 hello!

Before we start

Everyone should have R and R Studio installed.

Install R

Mac: https://cran.rstudio.com/bin/macosx/R-3.5.2.pkg

Windows: https://cran.rstudio.com/bin/windows/base/R-3.5.2-win.exe

Install R Studio

Mac: https://download1.rstudio.org/RStudio-1.1.463.dmg

Windows: https://download1.rstudio.org/RStudio-1.1.463.exe

And you should download the course materials here:

Open your browser and go to:

tinyurl.com/essamonash

(we'll do this later in the workshop as well)

An introduction to data science in R

Will Mackey and ESSA Monash

Monday 18th March, 2019

Workshop structure

- Getting started in R
- Scripts and coding
- Reading data
- Exploring with visuals
- Transforming our data
- Making maps
- *[Additional resources]*

Who am I?

I am Will (hello!), an Associate at the Grattan Institute.

Things I like:

- Data analysis and visualisation.
- Making R packages/functions.
- Write documentation and guides that make using R *easier*.

At Grattan, we use R for our data analysis and visualisations.

We use data to answer questions.

How Grattan uses R

Q: **Does living far away from your university increase your chance of dropping out?**

Wrangle a large dataset

- 16m anonymised observations with 120 variables

Join with other datasets

- join with 171m and 3.5m observation datasets

Generate new data

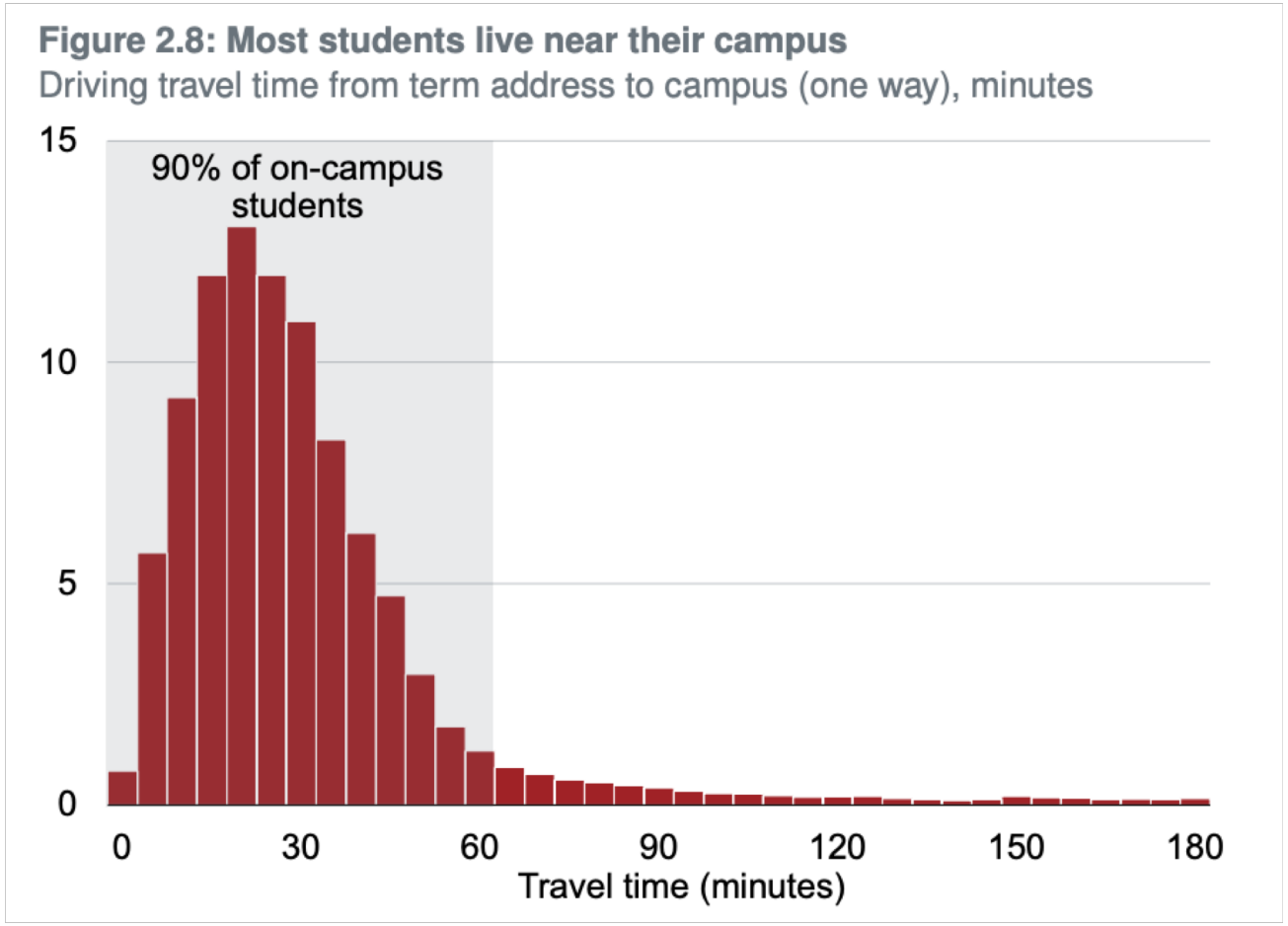
- driving time from each observed home postcode to campus, using a Google Maps API

Chart

- plot a distribution of travel distance

Analyse

- Logistic regression to understand the relationship between long travel times to university and degree completion.



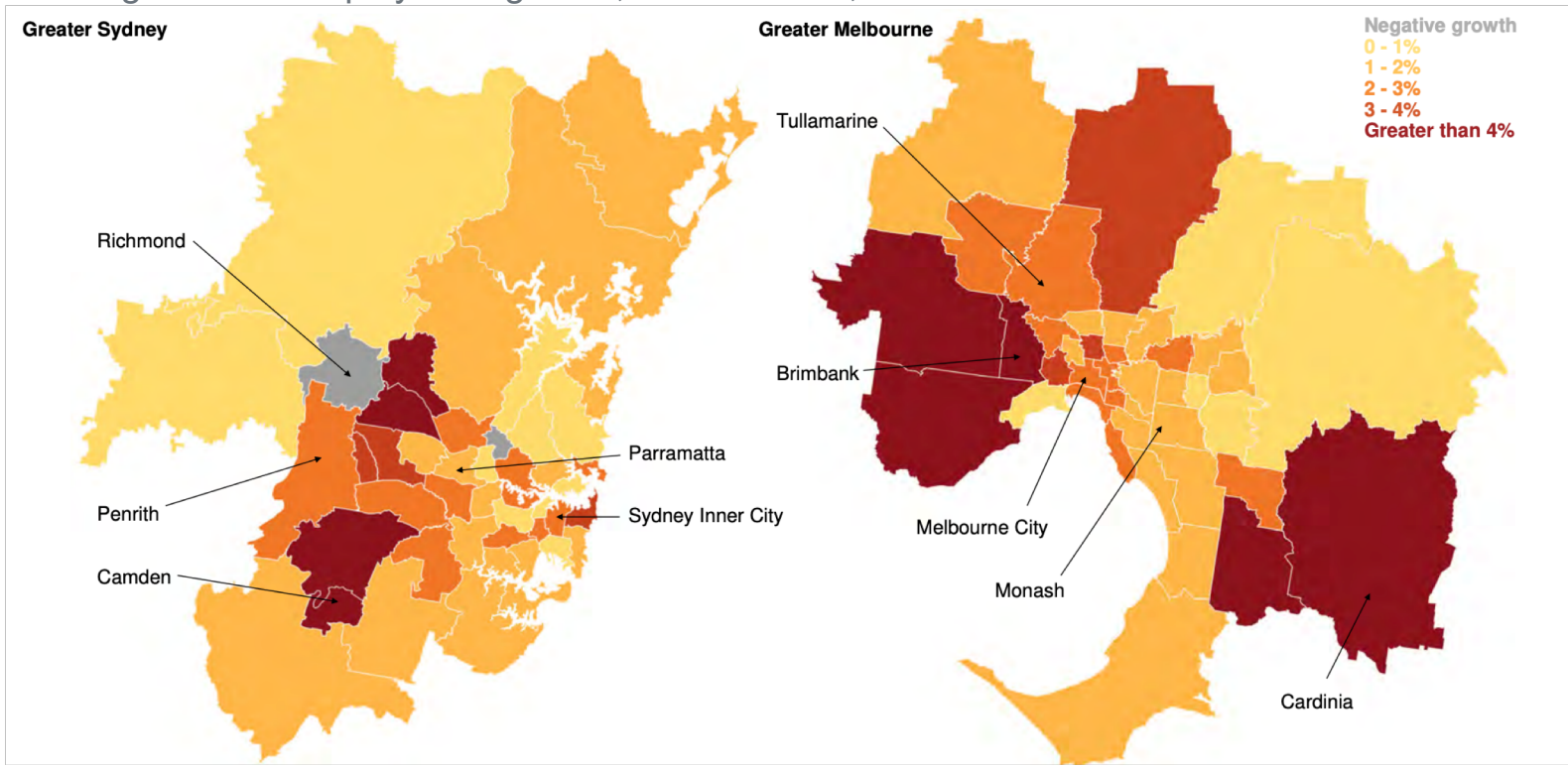
From *Cherastidtham, I., Norton, A., and Mackey, W. (2018). University attrition: what helps and what hinders university completion?. Grattan Institute*

How Grattan uses R

Q: **How has job growth been distributed around the major cities?**

- Get jobs data by area from the 2011 and 2016 Census
- Calculate growth in each area.
- Join with geometric data from the ABS.
- Plot maps of job growth.

Figure 2.5: In Sydney and Melbourne, jobs have grown in the CBD and central areas, but they've grown fastest in the outer areas
Average annual employment growth, 2011 to 2016, SA3



From *Terrill, M., Batrouney, H., Ha, James, and Hourani, D. (2018). Remarkably adaptive: Australian cities in a time of growth. Grattan Institute.*

Why are we all here?

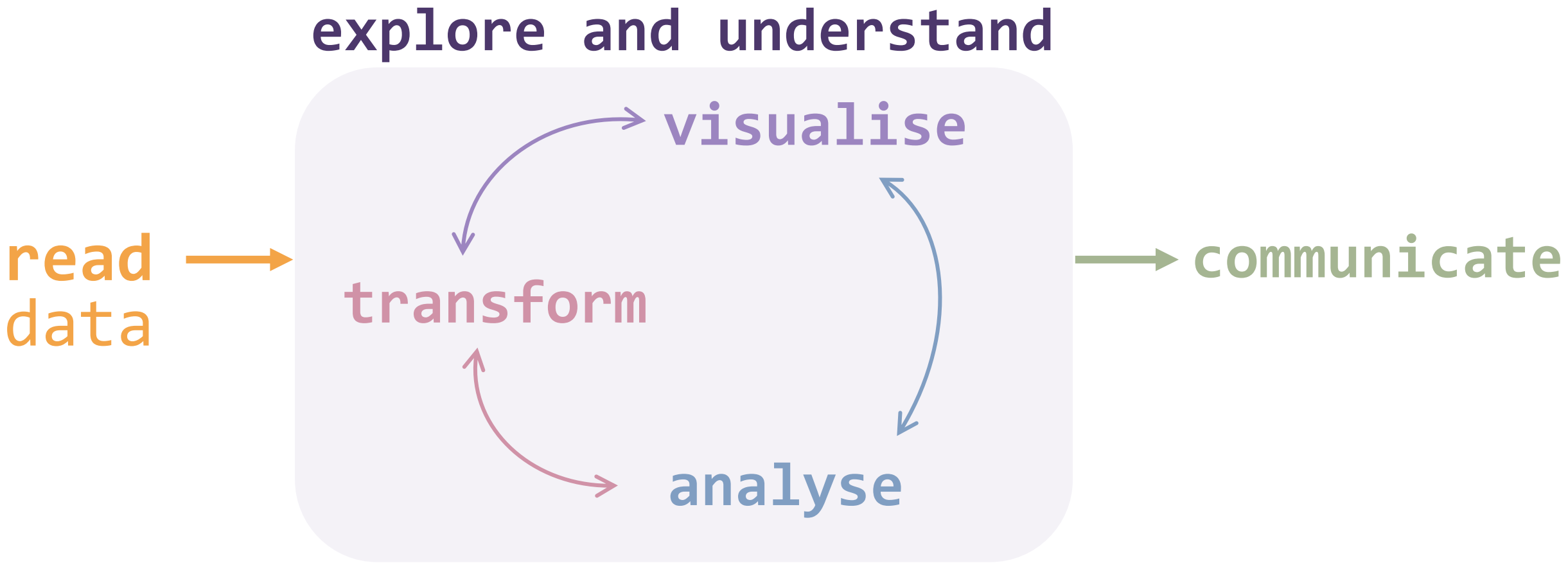
We are here to learn more about data science and R.

We are doing *this* because we want **extract insight** out of data.

We want to be able to **learn something** from the information we have.

Data it fun! But it can be hard to make sense of.

What we are doing with data science

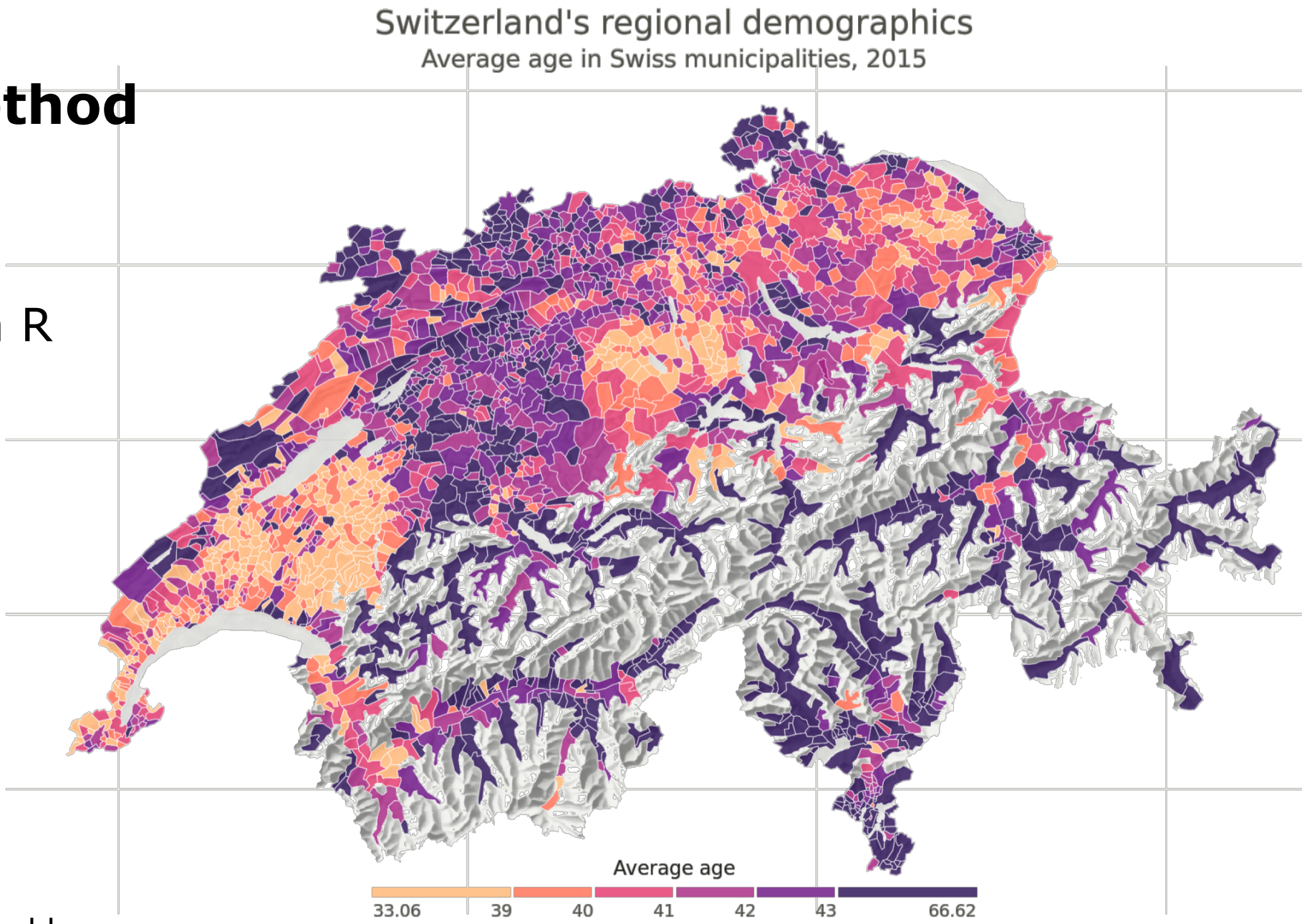


Model adapted from Wickham (2019) *R for Data Science*

The same method

Made by Timo Grossenbacher in R using `ggplot`.

It’s really nice.



The code used can be found here:

https://timogrossenbacher.ch/2016/12/beautiful-thematic-maps-with-ggplot2-only/

The purpose of today's workshop

I want to get you 'up and running' with R, and to see a bit of what R can do.

- How Grattan uses R
- How you can use R

I also want you to learn that **coding** is *learn-able*; bit by bit.

We'll go through **a lot of material in a short time**.

- You don't have to fully understand everything in this workshop.
- Just get a **vibe**.
- This wont be **on the exam**.

And I want to give you the confidence to look at the additional resources

- There are materials provided that you can go over later.
- From these, and others, you'll be able to do all the things you want to do.

And some notes before we start

You will make one-million errors.

- This will be frustrating and annoying.
- This is normal.

This error checking is called ‘debugging’.

- You will be bad it at to begin with.
- You will get better over time.

Make sure you close your brackets.

- Each (will need to be followed by a) at some point.
- Each “ will need to be followed by a ” at some point.

Help your neighbour

- Introduce yourself to your left and right.
- If you miss a step or can’t work something out, ask your neighbour.
- Receiving help from others is *wonderful*.
- Helping others is *wonderful*.

preamble

What is R?

R is a free, open-source statistical package.

It was created in New Zealand in the 1990s.

With more people using it, and more user-written functions and guides, it has become much easier to use.

A script-based language

R is a script-based program.

You write a list of instructions and R will follow it.

This is wonderfully handy:

- everything you have told it to do, it will do;
- you have a record of everything you have told it to do.

‘Coding’

Coding feels a bit intimidating. But it’s just a list of instructions.

If you have ever used Excel, you are a coder:

- You have used functions to create output. You have written code.

And we can greatly enhance the things we can do by using R.

‘Coding’

We often perform a list of instructions. For example, in Excel you:

1. Open a file.
2. Delete column that you don’t need.
3. Select all of your data and sort from highest to lowest.
4. Do a calculation using a formula.
5. Delete all the rows that have missing values.
6. Save your file.

We can do the same things when we code.

We just write down the list explicitly.

‘Coding’

We can do the same things when we code. We just write down the list explicitly.

This is beneficial:

- There is a **record** of everything you have done.
- It will do everything you want it to do **every time**:
  - You won’t miss a step!
  - And you can automate boring tasks.

Why R

There are **many** proprietary script-based programs for analysing data: `Stata`, `SAS`, `Matlab`, `Julia`, `Eviews`, `SPSS`.

Proprietary programs cost `$$$`:

- If you are not at a university/workplace that has the program, you can't use it.
- Or you will have to pay for a license yourself.
- You often can't **play** with the software outside of your institution

Why R

Proprietary programs are also *centrally controlled*.

The select functions are written by the company, and you can only use the set of functions they provide.

R thrives on user-written packages (collections of functions) that are available to everyone, for free. From a recent study:

> "In 2015, R added 1,357 packages, … or approximately 27,642 functions. During 2015 alone, R added more functions than SAS Institute has written in its entire history."

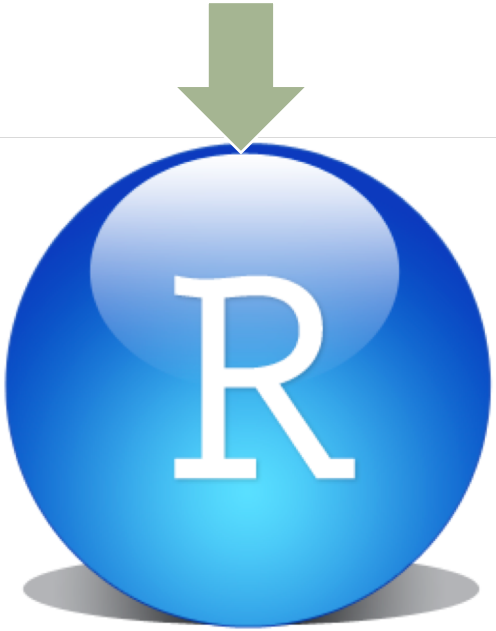
From r4stats.com/articles/popularity (the potential bias is indicated in its domain)

basics

Open R Studio

R
(not this one)

R Studio
(this one!)



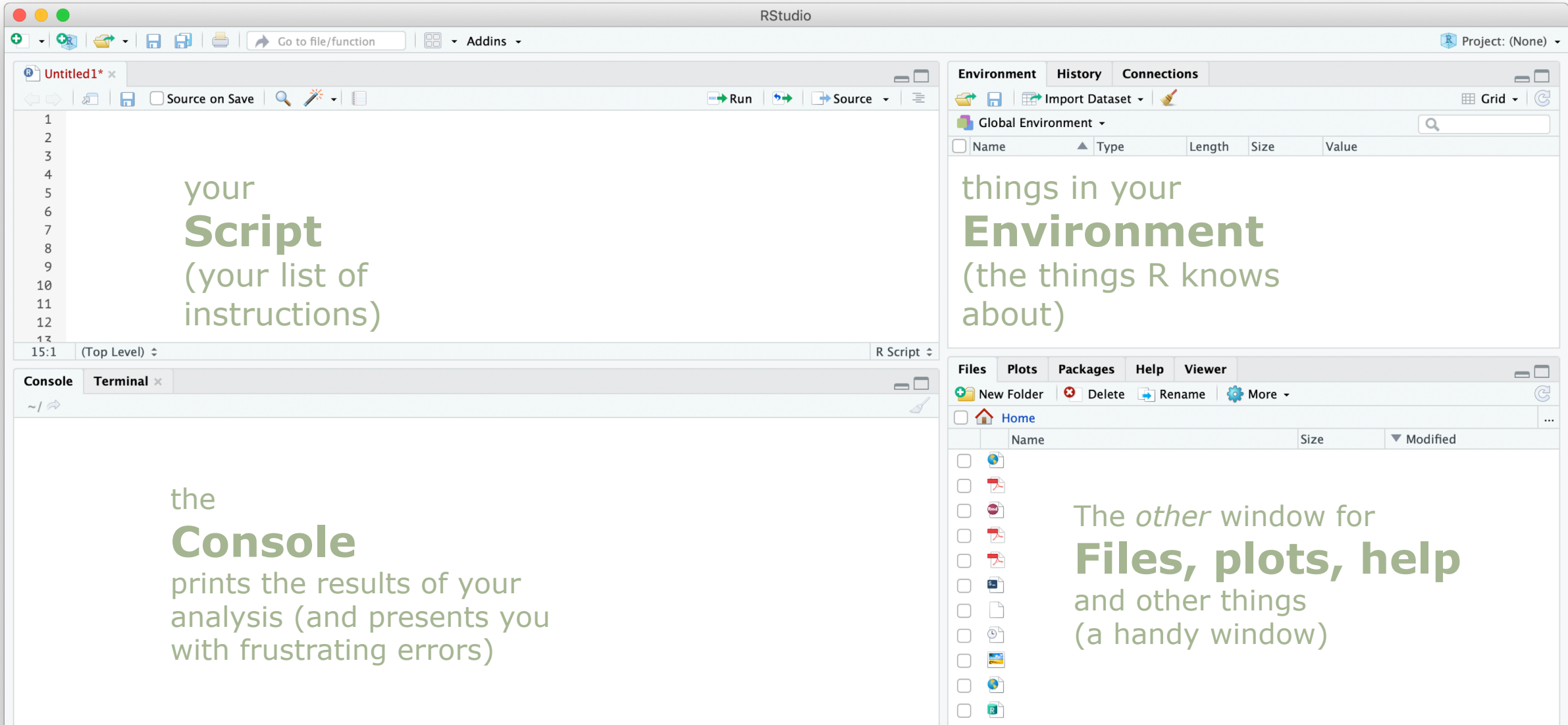
Open R Studio

From within R Studio, open a new script:

`File -> New File -> R Script`

What does R Studio look like?

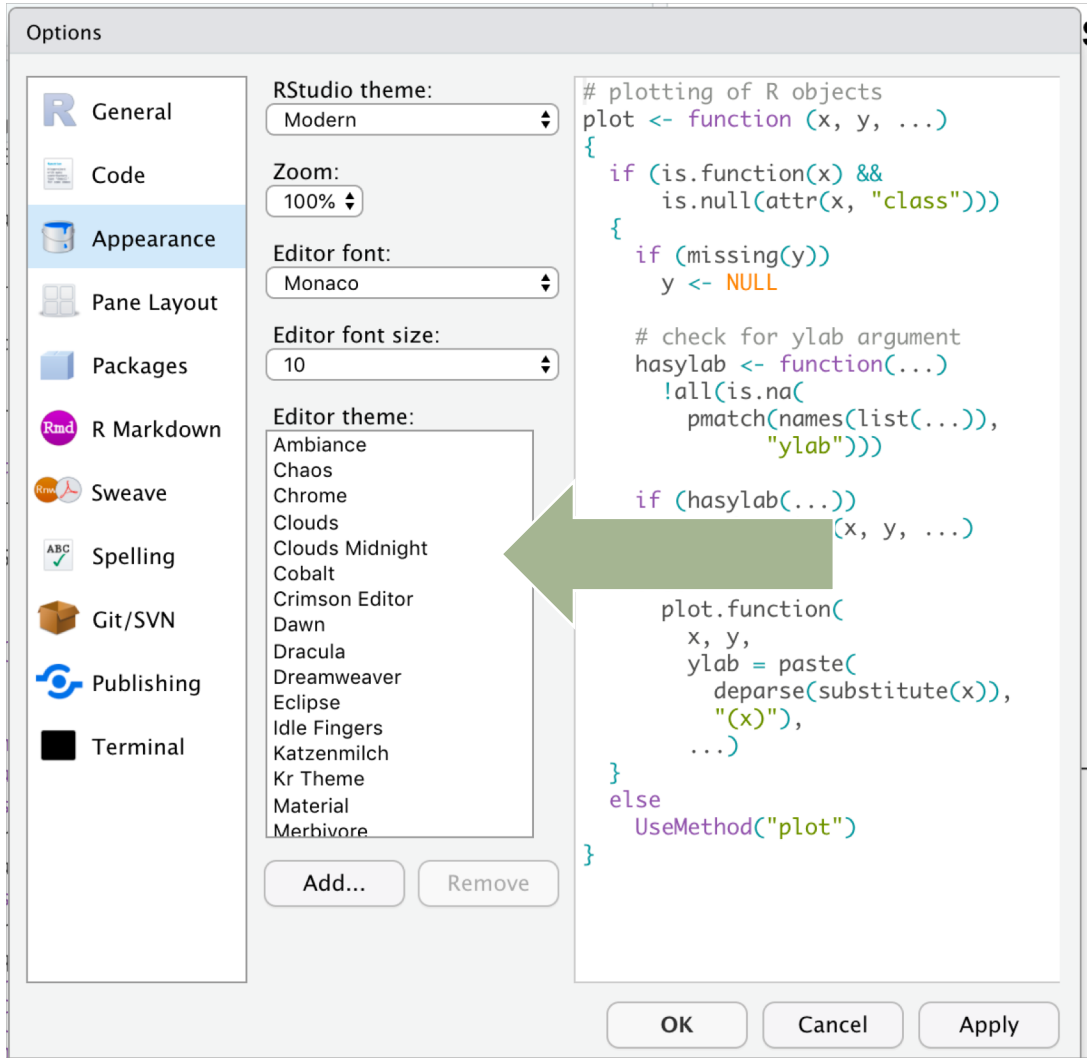
It should look at least a little bit like this



VERY IMPORTANT How to change the look of your R

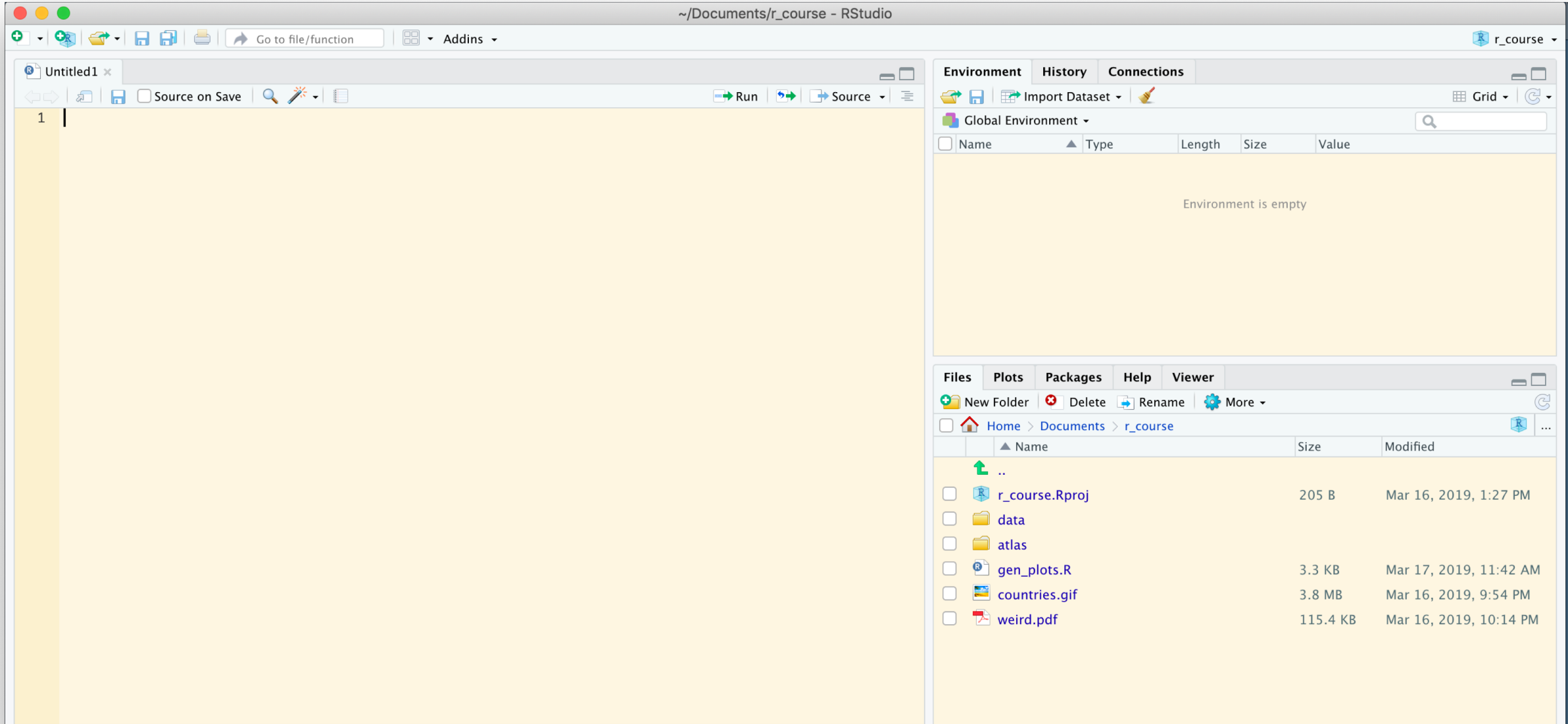
Go to:

Tools ->
Global options ->
Preferences ->
Appearance



And change the editor theme to something that you like.

What does your R Studio look like *now*?



Writing a script

Our script will be filled with a list of instructions for R to follow.

Let's do three things first:

1. Write a comment starting with # *(something R will ignore – it's just there for you)*

```
# Hello I am starting my R script
```

2. Define an **object**: assign the number 119 to the object named `goodnumber`

```
goodnumber <- 119
```

3. Calculate a thing:

```
2 * goodnumber
[1] 238
```

output is shown in the **console**

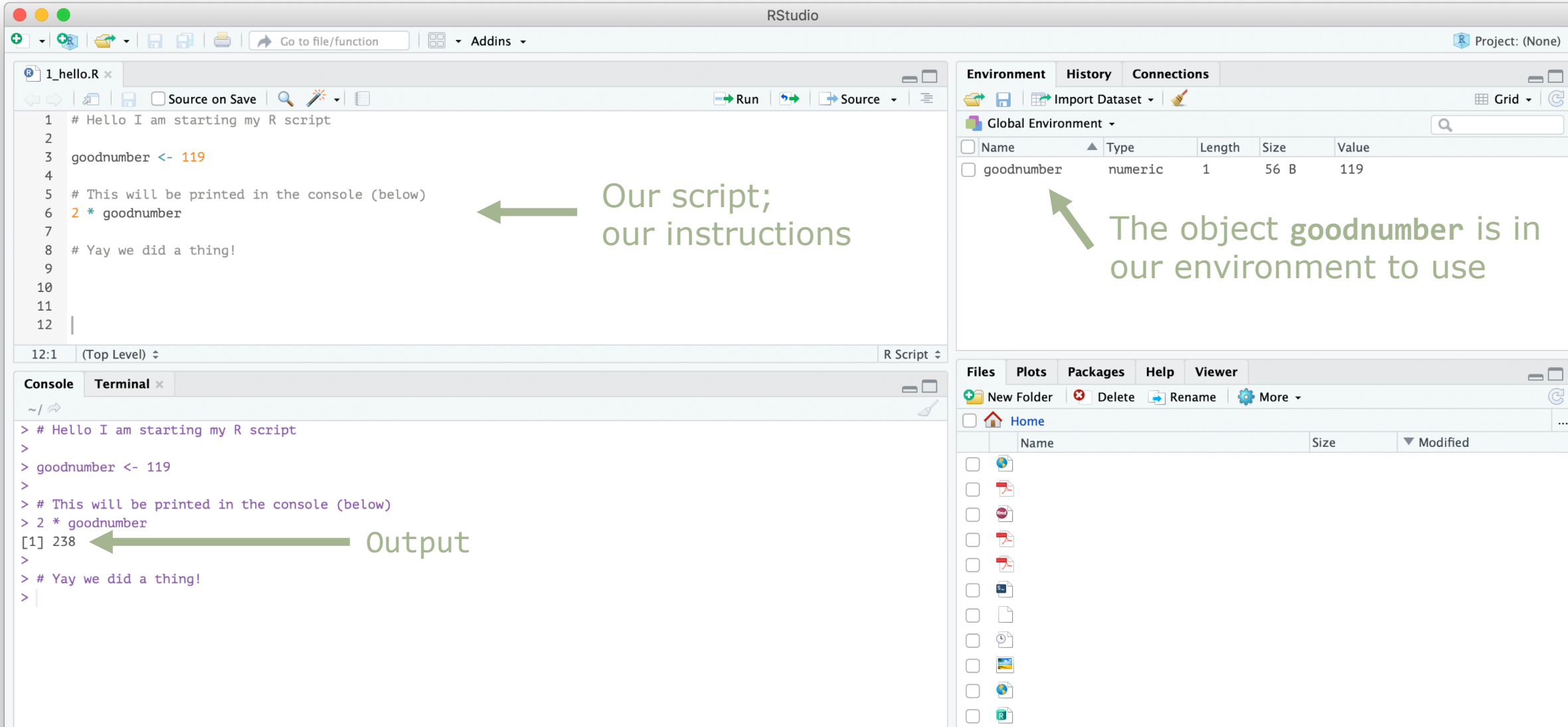
Note: running your code

To run a single line of code, the one your cursor is on:

`CTRL + ENTER` or `CMD + RETURN`

To run the whole script, click **Run** on the top-right of your script.

What does our screen look like now?



Exercise

Define an object to be equal to 90

```
my_number <- 90
```

Print your number to the console

```
print(my_number)
```

```
[1] 90
```

Multiply the object by 100

```
my_number * 100
```

```
[1] 9000
```

Divide the object by 10

```
my_number / 10
```

```
[1] 9
```

Functions

A **function** takes inputs (arguments) and produces outputs.

We can use the **c** function to combine *numbers* into a series of numbers (called a *vector*):

```
c(3, 4, 5)
```

```
[1] 3  4  5
```

Combine the numbers 3, 4 and 5 into a single vector

Functions

We can *nest* functions (put one inside another):


```
mean(c(3, 4, 5))
```

```
[1] 4
```

Combine the numbers 3, 4 and 5 into a vector, then take the mean of that vector.

But nesting is hard to read (especially when there are lots of steps)

Alternatively:

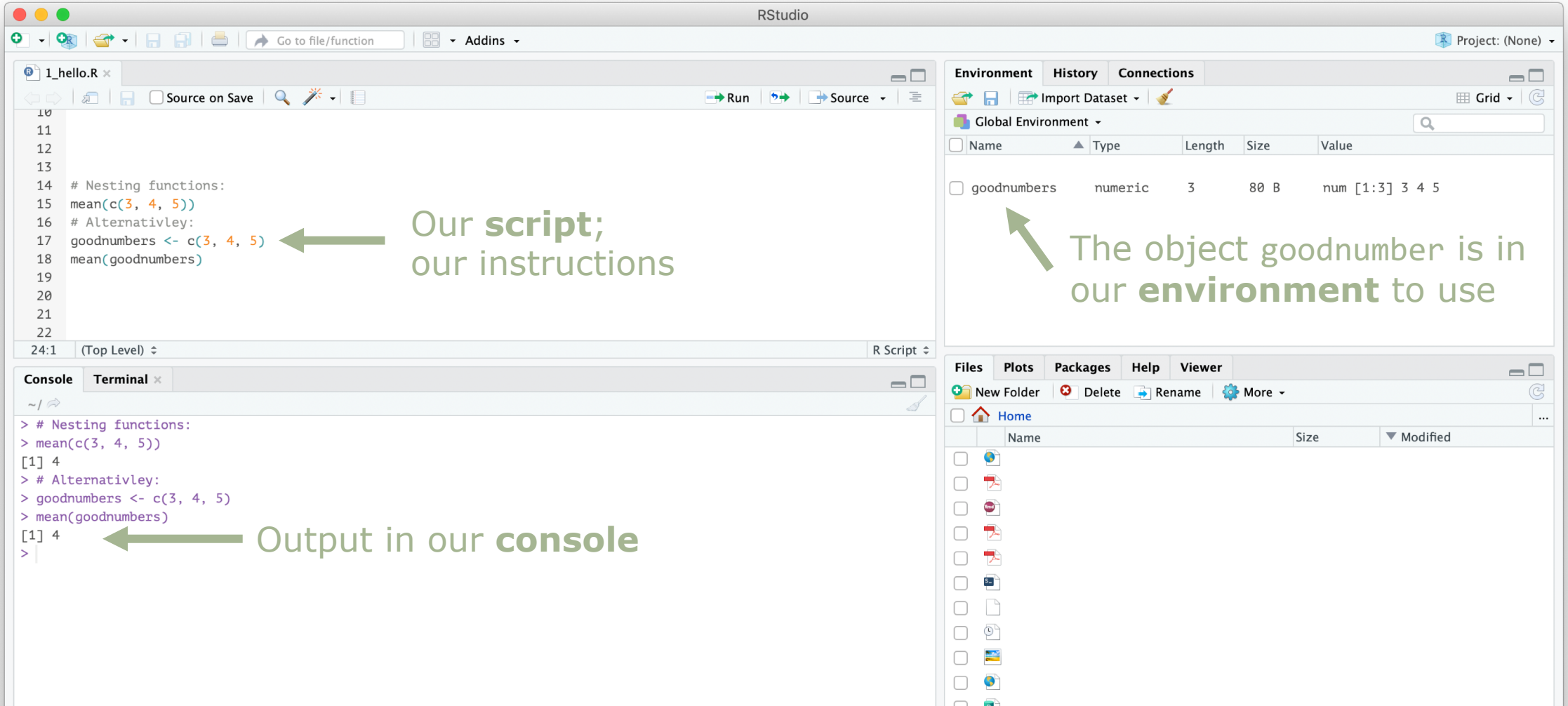
```
goodnumbers <- c(3, 4, 5)
mean(goodnumbers)
```

```
[1] 4
```

Combine the numbers 3, 4 and 5 into a vector and **assign** it to `goodnumbers`. Take the mean of `goodnumbers`.

Functions

We run our script and the console will show:



Looking at some data

R comes with some small datasets pre-installed.

One is called `mtcars`. You can look at it by printing it:

```
print(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

Hot tip

To find out more information about a function or dataset, use ?

```
?mean
```

```
?mtcars
```

Looking at some data

We can look at a single column using $:

```
print(mtcars$mpg)
```

Extract the mpg column from the mtcars dataset, and print it to the console

And we can use this to generate summary statistics about the dataset:

```
mean(mtcars$mpg)
```

```
[1] 20.09062
```

Extract the mpg column from the mtcars dataset, and then take the mean.

Exercise

Find the names of the variables in `mtcars` using the `names()` function.

```
names(mtcars)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"
 [7] "qsec" "vs"   "am"   "gear" "carb"
```

Assign the `wt` variable to an object.

```
weight <- mtcars$wt
```

Find the `median`, `mean` and standard deviation (`sd`) of your object.

```
mean(weight)
```

```
[1] 3.21725
```

```
median(weight)
```

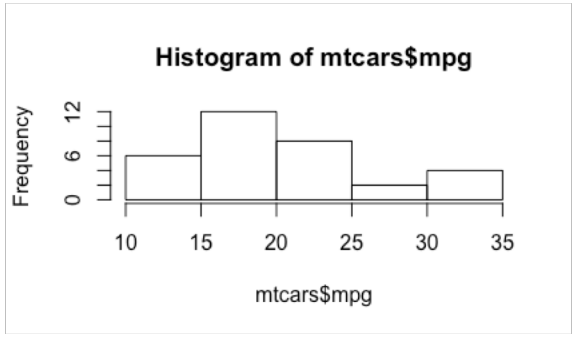
```
[1] 3.325
```

```
sd(weight)
```

```
[1] 0.9784
```

Plot a histogram of your object using `hist()`.

```
hist(weight)
```



Packages

R uses **functions** to do things. A **package** is a collection of functions.

Some packages are installed when you install R.

this is called “base R”

Other user-written packages can be installed using:

```
install.packages(“nameofpackage”)
```

Packages

The `tidyverse` is a package

(Well, the `tidyverse` is a collection of packages)

They are intuitive and consistent.

They help you **read**, **transform** and **visualise** your data with less hassle.

It’s best to **see** how these functions are used.

Packages

The `tidyverse` is a package

You can install the `tidyverse` by writing this in your **console**:

```
install.packages("tidyverse")
```

When you do this, your console **will explode with activity**. This is okay!

You only have to install a package **once**.

Packages

The `tidyverse` is a package

You have installed the `tidyverse`. Now you have to load it using **`library`**:

```
library(tidyverse)
```

You have to load the package **each time you use R**.

And now we're coders

We have R and R Studio up-and-running.

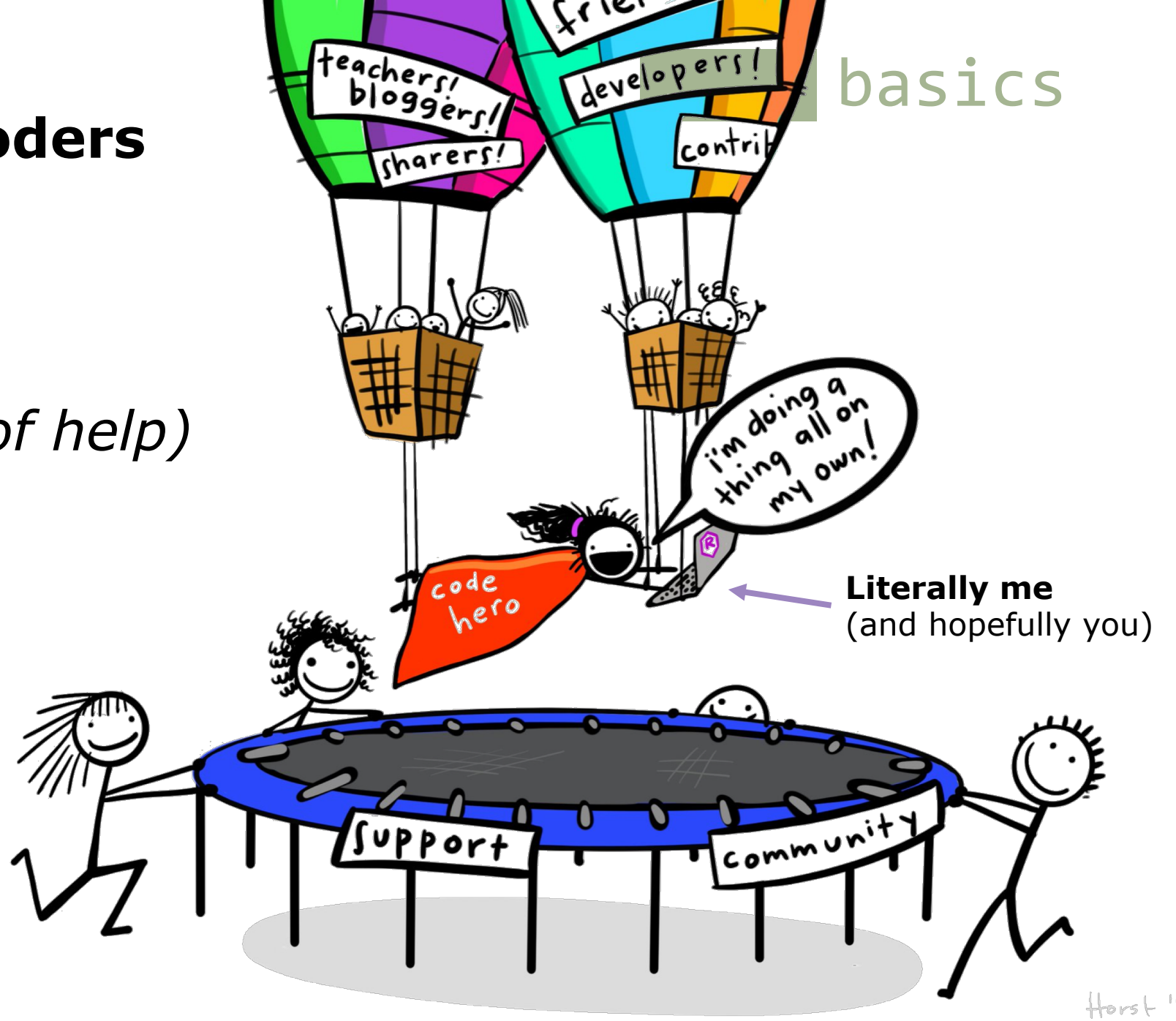
We have written a comment, defined an object and run a simple analysis.

Now we just have to learn how to ***do more***:

a process you will continue as long as you're using R.

And now we're coders

(and we have lots of help)



'Me "working independently"' by Allison Horst, @allison_horst


read
data

We need to get data into R

This means:

1. Starting an **R Project**.
2. Finding data.
3. Saving data onto your computer.
4. Reading the data into R.

read

We need to get data into R

[important that you do this bit!]

We're going to cheat a bit here

We have an **R Project** with **data** and a nice folder structure set up for us. To download it, go to this address in your web browser:

tinyurl.com/essamonash

Then:

Open this zipped folder and **move the zipped folder** somewhere nice, like your Desktop or Documents folder.

read

We need to get data into R

[important that you do this bit!]

We're going to cheat a bit here

With the **`r-course_essa-monash`** folder unzipped and saved somewhere:

- In R Studio:

  - Ignore or save your current scripts somewhere (we won't need them later).
  - Click **`File -> Open Project`**,
  - Then navigate to your **`r-course_essa-monash`** folder, and
  - Open the **`r-course_essa-monash.Rproj`** file.

read

We need to get data into R

[don't do this bit now – it's already set up**]**

1. Starting an R Project

To read things into and export out of R, it needs to know where it is.

A handy way to do this is by setting up an an R Project.

See a full discussion of this at https://r4ds.had.co.nz/workflow-projects.html

read

We need to get data into R

[don't do this bit now – it's already set up**]**

1. Starting an R Project

To start a new R project:

Within R Studio, click `File` -> `New Project`

Select `New Directory` -> `New Project`

(then follow the prompts)

read

We need to get data into R

[don't do this bit now – it's already set up**]**

2. Finding data

We are going to use a 'clean' version of the `gapminder` dataset.

It contains information about income and life expectancy of countries over time.

The gapminder dataset was created by https://www.gapminder.org/.
The cleaned version was made by Jenny Bryan and can be found at https://github.com/jennybc/gapminder.

read

We need to get data into R

[don't do this bit now – it's already set up**]**

3. Saving data on your computer.

This usually involves going to a website and downloading a file (eg an Excel file or csv file).

Then we would create a folder called **`data`** in our project folder: this is where would store our **original** data.

read

We need to get data into R

4. Reading data into R

We have our data. It is saved in our data folder.

We need to read it into our R environment.

We need to get data into R

4. Reading data into R

We will use the `read_csv` function from the tidyverse.

First, we have to load the tidyverse:

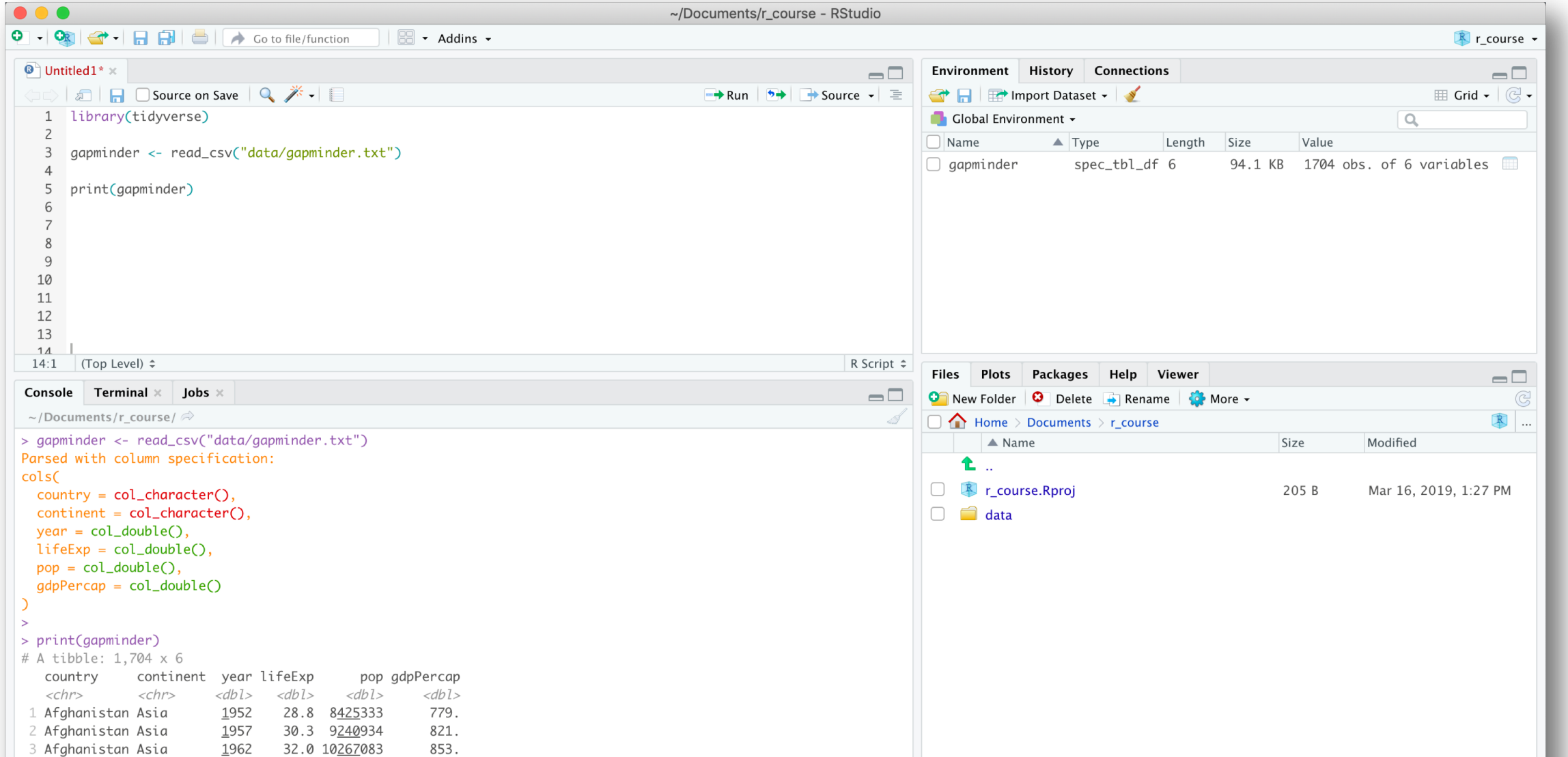
```
library(tidyverse)
```

Then, read our data file and assign it to an object:

```
gapminder <- read_csv(“data/gapminder.csv”)
```

read

All together now: our R Studio should look like this



read

We have our data in R

We have the tidyverse loaded.

We have read the `gapminder` dataset.

The `gapminder` dataset is in our environment.

Wonderful. Let’s do more!

Exercise

Find the **names** of the variables in `gapminder`.

```
names(gapminder)
```

View the `gapminder` dataset using the **View()** function.
Note the capital V.

```
View(gapminder)
```

Look at the **unique** values in the **year** column **$**.

```
unique(gapminder$year)
```

Create a **hist**ogram of the `gdpPercap` variable.

```
hist(gapminder$gdpPercap)
```

visualise

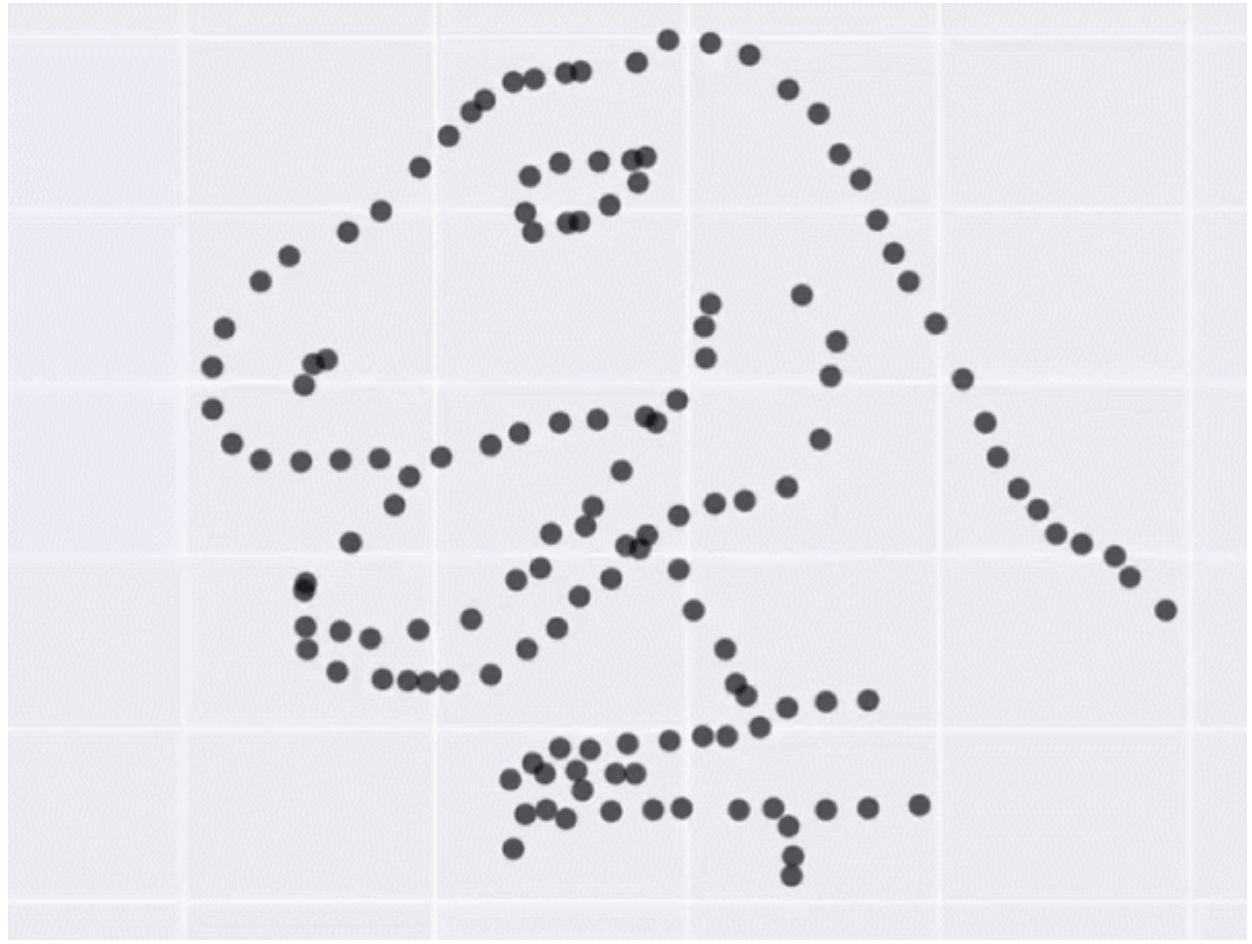
Visualising our data

It is important (and fun) to visualise our data.

The Datasaurus Dozen:

- **Averages** of the X and Y values are the same.
- **Standard deviation** is the same.
- **Same correlation** between the X and Y axes.

Scatterplots of the 'Datasaurus Dozen



From *Matejka & Fitzmaurice (2019) Same Stats, Different Graphs. See the summary here:* https://www.autodeskresearch.com/publications/samestats

Visualising our data

It is important (and fun) to visualise our data.

We do this with the `ggplot` package.

It requires:

- **Data:** what data to use.
- **Aesthetics:** mapping your data to aspects of the chart:
  - X axis, Y axis, colour, size, SHAPE, etc.
- **Geom:** how to put your aesthetics on the chart.

Visualising our data

visualise

don’t copy this code down; just watch for now

Data:

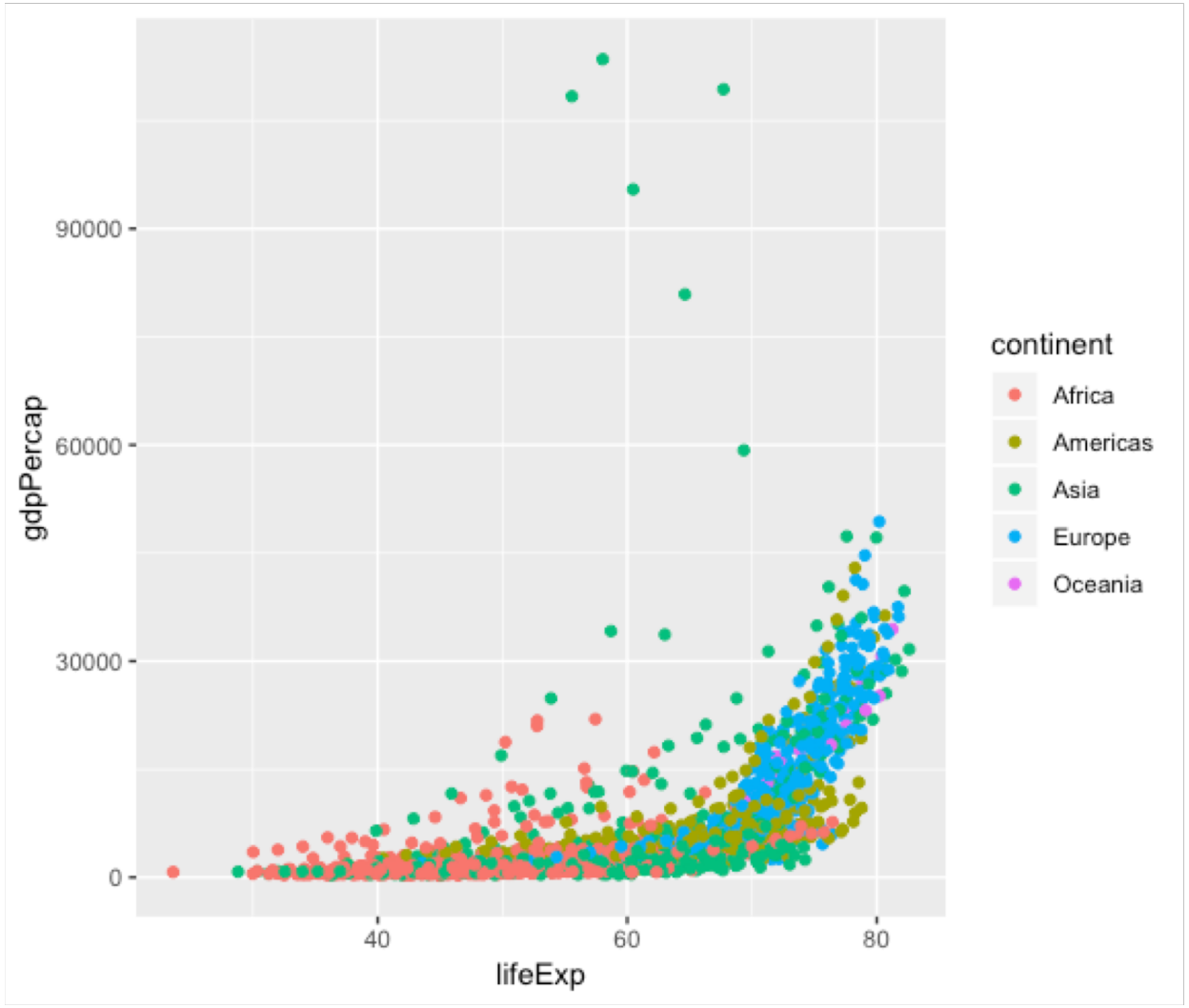
Aesthetics:

Geom:

```
ggplot(data = gapminder) +

aes(x = lifeExp,
    y = gdpPercap,
    colour = continent) +

geom_point()
```

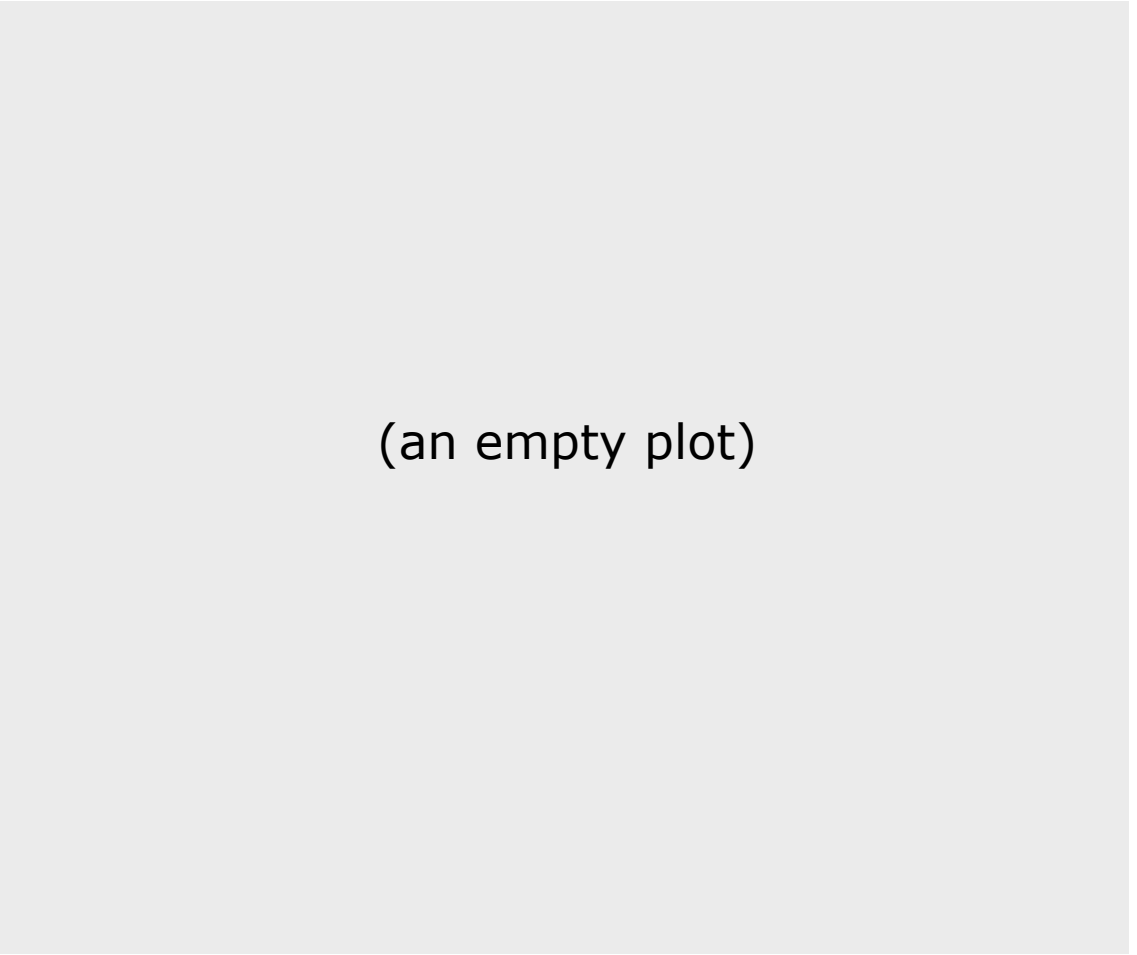


Visualising our data: step by step

visualise

don’t copy this code down; just watch for now

```
ggplot()
```



Visualising our data

visualise

don’t copy this code down; just watch for now

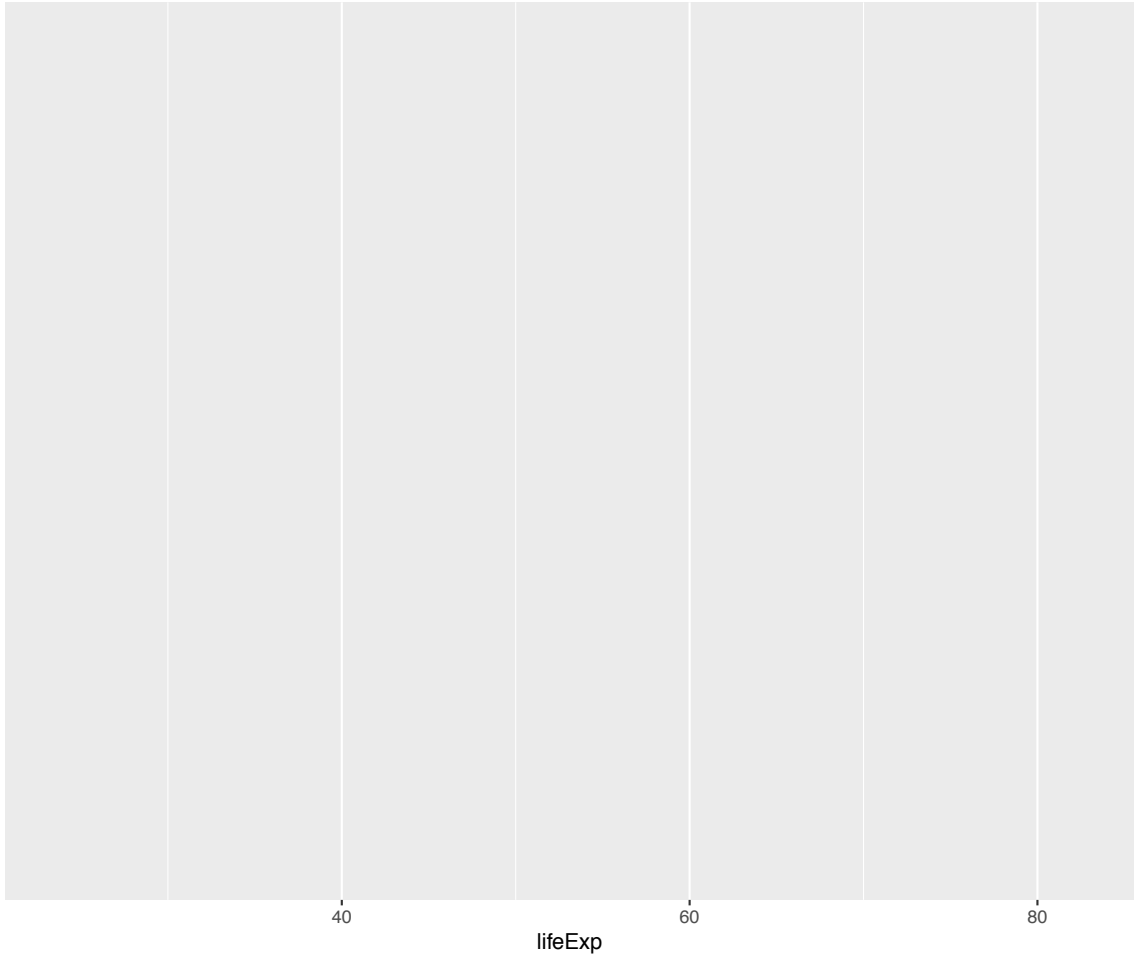
```
ggplot(data = gapminder)
```

Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp)
```

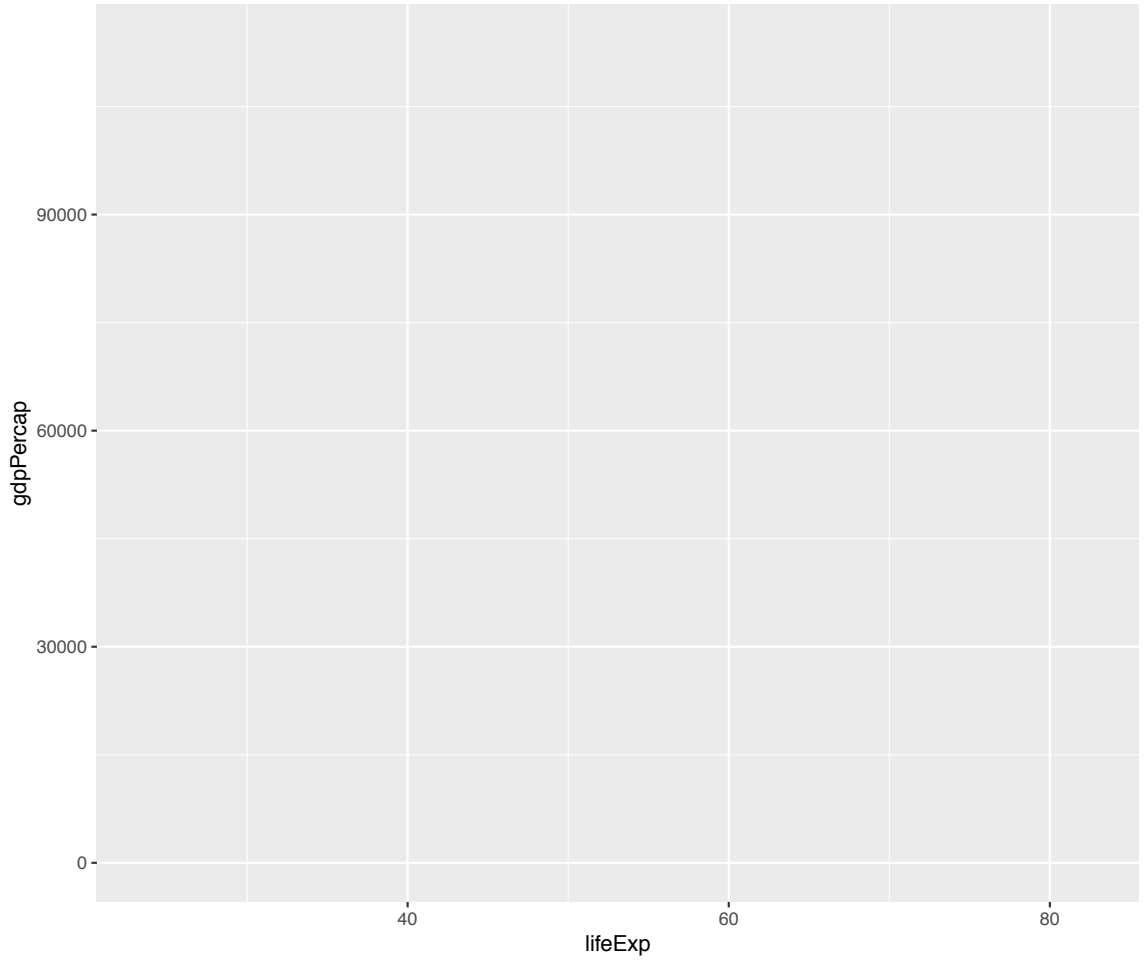


Visualising our data

visualise

don’t copy this code down; just watch for now

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

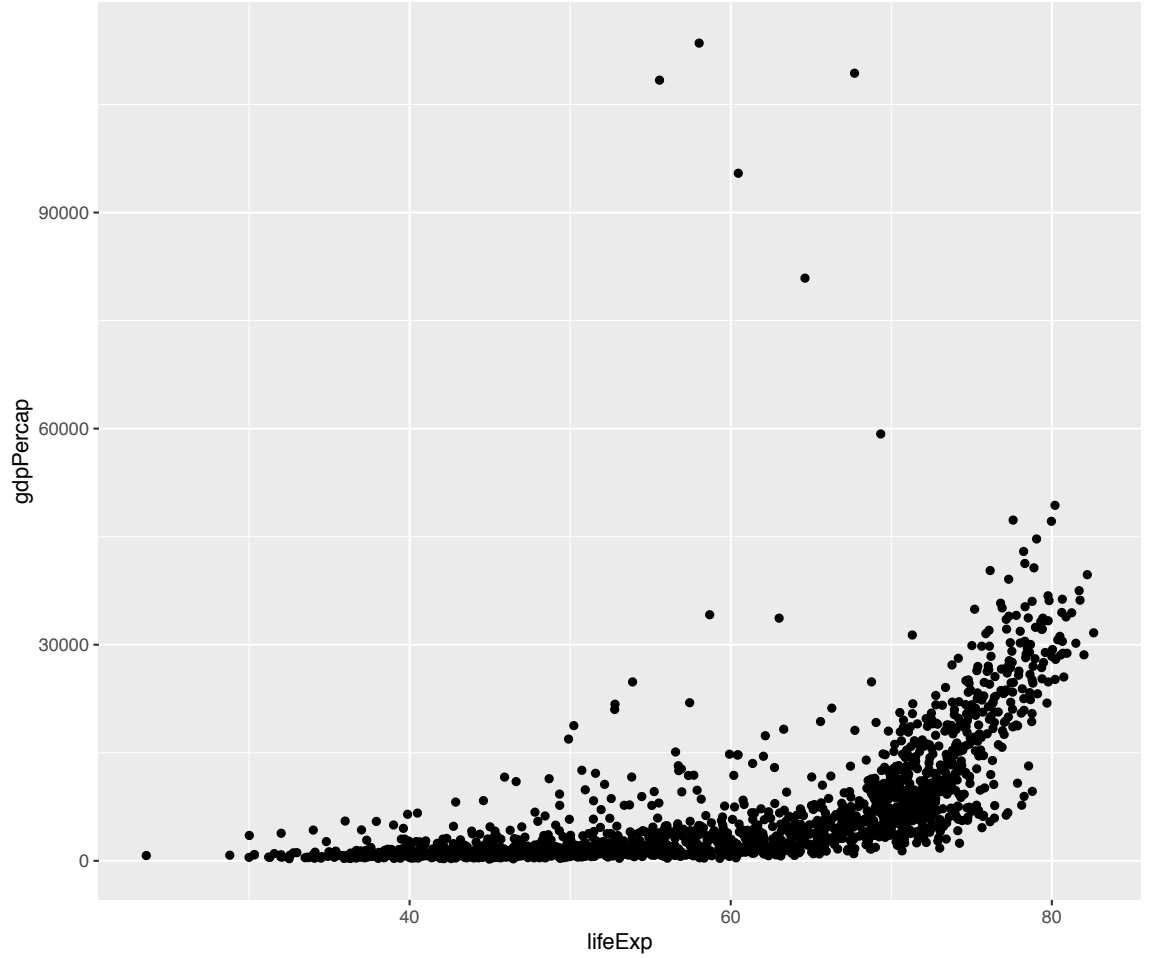


Visualising our data

visualise

don’t copy this code down; just watch for now

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



Visualising our data

visualise

don’t copy this code down; just watch for now

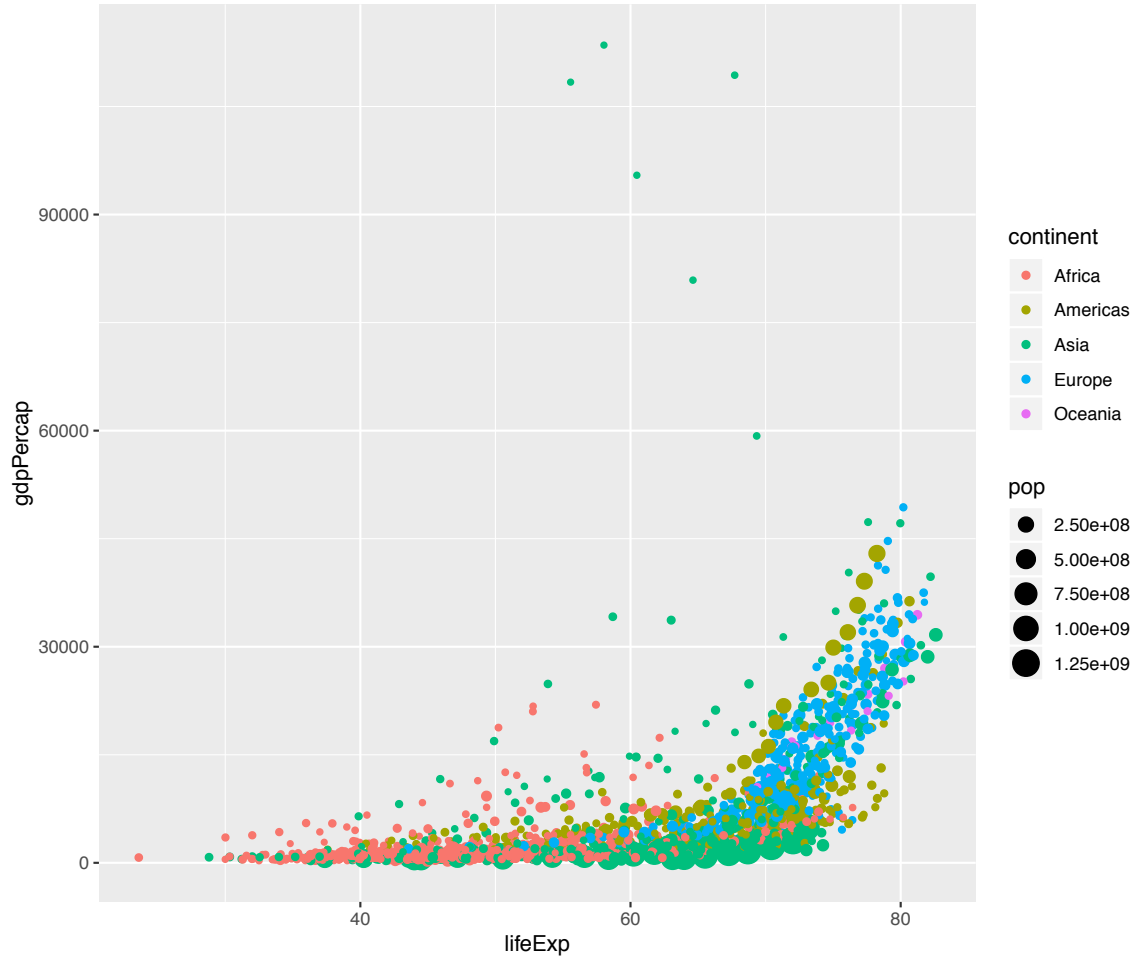
```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent) +
  geom_point()
```

Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point()
```

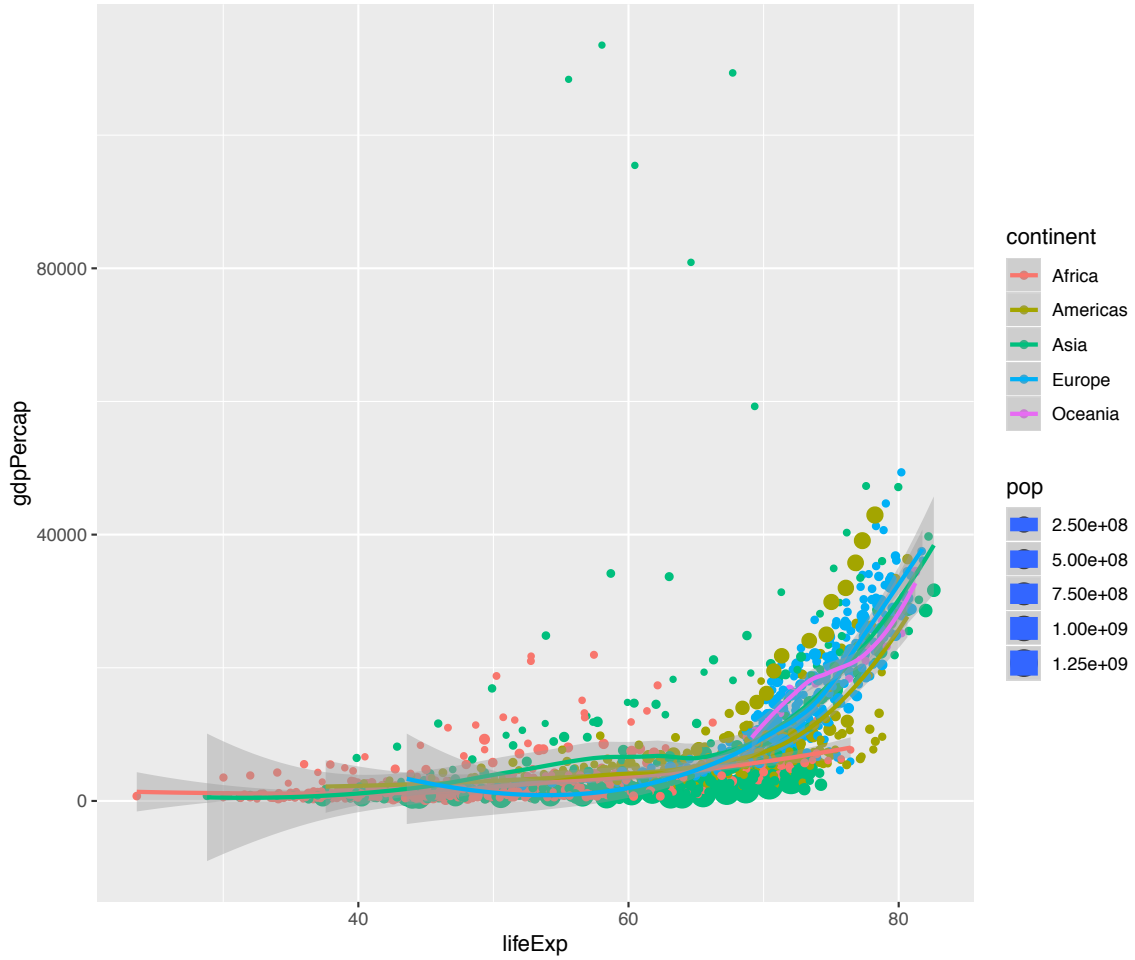


Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point() +
  geom_smooth()
```

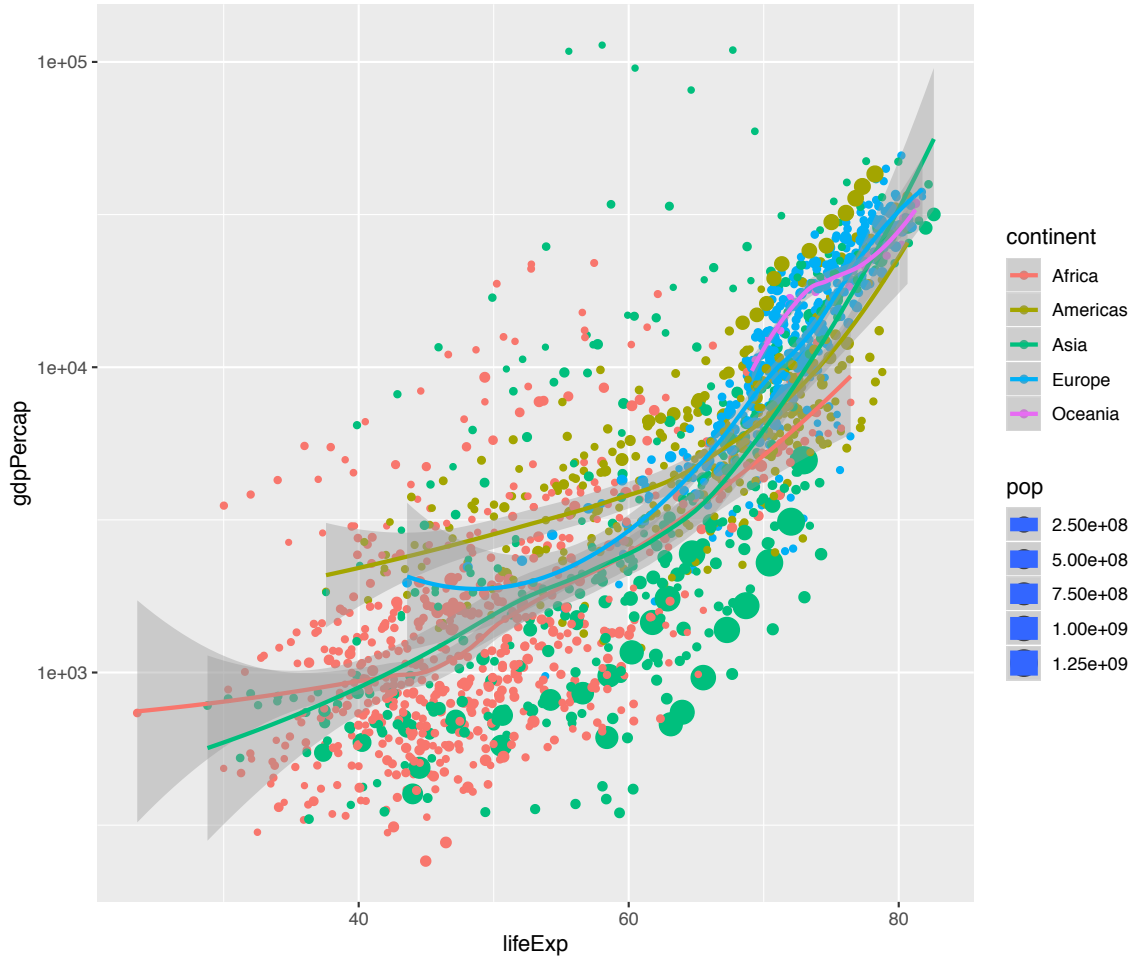


Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point() +
  geom_smooth() +
  scale_y_log10()
```

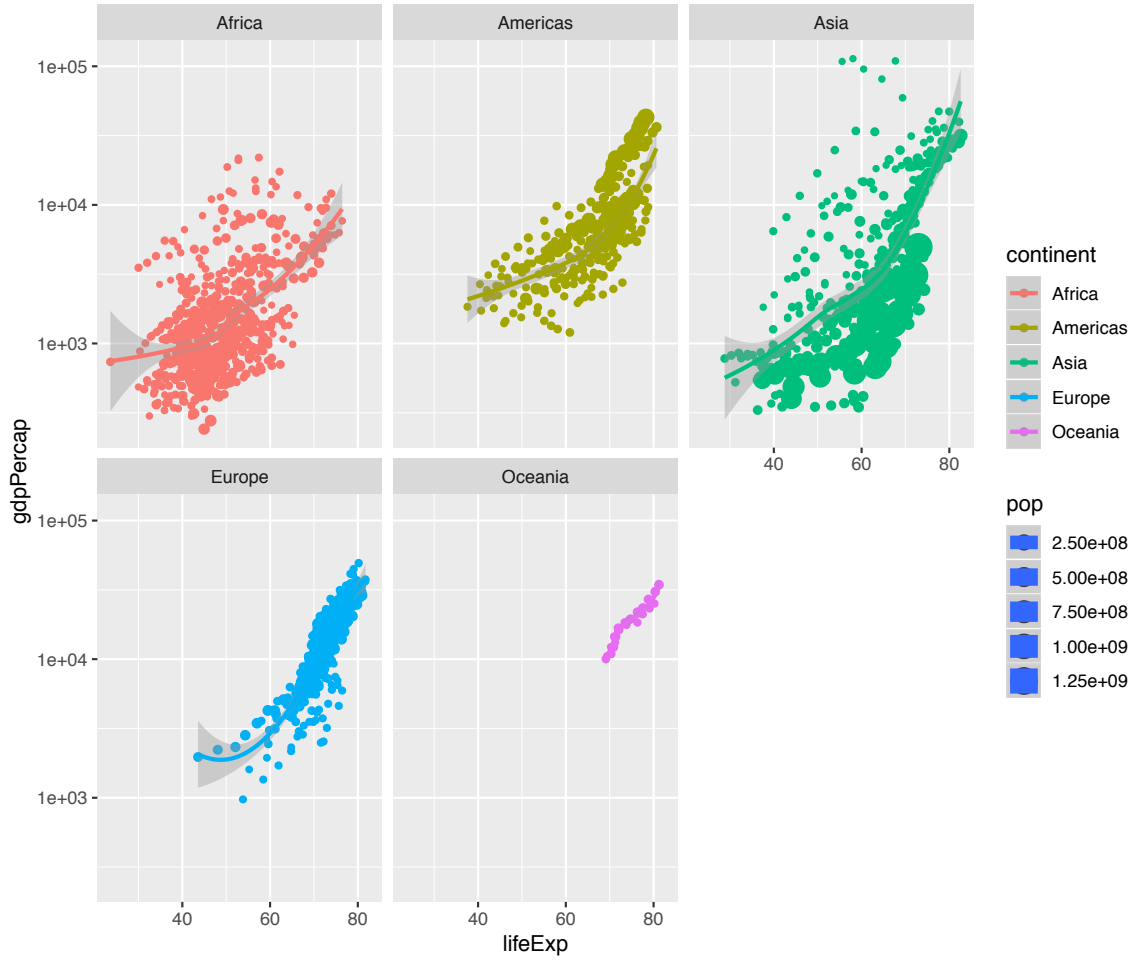


Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point() +
  geom_smooth() +
  scale_y_log10() +
  facet_wrap(~continent)
```

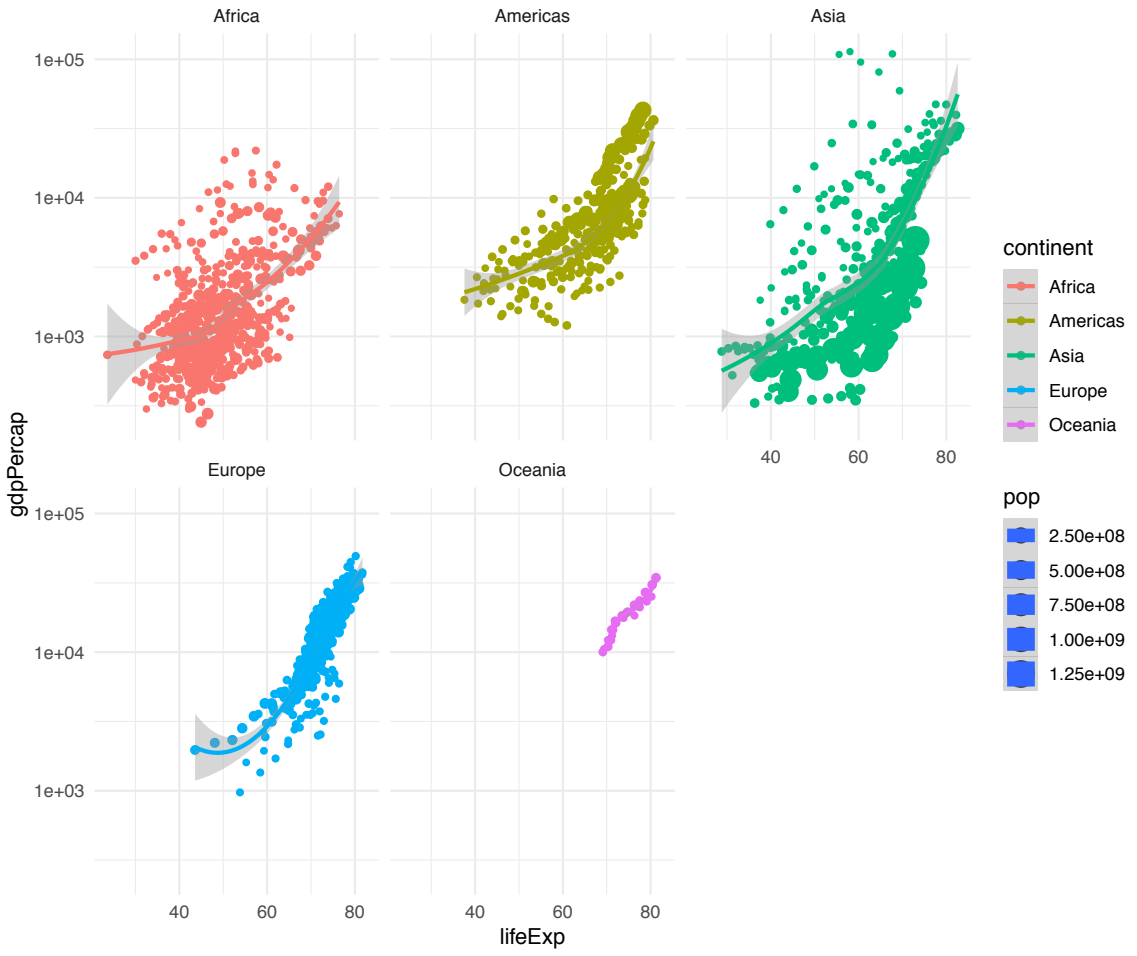


Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point() +
  geom_smooth() +
  scale_y_log10() +
  facet_wrap(~continent) +
  theme_minimal()
```

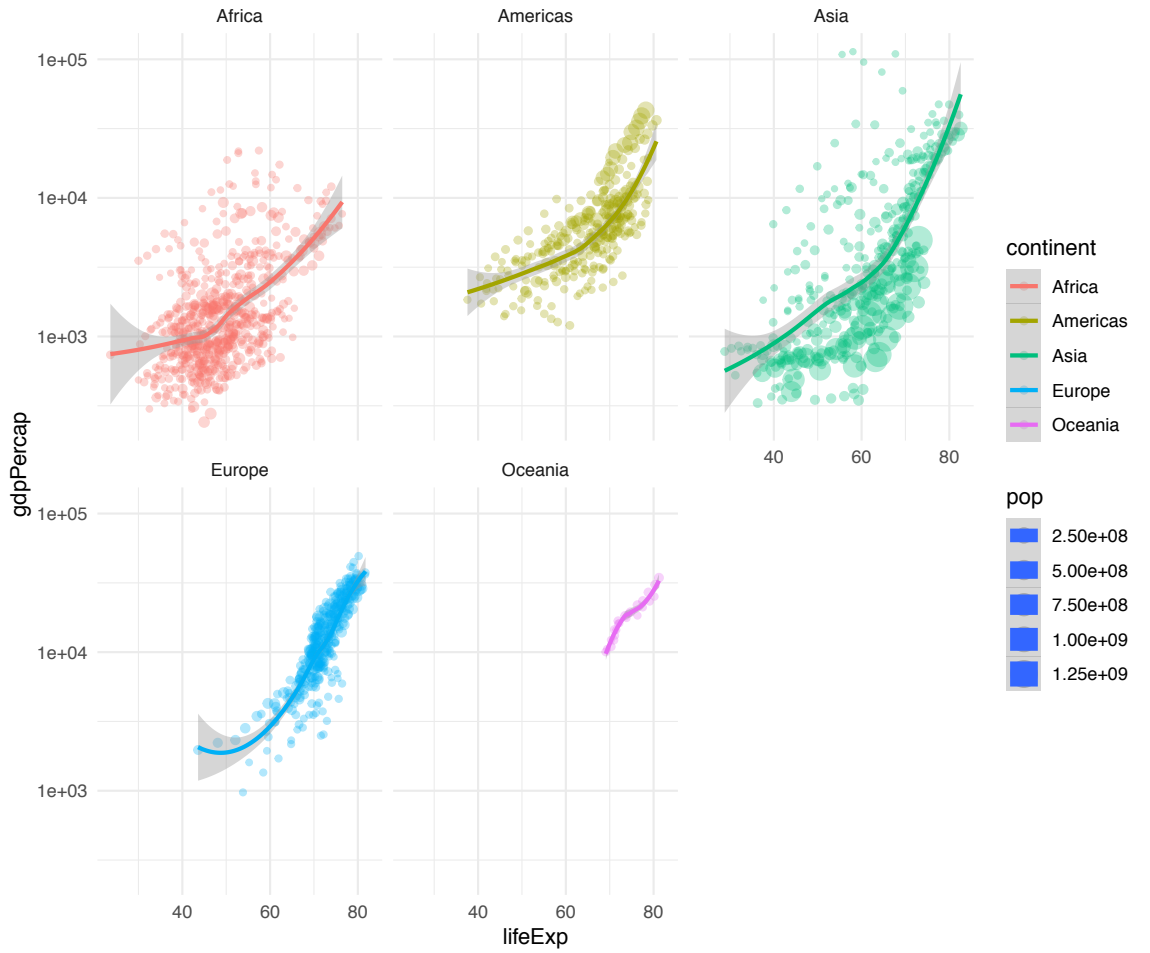


Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point(alpha = 0.3) +
  geom_smooth() +
  scale_y_log10() +
  facet_wrap(~continent) +
  theme_minimal()
```



Visualising our data

visualise

don’t copy this code down; just watch for now

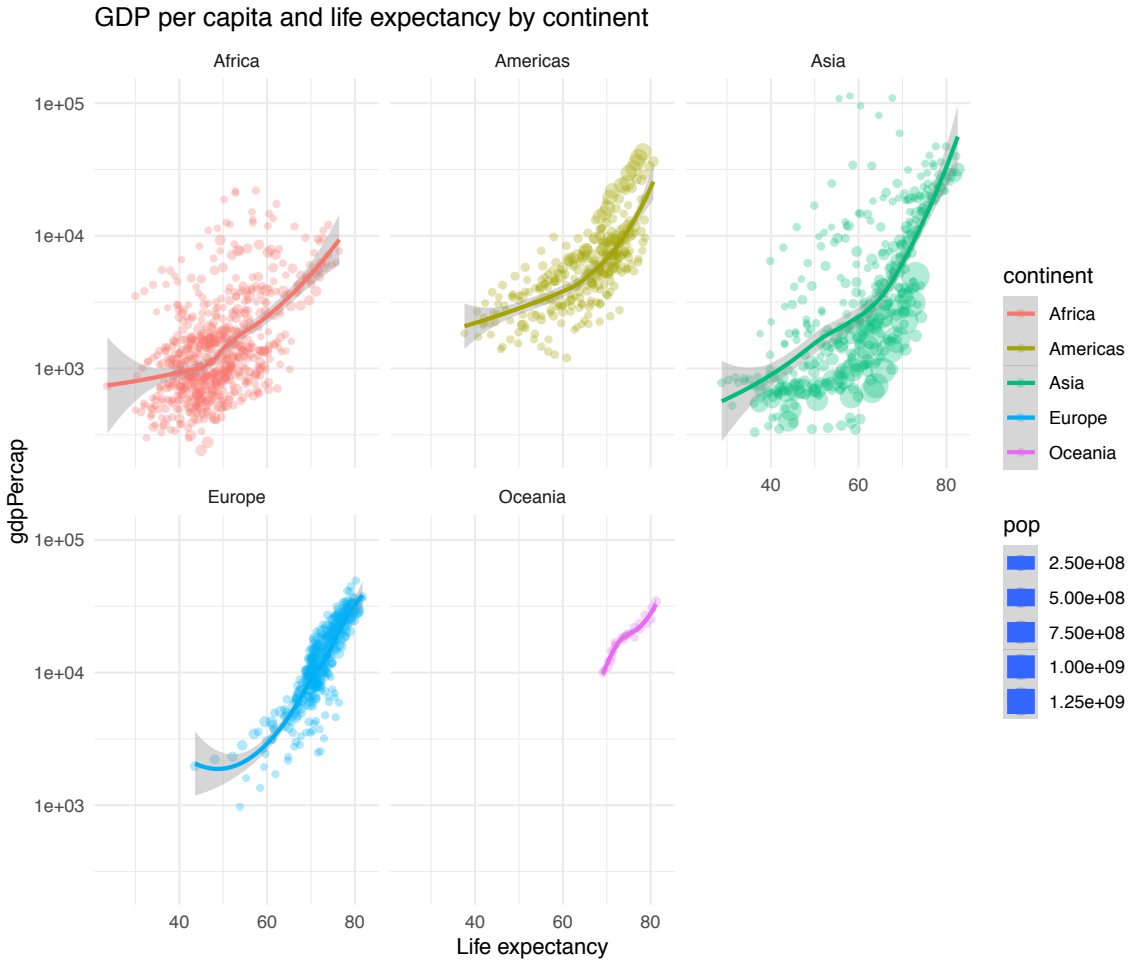
```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point(alpha = 0.3) +
  geom_smooth() +
  scale_y_log10() +
  facet_wrap(~continent) +
  theme_minimal() +
  labs(title = "GDP per capita and life expectancy by continent")
```

Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point(alpha = 0.3) +
  geom_smooth() +
  scale_y_log10() +
  facet_wrap(~continent) +
  theme_minimal() +
  labs(title = “GDP per capita and life expectancy by continent”,
       x = “Life expectancy”)
```

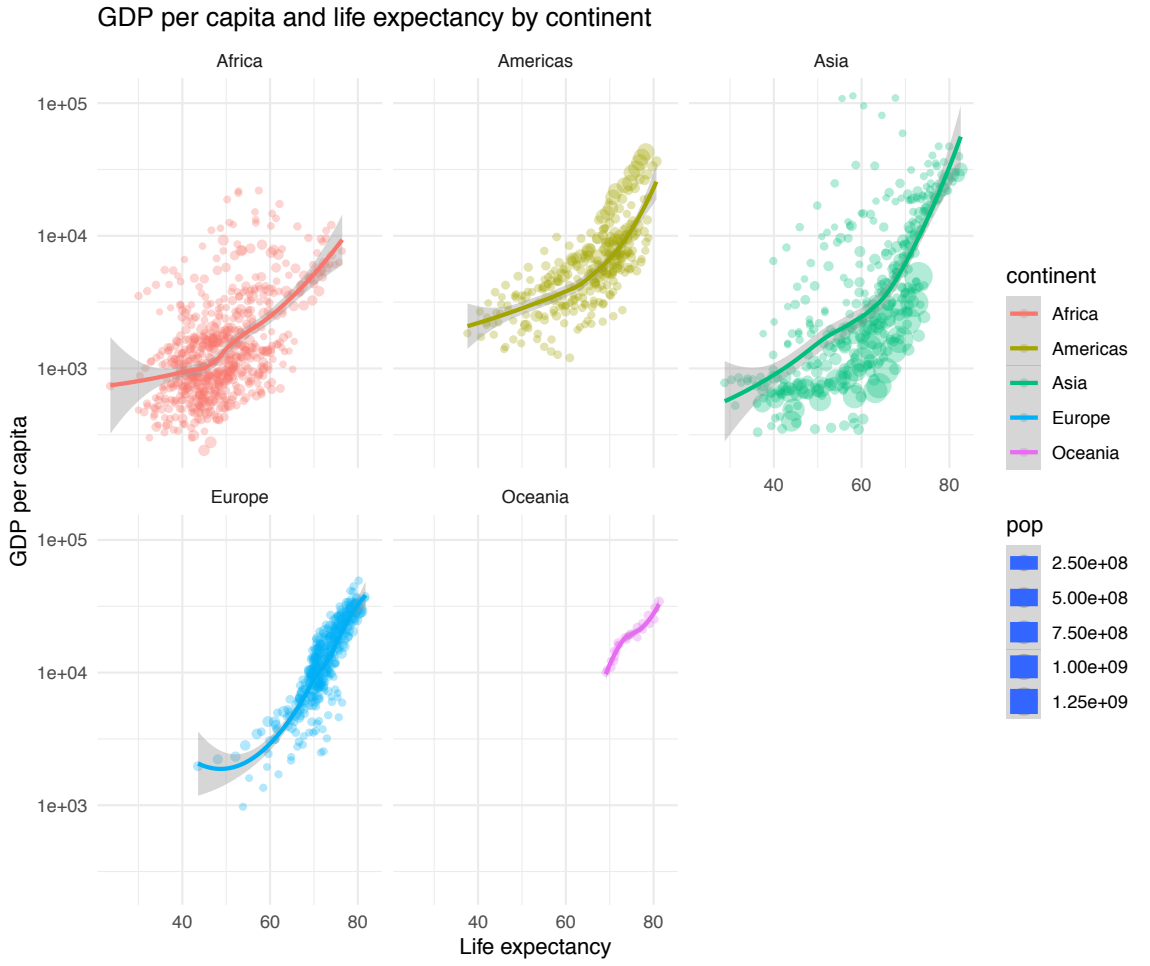


Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point(alpha = 0.3) +
  geom_smooth() +
  scale_y_log10() +
  facet_wrap(~continent) +
  theme_minimal() +
  labs(title = “GDP per capita and life expectancy by continent”,
       x = “Life expectancy”,
       y = "GDP per capita”)
```



Visualising our data

visualise

don’t copy this code down; just watch for now

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point(alpha = 0.3) +
  geom_smooth() +
  scale_y_log10() +
  facet_wrap(~continent) +
  theme_minimal() +
  labs(title = “GDP per capita and life expectancy by continent”,
       x = “Life expectancy”,
       y = "GDP per capita”,
       colour = "Continent", size = "Population”,
       caption = "Source: Gapminder https://www.gapminder.org/.")
```

Visualising our data

Recreate and save the final chart by copying the code:

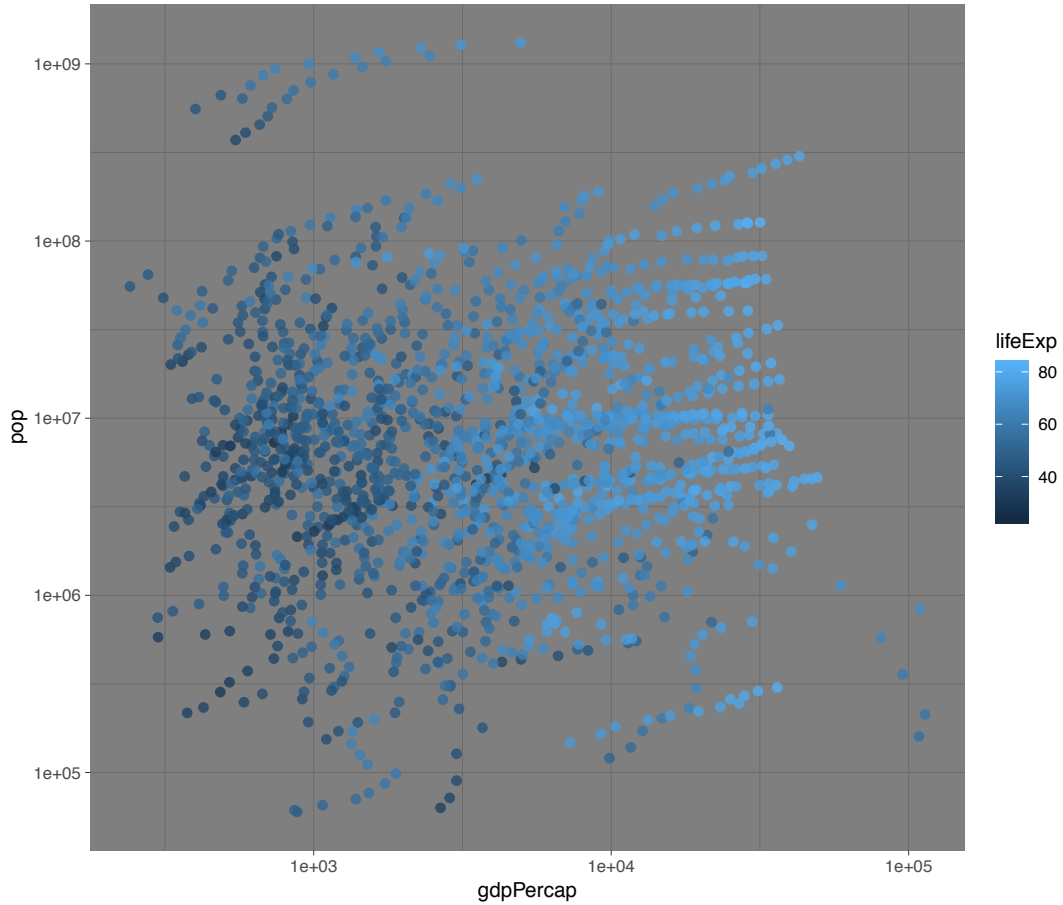
```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent,
      size = pop) +
  geom_point(alpha = 0.3) +
  geom_smooth() +
  scale_y_log10() +
  facet_wrap(~continent) +
  theme_minimal() +
  labs(title = “GDP per capita and life expectancy by continent”,
       x = “Life expectancy”,
       y = "GDP per capita”,
       colour = "Continent", size = "Population”,
       caption = "Source: Gapminder https://www.gapminder.org/.")

ggsave(“scatter_countries.png")
```

Exercise

Create a scatter plot using the gapminder dataset.

- Show **GDP per capita** on the x axis and **population** on the y axis.
- Colour should represent **life expectancy**.
- Set the **transparency** to 0.8.
  - *Using* `alpha = 0.8`
- Set the **size** to 3.
  - *Using* `size = 3`
- Set the x axis to scale to log 10.
- Set the y axis to scale to log 10.
- Set the theme to **theme_dark**.



Broody.

Making it interactive

We need to install a few additional packages:

```
install.packages("plotly")
library(plotly)
```

After that, we create a plot and assign it:

```
p_facet <- ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent) +
  geom_point(size = 3) +
  scale_y_log10() +
  facet_wrap(~continent)
```

Then we put our plot inside the `ggplotly` function:

```
ggplotly(p_facet)
```

Making it dance (making a **gif**)

We need to install a few additional packages:

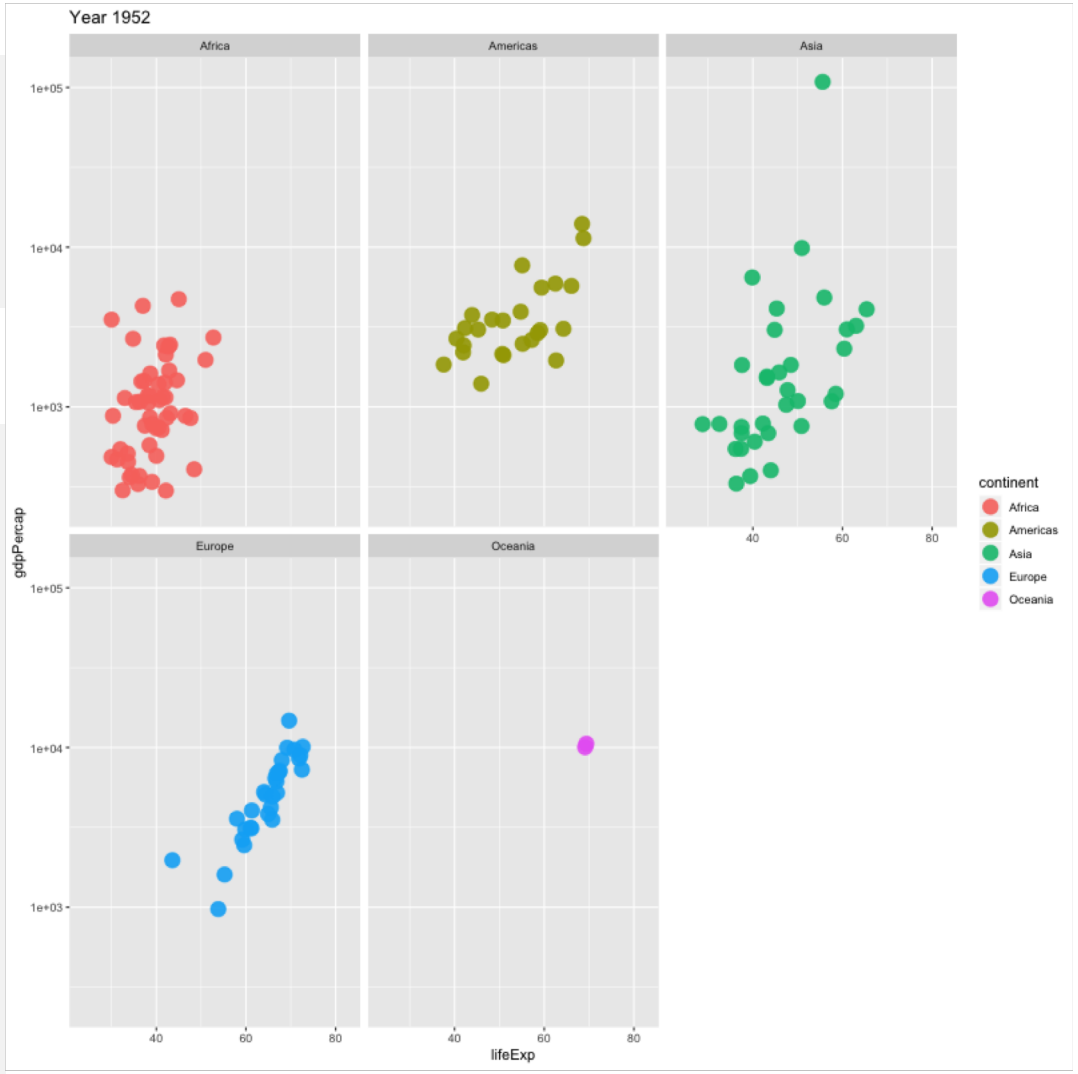
```
install.packages("gifski")
install.packages("png")
install.packages("gganimate")
library(gganimate)
```

After that, we just add **one line** to our normal plot:

```
ggplot(data = gapminder) +
  aes(x = lifeExp,
      y = gdpPercap,
      colour = continent) +
  geom_point(size = 3) +
  scale_y_log10() +
  facet_wrap(~continent) +

  labs(title = "Year {round(frame_time, 0)}") +

  transition_time(year)
```




transform

Transforming our data

We want transform our data as we wish.

We do this using functions in the **`dplyr`** package.

They have intuitive names like

- **filter** to filter
- **select** to select variables, and
- **summarise** (to summarise).

A quick side-note about pipes

This is a **pipe**: %>%

The keyboard shortcut is `CMD + SHIFT + M`.

It means:

take the thing on the left and pipe it **%>%**
into the first argument of the function on the right.

For example:

```
sum(10, 19)
```

```
[1] 19
```

is the same as

```
10 %>% sum(19)
```

```
[1] 19
```

Our **gapminder** dataset:

country	continent	year	lifeExp	pop	gdpPercap
…	…	…	…	…	…
Australia	Oceania	1952	69.1	8,691,212	10,040
Australia	Oceania	1957	70.3	9,712,569	10,950
Australia	Oceania	1962	70.9	10,794,968	12,217
Australia	Oceania	1967	71.1	11,872,264	14,526
Australia	Oceania	1972	71.9	13,177,000	16,789
Australia	Oceania	1977	73.5	14,074,100	18,334
Australia	Oceania	1982	74.7	15,184,200	19,477
Australia	Oceania	1987	76.3	16,257,249	21,889
Australia	Oceania	1992	77.6	17,481,977	23,425
Australia	Oceania	1997	78.8	18,565,243	26,998
Australia	Oceania	2002	80.4	19,546,792	30,688
Australia	Oceania	2007	81.2	20,434,176	34,435
…	…	…	…	…	…

Picture the `gapminder` dataset it in your mind.

In your mind:

- Create a new column `gdp` that is the product of `gdpPercap` and `pop`.
- Filter the observations to exclude those from `2007`.
- Remove the `gdpPercap` column.

What does this dataset look like now?

Add a column with `mutate()`

country	continent	year	lifeExp	pop	gdpPercap	gdp
…	…	…	…	…	…	…
Australia	Oceania	1952	69.1	8,691,212	10,040	**87,259,768,480**
Australia	Oceania	1957	70.3	9,712,569	10,950	**106,352,630,550**
Australia	Oceania	1962	70.9	10,794,968	12,217	**131,882,124,056**
Australia	Oceania	1967	71.1	11,872,264	14,526	**172,456,506,864**
Australia	Oceania	1972	71.9	13,177,000	16,789	**221,228,653,000**
Australia	Oceania	1977	73.5	14,074,100	18,334	**258,034,549,400**
Australia	Oceania	1982	74.7	15,184,200	19,477	**295,742,663,400**
Australia	Oceania	1987	76.3	16,257,249	21,889	**355,854,923,361**
Australia	Oceania	1992	77.6	17,481,977	23,425	**409,515,311,225**
Australia	Oceania	1997	78.8	18,565,243	26,998	**501,224,430,514**
Australia	Oceania	2002	80.4	19,546,792	30,688	**599,851,952,896**
Australia	Oceania	2007	81.2	20,434,176	34,435	**703,650,850,560**
…	…	…	…	…	…	…

Adding a variable with code

Create a new column `gdp` that is the product of `gdpPercap` and `pop`.

Create or change columns using **mutate**.

Redefine the object `gapminder`

to be the `gapminder` dataset **and**

```
gapminder07 <- gapminder %>%
               mutate(gdp = gdpPercap * pop)
```

mutated to have an additional column called `gdp` which is the product of `gdpPercap` and `pop`.

Filter with `filter()`

Dropped

Kept

country	continent	year	lifeExp	pop	gdpPercap	gdp
…	…	…	…	…	…	…
Australia	Oceania	1952	69.1	8,691,212	10,040	87,259,768,480
Australia	Oceania	1957	70.3	9,712,569	10,950	106,352,630,550
Australia	Oceania	1962	70.9	10,794,968	12,217	131,882,124,056
Australia	Oceania	1967	71.1	11,872,264	14,526	172,456,506,864
Australia	Oceania	1972	71.9	13,177,000	16,789	221,228,653,000
Australia	Oceania	1977	73.5	14,074,100	18,334	258,034,549,400
Australia	Oceania	1982	74.7	15,184,200	19,477	295,742,663,400
Australia	Oceania	1987	76.3	16,257,249	21,889	355,854,923,361
Australia	Oceania	1992	77.6	17,481,977	23,425	409,515,311,225
Australia	Oceania	1997	78.8	18,565,243	26,998	501,224,430,514
Australia	Oceania	2002	80.4	19,546,792	30,688	599,851,952,896
Australia	**Oceania**	**2007**	**81.2**	**20,434,176**	**34,435**	**703,650,850,560**
…	…	…	…	…	…	…

Filtering our dataset

Filter the observations to include only those from `2007`.

Filter observations using **filter**.

`==` means `is equal to`

`!=` means `is not equal to`

Redefine the object `gapminder`

to be the `gapminder07` dataset **and**

```
gapminder07 <- gapminder07 %>%
               filter(year == 2007)
```

filtered to keep all observations where year IS equal to 2007.

Drop a variable with `select()`

country	continent	year	lifeExp	pop	gdpPercap	gdp
…	…	…	…	…	…	…
Australia	Oceania	1952	69.1	8,691,212	10,040	87,259,768,480
Australia	Oceania	1957	70.3	9,712,569	10,950	106,352,630,550
Australia	Oceania	1962	70.9	10,794,968	12,217	131,882,124,056
Australia	Oceania	1967	71.1	11,872,264	14,526	172,456,506,864
Australia	Oceania	1972	71.9	13,177,000	16,789	221,228,653,000
Australia	Oceania	1977	73.5	14,074,100	18,334	258,034,549,400
Australia	Oceania	1982	74.7	15,184,200	19,477	295,742,663,400
Australia	Oceania	1987	76.3	16,257,249	21,889	355,854,923,361
Australia	Oceania	1992	77.6	17,481,977	23,425	409,515,311,225
Australia	Oceania	1997	78.8	18,565,243	26,998	501,224,430,514
Australia	Oceania	2002	80.4	19,546,792	30,688	599,851,952,896
Australia	**Oceania**	2007	**81.2**	**20,434,176**	**34,435**	**703,650,850,560**
…	…	…	…	…	…	…

Dropped

Dropping a variable with code

Remove the gdpPercap column.

Drop columns using **select**.

Redefine the object gapminder

to be the gapminder07 dataset **and**

```
gapminder07 <- gapminder07 %>%
                select(-gdpPercap)
```

Negatively select (drop) the variable gdpPercap

Combining it all

```
gapminder07 <- gapminder %>%
```
Take the gapminder dataset **and**

```
  mutate(gdp = gdpPercap * pop) %>%
```
Add a column called `gdp` which is the product of `gdpPercap` and `pop` **and**

```
  filter(year == 2007) %>%
```
Filter observations where year does not equal 2007 **and**

```
  select(-gdpPercap)
```
Negatively select (drop) the variable `gdpPercap`.

Now the **`gapminder`** dataset is what you pictured!

Exercise

Create a new dataset called `aus`

It should include:

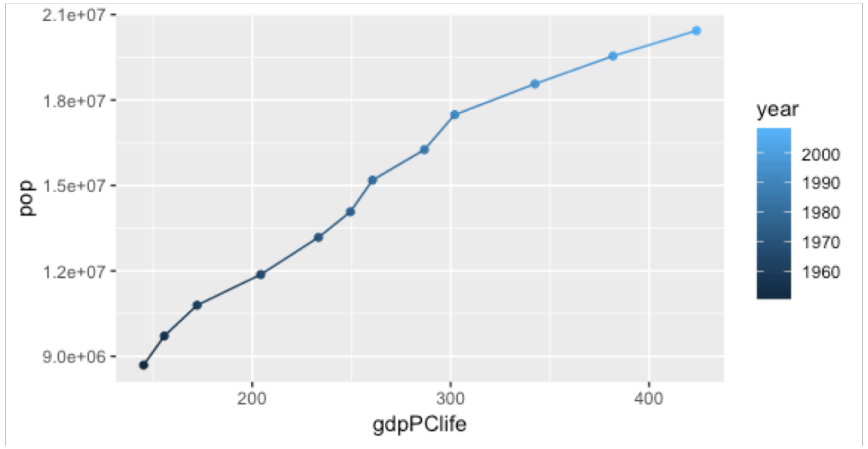
- Only observations from Australia.
  - *The filter will be:* `country == "Australia"`
- A new variable called `gdpPClife` which is GDP per capita divided by life expectancy.

- Plot a scatter plot using `aus`. It should show:
  - `gdpPClife` on one axis,
  - `pop` on the other, and
  - colour it by `year`.

- Then, add a line plot with `geom_line()`

```
aus <- gapminder %>%
     filter(country == "Australia") %>%
     mutate(gdpPClife = gdpPercap / lifeExp)

aus %>%
ggplot() +
  aes(gdpPClife,
      pop,
      colour = year) +
  geom_point() +
  geom_line()
```



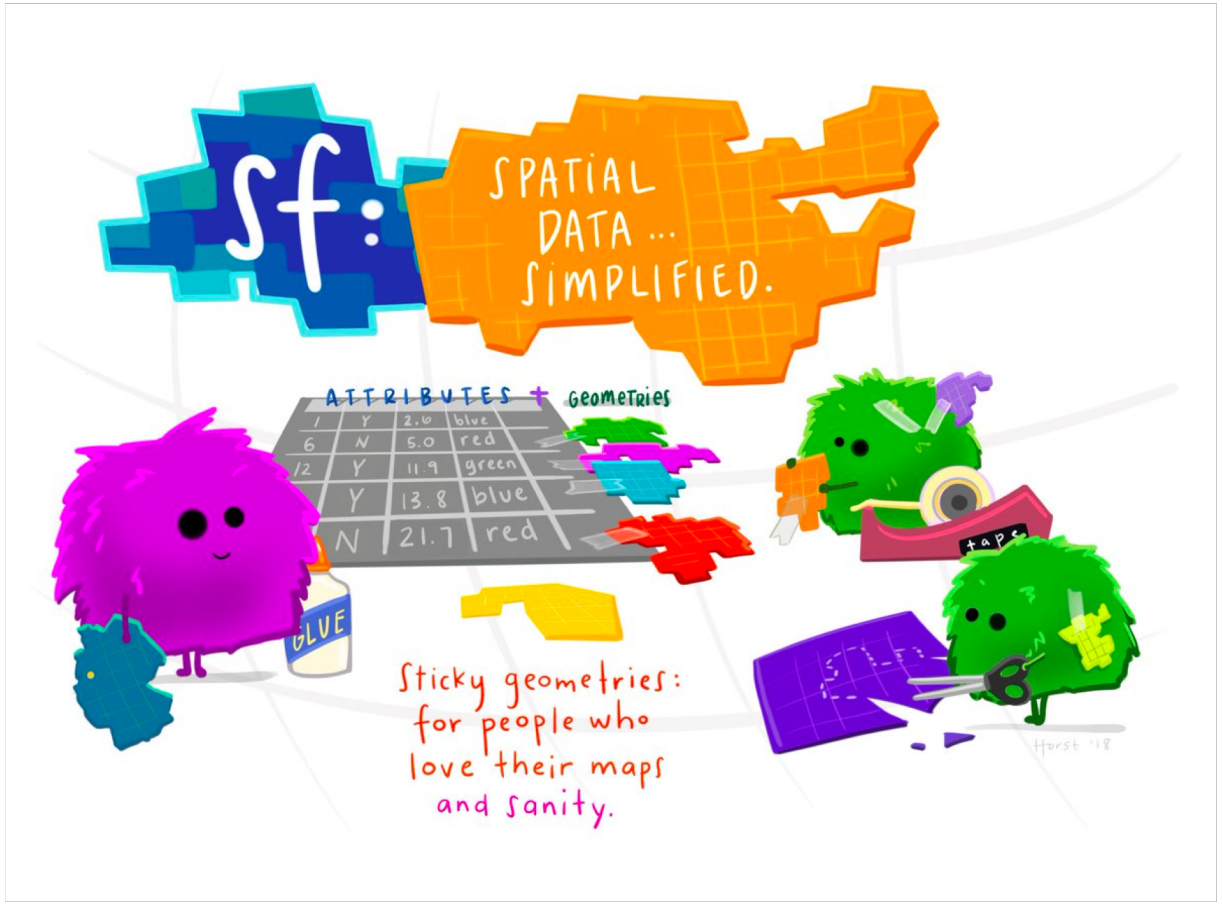
Making a map

We use the same `ggplot` ideas to plot maps.

We just need a variable that contains the **geometries** of each area.

`sf` (simple features) objects are great at this:

- they are normal datasets with a 'special' variable called `geometry`;
- this `geometry` variable contains the instructions on how to draw the area.



'Be a space wrangler' by Allison Horst, https://twitter.com/allison_horst/status/1071456081308614656

Making a map

There is a file in your **data** folder that contains this geometry variable, called **gapmapdata.Rds**

First, we install and load the **sf** package:

```
install.packages("sf")
```

```
library(sf)
```

Then we read the data file using `read_rds`:

```
gapmap <- read_rds("data/gapmapdata.Rds")
```

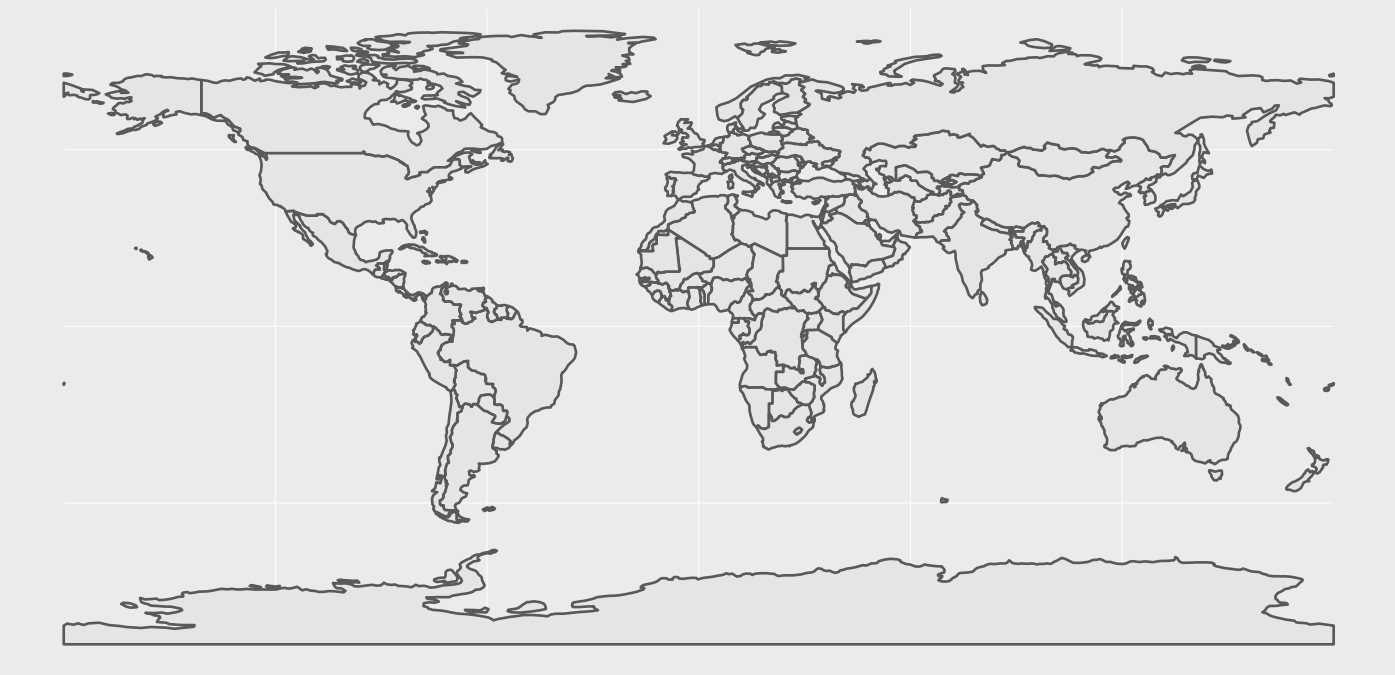
Making a map

Now we use the same ggplot syntax:

- **Data**
- **Aesthetics**
- **Geom**

We'll start by plotting an empty world map:

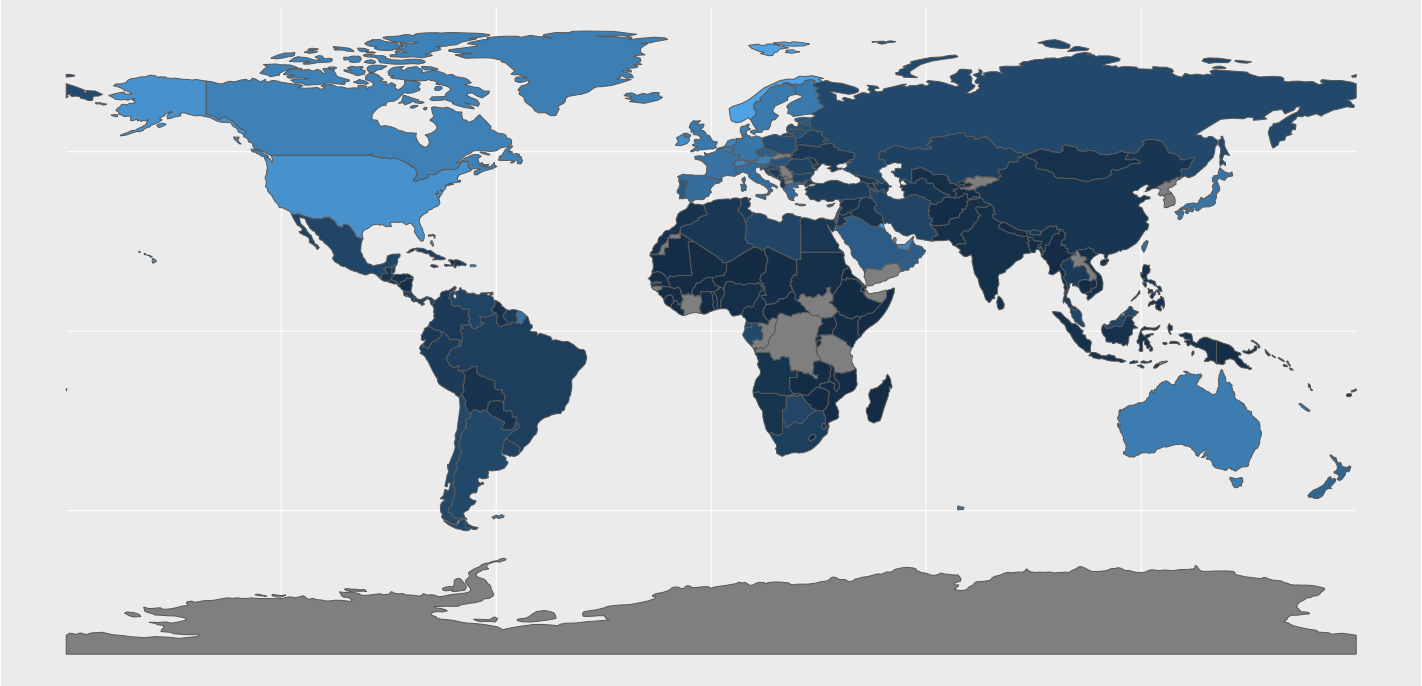
```
ggplot(data = gapmap) +
  aes(geometry = geometry) +
  geom_sf()
```



Making a map

And we can add an aesthetic: `fill` the countries by `gdpPercap`

```
ggplot(data = gapmap) +
  aes(geometry = geometry,
      fill = gdpPercap) +
  geom_sf()
```



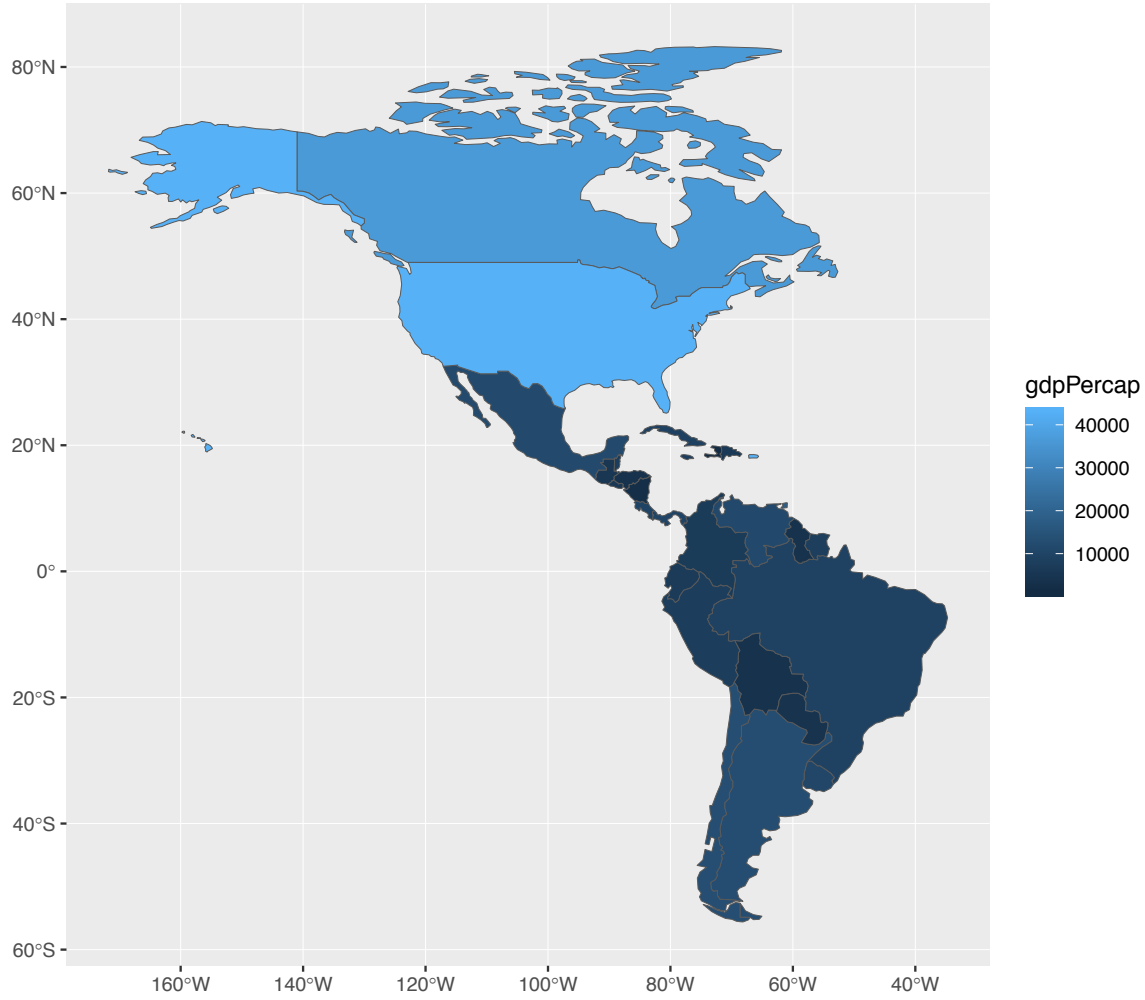
Making a map

With filtering, we can look at countries just in the Americas:

```
1. gapmap %>%
2.   filter(continent == "Americas") %>%
3.   ggplot() +
4.     aes(geometry = geometry,
           fill = gdpPercap) +
5.     geom_sf()
```

In words:

1. Take the **gapmap** dataset
2. And `filter` to keep the Americas continent
3. Pipe `%>%` this into the data argument of `ggplot()`
4. Define **aes**thetics
5. Use `geom_sf`, which plots geometries.




extra

extra

Using R Markdown

R Markdown is a type of file that produces documents.

It will process your R code and its output, then typeset/produce your file.

It uses the same wonderful math language as LaTeX, too.

Using an R Markdown document means your analysis, your visualisations and your writing are all in one place.

To start an R Markdown document, **click**:

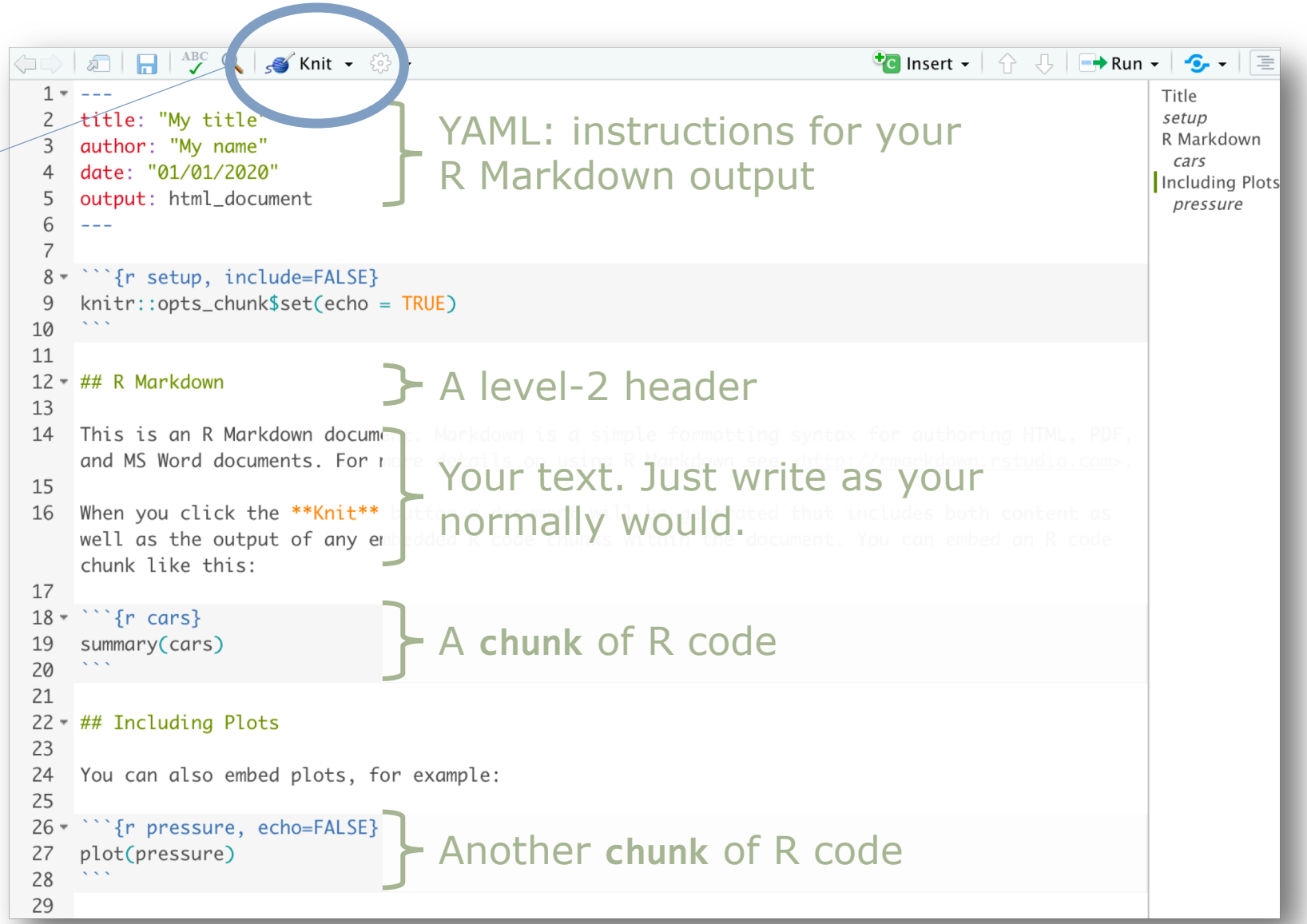
File ->
New File ->
R Markdown

extra

Using R Markdown

This will set you up and explain a few things.

Click the **Knit** button to compile your document.



extra

Resources

This short workshop can't cover everything. Below are some resources by topic that I have used in the past (and still use today!)

Note: they are all free.

Data visualisation with `ggplot`

Healy: *Data Visualization: A practical introduction* at https://socviz.co/

Wickham: *R for Data Science, chapter 3* at https://r4ds.had.co.nz/data-visualisation.html

Yau: *Flowing data* at https://flowingdata.com

R Studio: *ggplot cheatsheet* at https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Transforming data with `dplyr`

Wickham: *R for Data Science, chapter 5* at https://r4ds.had.co.nz/transform.html

R Studio: dplyr cheatsheet: https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

extra

Resources

Analysis:

Working with time series data

Wickham: *R for Data Science, chapter 16* at https://r4ds.had.co.nz/dates-and-times.html

Earo Wang (a Monash graduate and PHD candidate): *Melting the Clock: tidy time-series analysis* presentation at R Studio conference at *https://slides.earo.me/rstudioconf19*

The above presentation includes information about the tsibble and fable time-series packages, which were created by Di Cook and Earo Wang from Monash.

Communication:

R Markdown

R Studio: *R Markdown Quick Tour* at https://rmarkdown.rstudio.com/authoring_quick_tour.html

Wickham: *R for Data Science, chapter 27* at https://r4ds.had.co.nz/r-markdown.html

'Shiny' web applications

R Studio: Shiny tutorials at https://shiny.rstudio.com/tutorial/

and gallery at: https://shiny.rstudio.com/gallery/

extra

Resources

For everything else:

Google: asking well-phrased questions on Google will give you plenty of good answers.

If you're asking about `ggplot`, state `ggplot` in your search:

'How to change the default colours `ggplot`'

If you're asking about `dplyr`, state `dplyr` in your search:

'How to group observations together `dplyr`'

If you're not sure what package will do the trick, specify that you're using **R**:

'How to save a dataset in R'

It sounds silly, but being good at *googling* things will get you *a long, long way*.

Thanks for coming