Week 4: Visualisation

MY472: Data for Data Scientists
https://lse-my472.github.io/

Autumn Term 2025

Ryan Hübert
Associate Professor of Methodology

Slides last updated: 22 October 2025

Administrative items

- Starting this week: open door policy
- Formative project
- Update to GitHub Classroom workflow for seminars
  - GitHub Classroom assignments will have the seminar materials
  - Will be released after lecture
  - Review and do set up steps before seminar
  - Limited time for set up during seminars

Mid-term check-in: what MY472 is about

Data science is about bringing clarity to ambiguity

- Remember the data-to-information concept

MY472 is about learning how to be a data scientist

- Part of this is learning *technical skills*
  - Learning a wide range of relevant concepts (lectures)
  - Learning R code to do tasks (seminars)
- Part of this is learning *professional skills*
  - Managing time, preparing and following-up
  - Communicating clearly and professionally
  - Acting with integrity (e.g., appropriate AI use)

Learning to figure things out

New data scientists (and grad students) often underappreciate that **you must learn to do things yourself**

This means:

- Reading and understanding your own error messages
- Searching, experimenting, and debugging
- Using tools (Google, AI, docs) strategically

We're here to guide you, not tell you exactly what to do step-by-step

- Lectures and seminars give structured foundations
- Office hours offer targeted help

Why this matters

People who can figure things out:

- Get the best jobs
- Earn the highest salaries
- Make the biggest research contributions

Because they can:

- Define their own problems
- Work effectively even with incomplete information
- Deliver results without waiting for someone to tell them how

Most data science jobs are much less structured than MY472

High marks come from engaged learning

Post-graduate training is about learning, not marks

- Important: you *shouldn't* already know this stuff!

The final assessment will be fair and (very) achievable

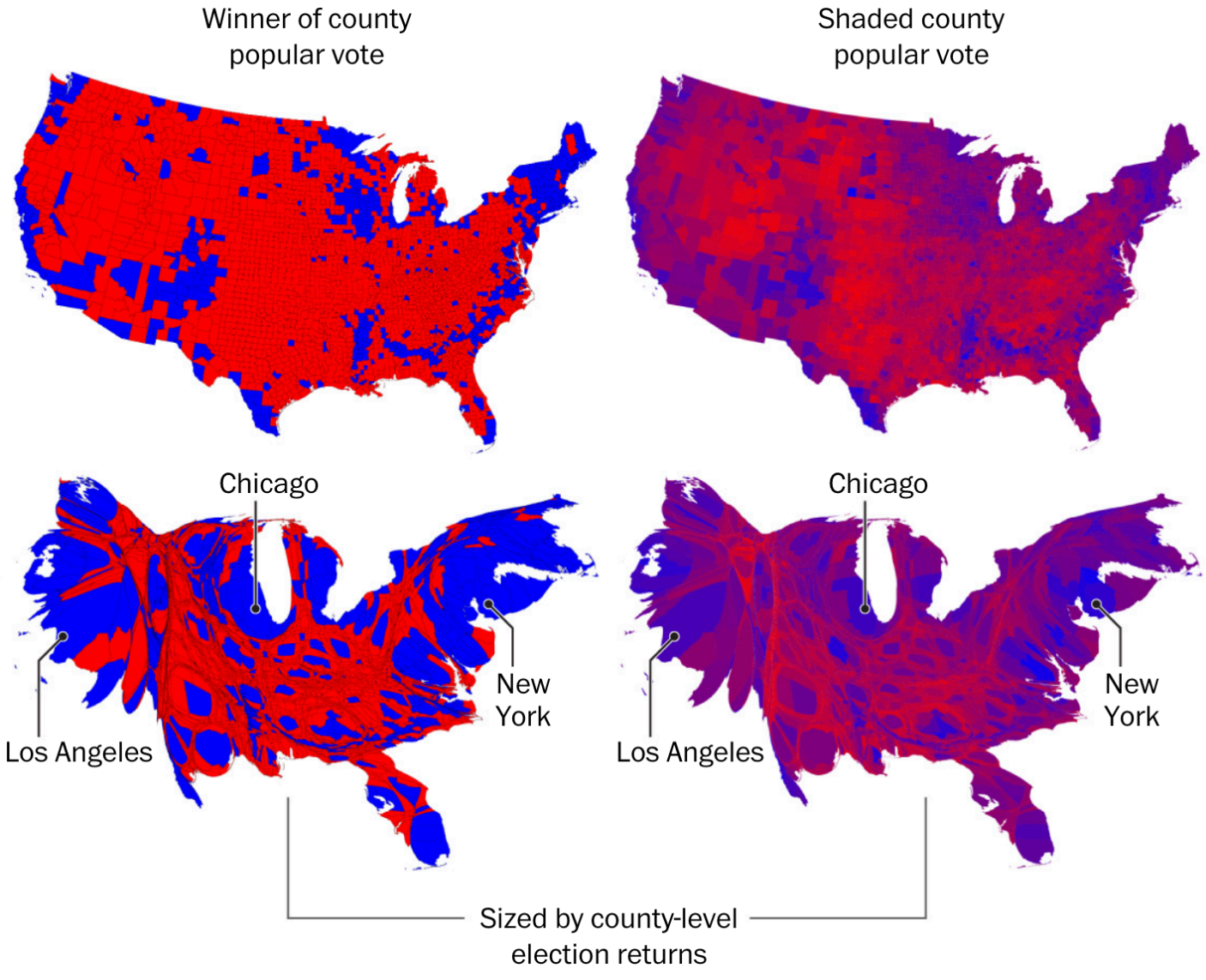
The best way to prepare is to be:

- highly engaged with the learning materials
- motivated to learn things on your own
- intellectually curious about your computer, data science, and the specific concepts we're teaching
- willing to push beyond the materials

You will get detailed information about format in due course

The importance of visualisation

The importance of visualisation



Source: https://www.washingtonpost.com/graphics/politics/2016-election/how-election-maps-lie/

Outline

1. Some principles of data visualisation
2. Grammar of graphics and `{ggplot}`
3. Spatial data

Reducing complexity to enable *learning* from data

A tabular dataset is a complex object—lots of observations (rows), lots of information about those observations (columns)

The typical human cannot look at a raw tabular dataset and draw meaning from it

Point of data analysis: reduce complexity, enable learning

- Remember: *data* exists in service of *information*

Reducing complexity to enable *learning* from data

To learn, you need to **summarise**, e.g.:

- Calculate means/medians/counts/etc. of each variable
- Calculate correlations between multiple variables
- Make plots of distributions (“shapes”) of variables

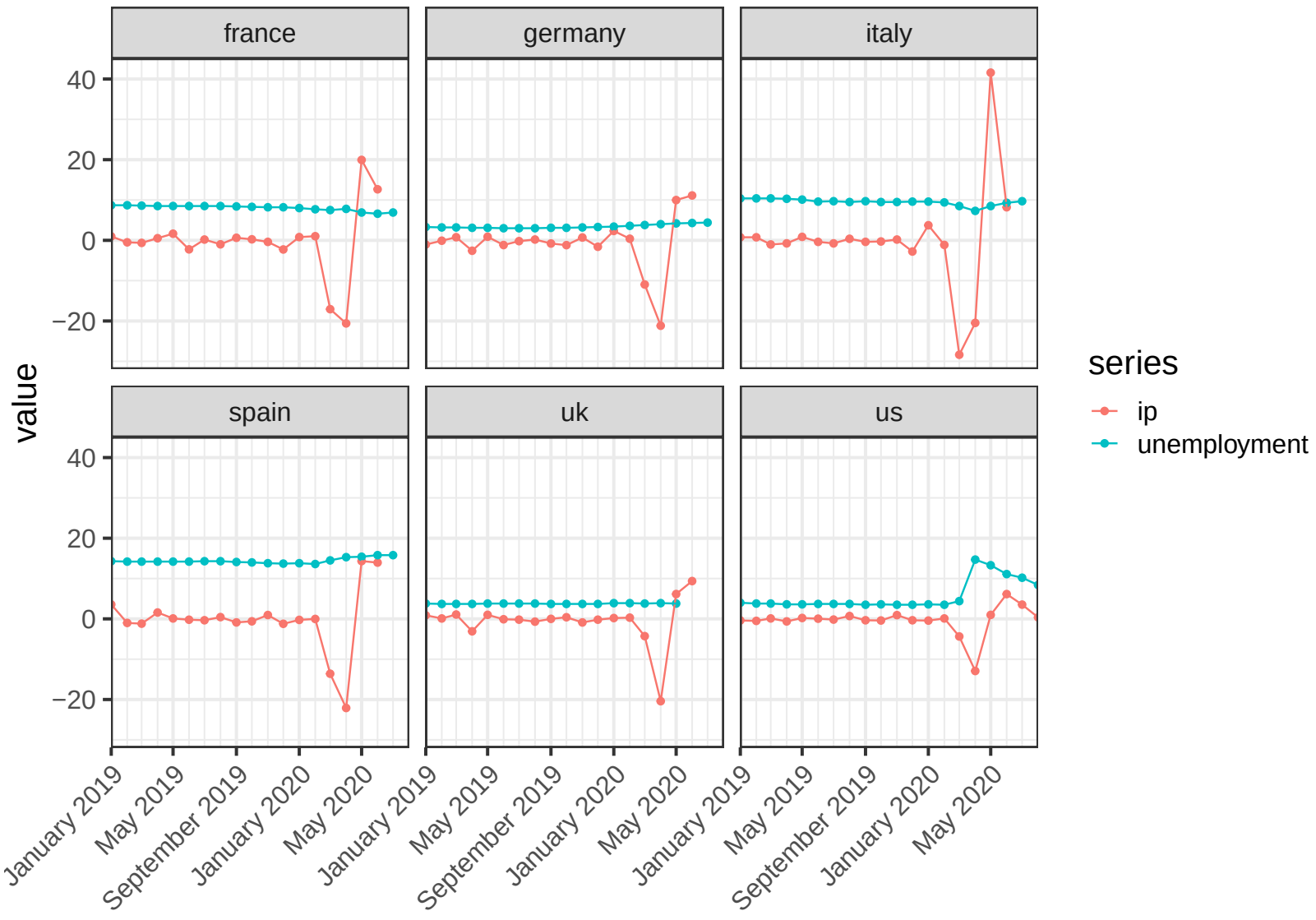
Visual communication of data: **visualisation**

- Warning: *lots* of discretion, need to do this well
- Could fill an entire course on this

What do we *learn* from this?

```
> print(ip_and_unemployment)
# A tibble: 223 × 4
   country date       series       value
   <chr>   <chr>      <chr>        <dbl>
 1 france  01.01.2019 ip           0.973
 2 france  01.01.2019 unemployment 8.7
 3 france  01.02.2019 ip          -0.496
 4 france  01.02.2019 unemployment 8.7
 5 france  01.03.2019 ip          -0.633
 6 france  01.03.2019 unemployment 8.6
 7 france  01.04.2019 ip           0.521
 8 france  01.04.2019 unemployment 8.5
 9 france  01.05.2019 ip           1.64
10 france  01.05.2019 unemployment 8.5
```

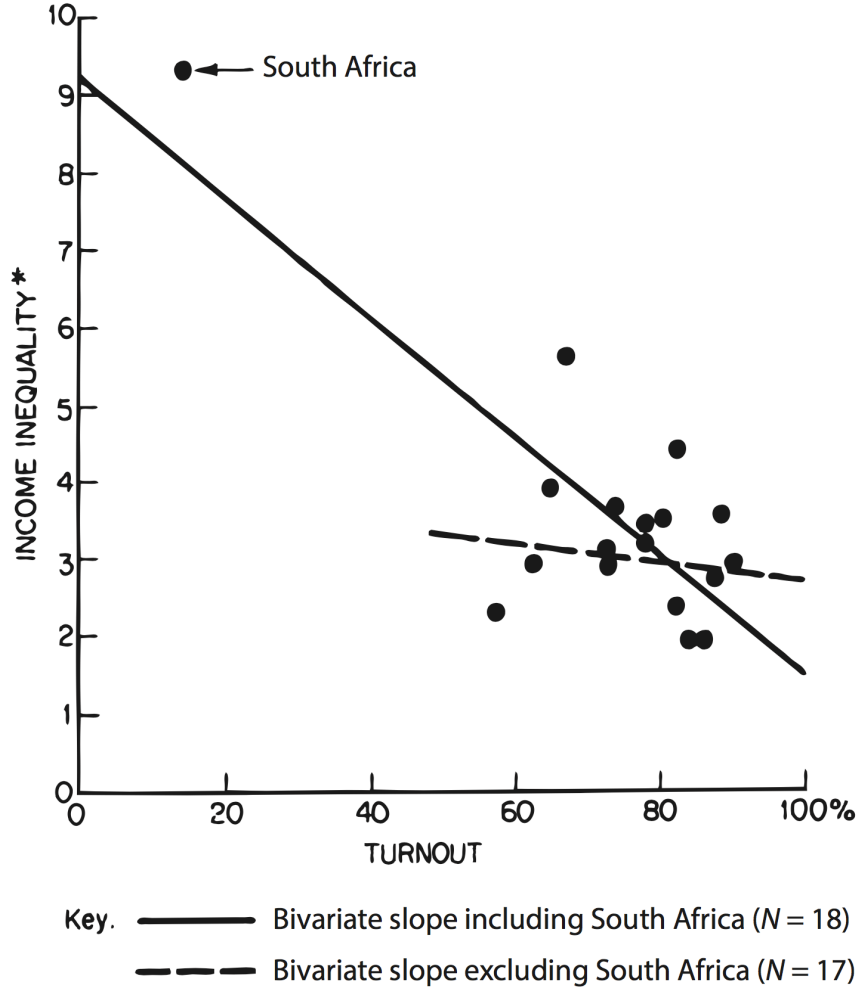
What do we *learn* from this?



Principles by Edward Tufte

- Show the data
- Avoid distorting what the data have to say
- Allow viewer to compare
- Serve a clear purpose: description, exploration, tabulation or decoration
- Be closely integrated with the statistical and verbal descriptions of the dataset
- Graphics can reveal data (e.g. Anscombe's Quartet)

Why you should look at data (from Healy 2019)



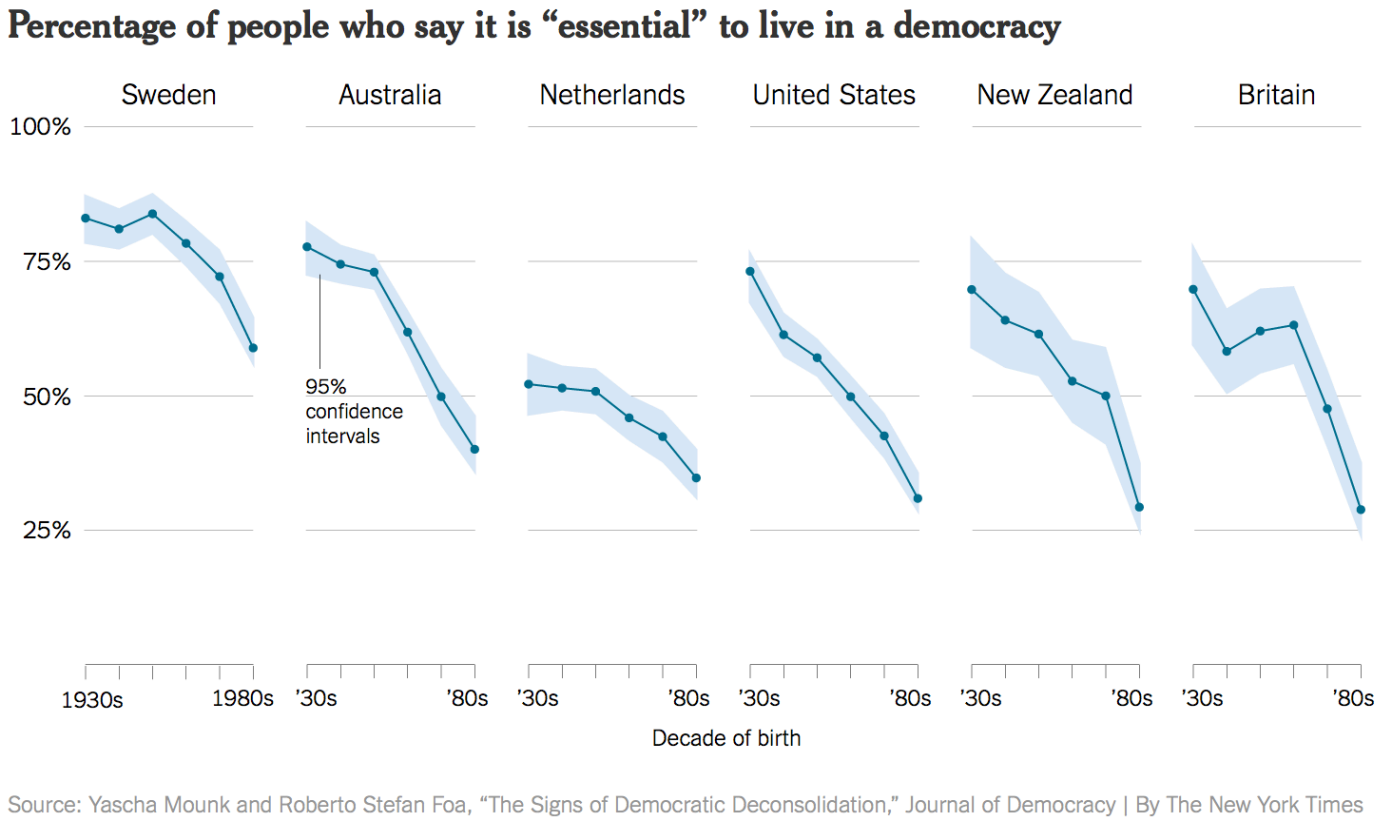
Source: Jackman, Robert M. 1980. "The Impact of Outliers on Income Inequality." *American Sociological Review* 45(2): 344–347.

What makes bad figures bad? (from Healy 2019)

1. Bad taste (e.g., too much “junk”)

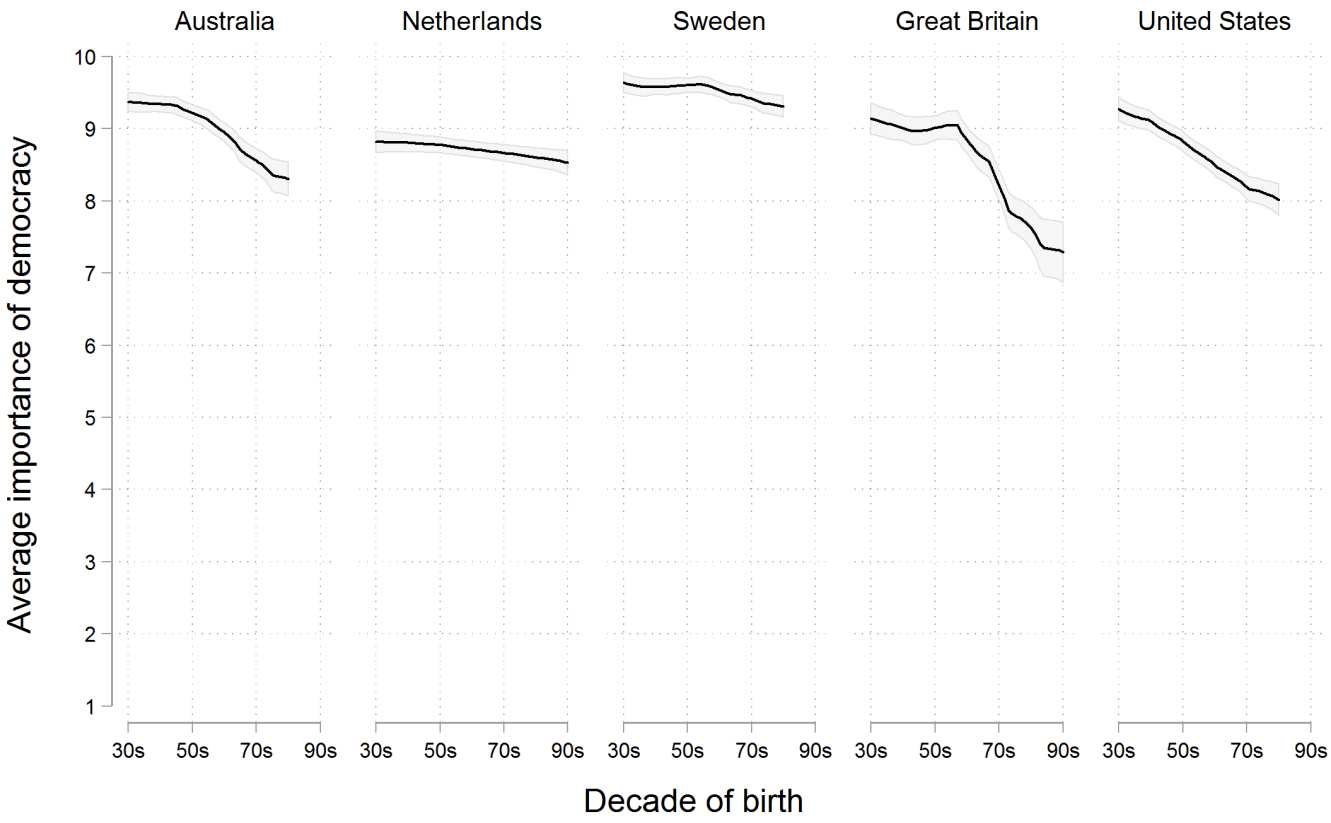
What makes bad figures bad? (from Healy 2019)

2. Bad data (e.g., cherry-picking, misleading)



What makes bad figures bad? (from Healy 2019)

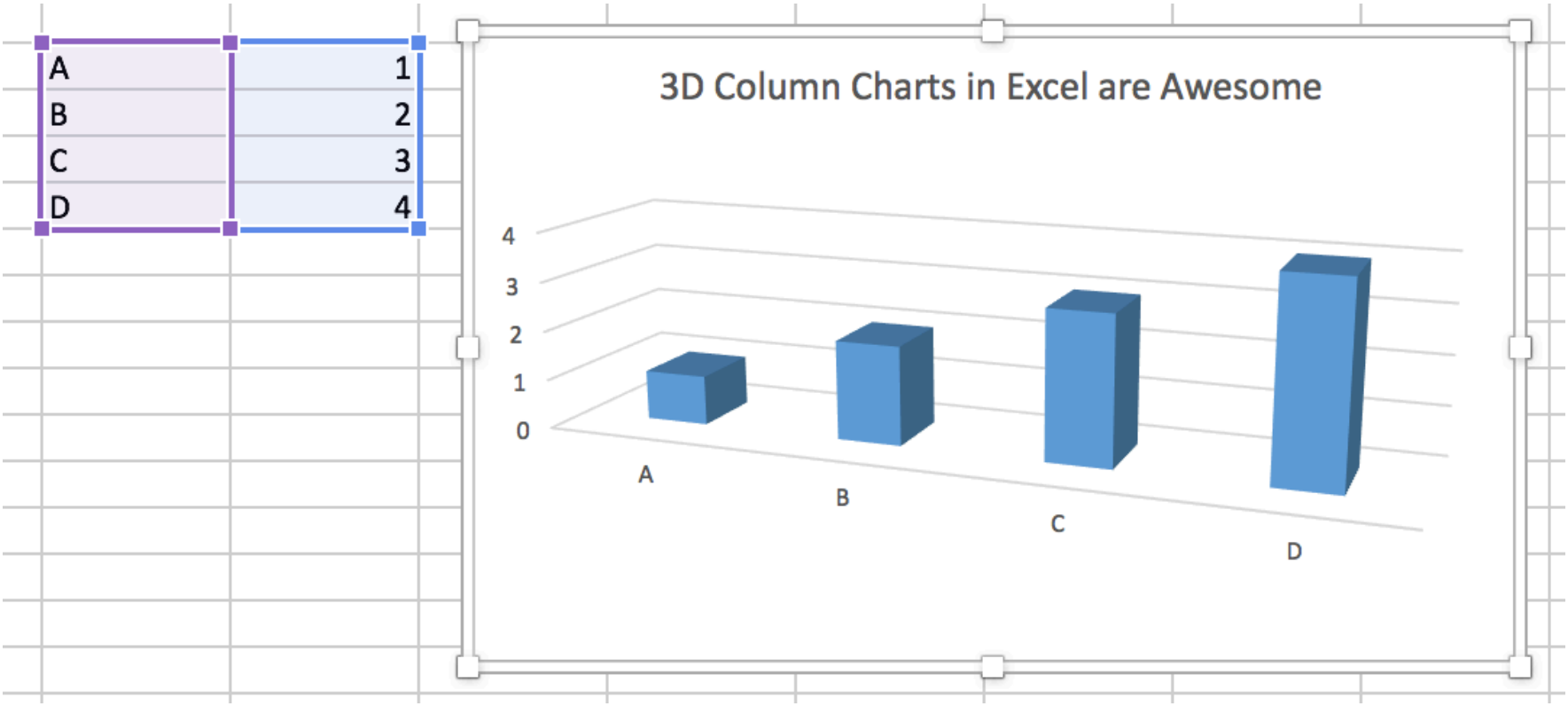
2. Bad data (e.g., cherry-picking, misleading)



Graph by Erik Voeten, based on WVS 5

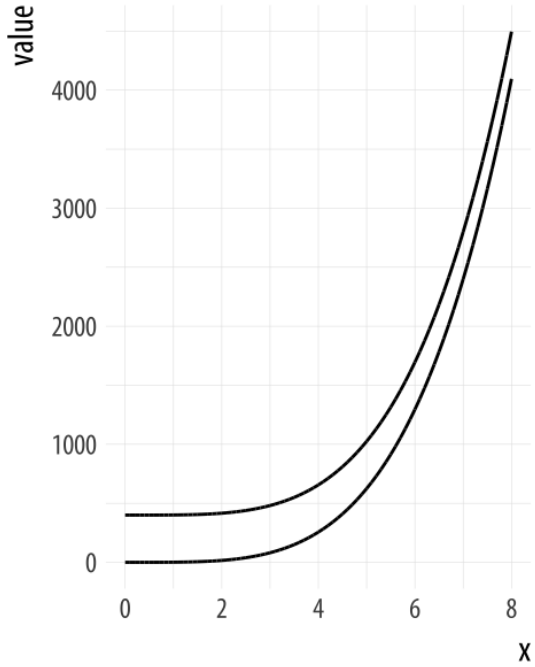
What makes bad figures bad? (from Healy 2019)

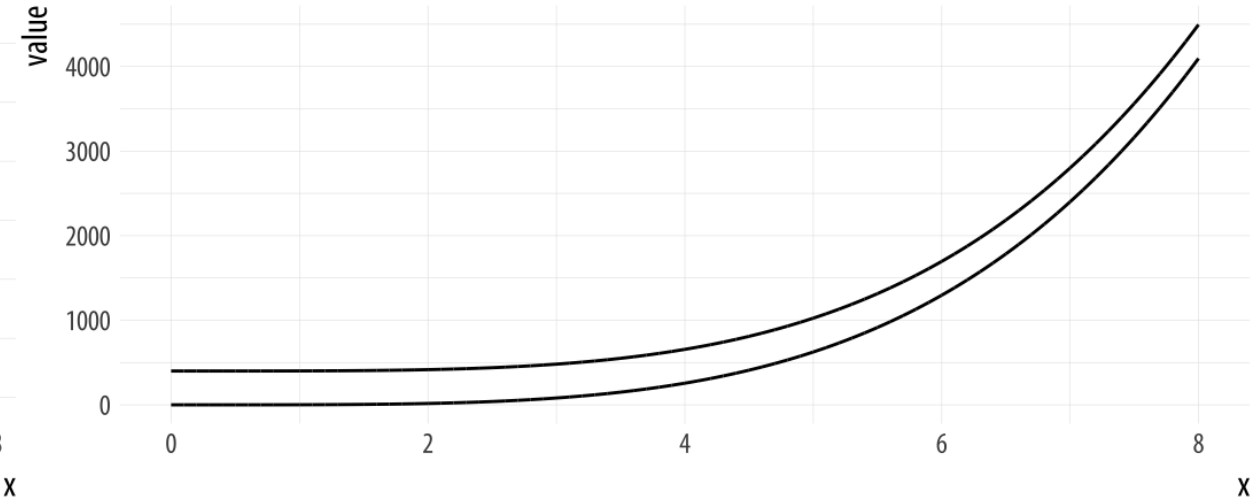
3. Bad perception



What makes bad figures bad? (from Healy 2019)

3. Bad perception





Some general guidelines

1. Maximize data-to-ink ratio
2. Avoid misleading decisions
   - ▷ Y axis starts at 0
   - ▷ Comparison of areas is hard
   - ▷ Use comparable units
   - ▷ Erase chart junk
3. Use text to inform and contextualise. Add annotations
4. Appropriate use of scales (x/y axes, colour, size, shape…)
5. Use small multiples to facilitate comparisons
6. Always cite sources
7. Consider accessibility and different use-cases, e.g., sizing, colour-blind palettes, web vs. print (`https://colorbrewer2.org/`)

Outline

1. Some principles of data visualisation
2. Grammar of graphics and `{ggplot}`
3. Spatial data

The “grammar of graphics”

Wilkinson (2005): (statistical) graphics have a “grammar”

- That is: a set of mathematical and aesthetic rules for creating visual representations from data

The big (somewhat subtle) idea: data visualisation isn’t limited to a constrained set of pre-defined and formulaic “charts”

- The grammar allows us to innovate and create new kinds of visualisations

R package `{ggplot2}`: Hadley Wickham’s layered version of Wilkinson’s grammar of graphics designed for use in R

- Very powerful package for beautiful, bespoke visualisations

The “grammar” of {ggplot2}

{ggplot2} creates visualisations from data using **layers**

- A visualisation can have more than one layer
- Intuitively: creating a visualisation = stacking layers

Each layer contains:

- `data`: data to visualise
- `mapping`: links variables in data to visual properties
- `stat`: statistical transformations of data
- `geom`: controls the *type* of plotting object (line, point, etc)
- `position`: adjust overlapping objects

Running example: penguin dataset

Purpose of {ggplot2} grammar: make bespoke plots

- Should still adhere to principles of good visualisation!

Best to see examples, like this nice one from R4DS:

```
library("palmerpenguins")
head(penguins)
```

```
# A tibble: 6 x 8
  species island    bill_length_mm bill_depth_mm
  <fct>   <fct>              <dbl>         <dbl>
1 Adelie  Torgersen           39.1          18.7
2 Adelie  Torgersen           39.5          17.4
3 Adelie  Torgersen           40.3          18
4 Adelie  Torgersen           NA            NA
5 Adelie  Torgersen           36.7          19.3
6 Adelie  Torgersen           39.3          20.6
# i 4 more variables: flipper_length_mm <int>,
#   body_mass_g <int>, sex <fct>, year <int>
```

Penguins: univariate visualisation

Penguins (like humans!) vary in their weight

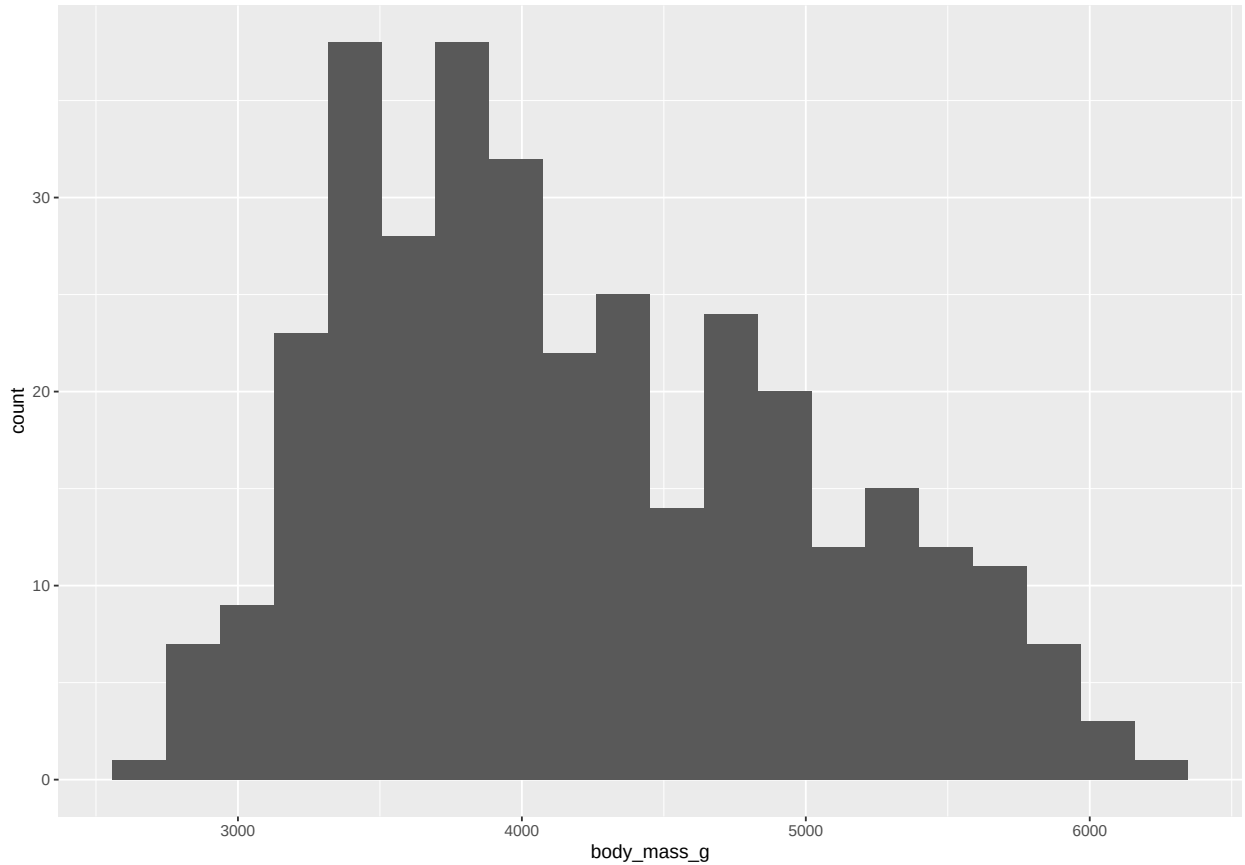
- How could we summarise this fact visually?

```
z <- ggplot()                                              # initialise plot

z <- z +                                                   # add layer with `+`
  geom_histogram(data = penguins,                          # specify data
                 mapping = aes(x = body_mass_g),           # map variable to axes
                 stat = "bin",                             # histogram default
                 position = "stack",                       # histogram default
                 bins = 20)                                # histogram setting
```

Penguins: univariate visualisation

```
print(z)
```



Penguins: bivariate visualisation

Now let's look at a bivariate relationship

- Flipper length and body mass

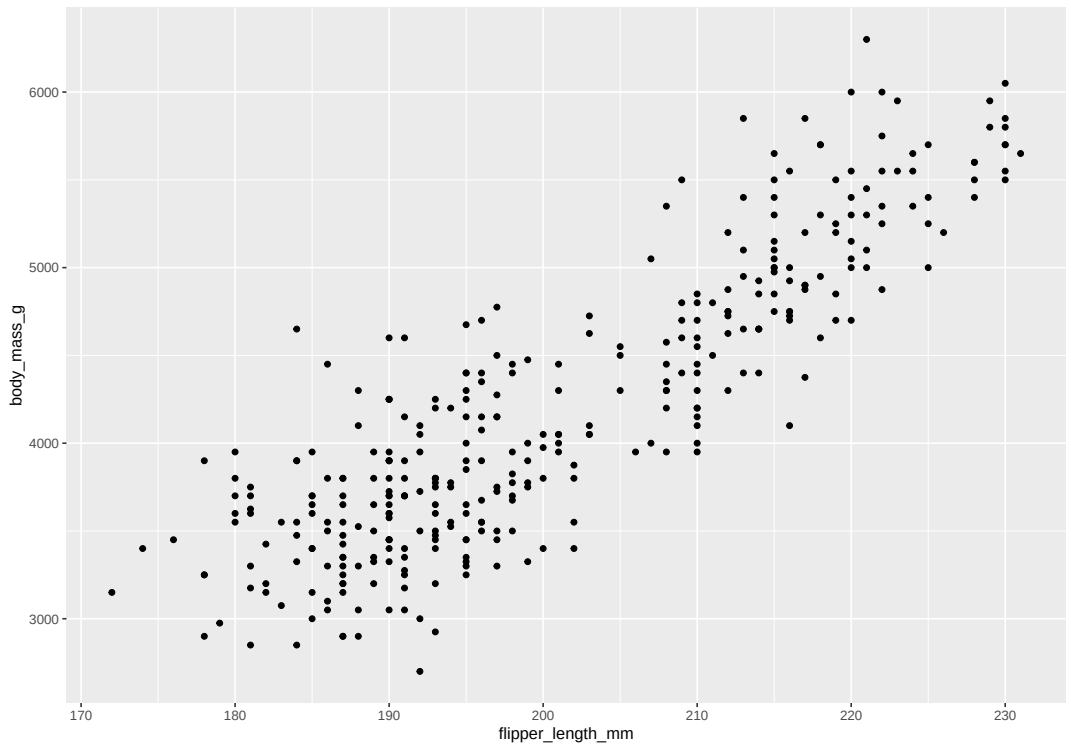
We can define `data` and `mapping` when we initialise

- All layers inherit these, unless otherwise specified

```
z <- ggplot(data = penguins,                          # initialise w/ data
            mapping = aes(x = flipper_length_mm,      #         and mapping
                          y = body_mass_g))
```

Penguins: add a `geom` (points)

```
z <- z +
  geom_point() # inherit data and mapping, use other defaults
print(z)
```



More of the “grammar” of {ggplot2}

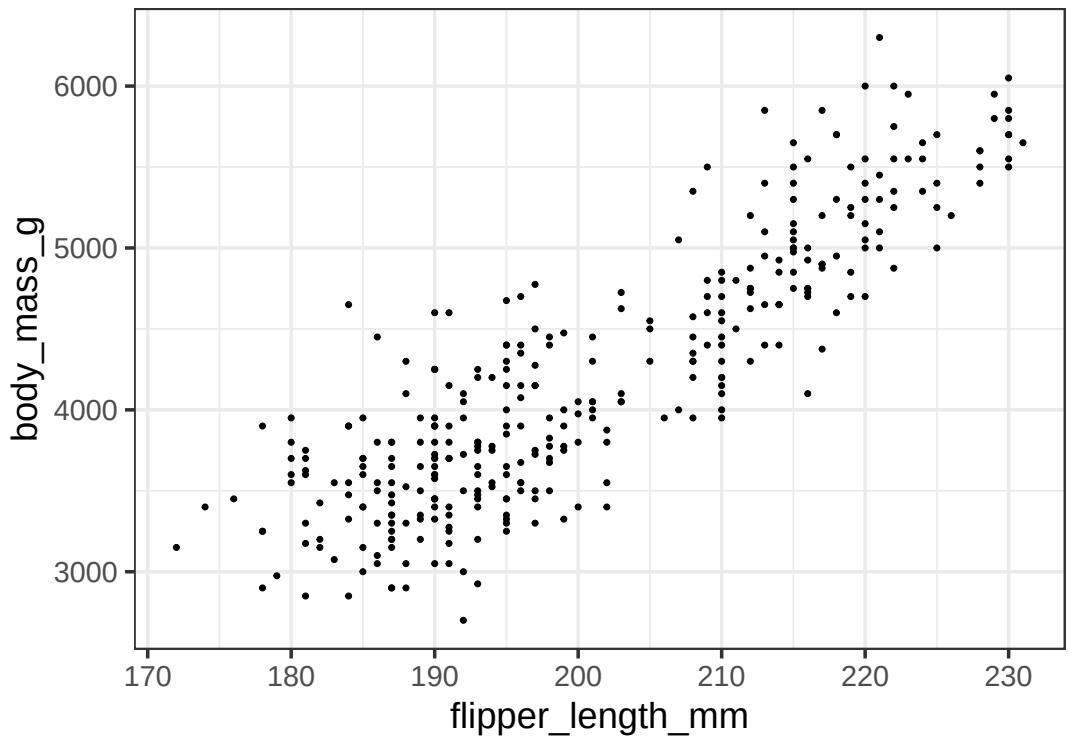
Layers are the most important component of the grammar, but there are four other major components

- **scales**: translation between variable ranges and graphical properties, e.g. linking values to colours and/or shapes
- **coordinates**: Coordinate system that, e.g. provides axes and gridlines
- **facets**: Breaking up the data into subsets, e.g. to be displayed independently on a grid
- **theme**: Parts that do not follow from the data, e.g. background colours, fonts, grid lines, legends etc.

Possibly a fifth: **metadata** (like labels/captions)

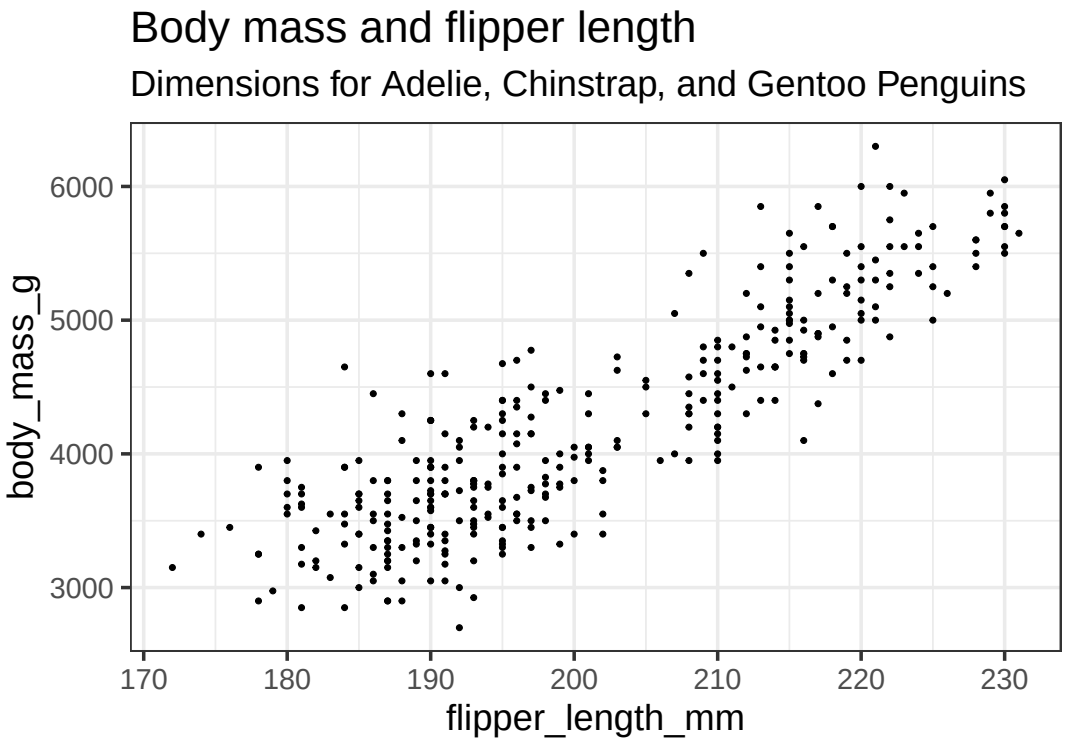
Penguins: adjust the (visual) theme

```
z <- z +
  theme_bw(base_size = 24) # make bigger & remove grey bg
print(z)
```



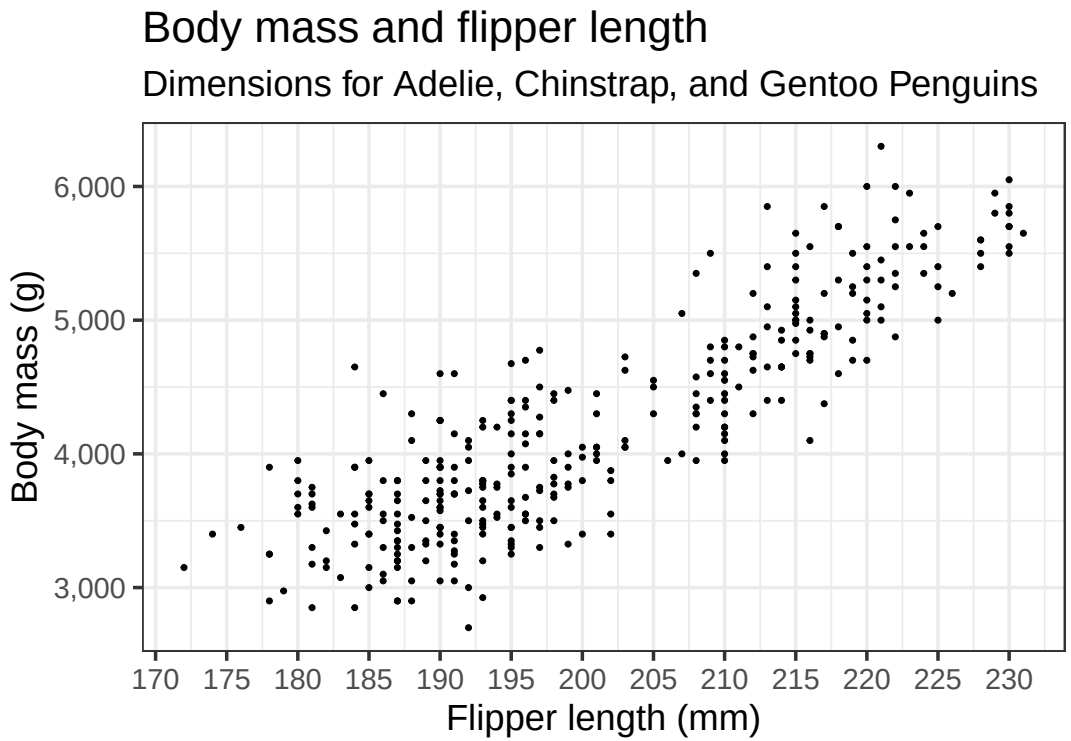
Penguins: add plot title

```
z <- z +
  labs(title = "Body mass and flipper length",
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins")
print(z)
```



Penguins: modify x and y scales

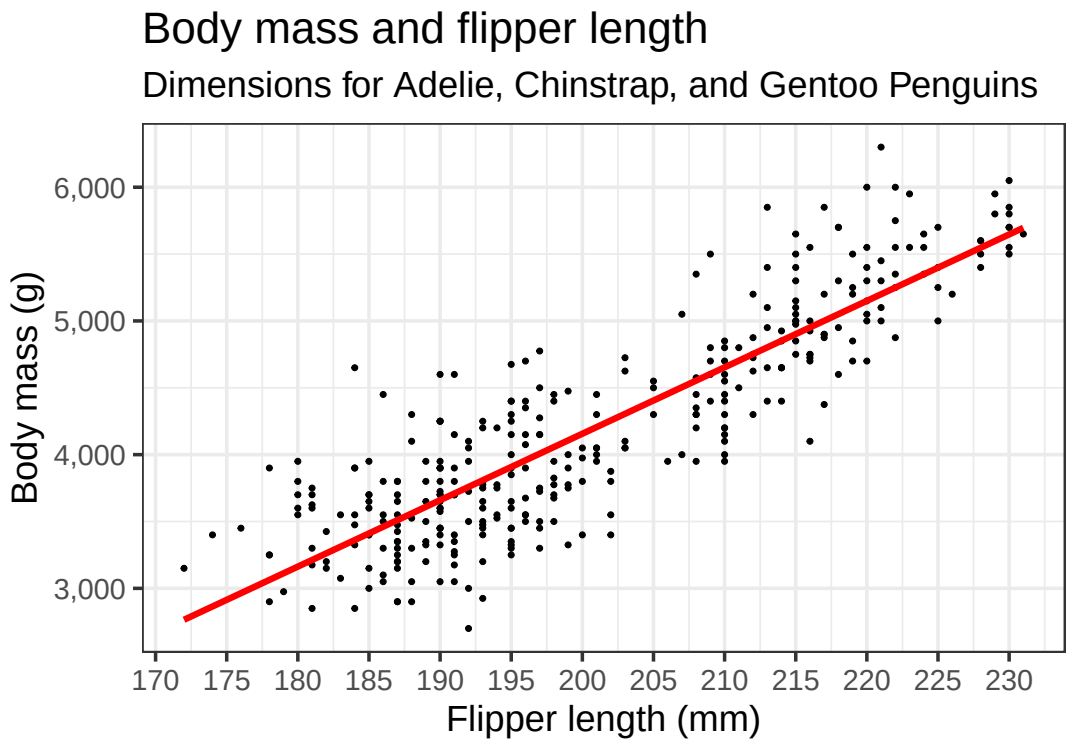
```
z <- z +
  scale_x_continuous("Flipper length (mm)", breaks = seq(170,230,5)) +
  scale_y_continuous("Body mass (g)", labels = scales::comma)
print(z)
```



Penguins: add a layer

Since layers are contained, we can overlay multiple layers

```
z <- z +
  stat_smooth(geom = "smooth", method = "lm", formula = "y ~ x",
              se = FALSE, linewidth = 2, colour = "red")
print(z)
```



Scales

Scales “translate” data ranges to property ranges

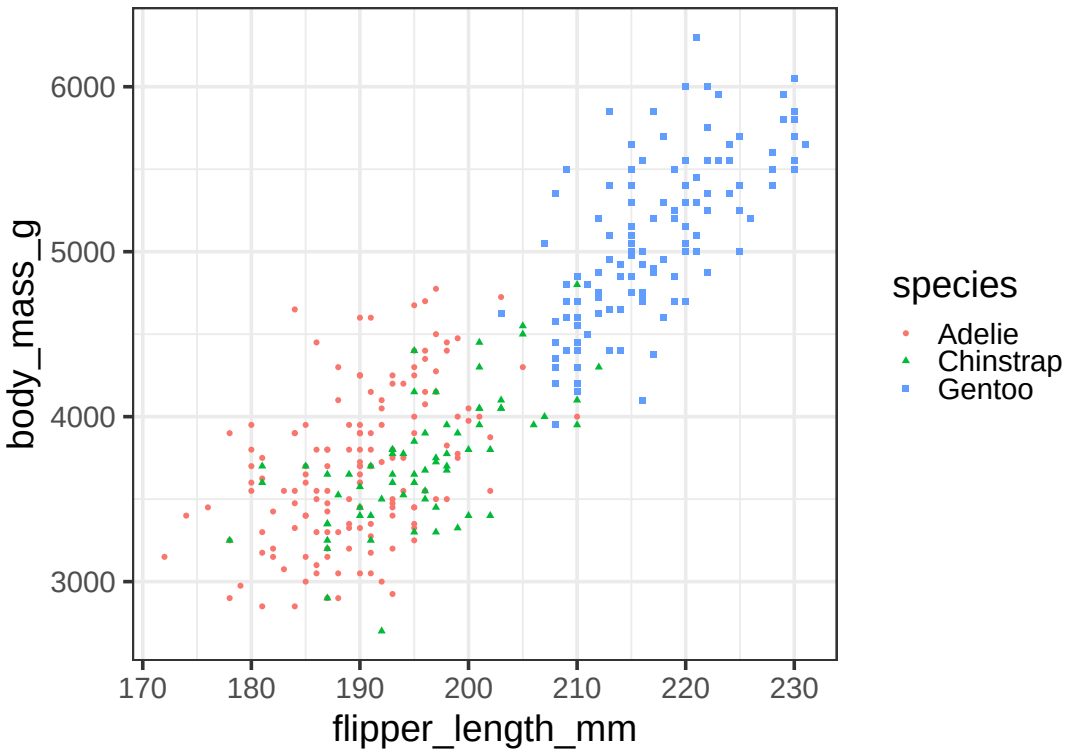
- Map continuous numeric data to a colour spectrum
- Translate categorical data to different shapes
- Map the size of a geom to some value (e.g. frequency)
- Etc.

Scales modify the geom object(s)

Penguins: map `species` to colour/shape

To colour penguin species, we have to start again

```
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(mapping = aes(colour = species, shape = species)) +
  theme_bw(base_size = 24)
```

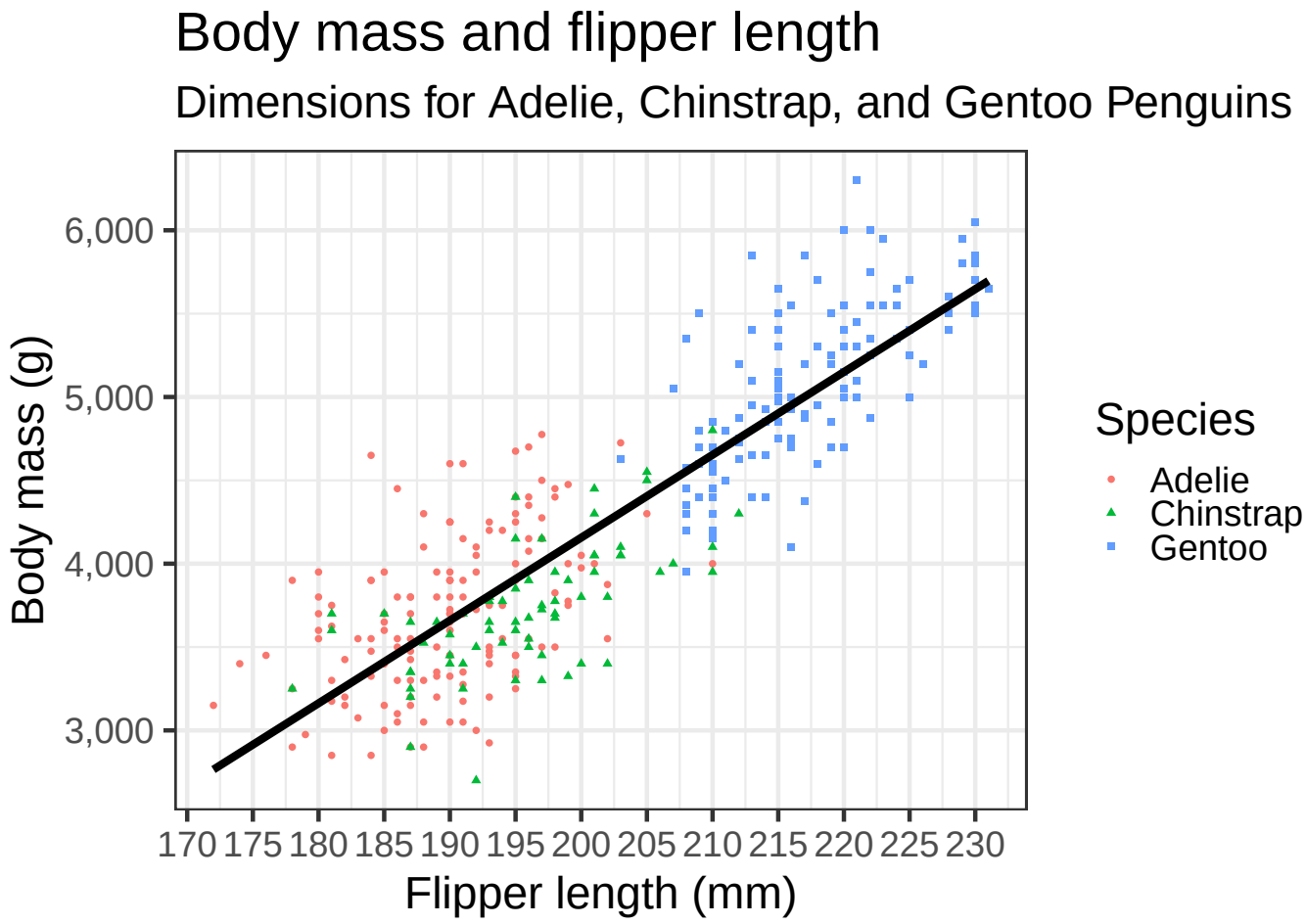


Penguins: map `species` to colour/shape

```
z1 <- penguins |>
  ggplot(mapping = aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(mapping = aes(colour = species, shape = species)) +
  theme_bw(base_size = 24) +
  scale_x_continuous("Flipper length (mm)", breaks = seq(170,230,5)) +
  scale_y_continuous("Body mass (g)", labels = scales::comma) +
  geom_smooth(method = "lm", formula = 'y ~ x',
              se = FALSE, linewidth = 2, colour = "black") +
  labs(title = "Body mass and flipper length",
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",
       colour = "Species", shape = "Species")
```

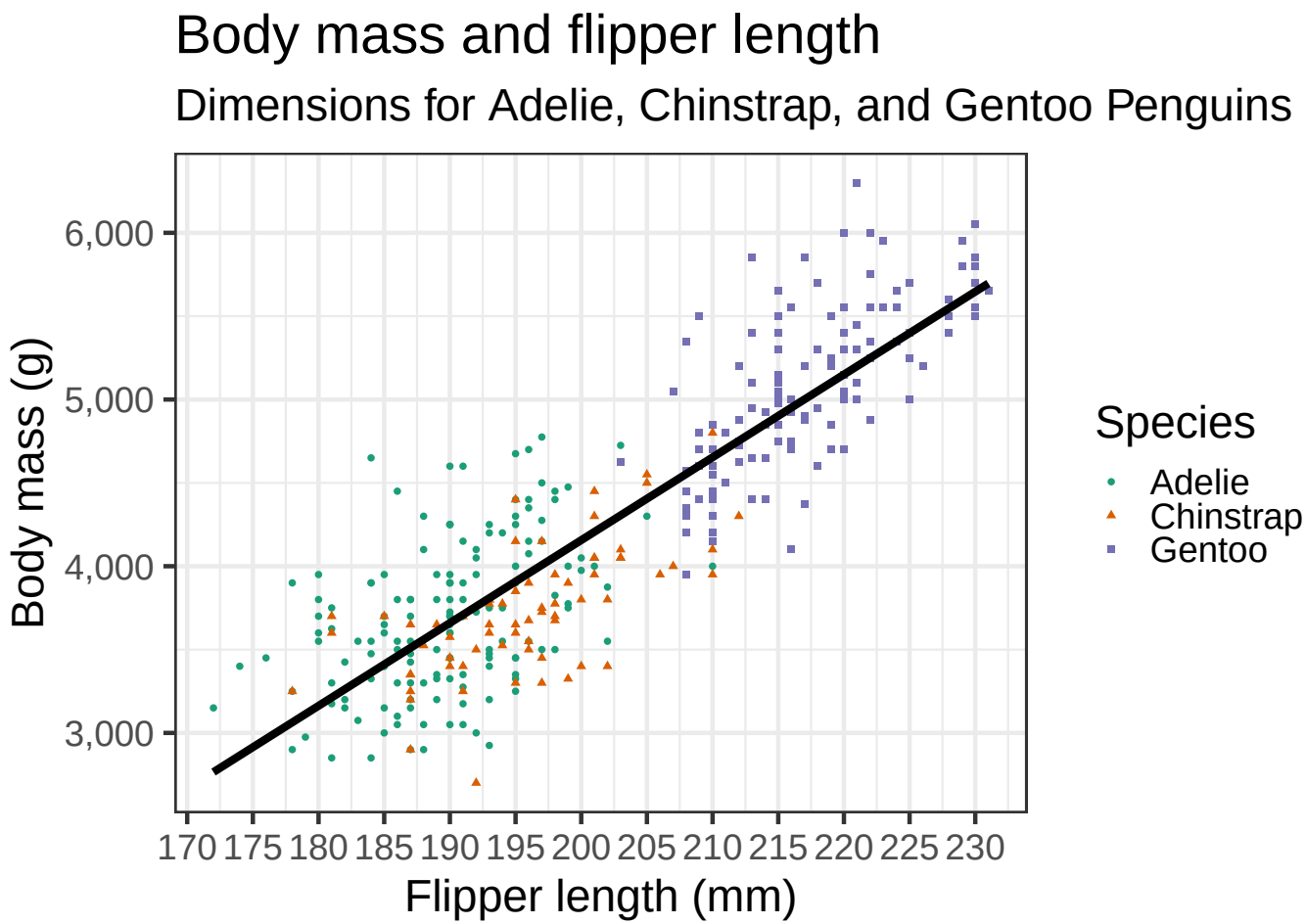
Penguins: map `species` to colour/shape

```
print(z1)
```



Penguins: modify colour scale

```
z1 + scale_colour_brewer(palette = "Dark2")
```



Redundant scales

In the previous slide:

- Shape of dots adds no *new* information
- We call this **redundancy**
  - When two (or more) scales translate the *same* variable to different aesthetics
- Redundancy can overly complicate plots…
  - … but can also add clarity, improve accessibility

Facets and coordinates

Facets allow for multiple plots by mapping subsets of data

- E.g. separate scatterplots by island
- When you facet by a single variable we use a **wrap**
- When we facet by two (or more) variables, we use a **grid**

Coordinate systems "map the position of objects onto the plane of the plot" (Wickham 2010, p.13)

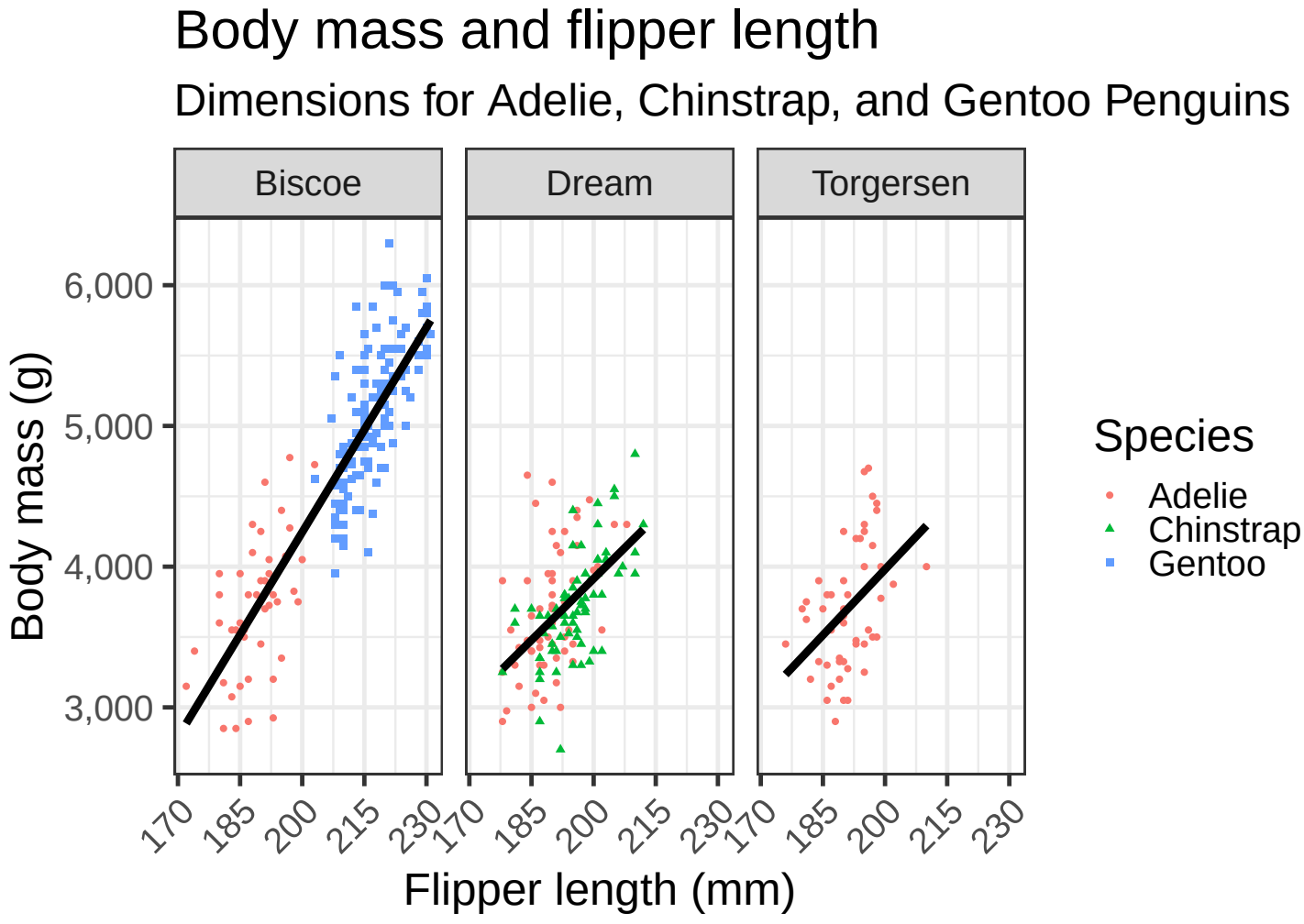
- In almost all cases we use **Cartesian coordinates**
  - Two orthogonal dimension (x, y)
- Alternative systems exist, like polar coordinates:
  - Allow you to draw circular distributions like pie-charts (eww!)

Penguins: create facets with `island`

```
z2 <- z1 +
  scale_x_continuous("Flipper length (mm)", breaks = seq(170,230,15)) +
  theme(axis.text.x = element_text(angle = 45, hjust=1)) +
  facet_grid(cols = vars(island))
```

Penguins: create facets with `island`

```
print(z2)
```



Outline

1. Some principles of data visualisation
2. Grammar of graphics and `{ggplot}`
3. Spatial data

What is spatial data?

Spatial data is data relating to physical space

Physical space is inherently three dimensional, but our data representations are two dimensional

For maps specifically:

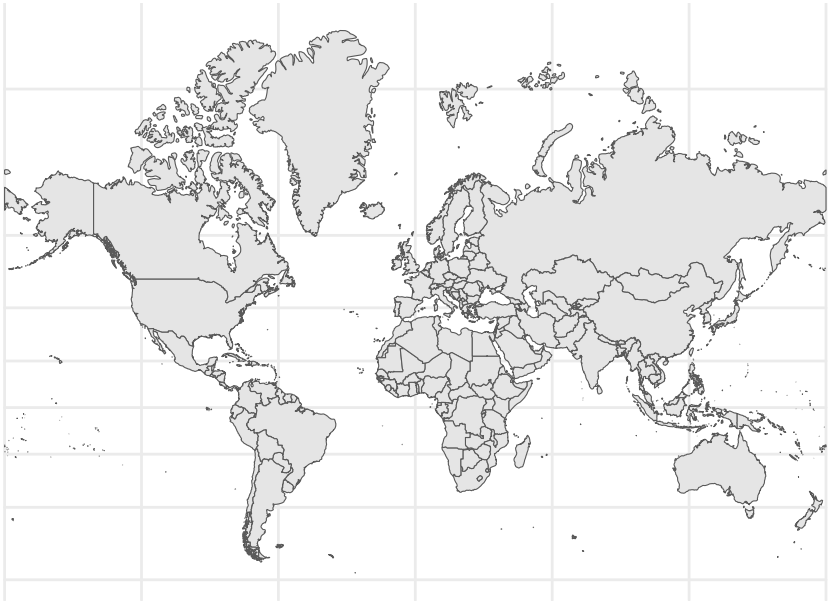
- the Earth is a 3D sphere (I hope you agree)
- but maps are depicted on 2D visualisations

So, mapping requires:

- a (Cartesian) plane (i.e., x and y axis) and
- a **projection** that "translates" 3D space to 2D space

Map projection

Mercator Projection
(without Antarctica)



Robinson Projection
(without Antarctica)

Map projection

Coordinate reference system (CRS)

- A way to link points in a 2D geometry with real places
- Set an origin, locations are defined as coordinates (e.g., x and y) giving distances from origin
- CRS is based on the chosen projection

You need to *choose* a projection that fits your use case

There are tradeoffs, since projections usually distort cardinal directions (north/south/east/west), distances and/or areas

Some commonly used CRS:

- WGS 84: Mercator projection used on web (e.g, Google Maps)
- British National Grid (BNG), used in the UK

Mapping in R

To make maps in R: `{sf}`

- Defines spatial data with “simple features” that represent geometric data types + projection
- Spatial data is loaded into an “sf data frame” with:
  - rows: geographic entities (countries, regions, etc)
  - columns: relevant data about each geographic entity plus one column with map drawing data (“geometries”)
  - (More on geometries this later in the course)
- The coordinates used for plotting are scaled by the projection

Many ways to store data for mapping, but commonly used format is the “shapefile” (`.shp`), which is what we will use

Mapping in R

You can read a shapefile into R using `read_sf()` from `{sf}`

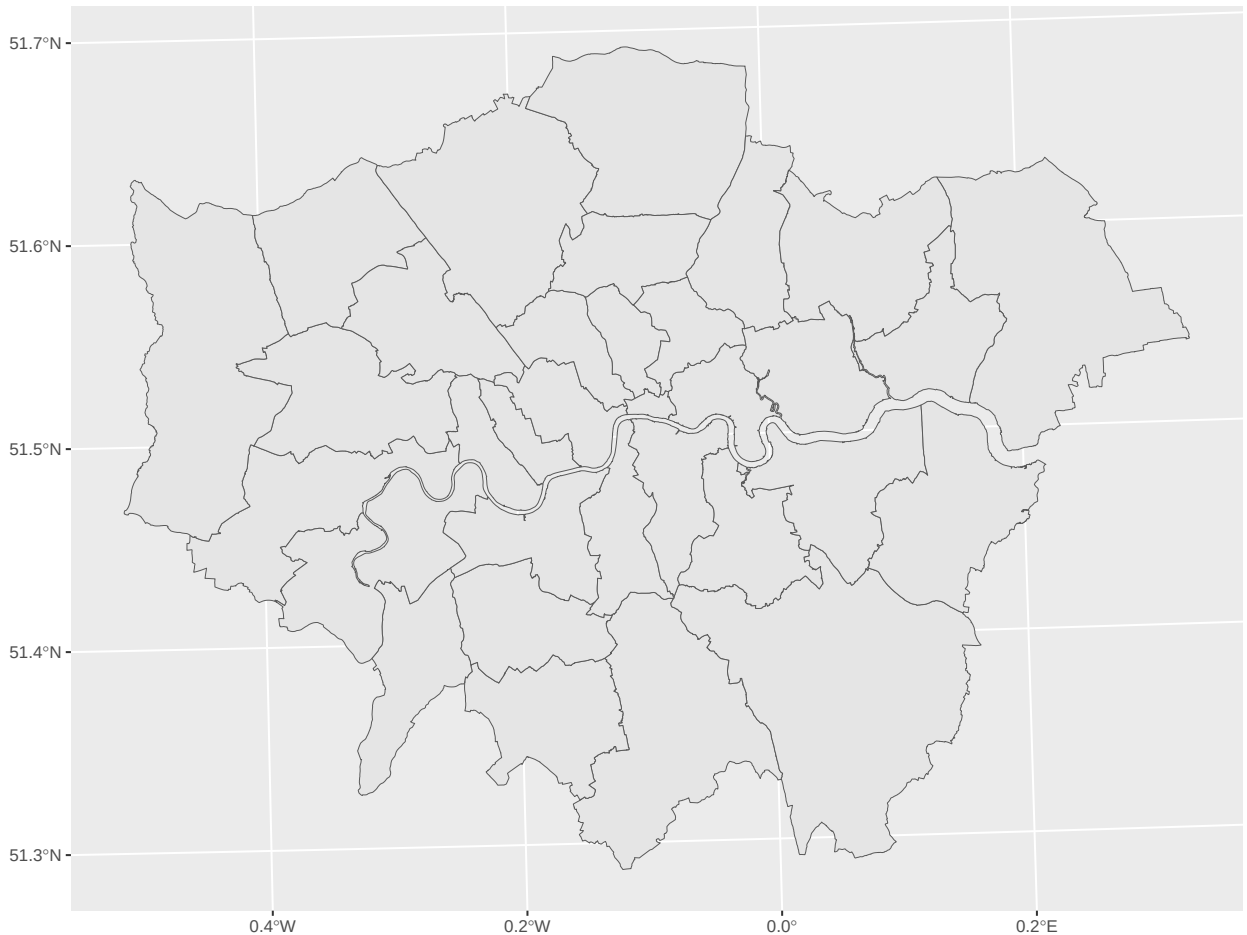
```
# Read shapefile data into R using
lf <- read_sf("data/ESRI/London_Borough_Excluding_MHW.shp")

# Look at the shape of the data
lf |> select(NAME, geometry) |> head()
```

```
Simple feature collection with 6 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 507007.4 ymin: 155850.8 xmax: 561957.5 ymax: 194889.3
Projected CRS: OSGB36 / British National Grid
# A tibble: 6 x 2
  NAME                                                  geometry
  <chr>                                          <MULTIPOLYGON [m]>
1 Kingston upon Thames (((516401.6 160201.8, 516407.3 16021~
2 Croydon              (((535009.2 159504.7, 535005.5 15950~
3 Bromley              (((540373.6 157530.4, 540361.2 15755~
4 Hounslow             (((521975.8 178100, 521967.7 178096.~
5 Ealing               (((510253.5 182881.6, 510249.9 18288~
6 Havering             (((549893.9 181459.8, 549894.6 18146~
```

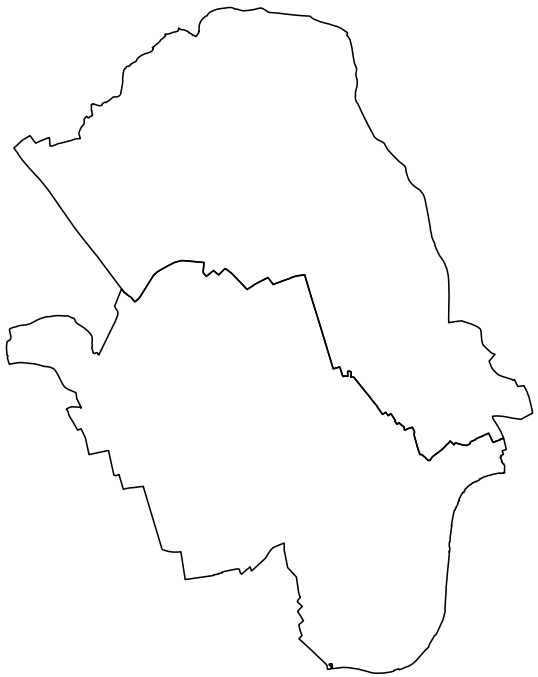
Mapping in R: initialise and show borders

```
ggplot(lf) +
  geom_sf() # optional here: aes(geometry = geometry)
```



Mapping in R

```
lf |>
  filter(str_detect(NAME, "(Westminster|Camden)")) |>
  ggplot() +
  geom_sf(colour = "black", fill = NA, linewidth = 0.5) +
  theme_void()
```



Mapping in R

You can add points to a map using latitude and longitude

- Find coordinates using Google Maps

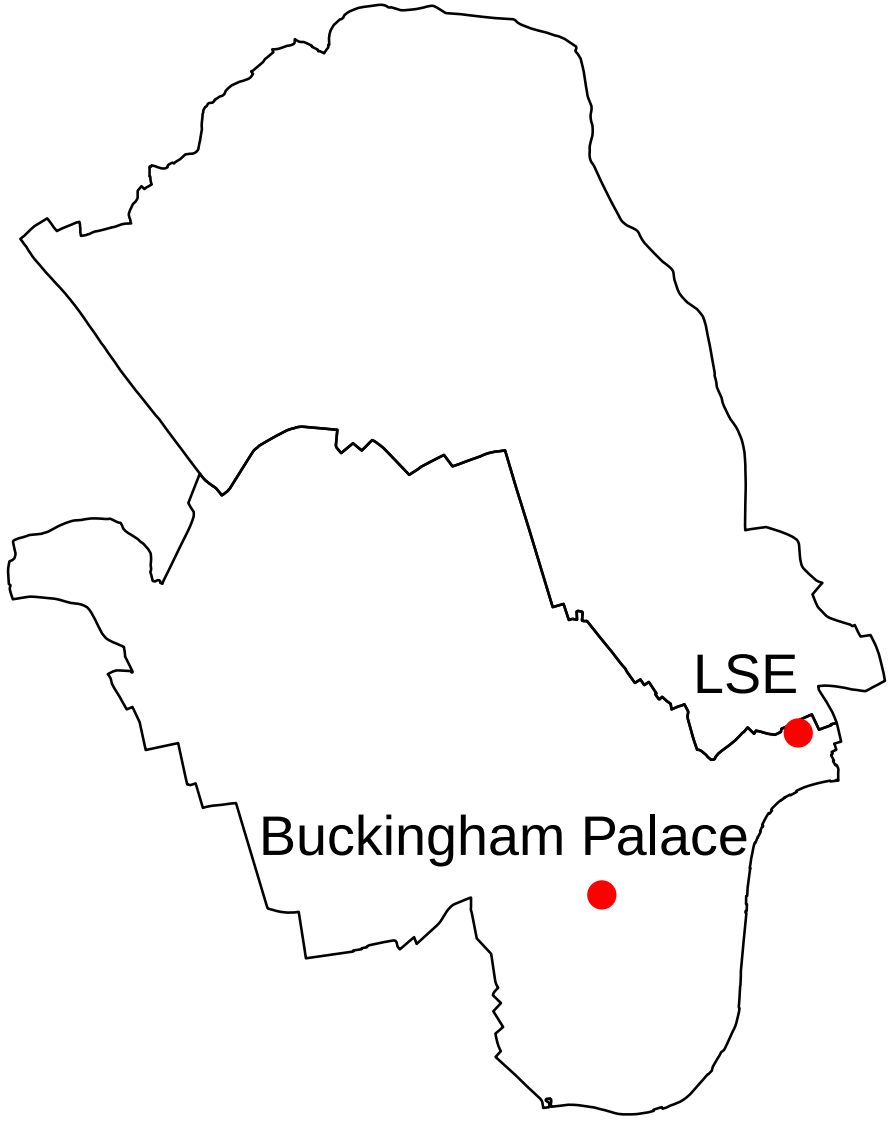
```
pf <- tibble(name = c("LSE", "Buckingham Palace"),
             lon = c(-0.1165, -0.1419),
             lat = c(51.5146, 51.5022)) |>
      st_as_sf(coords = c("lon", "lat"), crs = 4326) |>
      st_transform(crs = 27700) # optional, but smart
print(pf)
```

```
Simple feature collection with 2 features and 1 field
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 529065 ymin: 179776.9 xmax: 530792.3 ymax: 181200.9
Projected CRS: OSGB36 / British National Grid
# A tibble: 2 x 2
  name                          geometry
* <chr>                      <POINT [m]>
1 LSE                (530792.3 181200.9)
2 Buckingham Palace   (529065 179776.9)
```

Mapping in R

```
lf |>
  filter(str_detect(NAME, "(Westminster|Camden)")) |>
  ggplot() +
  geom_sf(colour = "black",
          fill = NA,
          linewidth = 0.5) +
  geom_sf(data = pf,
          colour = "red",
          size = 5) +
  geom_sf_text(data=pf,
               aes(label=name,
                   hjust = c(1,0.7),
                   vjust = c(-1,-1)),
               size=8) +
  theme_void()
```

Mapping in R



Mapping in R

```
lf |>
  ggplot() +
  geom_sf(colour = "gray70",
          fill = "gray95",
          linewidth = 0.5) +
  geom_sf(data = pf,
          colour = "red",
          size = 6) +
  geom_sf_text(data=pf,
               aes(label=name,
                   hjust = c(0.5,1.05),
                   vjust = c(-0.8,0.4)),
               size=10) +
  theme_void()
```

Mapping in R