Week 4: Data visualisation

Charlotte Hadley

Topics for today

1. Why do we use charts to tell stories?
2. Evidence-based visual perception theory
3. Advice on choosing charts
4. Advice on using colour in charts
5. Using this advice to tell stories with charts built with `{ggplot2}`

Why do we use charts??

~~A picture is worth a thousand words~~

Data visualisations are demonstrably useful

There is considerable experimental evidence for data visualisations improving:

- Comprehension of data
- Decision making accuracy and confidence

Evidence has been collected using eye-tracking, survey filling and interviews.

For a good overview of the available research see Eberhard 2021[1].

> Some of these studies consider tables to be a type of data visualisation.
>
> I agree with this! Tables are often awesome choices for presenting data - let's talk more about this later today.

Data visualisations are demonstrably useful

In 1973 Anscombe[2] published a paper designed to demonstrate…

> Graphs are essential to good statistical analysis.

To do so he simulated 4 datasets sharing **many** identical statistical properties.

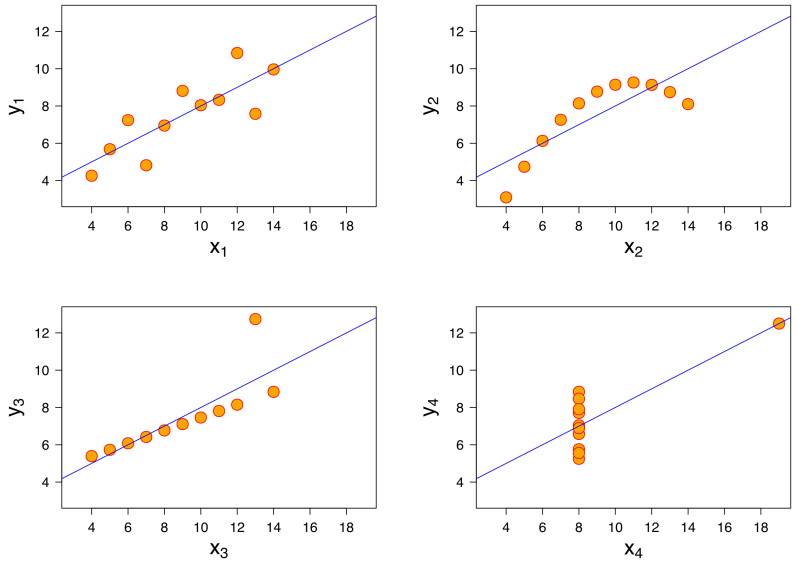
Each of the four data sets yields the same standard output from a typical regression program, namely

Number of observations $(n) = 11$
Mean of the x's $(\bar{x}) = 9.0$
Mean of the y's $(\bar{y}) = 7.5$
Regression coefficient (b_1) of y on $x = 0.5$
Equation of regression line: $y = 3 + 0.5\,x$
Sum of squares of $x - \bar{x} = 110.0$
Regression sum of squares $= 27.50$ (1 d.f.)
Residual sum of squares of $y = 13.75$ (9 d.f.)
Estimated standard error of $b_1 = 0.118$
Multiple $R^2 = 0.667$

Data visualisations are demonstrably useful

However, if you visualised the datasets it was obvious these datasets were **fundamentally** different to one another.

These charts are now known as *Anscombe's quartet*[2].

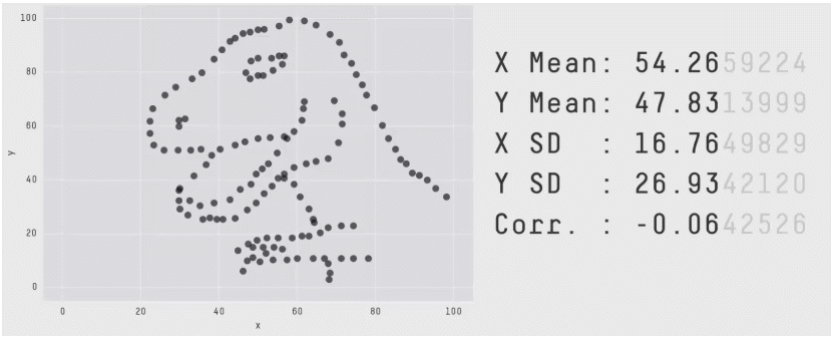


Data visualisations are demonstrably useful

The “Datasaurus Dozen” is a modern reimagining of the original quartet[3].

Datasaurus was originally created by Alberto Cairo[4].

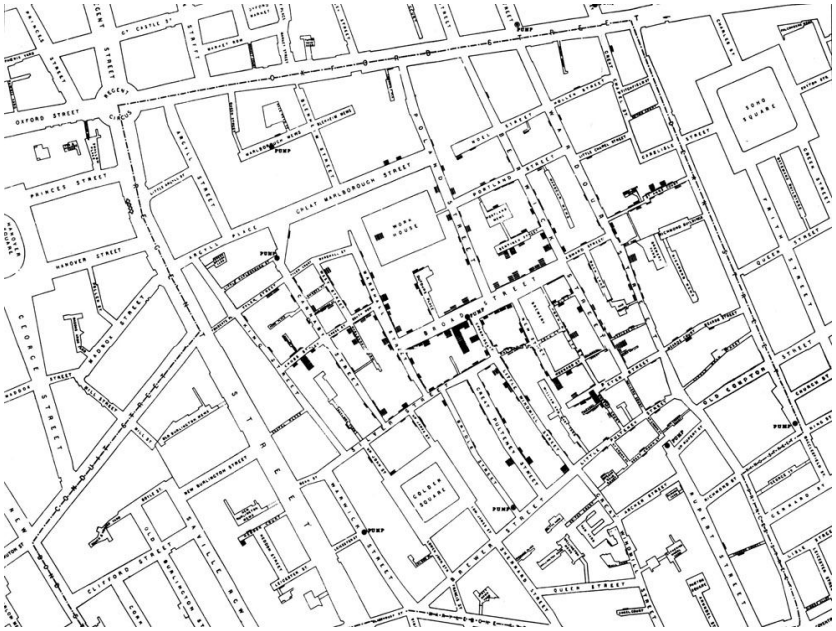
… there’s now an R package for building your own metamers
eliocamp.github.io/metamer/



ALWAYS.

Always visualise your datasets.

Data visualisations are demonstrably useful



There are several historical visualisations that have fundamentally changed social policy and behaviour.

This is a map from John Snow in 1855[5] that ties a cholera outbreak to a specific water pump.

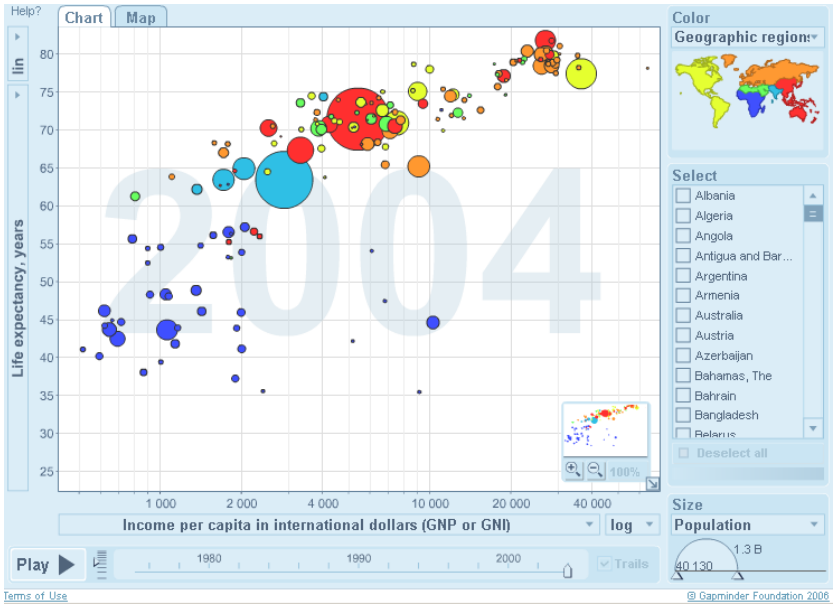
Combined with Snow's statistical analyses this was a significant step towards the development and acceptance of germ theory.

Data visualisations are demonstrably useful

Data visualisations are demonstrably useful

In 2006 Hans Rosling[7] gave an incredible TED talk where he introduced animated bubble charts as a tool to tell stories about global development.

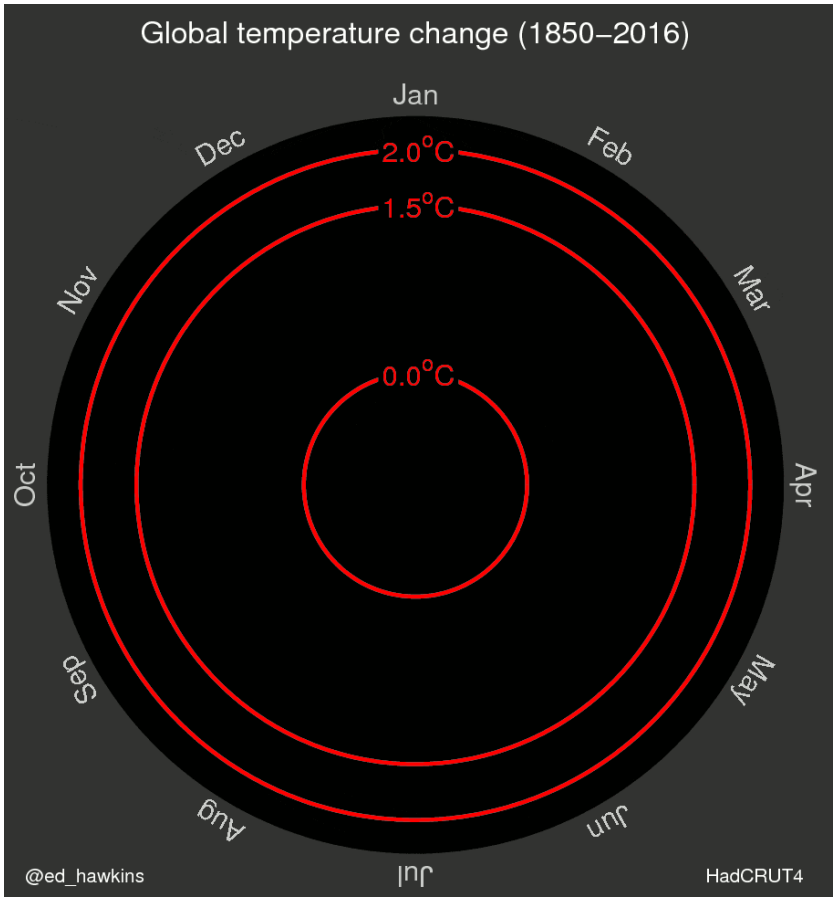
These charts helped demonstrate the value of interactive and animated data visualisations - which is why Google bought the tool behind the charts!



Data visualisations are demonstrably useful

A more recent example of a very powerful data visualisation is the spiralling global temperature GIF from 2016 by Ed Hawkins[8].

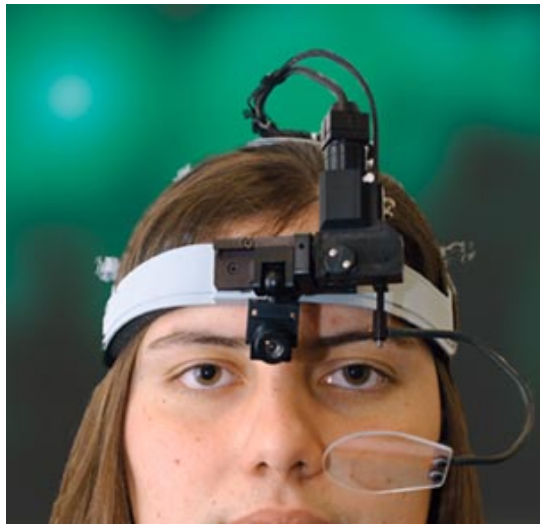
We can create animated GIF with `{ggplot2}` via the `{gganimate}` package. In fact, Pat Schloss[9] has a YouTube video and GitHub repo recreating this chart with R.



Evidence-based visual perception theory

Evidence-based visual perception theory

There is a wealth of evidence-based research in **how precisely or accurately** charts are perceived by readers.



Source: Wikimedia.org

Our evidence comes from:

- Eye tracking. We're really good at measuring where the eye is looking, for how long and how intently.
- Asking trial participants to estimate or compare values in charts.

> There are open debates[1] on **how our internal visual perception system works** - what the brain is doing.

Elementary perceptual tasks

Back in 1984 Cleveland & McGill[11] published their seminal paper on graphical perception theory where they defined "elementary perceptual tasks".

This study is the backbone of much of the research in this field.

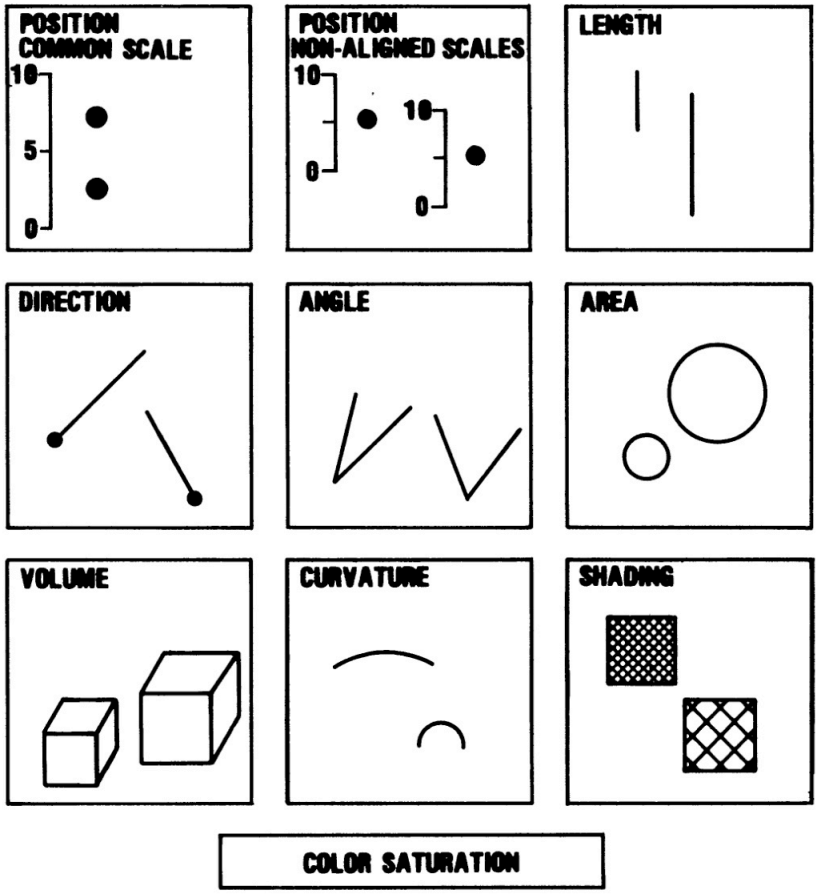


Figure 1. Elementary perceptual tasks.

Elementary perceptual tasks

Cleveland & McGill[11] designed many experiments where participants were asked to:

- Identify the largest/smallest segment
- Estimate what % the smaller segment was of the larger segment

The **accuracy** of subject estimates was then statistically analysed.

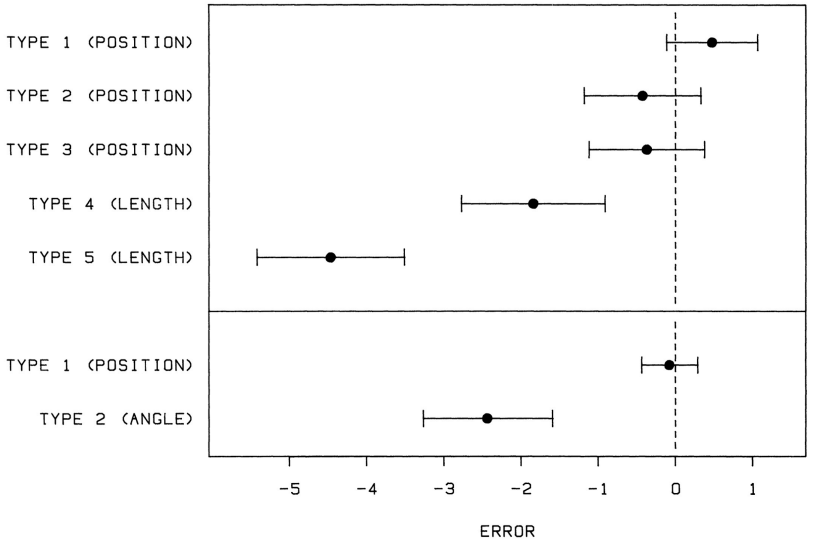


Figure 20. Error means and 95% confidence intervals for judgment types in position–length experiment (top) and position–angle experiment (bottom).

Crowd-sourced evidence for perception theory

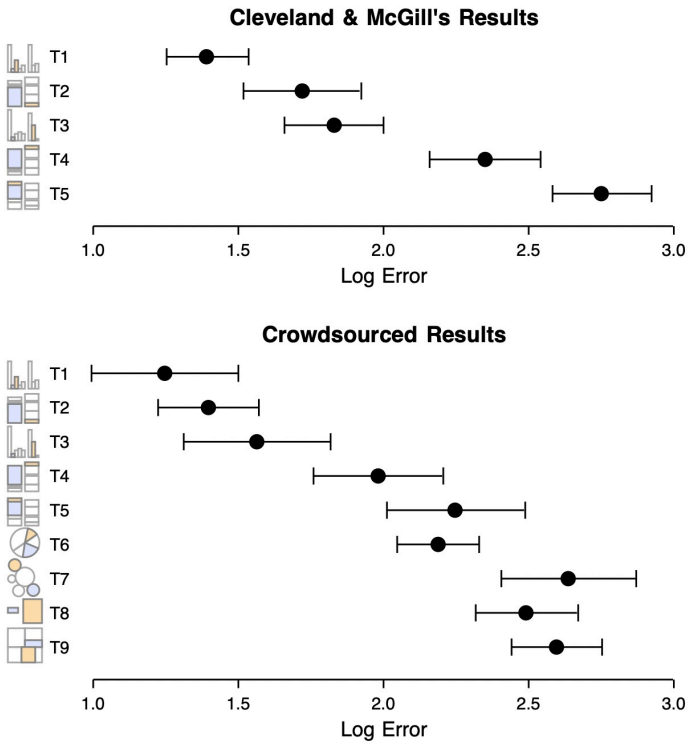


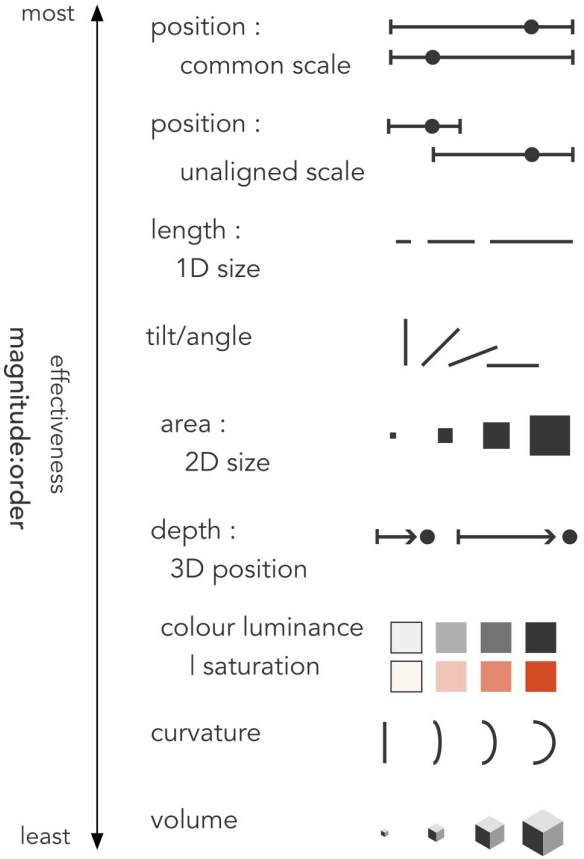
Figure 4: Proportional judgment results (Exp. 1A & B). Top: Cleveland & McGill's [7] lab study. Bottom: MTurk studies. Error bars indicate 95% confidence intervals.

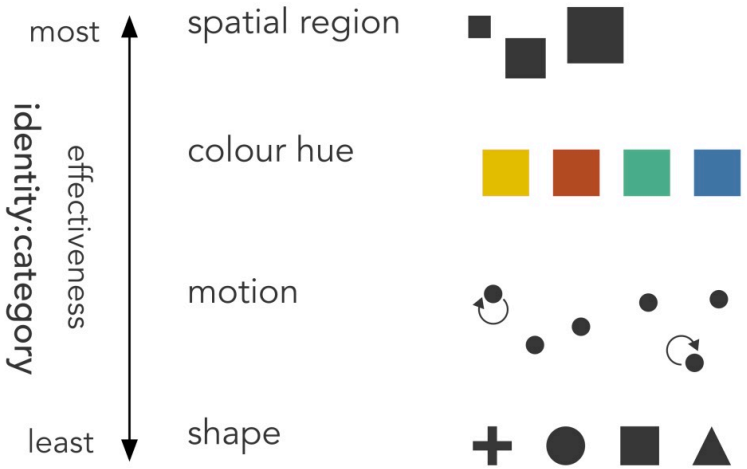
Heer & Bostock[12] replicated this study using Amazon's Mechanical Turk with 3,481 participants in 2010.

They validated the results of Cleveland & McGill[11] and provided further evidence that…

> There is a hierarchy of elementary perceptual tasks - or chart elements - when **accuracy** matters.

Ordering channels of communication (by accuracy)





Images from Beecham et al[13]

… real-world applications of visual perception theory (I)

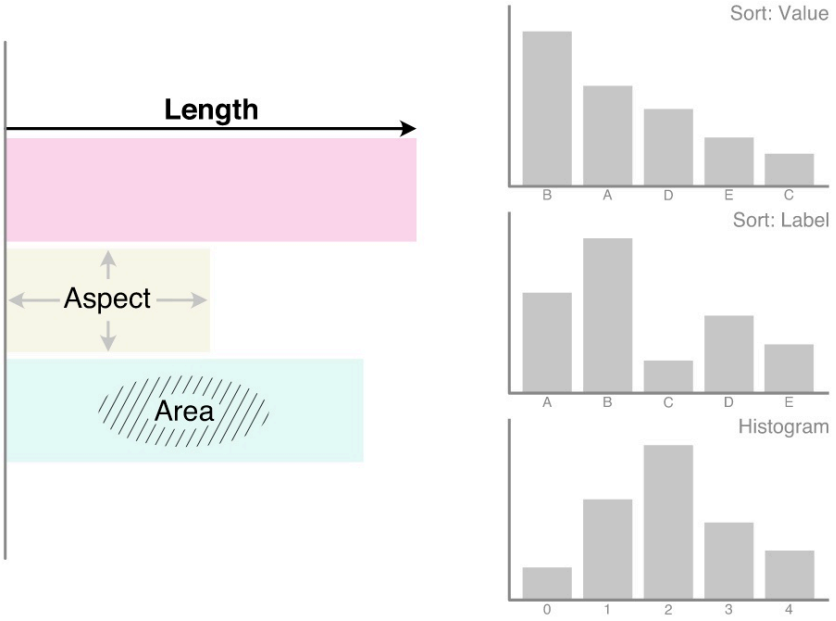


FIGURE 2. Bar charts encode values as their length, but also their area, aspect ratio, and overall shape. Sorting is often used for specific use cases.

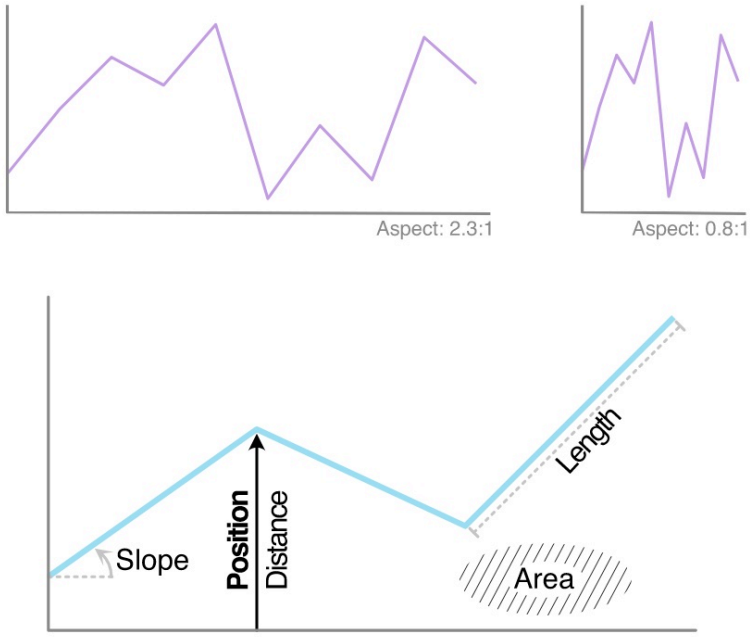


FIGURE 3. Line charts are specified by the location of the points connected by lines, but are read as slope and length, as well as area. Aspect ratio of the chart is also generally considered important.

Images from Robert Kosara[14]

… real-world applications of visual perception theory (II)

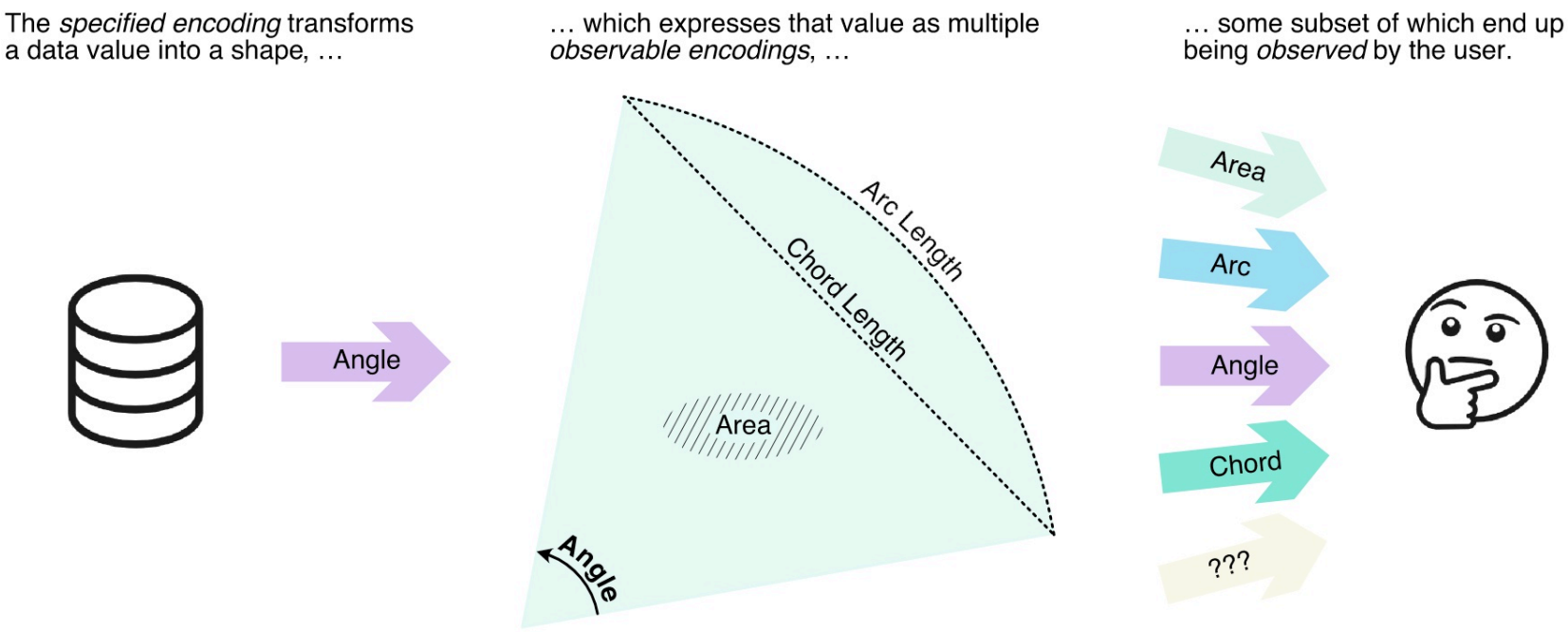


FIGURE 1. Pie charts are specified by angle, but may also be read by area, arc length, or even chord. Shape recognition is also likely for specific angles like 90°/25% and 180°/50%.

Images from Robert Kosara[14]

… real-world applications of visual perception theory (III)

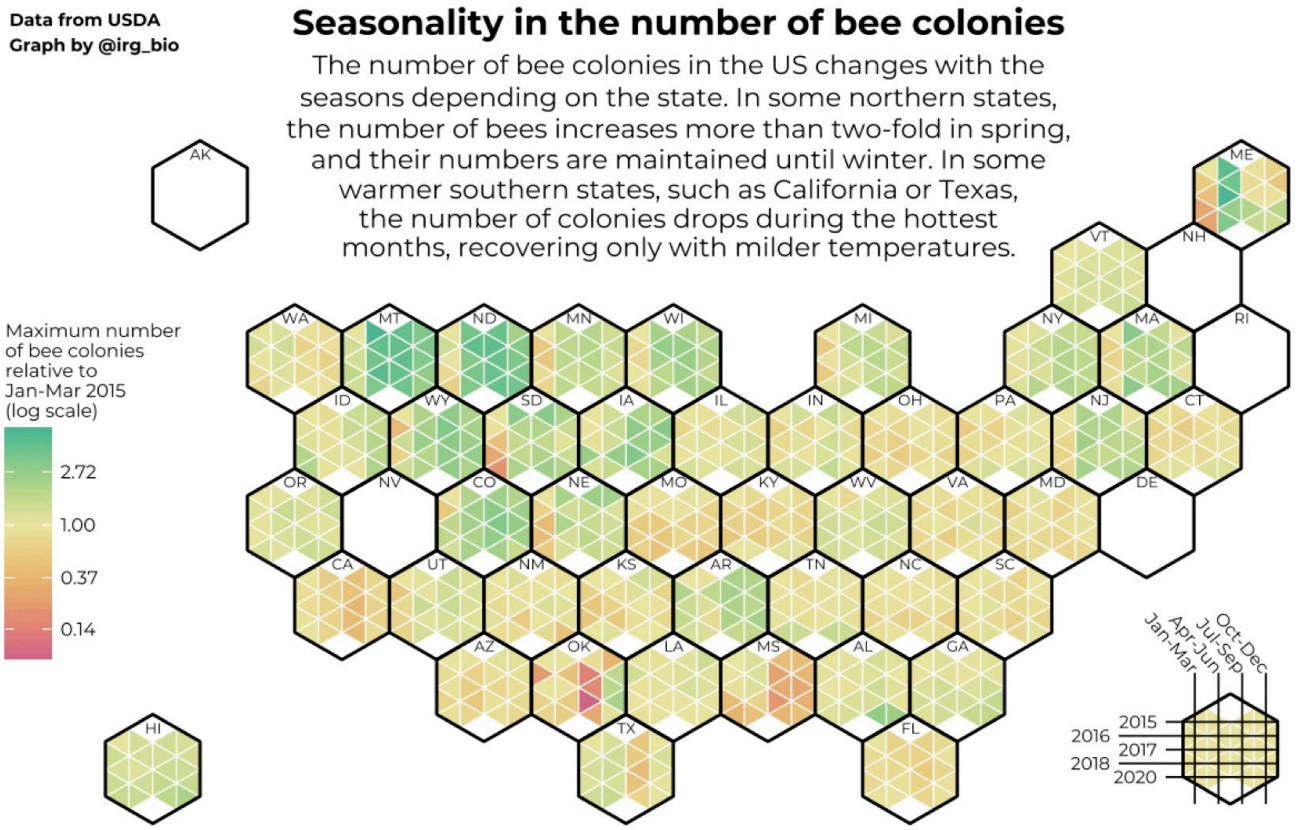


Image found on Twitter from @irg_bio[15] - code for chart available from GitHub[16].

Why is someone ~~reading~~ measuring your chart?

Why is someone ~~reading~~ measuring your chart?

To extract accurate values

> The **magnitude** of chart elements.

To quantatively compare values.

> The **part to whole** or **relative magnitude** of chart elements.

To find the largest/smallest value.

> The **ranking** of chart elements

To find *unusual* values.

> The **distribution**, **ranking** or **magnitude** of chart elements

Why is someone reading your chart?

You have a story you want to tell

> There's lots we can do to help guide the reader to understand your chart and follow the story you're telling. We'll cover some examples during this course.

The reader wants to see the data

> Charts (and tables) are the best way to see the "big picture" of a dataset - a single value (eg mean) is kind of useless. Interactivity is really useful to allow readers to properly **explore** the dataset.

The reader has a preconception about the data

> Readers might be approaching a chart biased with a particular theory about the data. We can do our best to make our charts easy to read and avoid common pitfalls.

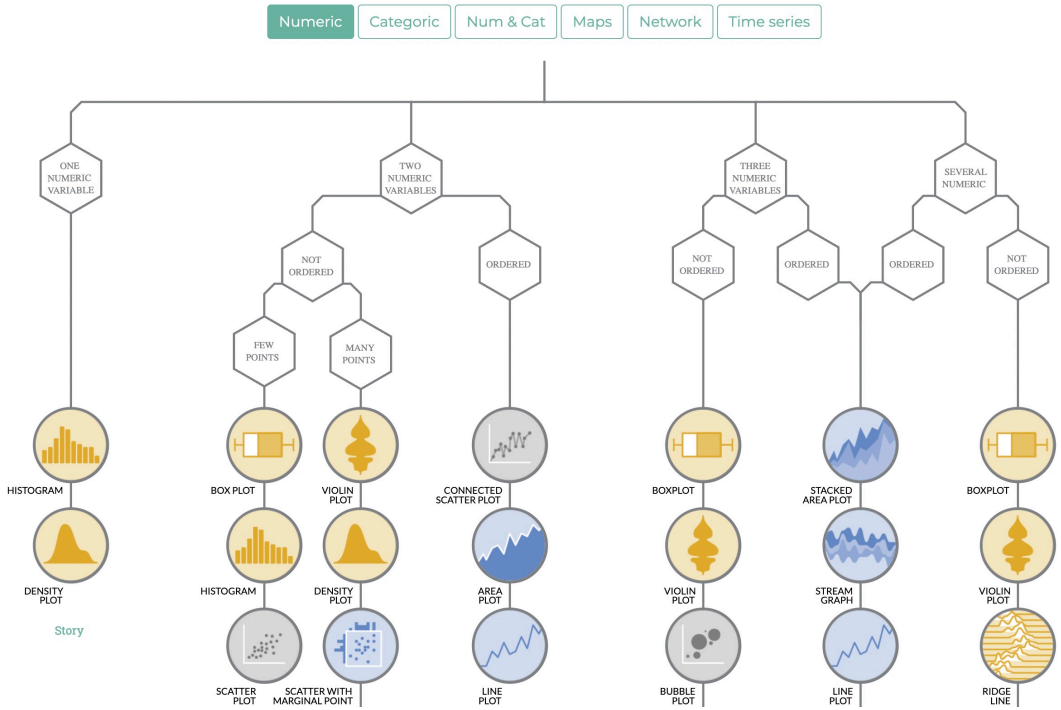
How do we choose a chart?

Use data columns to choose charts

```
> msleep
# A tibble: 83 × 11
   name    genus vore  order conse…¹ sleep…² sleep…³ sleep…⁴ awake  brainwt  bodywt
   <chr>   <chr> <chr> <chr> <chr>     <dbl>   <dbl>   <dbl> <dbl>    <dbl>   <dbl>
 1 Cheetah Acin… carni Carn… lc         12.1    NA      NA    11.9 NA         50
 2 Owl mo… Aotus omni  Prim… NA         17       1.8    NA     7    0.0155     0.48
 3 Mounta… Aplo… herbi Rode… nt         14.4     2.4    NA     9.6 NA          1.35
 4 Greate… Blar… omni  Sori… lc         14.9     2.3     0.133  9.1  0.00029    0.019
 5 Cow     Bos   herbi Arti… domest…     4       0.7     0.667 20    0.423    600
 6 Three-… Brad… herbi Pilo… NA         14.4     2.2     0.767  9.6 NA          3.85
 7 Northe… Call… carni Carn… vu          8.7     1.4     0.383 15.3 NA         20.5
 8 Vesper… Calo… NA    Rode… NA          7      NA      NA    17   NA          0.045
 9 Dog     Canis carni Carn… domest…    10.1     2.9     0.333 13.9  0.07      14
10 Roe de… Capr… herbi Arti… lc          3      NA      NA    21    0.0982    14.8
# … with 73 more rows, and abbreviated variable names ¹conservation, ²sleep_total,
#   ³sleep_rem, ⁴sleep_cycle
# ℹ Use `print(n = ...)` to see more rows
```

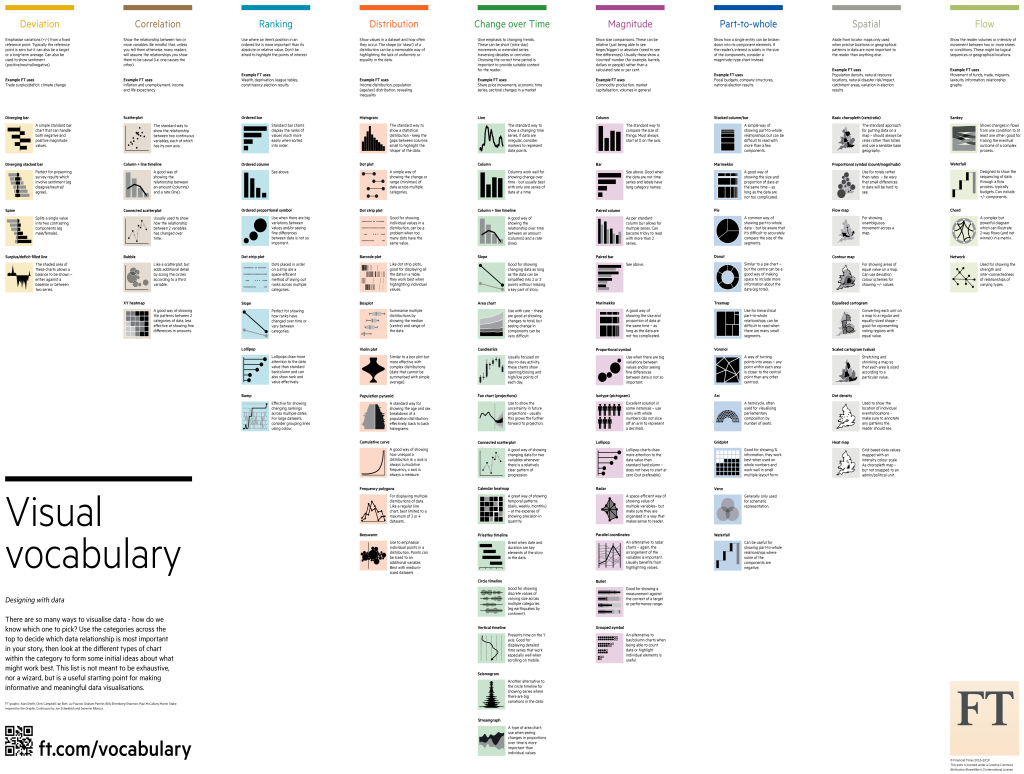
Use your story to choose charts

data-to-viz.com



> This site also provides simple to follow instructions for using {ggplot2} to build every single chart type you can find on the website.

ft-interactive.github.io/visual-vocabulary



{ggplot2} for charts

 Task: Setup a new project

SLIDE 1 OF 3

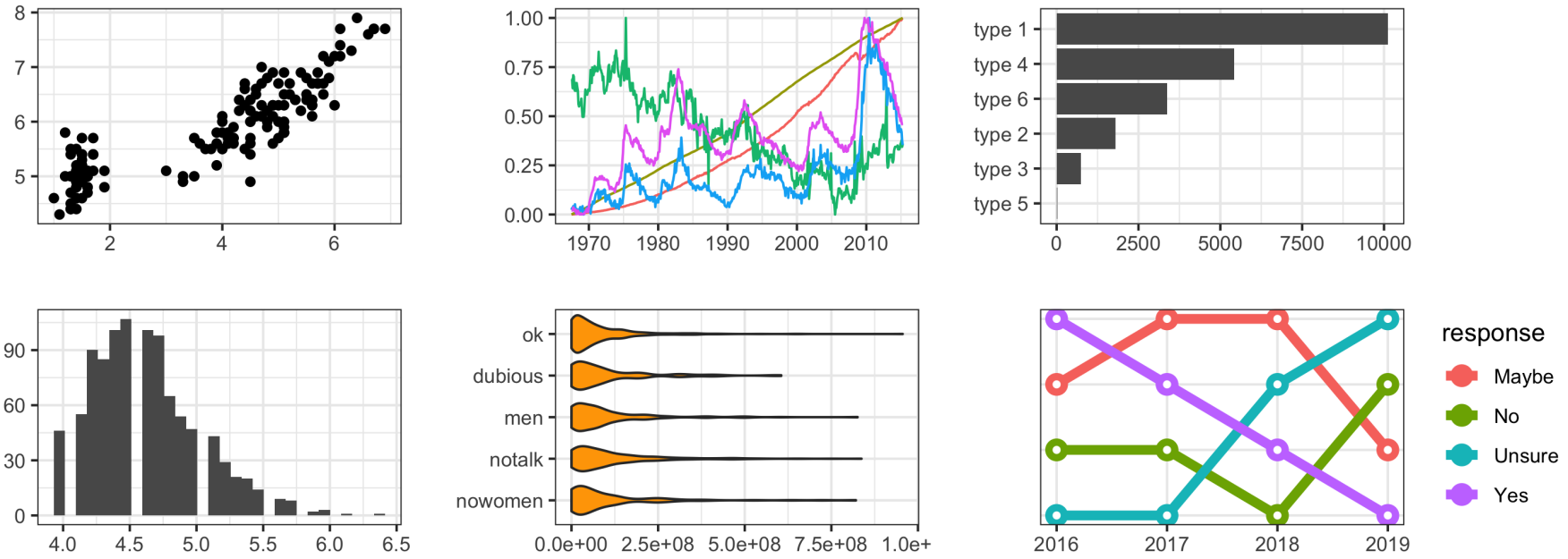
1. Create a new project called something like `week-4_dataviz.Rproj`
2. Add a new RMarkdown document called `ggplot2-notes.Rmd`

> We're going to do some structured and unstructured code during today. During the workshop I'll be asking to you to create your own charts.

ggplot2: A Grammar of Graphics

`{ggplot2}` is an incredibly powerful and flexible tool for building static dataviz.

We can build (almost)[1] **any** static chart we can conceive of.



[1] - Dual y-axis charts must be transformations of one another (for good reasons)

Building blocks of a {ggplot2} chart

Aesthetics

Geoms

Scales

Guides

Theme

Aesthetics

Aesthetics are used to create **mappings** between columns in our datasets and the coordinate systems of our chart:

```
msleep %>%
  ggplot() +
  aes(
    x = sleep_total,
    y = sleep_rem,
    colour = vore
  )
```

{ggplot2} uses *tidy evaluation* to allow us to use bare column names in our code.

Aesthetics

Where is aes() placed?	What it does
Inside `ggplot()` or on its own	Sets the aesthetics for the entire {ggplot2} object. These could be considered the *coordinate system aes()*
Inside `geom_*()`	Sets aesthetics for a specific geom *within* the existing coordinate system aes() for the {ggplot2} object. These should be considered *geom specific aes()*

Geoms

Geoms use the aesthetics to add layers to our charts.

```
msleep %>%
  ggplot() +
  aes(
    x = sleep_total,
    y = sleep_rem,
    colour = vore
  ) +
  geom_point()
```

There are 50+ geoms baked into the {ggplot2} package.

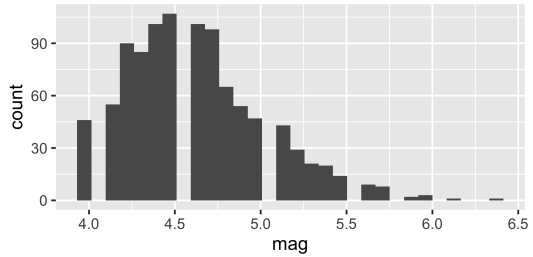
`geom_abline()`, `geom_area()`, `geom_bar()`, `geom_bin2d()`, `geom_blank()`, `geom_boxplot()`, `geom_col()`, `geom_contour()`, `geom_contour_filled()`, `geom_count()`, `geom_crossbar()`, `geom_curve()`, `geom_density()`, `geom_density_2d()`, `geom_density_2d_filled()`, `geom_density2d()`, `geom_density2d_filled()`, `geom_dotplot()`, `geom_errorbar()`, `geom_errorbarh()`, `geom_freqpoly()`, `geom_function()`, `geom_hex()`, `geom_histogram()`, `geom_hline()`, `geom_jitter()`, `geom_label()`, `geom_line()`, `geom_linerange()`, `geom_map()`, `geom_path()`, `geom_point()`, `geom_pointrange()`, `geom_polygon()`, `geom_qq()`, `geom_qq_line()`, `geom_quantile()`, `geom_raster()`, `geom_rect()`, `geom_ribbon()`, `geom_rug()`, `geom_segment()`, `geom_sf()`, `geom_sf_label()`, `geom_sf_text()`, `geom_smooth()`, `geom_spoke()`, `geom_step()`, `geom_text()`, `geom_tile()`, `geom_violin()`, `geom_vline()`

As we'll see later, there are *many* {ggplot2} extension packages that add **even more** geoms to the mix.

Some geoms are built from others (I)

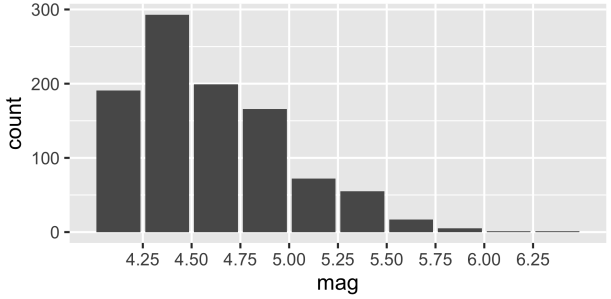
`geom_histogram()` has clever tricks to make useful histograms

```
1 ggplot(quakes, aes(mag)) +
2   geom_histogram()
```



It's built by calling `geom_bar()`

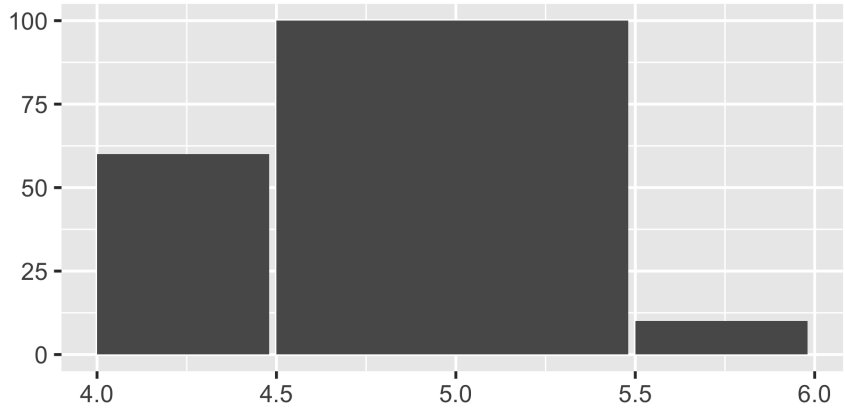
```
1 ggplot(quakes, aes(mag)) +
2   geom_bar() +
3   scale_x_binned()
```



Some geoms are built from others (II)

But `geom_bar()` itself is built from `geom_rect()`.

```
rect_data <- tribble(
  ~x_min, ~x_max, ~y_min, ~y_max,
  4, 4.48, 0, 60,
  4.5, 5.48, 0, 100,
  5.5, 5.98, 0, 10
)
rect_data %>%
  ggplot() +
  geom_rect(aes(xmin = x_min,
                xmax = x_max,
                ymin = y_min,
                ymax = y_max)) +
    theme_gray(base_size = 25)
```



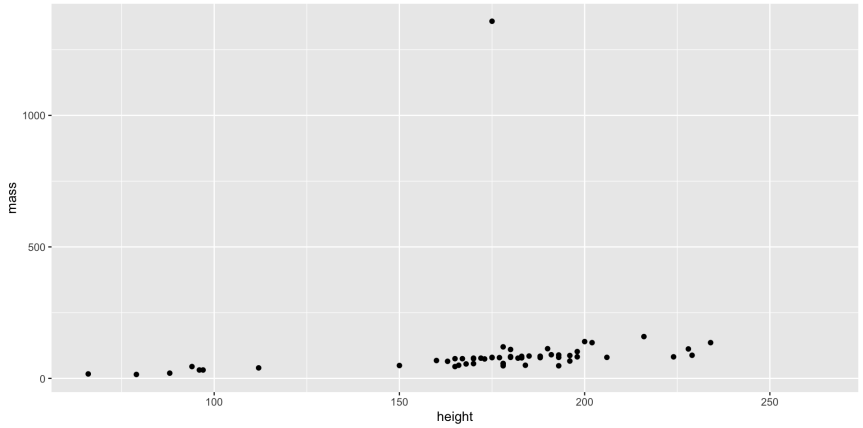
> There are **8 primitives** from which all other geoms are built:

`geom_blank()`, `geom_path()`, `geom_point()`, `geom_polygon()`, `geom_rect()`, `geom_ribbon()`, `geom_segment()`, `geom_text()`

All geoms have x and y aesthetics

These tell the geom where it needs to be drawn:

```
starwars %>%
  ggplot() +
  aes(x = height,
      y = mass) +
  geom_point()
```



Some geoms need more than just x and y

Let's `geom_segment()` to visualise some of the eras of the dinosaurs:

```
dinosaurs <- tribble(
  ~period, ~start, ~end,
  "Triassic Period", -251e6, -225e6,
  "Late Triassic Period", -225e6, -200e6,
  "Jurassic Period", -200e6, -150e6,
  "Late Jurassic Period", -150e6, -145e6
)
```

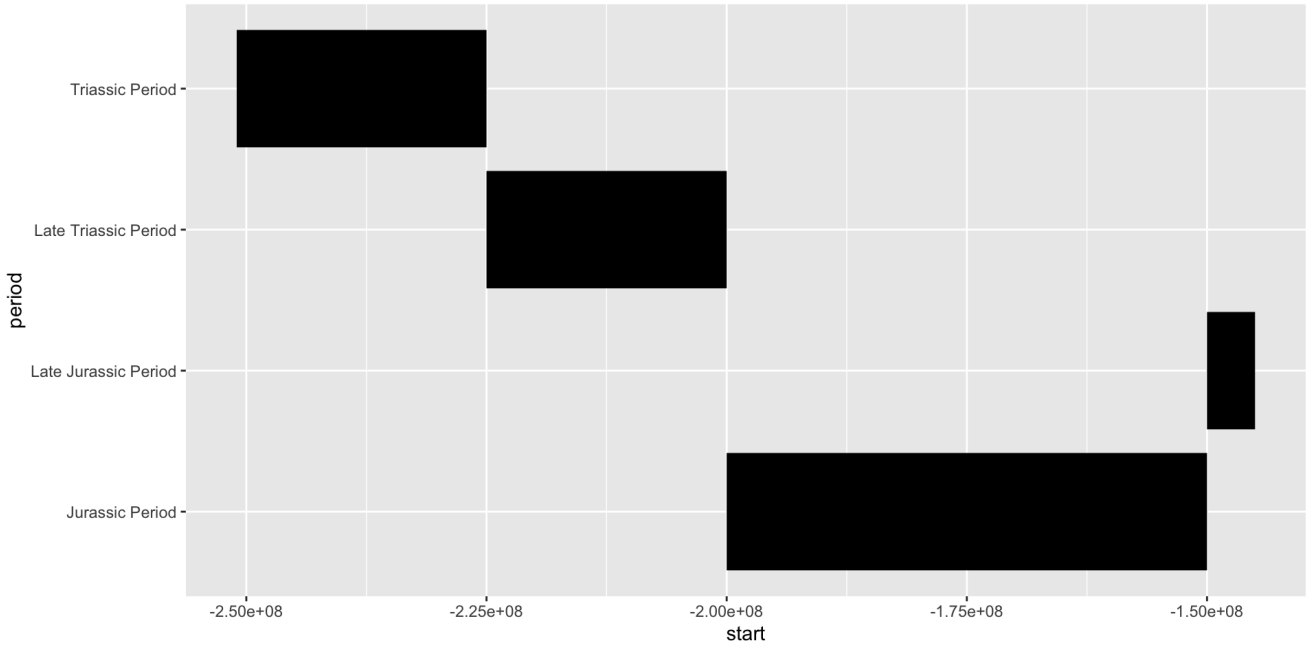
To build this chart we need to specify **all** of the following: x, xend, y and yend.

Use `size` to affect geom size

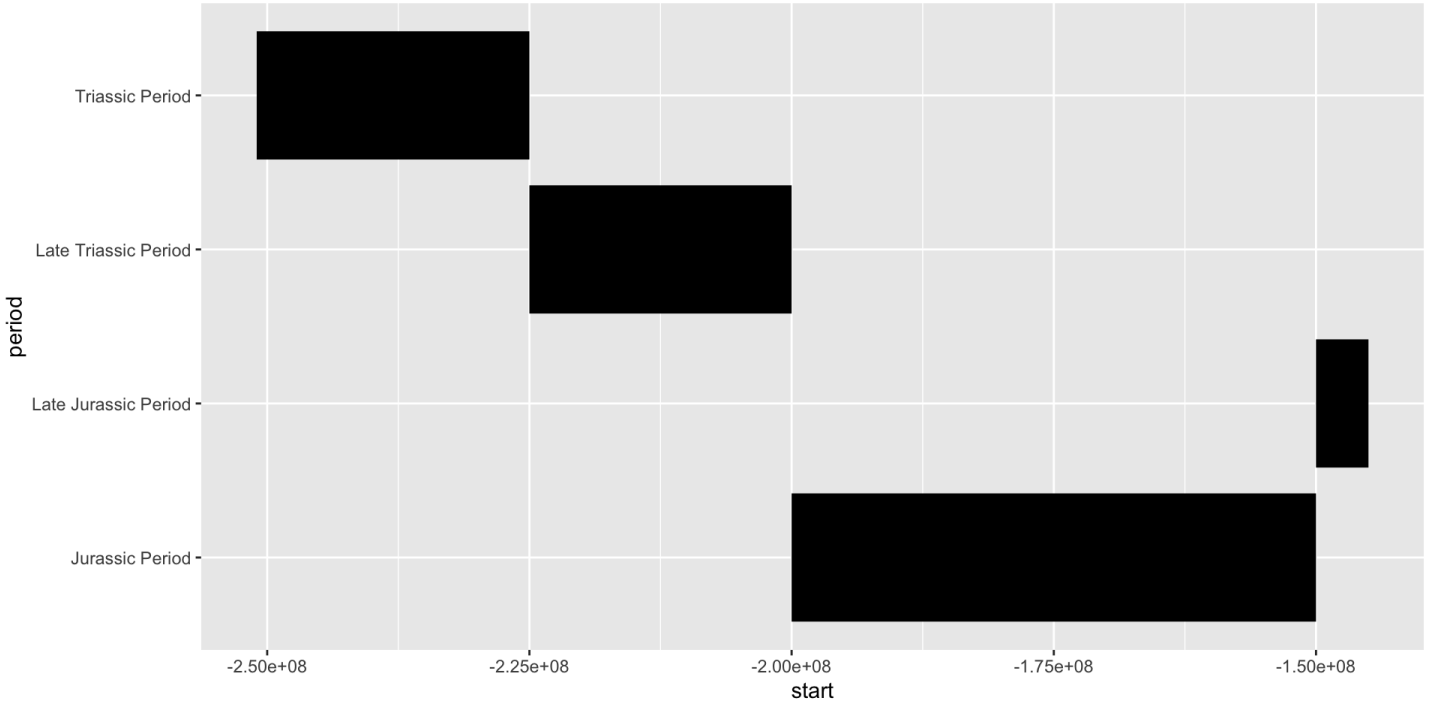
In many charts we want geoms to be **thicker**, **bigger** or *just be more prominent*.

Timeline (or Gantt charts) are good examples of this. We want the segments to be thicker to improve the readability of the chart - this comes down to the `size` aesthetic.

```
dinosaurs %>%
  ggplot() +
  aes(x = start, xend = end,
      y = period, yend = period) +
  geom_segment(size = 30)
```



Out of order dinosaurs



This is still a **bad chart**.

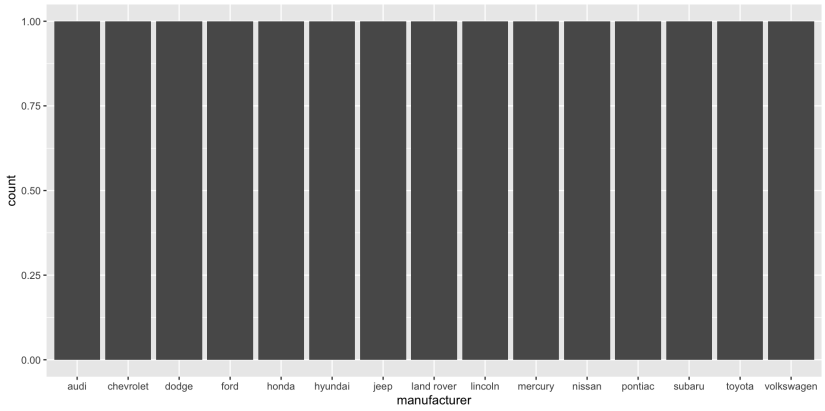
The eras are not ordered in geological time, instead they're ordered (reverse) alphabetically.

To control the order of things in `{ggplot2}` charts we must use **factors** - which are picked up by the **scales**.

Some geoms are designed to save time

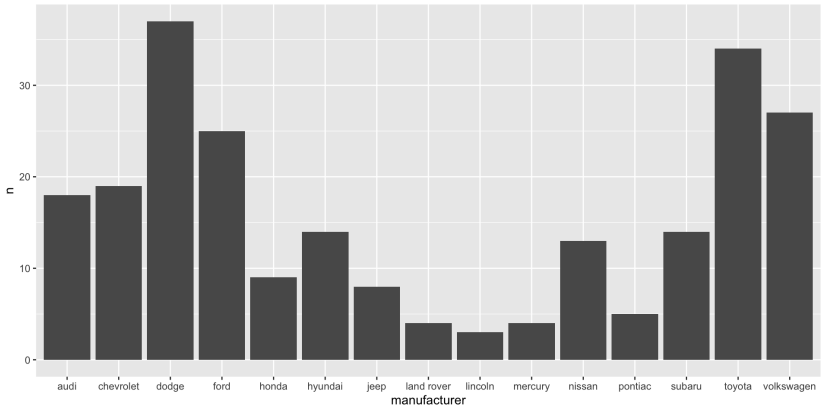
`geom_bar()` defaults to counting instances of a variable.

```
1  mpg %>%
2    count(manufacturer) %>%
3    ggplot() +
4    geom_bar(aes(manufacturer))
```



`geom_col()` uses a column to dictate the length of bars.

```
1  mpg %>%
2    count(manufacturer) %>%
3    ggplot() +
4    geom_col(aes(x = manufacturer, y = n))
```

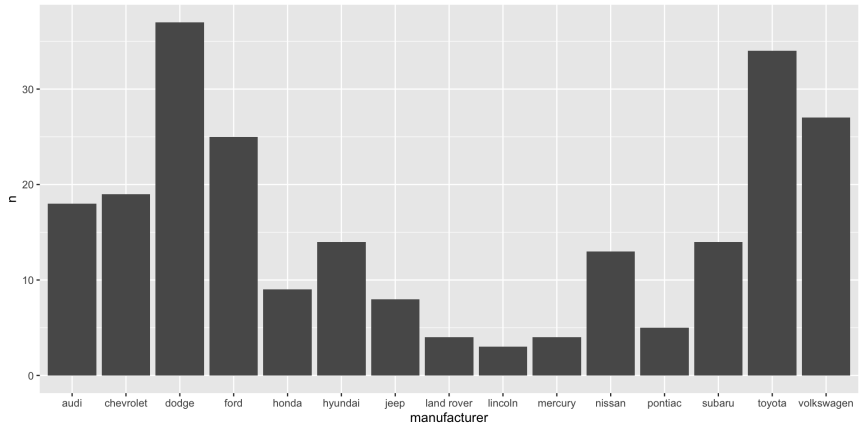


Some geoms depend on `stat` functions

The `geom_bar()` function has a `stat` argument with the default value of `"count"`.

We can force the geom to behave like `geom_col()` by changing the stat:

```
mpg %>%
  count(manufacturer) %>%
  ggplot() +
  geom_bar(aes(x = manufacturer,
               y = n),
           stat = "identity")
```



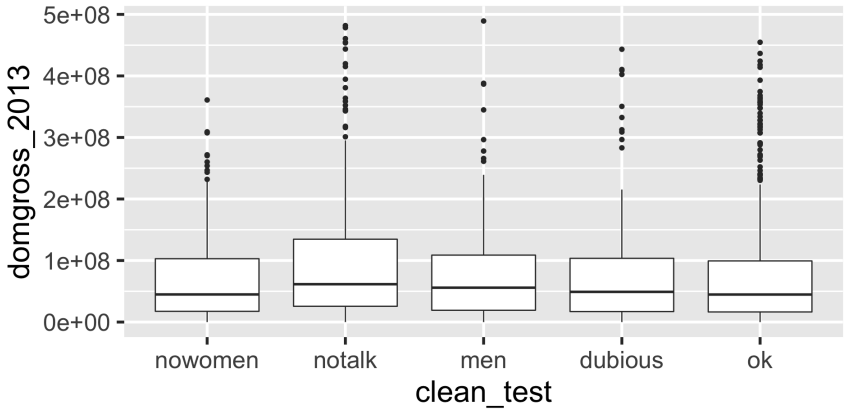
All of the goodness from the `stat` argument comes from the `stat_identity()` and `stat_count()` functions.

If you're building a complex chart it might be useful to directly call a stat_() function.

Position things to resolve overlapping (I)

Box and whisker diagrams hide a lot of detail

```
bechdel %>%
  filter(complete.cases(.),
         domgross_2013 < 0.5e9) %>%
  ggplot(aes(clean_test,
             domgross_2013)) +
  geom_boxplot() +
  theme_gray(base_size = 25)
```



Let's add the data points to this chart with `geom_point()` and look at the position argument.

Position things to resolve overlapping (II)

The position argument can also be used to create three different types of bar chart:

- “stack” creates a stacked bar chart
- “fill” creates a proportional bar chart
- “dodge” creates a grouped bar chart

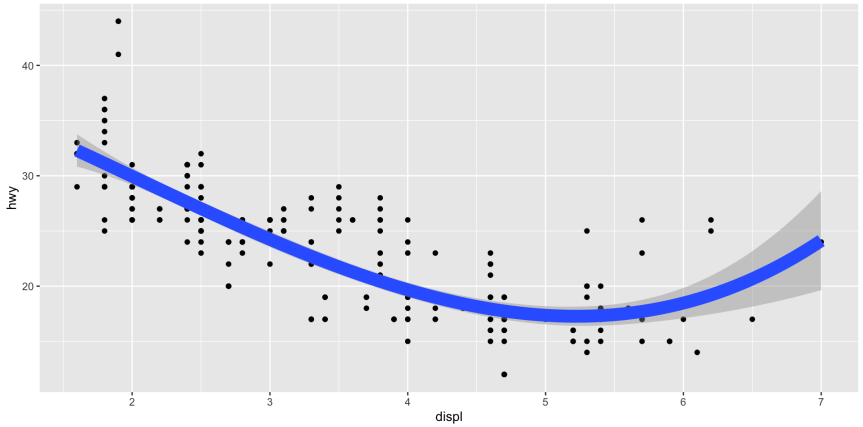
Let’s create all 3 of these for the following dataset:

```
gss_cat %>%
  count(relig, marital)
```

```
# A tibble: 78 × 3
   relig      marital           n
   <fct>      <fct>         <int>
 1 No answer  No answer         4
 2 No answer  Never married    22
 3 No answer  Separated         3
 4 No answer  Divorced         13
 5 No answer  Widowed           7
 6 No answer  Married          44
 7 Don't know Never married     6
 8 Don't know Separated         3
 9 Don't know Divorced          1
10 Don't know Married           5
# … with 68 more rows
```

Geom layers are placed on top of one another

```
ggplot(mpg,
       aes(displ, hwy)) +
  geom_point() +
  geom_smooth(method = lm,
              formula = y ~ splines::bs(x, 3),
              size = 5)
```



The `geom_smooth()` line is hiding data points.

We could either swap the order of these geoms or change the `alpha` aesthetic.

⚖️ Scales

Scales determine the appearance of an aesthetic within the chart, including:

- Axes labels and breaks
- Colours used for `colour` and `fill` aesthetics

```
msleep %>%
  ggplot() +
  aes(
    x = sleep_total,
    y = sleep_rem,
    colour = vore
  ) +
  geom_point() +
  scale_colour_manual(
    values = c("carni" = "#c03728",
               "omni" = "#fd8f24",
               "insecti" = "#f5c04a",
               "herbi" = "#919c4c",
               "NA" = "#e68c7c")
  )
```

Scales also determine the **order** in which elements are shown in a chart.

To change the order of discrete/categorical columns we need to use **factors**.

Scales and {scales}

{ggplot2} uses the {scales} package under the hood to build all of the scales that we see - including continuous and discrete scales.

The `{scales}` package also contains many utility functions that are useful for us to format our axes and other scales.

We can either load the `{scales}` package itself or call functions specifically with `scales::label_percent()`

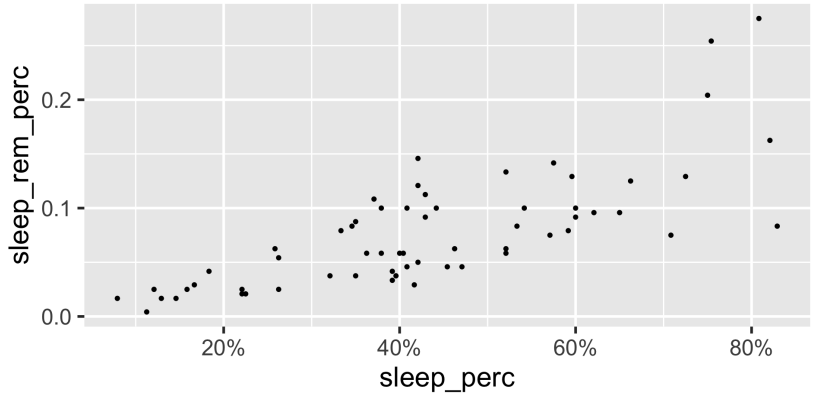
{scales} and deprecation (I)

Until recently the way we'd use `{scales}` would be as follows

```
percent(c(0.3, 0.5, 0.6))
```

```
[1] "30%" "50%" "60%"
```

```
msleep %>%
  mutate(sleep_perc = sleep_total / 24,
         sleep_rem_perc = sleep_rem / 24) %>%
  ggplot() +
  aes(x = sleep_perc,
      y = sleep_rem_perc) +
  geom_point() +
  scale_x_continuous(label = percent_format()) +
  theme_gray(base_size = 24)
```



There was a function called `percent(x)` for formatting a vector of values `x` and `percent_format()` for modifying the appearance of percentages in a {ggplot2} chart.

{scales} and deprecation (II)

These functions have now been deprecated. This means there are new alternatives to these functions.

Deprecation is a fact of life in software development. But the details of how things are deprecated are variable.

Sometimes things are deprecated with the intention of removing them in the future. Other times, the deprecated functions will continue to exist far into the future.

It seems like the intention is for these functions to continue to work into the future. But they might be removed in several years time.

> Let's use the new approach for formatting scales so that you can read modern documentation and so you're not learning deprecated functions.

{scales} and deprecation (III)

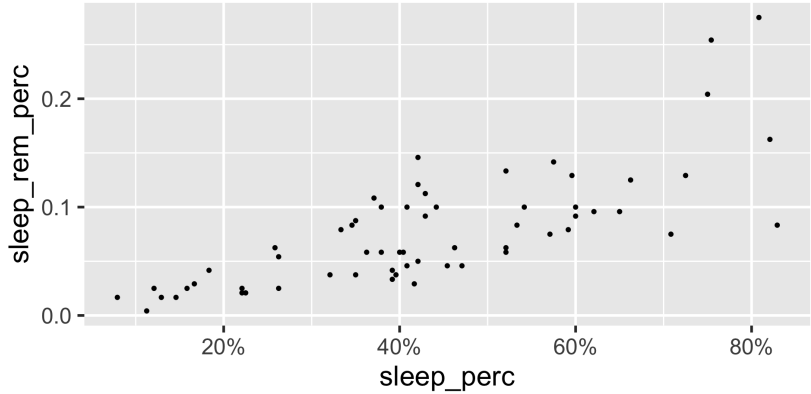
We now use `label_percent()` for both types of operation.

```
label_percent()(c(0.3, 0.5, 0.6))
```

```
[1] "30%" "50%" "60%"
```

> This is known as a function factory.
>
> Function factories are cool. But I wish you didn't have to learn this syntax.

```
msleep %>%
  mutate(sleep_perc = sleep_total / 24,
         sleep_rem_perc = sleep_rem / 24) %>%
  ggplot() +
  aes(x = sleep_perc,
      y = sleep_rem_perc) +
  geom_point() +
  scale_x_continuous(label = label_percent()) +
  theme_gray(base_size = 24)
```

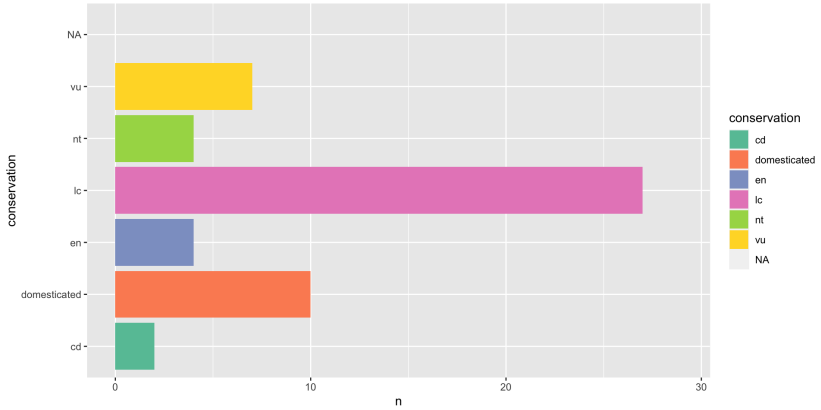


{scales} and colours (I)

There are many built-in colour palettes in {scales} - let me introduce two families of palettes.

{scales} and colours (II)

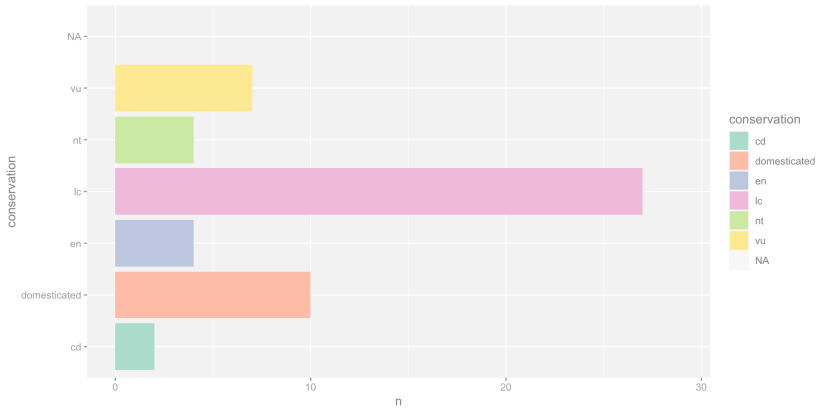
There are many built-in colour palettes in {scales} - let me introduce two families of palettes.



There are some pretty good palettes for discrete/categorical variables in this family of palettes.

{scales} and colours (III)

There are many built-in colour palettes in {scales} - let me introduce two families of palettes.

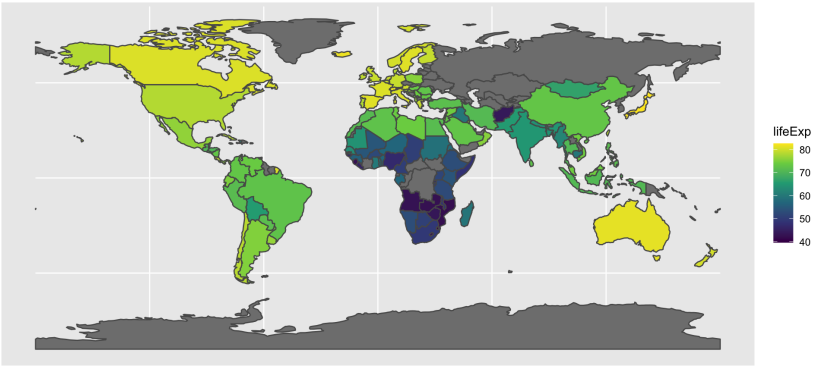


There are some pretty good palettes for discrete/categorical variables in this family of palettes.

But for continuous variables I strongly recommend using the viridis family of palettes.

These are designed to be both perceptually uniform and to work for folks with colour blindness.

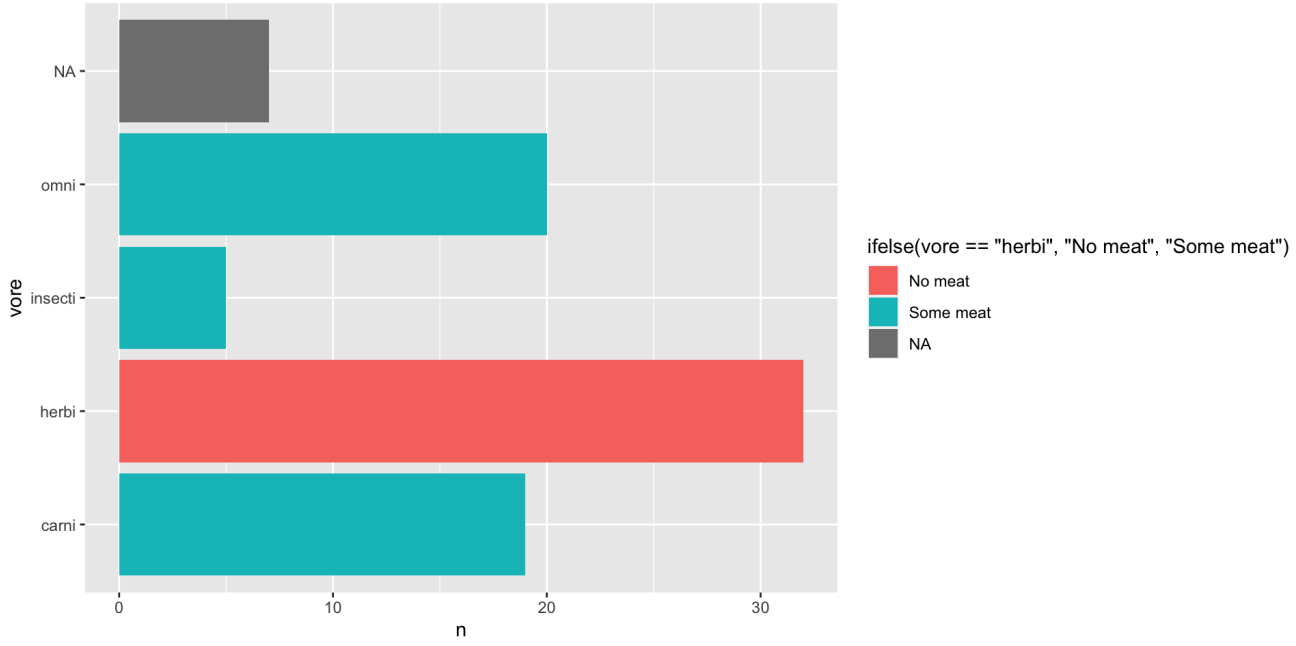
```
countries110 %>%
  st_as_sf() %>%
  left_join(filter(gapminder, year == 2007),
            by = c("name" = "country")) %>%
  ggplot() +
  geom_sf(aes(fill = lifeExp)) +
  scale_fill_viridis_c()
```



Setting custom colours (I)

One of the first frustrations people find with {ggplot2} is setting our own custom colours, eg in this chart:

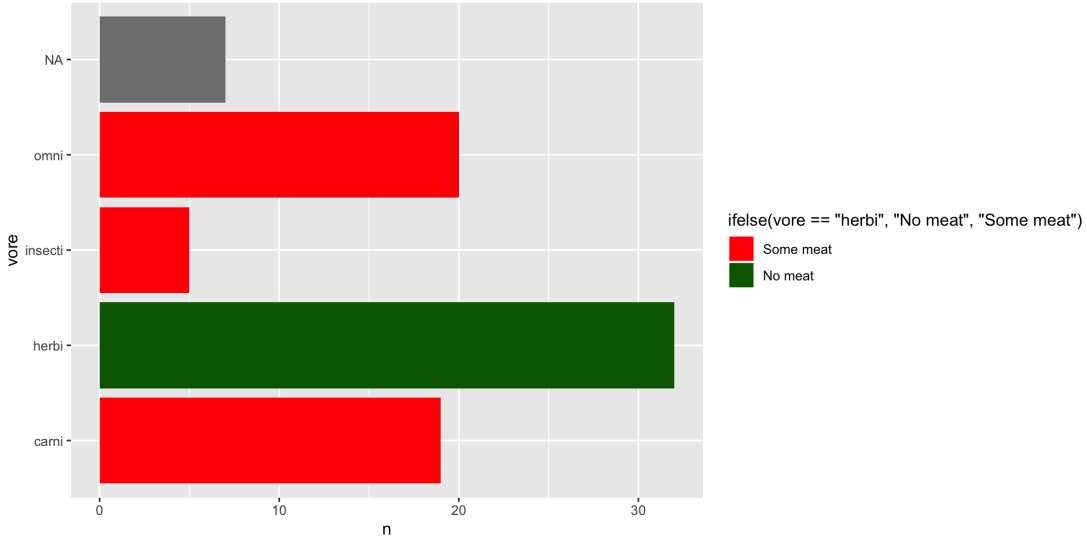
```
1  msleep %>%
2    count(vore) %>%
3    ggplot() +
4    aes(x = n,
5        y = vore,
6        fill = ifelse(vore == "herbi", "No meat", "Some meat")) +
7    geom_col()
```



Setting custom colours (II)

We need to use `scale_fill_manual()`

```
msleep %>%
  count(vore) %>%
  ggplot() +
  aes(x = n,
      y = vore,
      fill = ifelse(vore == "herbi", "No meat", "Some meat")) +
  geom_col() +
  scale_fill_manual(values = c("Some meat" = "red",
                               "No meat" = "darkgreen"))
```



We'll come back to this chart in the section on guides().

{scales} and factors (I)

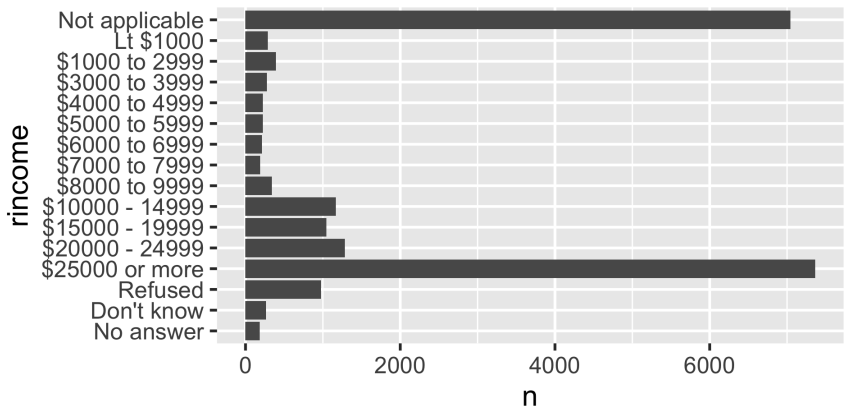
Factors are R's categorical data type. They allow us to create a variable with fixed values (`levels`) and to set the *order* of those levels.

Let's look at a pre-existing dataset with factors:

```
gss_cat %>%
  head() %>%
  pull(rincome)
```

```
[1] $8000 to 9999  $8000 to 9999  Not applicable Not applicable Not applicable
[6] $20000 - 24999
16 Levels: No answer Don't know Refused $25000 or more ... Not applicable
```

```
gss_cat %>%
  count(rincome) %>%
  ggplot() +
  aes(x = n,
      y = rincome) +
  geom_col() +
  theme_gray(base_size = 24)
```



{scales} and factors (II)

The base R tools for creating and manipulating factors are messy and frustrating to use.

We're going to use the {forcats} package which is loaded when we run `library(tidyverse)`.

Almost all of the functions begin with `fct_*()` to let you know we're dealing with factors.

factors and msleep

Let's think of the different ways we could order this dataset:

```
1 msleep %>%
2   count(vore)
```

```
# A tibble: 5 × 2
  vore        n
  <chr>   <int>
1 carni      19
2 herbi      32
3 insecti     5
4 omni       20
5 <NA>        7
```

Count order

In this ordering we will arrange the `vore` column according to values in the `n` column.

This is usually what we want in count bar charts.

Canonical order

In this ordering we'll arrange the `vore` column from the diet with the most meat to the least meat.

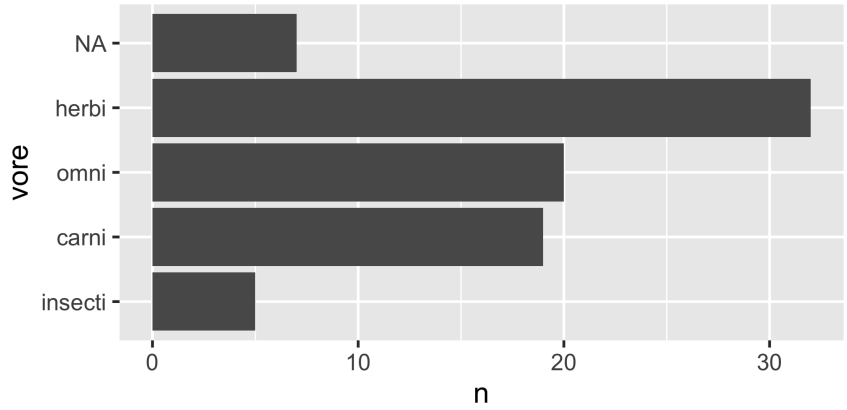
This is usually what we want in visualising survey datasets,

- eg Strong disagree, Disagree, Neither agree or disagree, Agree, Strong Agree

msleep factor: count order (I)

We use `fct_reorder()` to order a factor by another column.

```
msleep %>%
  count(vore) %>%
  mutate(vore = fct_reorder(vore, n)) %>%
  ggplot() +
  aes(x = n,
      y = vore) +
  geom_col() +
  theme_gray(base_size = 24)
```

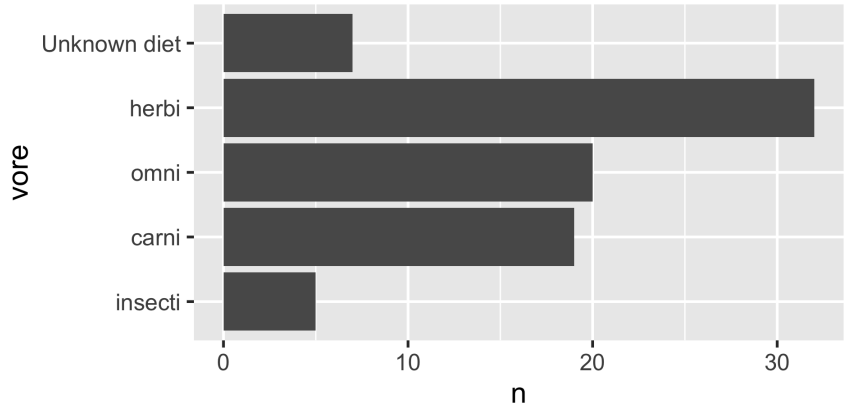


But what about the NA values? What should we do?

msleep factor: count order (II)

We can replace NA values nicely with `fct_explicit_na()`

```
msleep %>%
  count(vore) %>%
  mutate(vore = fct_reorder(vore, n),
         vore = fct_explicit_na(vore, "Unknown diet"))
  ggplot() +
  aes(x = n,
      y = vore) +
  geom_col() +
  theme_gray(base_size = 24)
```

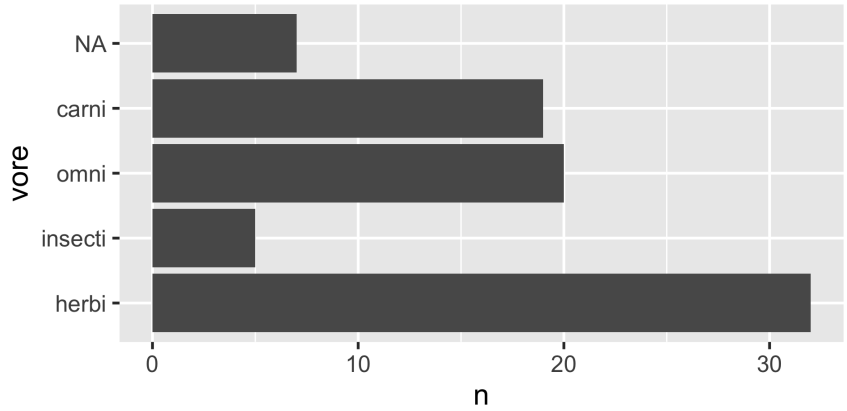


> Let's come back to moving the position of the NA level.

msleep factor: canonical order (I)

To set our own canonical order we use `fct_relevel()` and provide a vector with our preferred order.

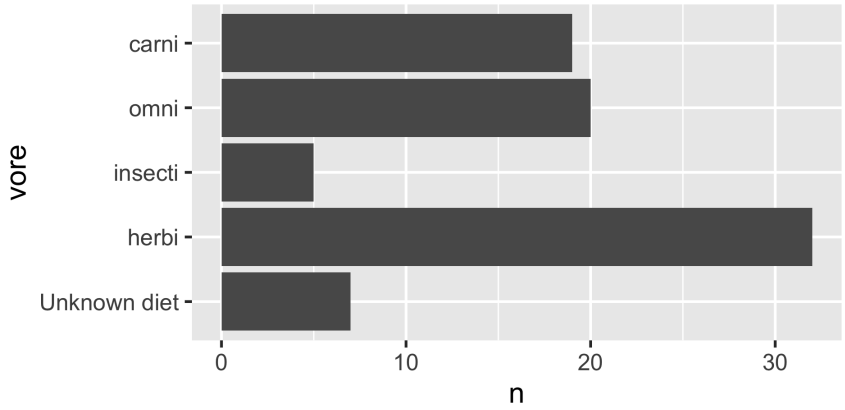
```
order_vore <- c("carni", "omni", "insecti", "herbi")

msleep %>%
  count(vore) %>%
  mutate(vore = fct_relevel(vore, order_vore),
         vore = fct_rev(vore)) %>%
  ggplot() +
  aes(x = n,
      y = vore) +
  geom_col() +
  theme_gray(base_size = 24)
```



msleep factor: canonical order (II)

We can also use `fct_relevel()` to modify the position of a specific

```
msleep %>%
  count(vore) %>%
  mutate(vore = fct_relevel(vore, order_vore),
         vore = fct_rev(vore),
         vore = fct_explicit_na(vore, "Unknown diet"),
         vore = fct_relevel(vore, "Unknown diet", afte
  ggplot() +
  aes(x = n,
      y = vore) +
  geom_col() +
  theme_gray(base_size = 24)
```



📝 Task: Global Burden of Disease factors

SLIDE 1 OF 3

> These are the same steps you've repeated before

1. Add a sub-folder to your project called `data`
2. Inside of the `data` folder add a script called `obtain-data.R`
3. Add this code

```
download.file("https://raw.githubusercontent.com/charliejhadley/eng7218_data-science-for-healthcare-applications_bcu-
              destfile = "data/global-burden-of-disease-data.csv")
```

5. Run the code

📝 Task: Global Burden of Disease factors

SLIDE 2 OF 3

1. Add a new heading for the GBD Dataset to your .Rmd
2. Filter the dataset as follows:

- Most recent year
- `location_name` starts with “World Bank”
- `metric_name` is “Number”
- `cause_name` is “Injuries”

3. Select only these columns

- `location_name`, `cause_name`, `val`

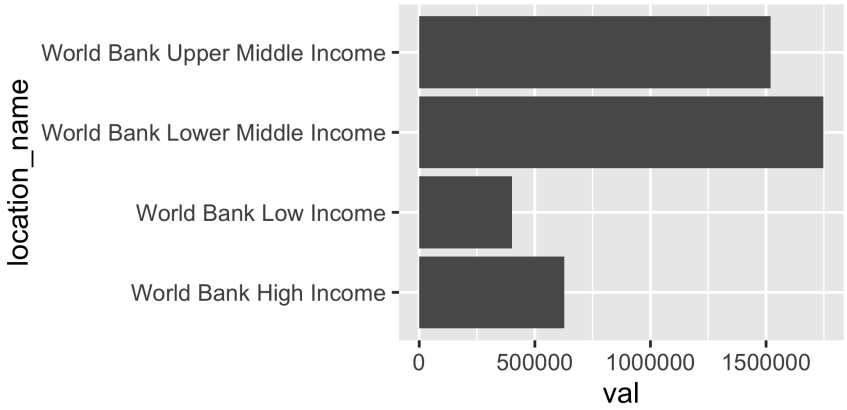
📝 Task: Global Burden of Disease factors

SLIDE 2 OF 3

Create two versions of this chart:

- Bars are ordered by their size
- Bars are ordered from “World Bank High Income” to “World Bank Low Income”

```
gdb_injuries %>%
  ggplot() +
  aes(x = val,
      y = location_name) +
  geom_col() +
  theme_gray(base_size = 24)
```

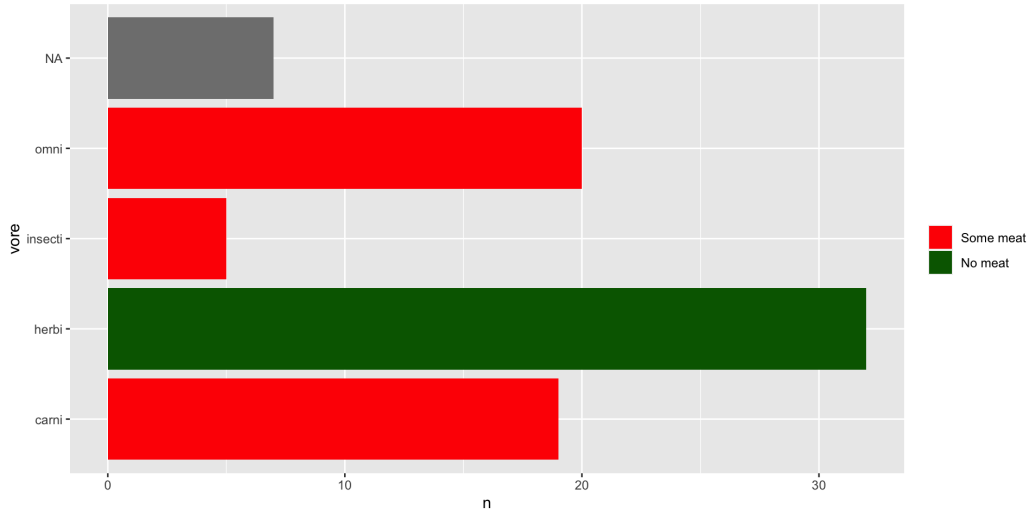


🏷️ Guides

We (kind of) use “guides” and “legends” interchangeably in {ggplot2}.

```
ggplot() +
  geom_line(show.legend = FALSE) +
  guides(alpha = guide_legend()) +
  theme(legend.position = "bottom")
```

Guides can **only** be created through a corresponding aesthetic and scale.

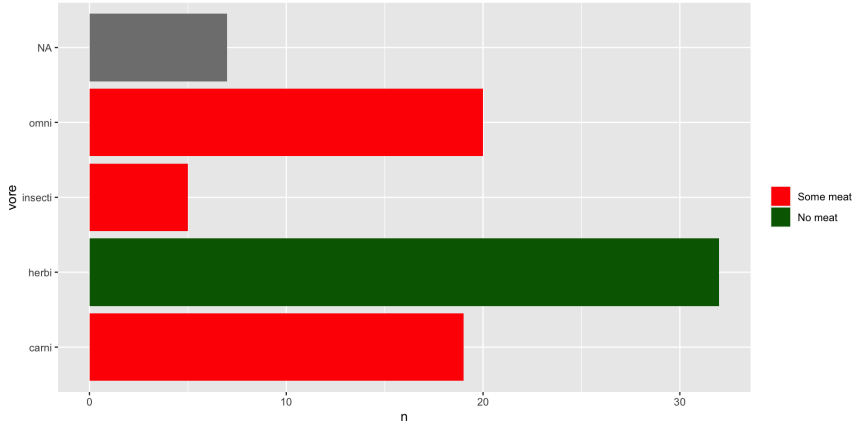


“Manual legends” (I)

Sometimes we want to add additional legend items - usually for NA values, and particularly for maps.

Let’s continue with this chart from before:

```
msleep %>%
  count(vore) %>%
  ggplot() +
  aes(x = n,
      y = vore,
      fill = ifelse(vore == "herbi",
                    "No meat",
                    "Some meat")) +
  geom_col() +
  scale_fill_manual(values = c("Some meat" = "red",
                               "No meat" = "darkgreen"
                    name = "")
```

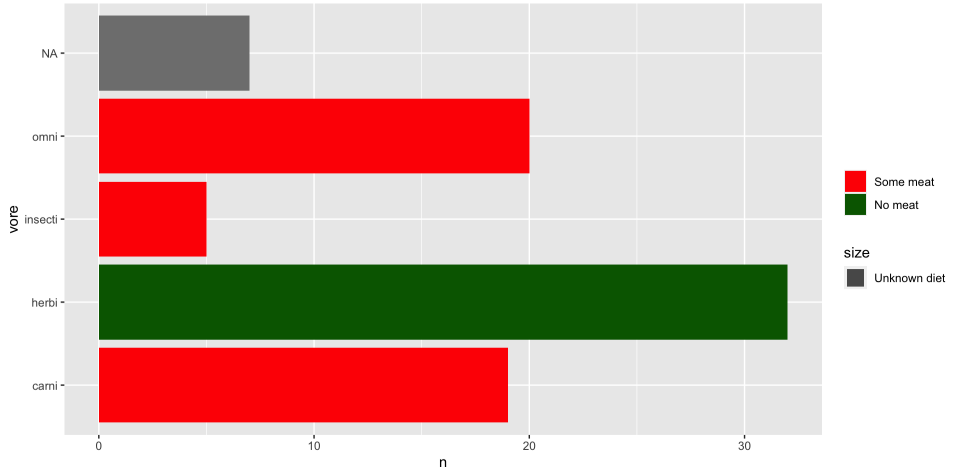


“Manual legends” (II)

We need to choose an aesthetic that works for `geom_col()` but we’re not using elsewhere in the chart.

This will change depending on your chart. In this instance we can use `size`

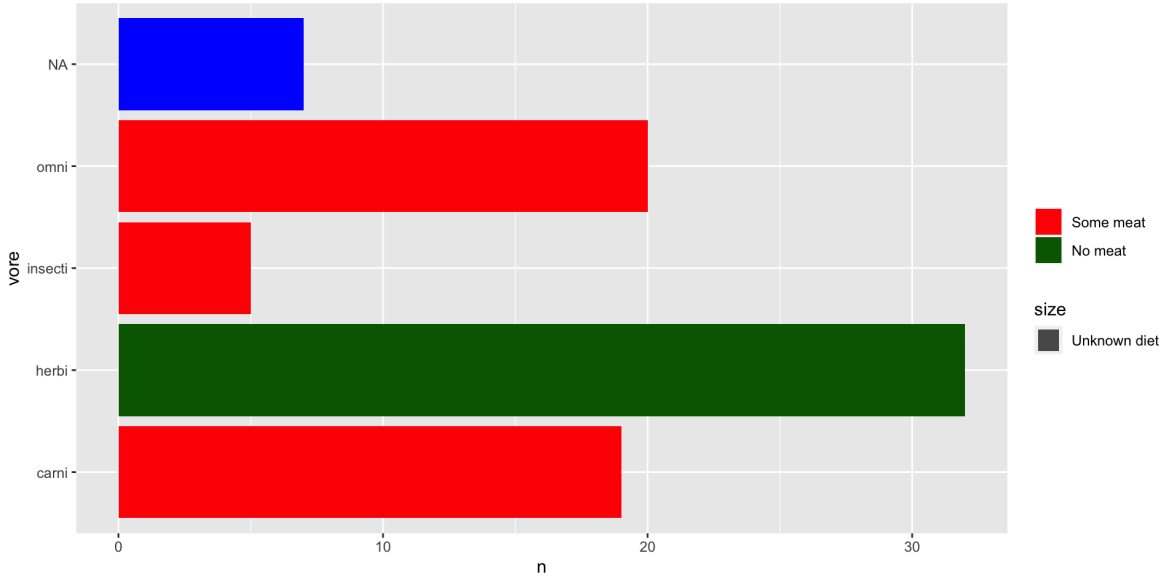
```
msleep %>%
  count(vore) %>%
  ggplot() +
  aes(x = n,
      y = vore,
      fill = ifelse(vore == "herbi", "No meat", "Some meat")) +
  geom_col(aes(size = "Unknown diet")) +
  scale_fill_manual(values = c("Some meat" = "red",
                               "No meat" = "darkgreen"),
                    name = "")
```



“Manual legends” (III)

We now set the `na.value` colour for the original `scale_fill_manual()` scale

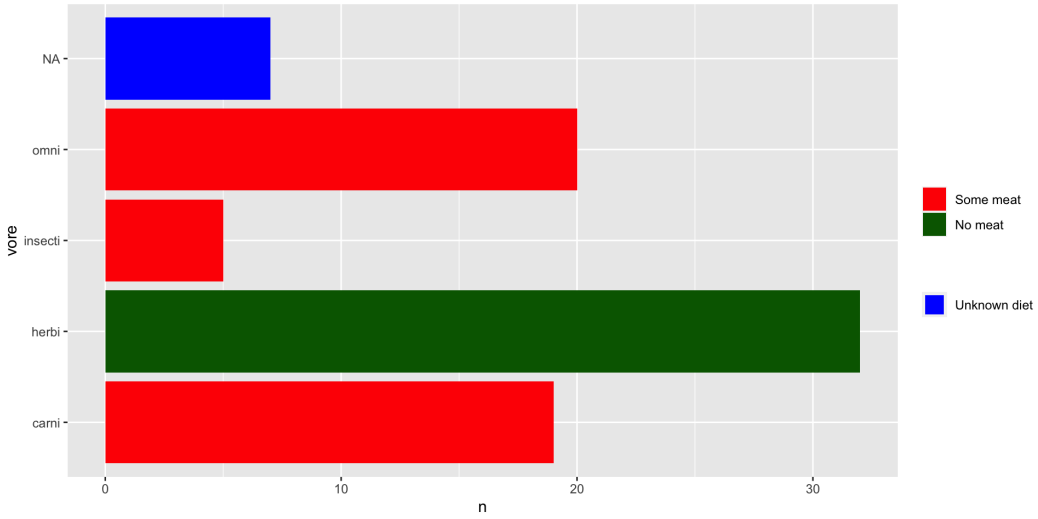
```
msleep %>%
  count(vore) %>%
  ggplot() +
  aes(x = n,
      y = vore,
      fill = ifelse(vore == "herbi", "No meat", "Some meat")) +
  geom_col(aes(size = "Unknown diet")) +
  scale_fill_manual(values = c("Some meat" = "red",
                               "No meat" = "darkgreen"),
                    name = "",
                    na.value = "blue")
```



“Manual legends” (IV)

Next we use the `guides()` function to override the values for the `size` legend

```
msleep %>%
  count(vore) %>%
  ggplot() +
  aes(x = n,
      y = vore,
      fill = ifelse(vore == "herbi", "No meat", "Some meat")) +
  geom_col(aes(size = "Unknown diet")) +
  scale_fill_manual(values = c("Some meat" = "red",
                               "No meat" = "darkgreen"),
                    name = "",
                    na.value = "blue") +
  guides(size = guide_legend(title = "",
                             override.aes = list(fill = "blue")))
```



More about guides()

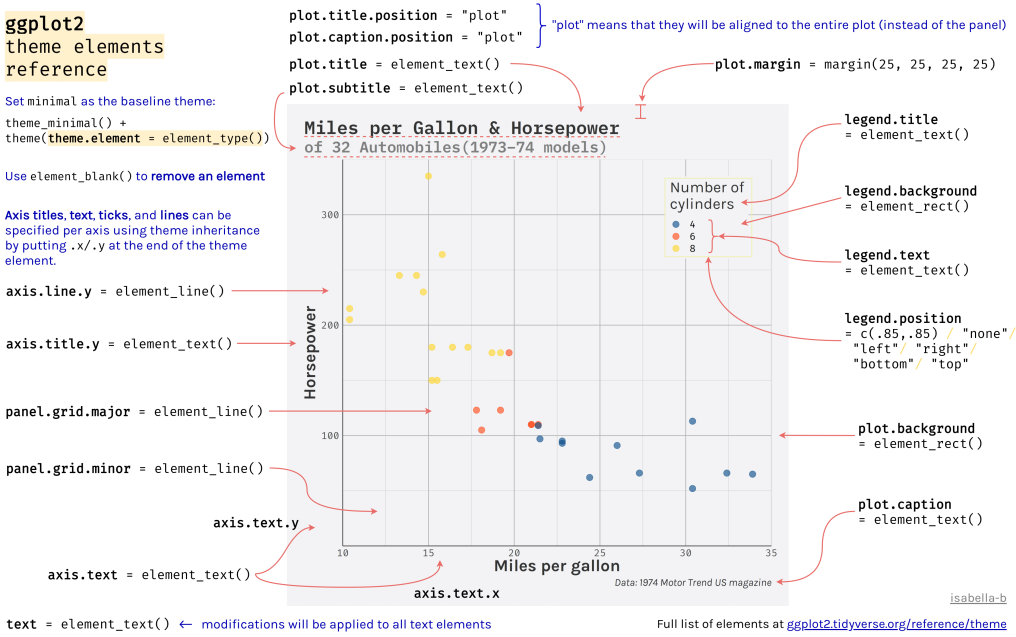
If we want to modify the size of legend items we have two choices:

- `guides(fill = guide_colourbar(barwidth = 0.5, barheight = 10))`
- … or to set the sizes in the `theme()`.

Theme

There are over 92 arguments to the theme() function for controlling chart appearance.

Remembering them all is challenging - I usually google them! Or use guides like this one:



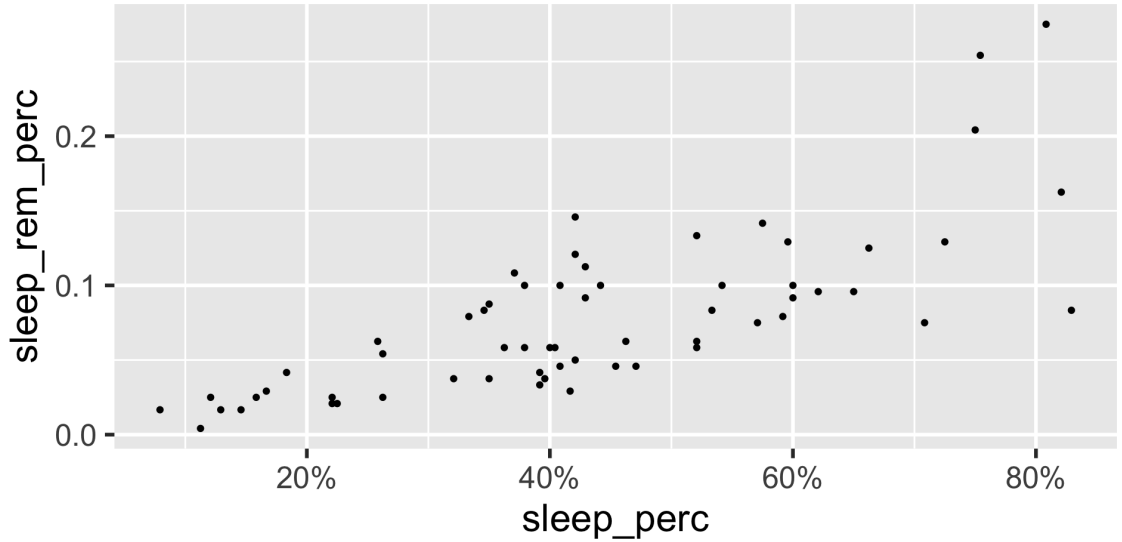
Source: https://bookdown.org/alapo/learnr/data-visualisation.html

What themes are there? (I)

{ggplot2} has several built-in themes. They have several arguments for quickly customising them.

I've been using the default `theme_gray()` to change text size in charts.

```
msleep %>%
  mutate(sleep_perc = sleep_total / 24,
         sleep_rem_perc = sleep_rem / 24) %>%
  ggplot() +
  aes(x = sleep_perc,
      y = sleep_rem_perc) +
  geom_point() +
  scale_x_continuous(label = label_percent()) +
  theme_gray(base_size = 24)
```



What themes are there? (II)

The {ggthemes} package contains lots of really useful - and beautiful - themes.

It's recommended that you choose a theme close to what to want and then customise it.

```
theme_fivethirtyeight() +
  theme(panel.grid.major = element_line(colour = "red"))
```

element_*() functions in legends

Most of the legend arguments expect one of these functions:

- element_line()
- element_text()
- element_rect()

Or `element_black()` if you want to remove a theme element.

References

1. Eberhard, K. The effects of visualization on judgment and decision-making: A systematic literature review. *Management Review Quarterly* (2021) doi:10.1007/s11301-021-00235-8.
2. Anscombe, F. J. Graphs in Statistical Analysis. *The American Statistician* **27**, 17–21 (1973).
3. Matejka, J. & Fitzmaurice, G. Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing. in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* 1290–1294 (Association for Computing Machinery, 2017). doi:10.1145/3025453.3025912.
4. Cairo, A. Download the Datasaurus: Never trust summary statistics alone; always visualize your data. (2016).
5. Snow, J. *On the mode of communication of cholera*. (John Churchill, 1855).
6. Nightingale, F. *Notes on Matters Affecting the Health, Efficiency and Hospital Administration of the British Army*. (Harrison & Sons, 1858).
7. Hans Rosling. The best stats you've ever seen [Video]. *The best stats you've ever seen* (2006).
8. Hawkins, E. Spiralling global temperatures | Climate Lab Book. (2016).
9. Pat Schloss. Recreating animated climate temperature spirals in R with Ggplot2 and gganimate (CC219). (2022).
10. Kosara, R. & Skau, D. Judgment Error in Pie Chart Variations. *EuroVis 2016 - Short Papers* 5 pages (2016) doi:10.2312/EUROVISSHORT.20161167.
11. Cleveland, W. S. & McGill, R. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association* **79**, 531–554 (1984).
12. Heer, J. & Bostock, M. Crowdsourcing graphical perception: Using mechanical turk to assess visualization design. in *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10* 203 (ACM Press, 2010). doi:10.1145/1753326.1753357.
13. Beecham, R., Dykes, J., Hama, L. & Lomax, N. On the Use of "Glyphmaps" for Analysing the Scale and Temporal Spread of COVID-19 Reported Cases. *ISPRS International Journal of Geo-Information* **10**, 213 (2021).
14. Kosara, R. More Than Meets the Eye: A Closer Look at Encodings in Visualization. *IEEE Computer Graphics and Applications* **42**, 110–114 (2022).
15. Iker Rivas-González [@irg_bio]. I am also joining the hexbin fever! 🐝 For this week's #TidyTuesday, I plotted the number of bee colonies in the US by year and season. It seems like cold and warm states have different patterns of seasonal changes. Code: https://github.com/rivasiker/TidyTuesday/blob/main/2022/2022-01-11/analysis_2022-01-11.Rmd #RStats #DataViz #ggplot2 https://t.co/OYGyg2az7M. *Twitter* (2022).