Week 3: Data science with the tidyverse

Charlotte Hadley

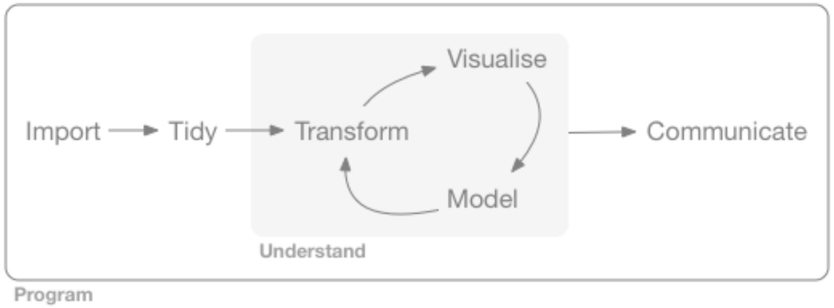
Topics for today

1. Understanding `data.frame` and `tibble`
2. Using `{readr}` for reading data
3. Using `{dplyr}` for cleaning and wrangling data
4. Using `{ggplot2}` for some basic exploratory data analysis

What is the tidyverse?

The tidyverse is two things

- A collection of packages that are designed to work together really well.
- A collection of packages for all stages of the data science workflow.



Source: R for Data Science[1]

- An opinionated framework for how to work with data.

Some people[2] would describe the tidyverse as being an alternative to using base R.

It’s not.

The tidyverse is a useful approach to working with data that has a large ecosystem of tools. We’ll use it to move quickly.

> After we’ve been using the tidyverse for a while I’ll come back to this subject and provide further context.

📝 Task: Create a week-3 project

SLIDE 1 OF 1

1. Create a new RStudio project called something like `week-3`.

Installing and working with the tidyverse

To install the tidyverse collection of packages you need to run this code in the console.

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

The tidyverse packages are split into two groups:

- Core tidyverse packages which are loaded with this code

```r
library(tidyverse)
```

- “Specialised” packages that need to be explicitly loaded, like `{readxl}` for importing data from Excel files.

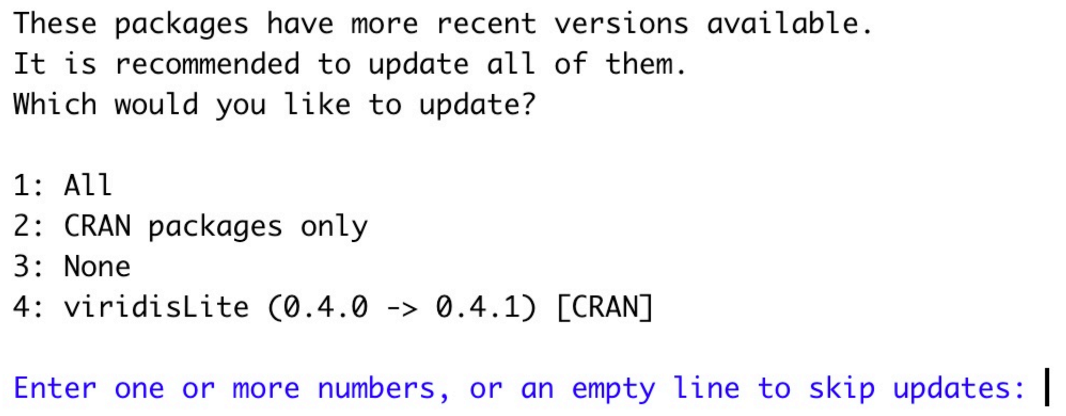
```r
library(tidyverse)
library(readxl)
```

Updating the tidyverse (I)

In terms of real-world usage, keeping the tidyverse up to date is identical to keeping any R package up to date.

Thee

- When installing a *new* package the console might prompt you to update to new versions of packages



```
These packages have more recent versions available.
It is recommended to update all of them.
Which would you like to update?

1: All
2: CRAN packages only
3: None
4: viridisLite (0.4.0 -> 0.4.1) [CRAN]

Enter one or more numbers, or an empty line to skip updates:
```

Updating the tidyverse (II)

In terms of real-world usage, keeping the tidyverse up to date is identical to keeping any R package up to date.

There are 3 different ways you might discover you should update a package.

- When installing a *new* package the console might prompt you to update to new versions of packages.
- When installing a *new* package the install fails due to an old package.
- You hear about an exciting new update to a package.

> There is a `tidyverse::tidyverse_update()` function but in practice I think it’s very rarely used.

Datasets we’ll be using today (I)

We’re going to be using at least 3 different datasets today:

- The Global Burden of Disease study from the Global Health Data Exchange[3].

The Global Burden of Disease study is an extremely useful and rich dataset for understanding global (and comparative) health challenges.

There’s an excellent interactive tool for downloading data from the survey - but you do need to register for a free account to use it.

Datasets we’ll be using today (II)

We’re going to be using at least 3 different datasets today:

- The Global Burden of Disease study from the Global Health Data Exchange[3].

- The `msleep` dataset from within the `{ggplot2}` package

Lots (and lots) of R packages have datasets built into them, usually to demonstrate how to use functions inside the package.

The `msleep` dataset has data about mammalian sleep cycles from Savage and West[4]

```r
glimpse(msleep)
```

```
Rows: 83
Columns: 11
$ name         <chr> "Cheetah", "Owl monkey", "Mountain
beaver", "Greater shor…
$ genus        <chr> "Acinonyx", "Aotus", "Aplodontia",
"Blarina", "Bos", "Bra…
$ vore         <chr> "carni", "omni", "herbi", "omni",
"herbi", "herbi", "carn…
$ order        <chr> "Carnivora", "Primates",
"Rodentia", "Soricomorpha", "Art…
$ conservation <chr> "lc", NA, "nt", "lc",
"domesticated", NA, "vu", NA, "dome…
$ sleep_total  <dbl> 12.1, 17.0, 14.4, 14.9, 4.0, 14.4,
8.7, 7.0, 10.1, 3.0, 5…
$ sleep_rem    <dbl> NA, 1.8, 2.4, 2.3, 0.7, 2.2, 1.4,
NA, 2.9, NA, 0.6, 0.8, …
$ sleep_cycle  <dbl> NA, NA, NA, 0.1333333, 0.6666667,
0.7666667, 0.3833333, N…
$ awake        <dbl> 11.9, 7.0, 9.6, 9.1, 20.0, 9.6,
15.3, 17.0, 13.9, 21.0, 1…
$ brainwt      <dbl> NA, 0.01550, NA, 0.00029, 0.42300,
NA, NA, NA, 0.07000, 0…
$ bodywt       <dbl> 50.000, 0.480, 1.350, 0.019,
600.000, 3.850, 20.490, 0.04
```

Datasets we’ll be using today (III)

We’re going to be using at least 3 different datasets today:

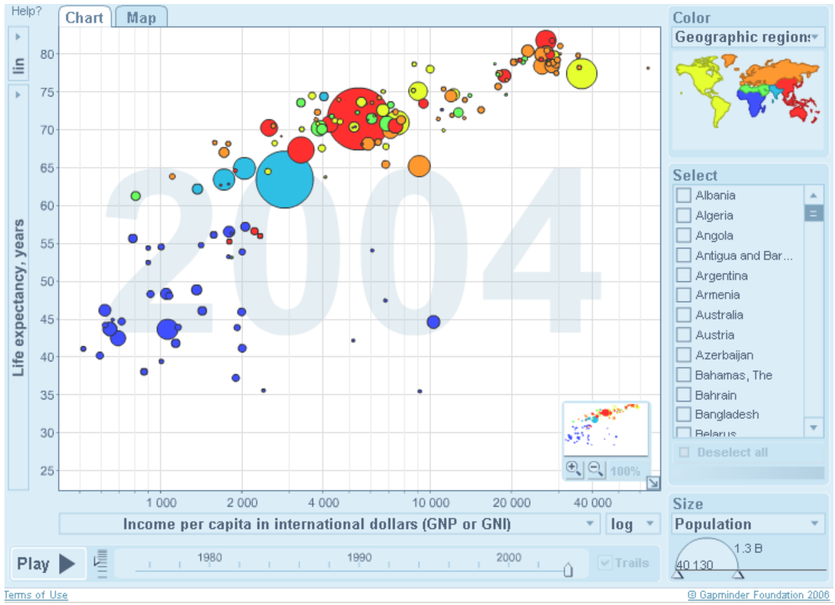
- The Global Burden of Disease study from the Global Health Data Exchange[3].
- The `msleep` dataset from within the `{ggplot2}` package

- The `gapminder` dataset from the `{gapminder}` package

In 2006 Hans Rosling[5] gave an incredible TED talk where he introduced animated bubble charts as a tool to tell stories about global development.

Hans Rosling also founded the Gapminder Foundation to promote sustainable global development.

The `{gapminder}` package contains a subset of their data.



msleep

📝 Task: Get the msleep dataset

SLIDE 1 OF 1

1. Add a heading for the msleep dataset.
2. Load the {tidyverse} package in the setup code chunk
3. Add a new code chunk and print the object msleep to the console

```
msleep
```

Understanding our object

The datasets embedded into `{tidyverse}` packages and those generated by reading in data files with {readr}, {readxl} and {haven} are objects known as “tibbles”.

It’s important to understand how this relates to and is different from `data.frame`.

data.frames (I)

The `data.frame` is R’s general purpose rectangular data store, it’s therefore the data structure required to build `{ggplot2}` charts.

The `{datasets}` package has a number of built-in `data.frames`, for instance:

```
head(quakes)
```

```
     lat   long depth mag stations
1 -20.42 181.62   562 4.8       41
2 -20.62 181.03   650 4.2       15
3 -26.00 184.10    42 5.4       43
4 -17.97 181.66   626 4.1       19
5 -20.42 181.96   649 4.0       11
6 -19.68 184.31   195 4.0       12
```

The `class()` function is the way to determine what type of *thing/object* you’re working with:

```
class(quakes)
```

```
[1] "data.frame"
```

data.frames (II)

Because they’re rectangular, `data.frames` have rows and columns which we can extract separately.

We can print the dimensions of a `data.frame` with `dim()`

```
dim(quakes)
```

```
[1] 1000    5
```

In Base R there are two ways to extract columns:

- The **$** operator allows us to extract columns via their name, with autocompletion:

```
quakes$mag
```

- The more flexible **[** operator allows us to extract both rows and columns, including by their index

```
quakes[, "mag"]
```

```
quakes[, 1]
```

data.frame is dead, long live tibble… (I)

The `tidyverse` introduces an augmented `data.frame`, called a `tibble`.

Let's demonstrate the differences after loading the `tidyverse`.

```r
library("tidyverse")
```

The first thing we notice about a `tibble` is that it prints differently to the console:

```r
quakes
```

```
      lat   long depth mag stations
1  -20.42 181.62   562 4.8       41
2  -20.62 181.03   650 4.2       15
3  -26.00 184.10    42 5.4       43
4  -17.97 181.66   626 4.1       19
5  -20.42 181.96   649 4.0       11
6  -19.68 184.31   195 4.0       12
7  -11.70 166.10    82 4.8       43
8  -28.11 181.93   194 4.4       15
9  -28.74 181.74   211 4.7       35
10 -17.47 179.59   622 4.3       19
11 -21.44 180.69   583 4.4       13
12 -12.26 167.00   249 4.6       16
13 -18.54 182.11   554 4.4       19
14 -21.00 181.66   600 4.4       10
15 -20.70 169.92   139 6.1       94
16 -15.94 184.95   306 4.3       11
17 -13.64 165.96    50 6.0       83
18 -17.83 181.50   590 4.5       21
19 -23.50 179.78   570 4.4       13
20 -22.63 180.31   598 4.4       18
21 -20.84 181.16   576 4.5       17
22 -10.98 166.32   211 4.2       12
23 -23.30 180.16   512 4.4       18
```

```r
starwars
```

```
# A tibble: 87 × 14
   name         height  mass hair_…¹ skin_…² eye_c…³
birth…⁴ sex    gender homew…⁵
   <chr>         <int> <dbl> <chr>   <chr>   <chr>
<dbl> <chr> <chr>  <chr>
 1 Luke Skywa…     172    77 blond   fair    blue
19    male  mascu… Tatooi…
 2 C-3PO           167    75 <NA>    gold    yellow
112   none  mascu… Tatooi…
 3 R2-D2            96    32 <NA>    white,… red
33    none  mascu… Naboo
 4 Darth Vader     202   136 none    white   yellow
41.9 male  mascu… Tatooi…
 5 Leia Organa     150    49 brown   light   brown
19    fema… femin… Aldera…
 6 Owen Lars       178   120 brown,… light   blue
52    male  mascu… Tatooi…
 7 Beru White…     165    75 brown   light   blue
47    fema… femin… Tatooi…
 8 R5-D4            97    32 <NA>    white,… red
NA    none  mascu… Tatooi…
 9 Biggs Dark…     183    84 black   light   brown
24    male  mascu… Tatooi…
10 Obi-Wan Ke      182    77 auburn  fair    blue-g
```

data.frame is dead, long live tibble… (II)

In this course we're not going to use the `$` or `[` operators for extracting columns from a `data.frame` or `tibble`.

The `{dplyr}` provides the extremely flexible `select()` function:

```r
select(quakes, mag)
```

```
     mag
1    4.8
2    4.2
3    5.4
4    4.1
5    4.0
6    4.0
7    4.8
8    4.4
9    4.7
10   4.3
11   4.4
12   4.6
13   4.4
14   4.4
15   6.1
16   4.3
17   6.0
18   4.5
19   4.4
20   4.4
21   4.5
22   4.2
23   4.4
```

```r
select(starwars, name)
```

```
# A tibble: 87 × 1
   name
   <chr>
 1 Luke Skywalker
 2 C-3PO
 3 R2-D2
 4 Darth Vader
 5 Leia Organa
 6 Owen Lars
 7 Beru Whitesun lars
 8 R5-D4
 9 Biggs Darklighter
10 Obi-Wan Kenobi
# … with 77 more rows
```

data.frame is dead, long live tibble… (III)

The `select()` function returns a `data.frame` or `tibble`, but sometimes *we need a vector*.

A vector is a one-dimensional atomic object… we usually come across them via the `c()` function:

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

```
[1] "1" "2" "3"
```

If we want to extract a column from a `tibble` as a vector, we need to use `pull()`:

```
pull(starwars, name)
```

```
 [1] "Luke Skywalker"        "C-3PO"                 "R2-D2"
 [4] "Darth Vader"           "Leia Organa"           "Owen Lars"
 [7] "Beru Whitesun lars"    "R5-D4"                 "Biggs Darklighter"
[10] "Obi-Wan Kenobi"        "Anakin Skywalker"      "Wilhuff Tarkin"
[13] "Chewbacca"             "Han Solo"              "Greedo"
[16] "Jabba Desilijic Tiure" "Wedge Antilles"        "Jek Tono Porkins"
[19] "Yoda"                  "Palpatine"             "Boba Fett"
[22] "IG-88"                 "Bossk"                 "Lando Calrissian"
[25] "Lobot"                 "Ackbar"                "Mon Mothma"
[28] "Arvel Crynyd"          "Wicket Systri Warrick" "Nien Nunb"
[31] "Qui-Gon Jinn"          "Nute Gunray"           "Finis Valorum"
[34] "Jar Jar Binks"         "Roos Tarpals"          "Rugor Nass"
[37] "Ric Olié"              "Watto"                 "Sebulba"
[40] "Quarsh Panaka"         "Shmi Skywalker"        "Darth Maul"
[43] "Bib Fortuna"           "Ayla Secura"           "Dud Bolt"
[46] "Gasgano"               "Ben Quadinaros"        "Mace Windu"
[49] "Ki-Adi-Mundi"          "Kit Fisto"             "Eeth Koth"
[52] "Adi Gallia"            "Saesee Tiin"           "Yarael Poof"
[55] "Plo Koon"              "Mas Amedda"            "Gregar Typho"
[58] "Cordé"                 "Cliegg Lars"           "Poggle the Lesser"
```

Exploring our dataset

Let’s get to grips with our dataset.

How many animals do we have for each diet type?

We can calculate this using the `count()` function

```
count(msleep, 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 documentation

If we consult the documentation for an explanation of `count()`, we're introduced to this beast:

> %>%

```
starwars %>%
 count(species, homeworld, sort = TRUE)
```

The code we will use to split the `start.location` column also uses `%>%`

```
bechdel %>%
  count(binary)
```

Let's address what `%>%` does…

Little Bunny Foo Foo

To introduce pipes, we’re going to borrow an example from Hadley Wickham:

How can we convert this poem into code?

> Little bunny Foo Foo
>
> Went hopping through the forest
>
> Scooping up the field mice
>
> And bopping them on the head

Coding up little bunny foo foo

> Little bunny Foo Foo
> Went hopping through the forest
> Scooping up the field mice
> And bopping them on the head

Let’s create an instance of a bunny called `foo_foo`

```
foo_foo <- little_bunny()
```

Now let’s write the poem out as code:

```
bop_on(
  scoop_up(
    hop_through(foo_foo, forest),
    field_mouse
  ),
  head
)
```

Understanding our code

In order to understand what our code does, we need to parse it:

- Find the deepest expression (the first thing that happens)
- Work backwards (or up) the code

```
bop_on(
  scoop_up(
   hop_through(foo_foo, forest),
    field_mouse
  ),
  head
)
```

This is exactly counter to the order of operations in the original poem.

Piping little bunny foo foo

Let’s instantiate a bunny called `foo_foo`

```
foo_foo <- little_bunny()
```

Now write the same code as before but using pipes:

```
foo_foo %>%
  hop_through(forest) %>%
  scoop_up(field_mouse) %>%
  bop_on(head)
```

The order we read operations is exactly the same as the order in which the operations happen!

Comparing the two

Independent of pipes, we create ourselves a little bunny:

```r
foo_foo <- little_bunny()
```

Now comparing the two code samples, the one with pipes is easier to parse by eye.

```r
bop_on(
  scoop_up(
    hop_through(foo_foo, forest),
    field_mouse
  ),
  head
)
```

```r
foo_foo %>%
  hop_through(forest) %>%
  scoop_up(field_mouse) %>%
  bop_on(head)
```

> Admitedly, this doesn’t explain what %>% actually does!

Simpler %>% example

The pipe operator takes the left-hand side of your expression and inserts it into the first argument of the right-hand side of the expression:

```
"cats" %>% rep(4)
```

There’s nothing special about `rep`, it’s `%>%` which is doing the work.

`%>%` is an example of what’s called **syntactic sugar** it makes code easier to write/read.

Pushing the pipe further

In some cases you don’t want the left-hand side in the first argument, you can explicitly shove it somewhere else by using a period `.`

```
"cats" %>% paste(., "are great", "but one can have too many", .)
```

Where does `%>%` come from?

`magrittr` is the package that gives us `%>%`, it was first introduced in 2014 and since then has become ridiculously popular.

The pipe is now an intrinsic part of the `tidyverse` and made available to us when we load it.

If you want to use %>% in your own packages, then consider using the usethis package

```
usethis::use_pipe()
```

Advice on using %>%

The pipe isn’t a hammer to be used without exception, some code is both harder to write and read with pipes.

Try to break pipe chains into blocks of similar operatios to make your code easier to understand at a glance:

```r
raw_data <- read_csv("data-raw/the-file.csv")

clean_data <- raw_data %>%
  clean() %>%
  clean_it() %>%
  cleaning() %>%
  cleaned()

clean_data <- clean_data %>%
  normalise() %>%
  normaler() %>%
  norm_it()
```

What if I hate `%>%`?

It's perfectly acceptable to hate `%>%`.

That's fine.

It's just sugar to sweeten the already lovely R.

However, you need a basic understanding of it to read most documentation pages in the `tidyverse` (and beyond).

Hierarchical counting

We can count by as many attributes as we like:

```
msleep %>%
  count(order, vore, sort = TRUE)
```

```
# A tibble: 32 × 3
   order          vore        n
   <chr>          <chr>   <int>
 1 Rodentia       herbi      16
 2 Carnivora      carni      12
 3 Primates       omni       10
 4 Artiodactyla   herbi       5
 5 Cetacea        carni       3
 6 Perissodactyla herbi       3
 7 Rodentia       <NA>        3
 8 Soricomorpha   omni        3
 9 Chiroptera     insecti     2
10 Hyracoidea     herbi       2
# … with 22 more rows
```

Is there any missing data? (I)

The `filter()` function allows us to query a dataset:

```
msleep %>%
  filter(sleep_total > 12)
```

We use == for equivalence tests:

```
msleep %>%
  filter(vore == "carni")
```

We can negate conditions in two different ways:

```
msleep %>%
  filter(vore != "carni")
```

```
msleep %>%
  filter(!vore == "carni")
```

Is there any missing data? (II)

We can’t use an equivalence test to filter for NA values, instead we need to use `is.na()`:

```r
msleep %>%
  filter(is.na(conservation))
```

The `drop_na()` function returns only those rows containing zero NA values:

```r
msleep %>%
  drop_na()
```

{naniar} and {ggplot2} (I)

We’re going to introduce {ggplot2} today which allows us to build data visualisations from scratch.

There are lots of packages that *extend* the capabilities of {ggplot2}.

Some of these packages provide additional **geoms** to help you build up charts.

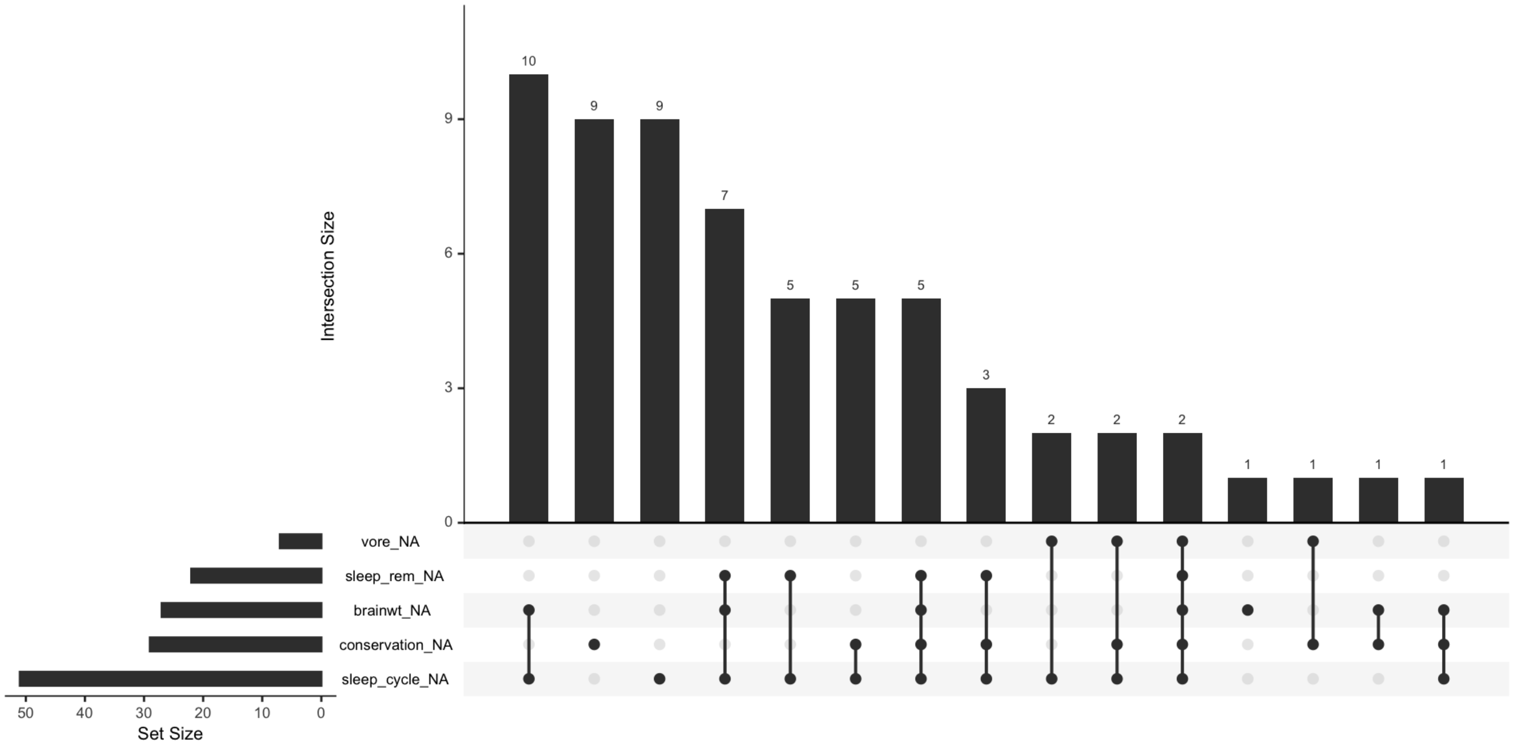
Some of these packages provide ready made data visualisations!

{naniar} is an example of this

{naniar} and {ggplot2} (II)

1. Install and load the {naniar} package
2. Run this code:

```r
msleep %>%
  gg_miss_upset()
```



Making our own data visualisations

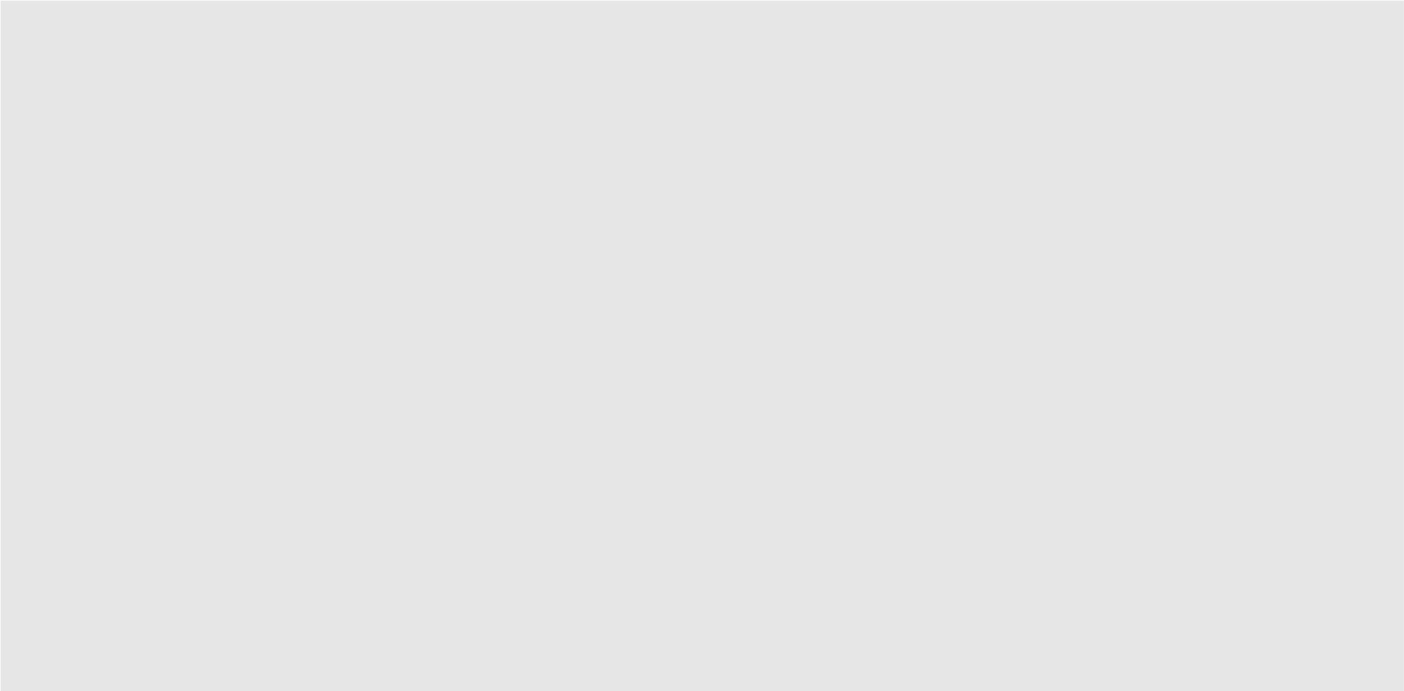
We’re going to make two different types of dataviz with the {msleep} dataset:

- Scatter plot of sleep_rem vs sleep_total
- Bar chart of mean sleep_total per vore

sleep_rem scatter plot: ggplot()

We start ggplot2 charts by providing a dataset to the ggplot() function

```r
msleep %>%
  ggplot()
```



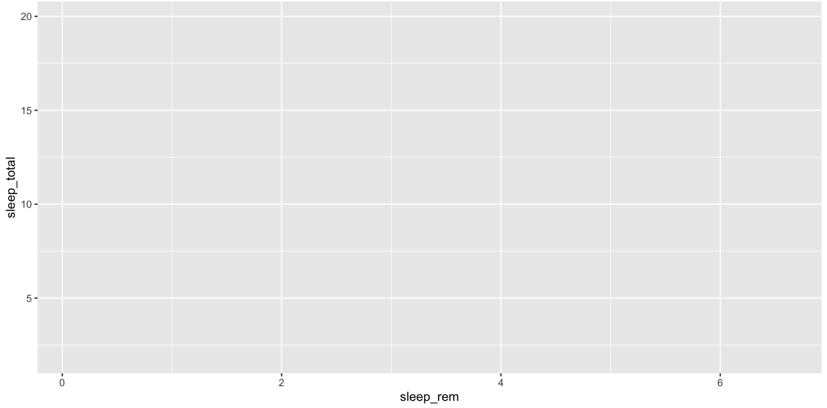
This creates an infamous grey rectangle. We need to provide more information to {ggplot2} so it can create a meaningful dataviz.

sleep_rem scatter plot: aes()

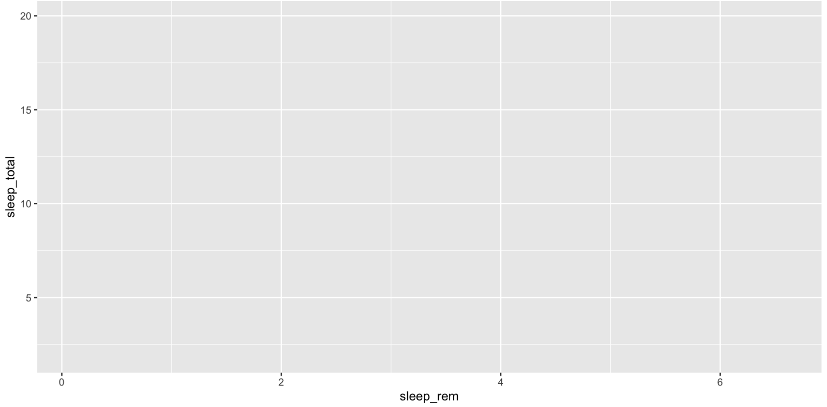
We need to tell {ggplot2} how to map columns in our dataset to coordinate systems in the chart.

We do this with the aes() function. There are two different ways we can write this in two different ways:

```r
msleep %>%
  ggplot(aes(x = sleep_rem,
             y = sleep_total))
```



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



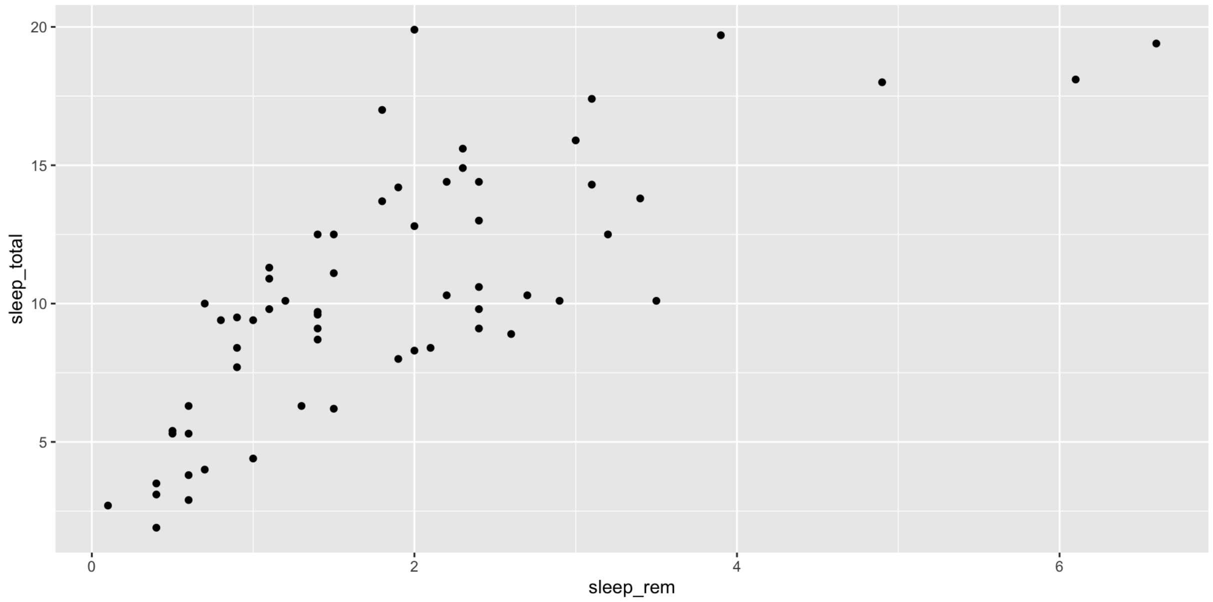
- {ggplot2} uses the columns to create new **scales** in the chart. As both columns are numeric we get an x and y continuous scale.
- The {tidyverse} functions are written specially to allow us to provide “naked” or “bare” column names[a] thanks to tidy evaluation.

sleep_rem scatter plot: geom_point()

We now add geoms to our charts. These use the aesthetic mappings to add geometric shapes to our chart.

- {ggplot2} was invented **before** the pipe existed so we use + to add layers to the chart.

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



sleep_rem scatter plot: improving

This is a pretty useless chart. It doesn’t tell any stories and is almost useless.

> What can we do to improve the chart?

sleep_rem bar chart: calculating (I)

Before we can create a bar chart of the mean sleep time per diet we need to calculate these values!

This means introducing the `group_by()` function for calculating in-group measures.

1. Add groups to data

2. Calculate in group measures

13 Ungroup the data when finished.

```r
msleep %>%
  group_by(vore)
```

```
# A tibble: 83 × 11
# Groups:   vore [5]
   name         genus vore  order conse…¹ sleep…² sleep…³ sleep…⁴
awake  brainwt
   <chr>        <chr> <chr> <chr> <chr>     <dbl>   <dbl>   <dbl>
<dbl>    <dbl>
 1 Cheetah      Acin… carni Carn… lc         12.1    NA    NA
11.9 NA
 2 Owl monkey   Aotus omni  Prim… <NA>       17       1.8  NA        7
0.0155
 3 Mountain be… Aplo… herbi Rode… nt         14.4     2.4  NA
9.6 NA
 4 Greater sho… Blar… omni  Sori… lc         14.9     2.3   0.133
9.1  0.00029
 5 Cow          Bos   herbi Arti… domest…     4       0.7   0.667  20
0.423
 6 Three-toed … Brad… herbi Pilo… <NA>       14.4     2.2   0.767
9.6 NA
 7 Northern fu… Call… carni Carn… vu          8.7     1.4   0.383
15.3 NA
 8 Vesper mouse Calo… <NA>  Rode… <NA>        7      NA    NA       17
NA
 9 Dog          Canis carni Carn… domest…    10.1     2.9   0.333
13.9   0.07
```

sleep_rem bar chart: calculating (II)

Before we can create a bar chart of the mean sleep time per diet we need to calculate these values!

This means introducing the `group_by()` function for calculating in-group measures.

1. Add groups to data
2. Calculate in group measures

- mutate() leaves all rows

3. Ungroup the data when finished.

```r
msleep %>%
  group_by(vore) %>%
  mutate(mean_sleep_total = mean(sleep_total))
```

```
# A tibble: 83 × 12
# Groups:   vore [5]
   name           genus vore  order conse…¹ sleep…² sleep…³ sleep…⁴
awake  brainwt
   <chr>          <chr> <chr> <chr> <chr>     <dbl>   <dbl>   <dbl>
<dbl>    <dbl>
 1 Cheetah        Acin… carni Carn… lc         12.1    NA     NA
11.9 NA
 2 Owl monkey     Aotus omni  Prim… <NA>       17       1.8   NA        7
0.0155
 3 Mountain be… Aplo… herbi Rode… nt           14.4     2.4   NA
9.6 NA
 4 Greater sho… Blar… omni  Sori… lc           14.9     2.3    0.133
9.1  0.00029
 5 Cow            Bos   herbi Arti… domest…     4       0.7    0.667  20
0.423
 6 Three-toed … Brad… herbi Pilo… <NA>         14.4     2.2    0.767
9.6 NA
 7 Northern fu… Call… carni Carn… vu            8.7     1.4    0.383
15.3 NA
 8 Vesper mouse Calo… <NA>  Rode… <NA>          7      NA     NA       17
NA
 9 Dog            Canis carni Carn… domest…    10.1     2.9    0.333
13.9   0.07
```

sleep_rem bar chart: calculating (III)

Before we can create a bar chart of the mean sleep time per diet we need to calculate these values!

This means introducing the `group_by()` function for calculating in-group measures.

1. Add groups to data

2. Calculate in group measures

- mutate() leaves all rows and columns
- group_by() throws away columns

3. Ungroup the data when finished.

```r
msleep %>%
  group_by(vore) %>%
  summarise(mean_sleep_total = mean(sleep_total))
```

```
# A tibble: 5 × 2
  vore    mean_sleep_total
  <chr>              <dbl>
1 carni              10.4 
2 herbi               9.51
3 insecti            14.9 
4 omni               10.9 
5 <NA>               10.2 
```

sleep_rem bar chart: calculating (IV)

Before we can create a bar chart of the mean sleep time per diet we need to calculate these values!

This means introducing the `group_by()` function for calculating in-group measures.

1. Add groups to data
2. Calculate in group measures

- mutate() leaves all rows and columns
- group_by() throws away columns

```
msleep %>%
  group_by(vore) %>%
  summarise(mean_sleep_total = mean(sleep_total)) %>%
  ungroup()
```

```
# A tibble: 5 × 2
  vore    mean_sleep_total
  <chr>              <dbl>
1 carni              10.4
2 herbi               9.51
3 insecti            14.9
4 omni               10.9
5 <NA>               10.2
```

3. Ungroup the data when finished.

sleep_rem bar chart: calculating (V)

Before we can create a bar chart of the mean sleep time per diet we need to calculate these values!

This means introducing the `group_by()` function for calculating in-group measures.

1. Add groups to data

2. Calculate in group measures

- mutate() leaves all rows and columns
- group_by() throws away columns

3. Ungroup the data when finished.

Don’t forget to make an assignment for this useful subset/analysis of the dataset

```
msleep %>%
  group_by(vore) %>%
  summarise(mean_sleep_total = mean(sleep_total)) %>%
  ungroup()
```

```
# A tibble: 5 × 2
  vore    mean_sleep_total
  <chr>              <dbl>
1 carni              10.4 
2 herbi               9.51
3 insecti            14.9 
4 omni               10.9 
5 <NA>               10.2 
```

```
mean_sleep_by_vore <- msleep %>%
  group_by(vore) %>%
  summarise(mean_sleep_total = mean(sleep_total)) %>%
  ungroup()
```

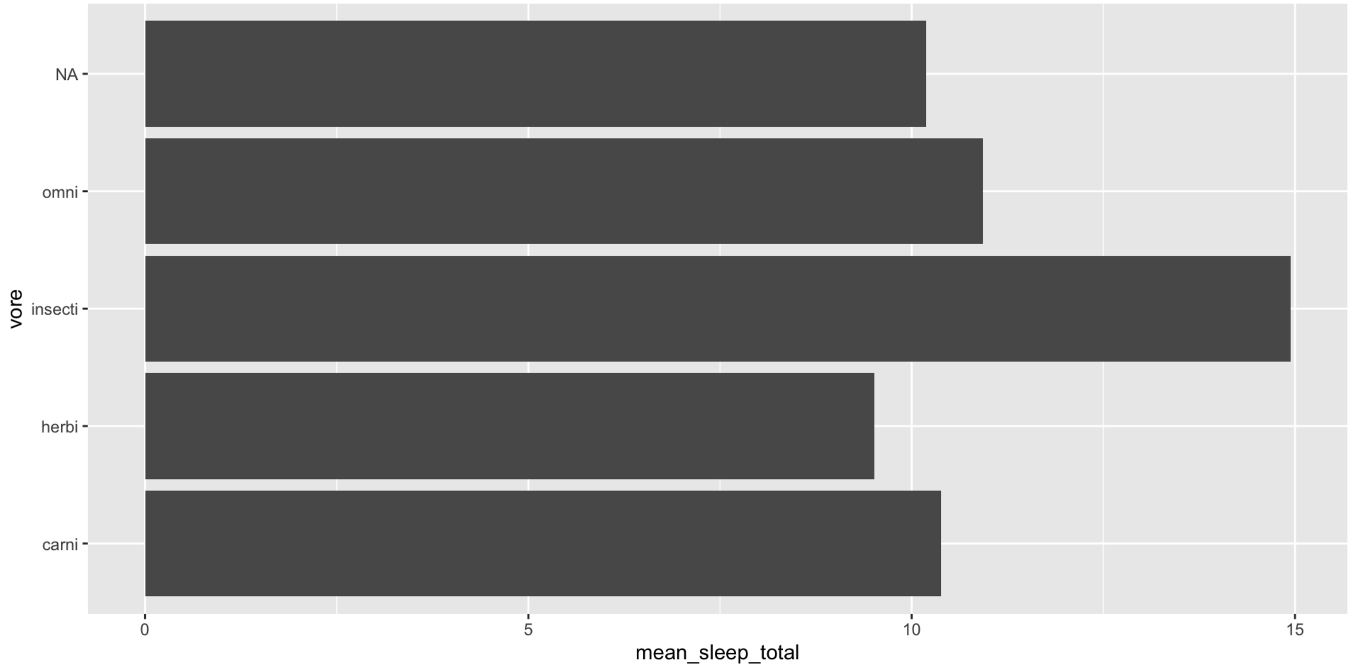
sleep_rem bar chart: Charting (I)

Please setup a {ggplot2} with the following specifications:

- x axis should be the “mean_sleep_total” column
- y axis should be the “vore” column

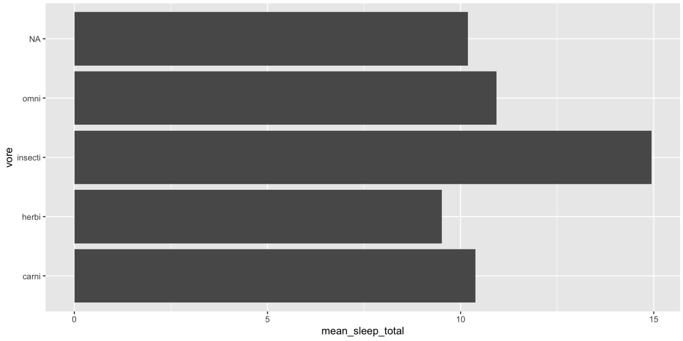
sleep_rem bar chart: Charting (II)

```r
mean_sleep_by_vore %>%
  ggplot() +
  aes(x = mean_sleep_total,
      y = vore) +
  geom_col()
```



> How can we improve this chart?!

sleep_rem bar chart: Ordering (I)



This chart is confusing to read because of the order bars are shown.

{ggplot2} uses information in the column to decide the order of scales.

- In the case of `chr` (character) columns alphabetical ordering is used
- In the case of `fct` (factor) columns are ordered by the levels in the factor.

> We’re going to come back to factors in the data visualisation week.

Mutating multiple columns (I)

Often we need to target multiple columns at once. For instance, it could be useful to convert all `sleep*` columns in the dataset to fractions of a day.

This is achieved with the `across()` function

The across() function takes 2 arguments.

```
msleep %>%
  mutate(across(argument_1, argument_2))
```

Mutating multiple columns (II)

Often we need to target multiple columns at once. For instance, it could be useful to convert all `sleep*` columns in the dataset to fractions of a day.

This is achieved with the `across()` function

The first argument is how we target specific columns.

This is achieved with a tidy-select function.

```r
msleep %>%
  mutate(across(argument_1, argument_2))
```

```r
msleep %>%
  mutate(across(starts_with("sleep"),
                argument_2))
```

Mutating multiple columns (III)

Often we need to target multiple columns at once. For instance, it could be useful to convert all `sleep*` columns in the dataset to fractions of a day.

This is achieved with the `across()` function

The 2nd argument is where we specify what happens to each column that's selected.

We need to write a function here.

That's **usually** achieved with the ~ shorthand

```
~ . / 24
```

```
msleep %>%
  mutate(across(argument_1, argument_2))
```

```
msleep %>%
  mutate(across(starts_with("sleep"),
                ~ . / 24))
```

```
# A tibble: 83 × 11
   name         genus vore  order conse…¹
sleep…² sleep…³ sleep_…⁴ awake  brainwt
   <chr>        <chr> <chr> <chr> <chr>
<dbl>   <dbl>    <dbl> <dbl>     <dbl>
 1 Cheetah      Acin… carni Carn… lc
0.504 NA       NA         11.9 NA
 2 Owl monkey   Aotus omni  Prim… <NA>
0.708  0.075   NA          7    0.0155
 3 Mountain b… Aplo… herbi Rode… nt         0.6
0.1    NA         9.6 NA
 4 Greater sh… Blar… omni  Sori… lc
0.621  0.0958  0.00556   9.1  0.00029
 5 Cow          Bos   herbi Arti… domest…
0.167  0.0292  0.0278   20    0.423
 6 Three-toed… Brad… herbi Pilo… <NA>      0.6
0.0917  0.0319    9.6 NA
 7 Northern f… Call… carni Carn… vu
0.362  0.0583  0.0160   15.3 NA
 8 Vesper mou… Calo… <NA>  Rode… <NA>
0.292 NA       NA         17   NA
 9 Dog          Canis carni Carn… domest…
0.421  0.121   0.0139   13.9  0.07
10 Roe deer     Capr  herbi Arti  lc
```

Mutating multiple columns (IV)

Often we need to target multiple columns at once. For instance, it could be useful to convert all `sleep*` columns in the dataset to fractions of a day.

This is achieved with the `across()` function

The 2nd argument is where we specify what happens to each column that's selected.

We need to write a function here.

```r
msleep %>%
  mutate(across(argument_1, argument_2))
```

That's **usually** achieved with the ~ shorthand

```r
~ . / 24
```

```r
msleep %>%
  mutate(across(starts_with("sleep"),
                ~ . / 24))
```

All tidyverse wrangling is done "column-wise" unless you explicitly specify row-wise operations with the `rowwise()`. We'll see some examples of this during the course.

gapminder

📝 Task: Get the gapminder dataset

SLIDE 1 OF 3

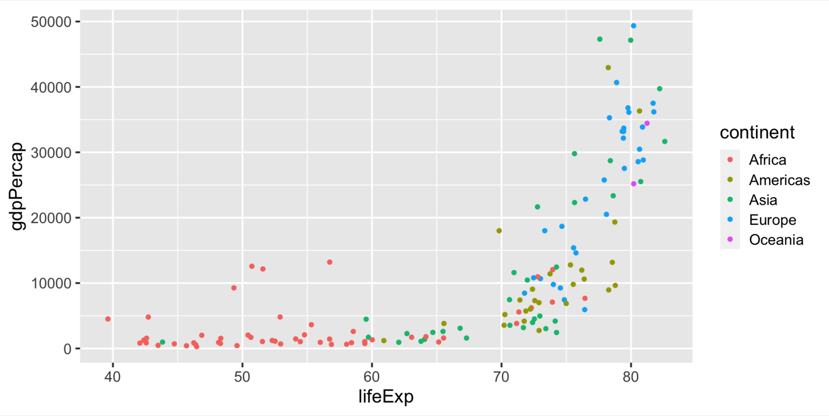
1. Add a heading for the gapminder dataset.
2. Load the `{gapminder}` package in the setup code chunk
3. Add a new code chunk and print the object `gapminder` to the console

```
gapminder
```

📝 Task: gapminder scatter plot

SLIDE 2 OF 3

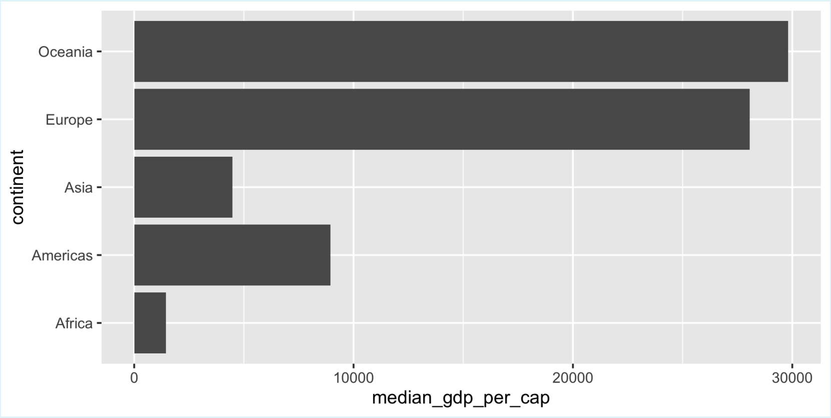
Create this scatter plot of the {gapminder} dataset for the year 2007.



📝 Task: gapminder barchart

SLIDE 3 OF 3

Create this bar chart of the {gapminder} dataset for the year 2007.



GBD Dataset

📝 Task: Get the GBD dataset

SLIDE 1 OF 2

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: Get the GBD dataset

SLIDE 2 OF 2

1. Add a new heading for the GBD Dataset to your .Rmd

Reading data into R

The `{readr}` package provides excellent tools for reading *rectangular* data from *plain-text* files like `.csv` and `.tsv` files.

We need to think about creating reproducible file paths. The easiest way to do so is as follows:

1. Add a code chunk to your `.Rmd`
2. Choose a name for the dataset you're importing, it's recommended to use `raw` or something similar to denote this is your data pre-wrangling.

   ```
   disease_burden_raw <-
   ```

3. Call the appropriate function from `{readr}` for your data, add `""` in the 1st argument

   ```
   disease_burden_raw <- read_csv("")
   ```

4. Press TAB with your cursor inside the quotation marks to bring up an interactive file tree and select your file.

   ```
   disease_burden_raw <- read_csv("data/global-burden-of-disease-data.csv")
   ```

Matching text in R (I)

A lot of data wrangling comes down to filtering, matching or otherwise manipulating text. In computer science we usually call text “strings” but R is a bit different and uses the term “character”.

The GBD dataset gives us a good example of this.

There are two different types of region in the dataset

- World Bank regions
- Geographic regions

```
disease_burden_raw %>%
  count(location_name)
```

```
# A tibble: 10 × 2
   location_name                     n
   <chr>                         <int>
 1 African Region                  720
 2 Eastern Mediterranean Region    720
 3 European Region                 720
 4 Region of the Americas          720
 5 South-East Asia Region          720
 6 Western Pacific Region          720
 7 World Bank High Income          720
 8 World Bank Low Income           720
 9 World Bank Lower Middle Income  720
10 World Bank Upper Middle Income  720
```

The tidyverse gives us an entire package called `{string}` for parsing/manipulating strings.

Matching text in R (II)

We can search the beginning of the strings with `str_starts()`.

```r
disease_burden_raw %>%
  filter(str_starts(location_name, "World Bank"))
```

```
# A tibble: 2,880 × 16
   measu…¹ measu…² locat…³ locat…⁴ sex_id sex_n…⁵ age_id
age_n…⁶ cause…⁷ cause…⁸
     <dbl> <chr>     <dbl> <chr>    <dbl> <chr>    <dbl>
<chr>     <dbl> <chr>
 1       1 Deaths    44575 World …      3 Both        22
All ag…     409 Non-co…
 2       1 Deaths    44575 World …      3 Both        22
All ag…     409 Non-co…
 3       1 Deaths    44575 World …      3 Both        22
All ag…     409 Non-co…
 4       1 Deaths    44575 World …      3 Both        22
All ag…     294 All ca…
 5       1 Deaths    44575 World …      3 Both        22
All ag…     294 All ca…
 6       1 Deaths    44575 World …      3 Both        22
All ag…     294 All ca…
 7       1 Deaths    44575 World …      3 Both        22
All ag…     295 Commun…
 8       1 Deaths    44575 World …      3 Both        22
All ag…     295 Commun…
 9       1 Deaths    44575 World …      3 Both        22
All ag…     295 Commun…
10       1 Deaths    44576 World        3 Both        22
```

Matching text in R (II)

For more complex string matching we have to make use of REGEX.

REGEX stands for regular expressions and is an approach to string manipulation that is implemented in all programming languages.

We can use regular expressions in the `pattern` argument for all `{stringr}` functions.

regex101.com is a really useful tool for building up complex regular expressions.

```r
disease_burden_raw %>%
  filter(str_detect(location_name, "^World Bank"))
```

```
# A tibble: 2,880 × 16
   measu…¹ measu…² locat…³ locat…⁴ sex_id sex_n…⁵ age_id
age_n…⁶ cause…⁷ cause…⁸
     <dbl> <chr>     <dbl> <chr>    <dbl> <chr>    <dbl>
<chr>     <dbl> <chr>
 1       1 Deaths    44575 World …      3 Both        22
All ag…     409 Non-co…
 2       1 Deaths    44575 World …      3 Both        22
All ag…     409 Non-co…
 3       1 Deaths    44575 World …      3 Both        22
All ag…     409 Non-co…
 4       1 Deaths    44575 World …      3 Both        22
All ag…     294 All ca…
 5       1 Deaths    44575 World …      3 Both        22
All ag…     294 All ca…
 6       1 Deaths    44575 World …      3 Both        22
All ag…     294 All ca…
 7       1 Deaths    44575 World …      3 Both        22
All ag…     295 Commun…
 8       1 Deaths    44575 World …      3 Both        22
All ag…     295 Commun…
 9       1 Deaths    44575 World …      3 Both        22
All ag…     295 Commun…
10       1 Deaths    44576 World        3 Both        22
```

GBD data visualisation (I)

I’d like you to filter the dataset so that all of these are true:

- We can only see data for the most recent year
- We can see the **percent** of deaths for each cause_name
- There are 16 rows in the filtered dataset

GBD data visualisation (II)

Once we’ve filtered down the data, let’s select only the interesting columns:

```
disease_burden_percent_deaths %>%
  select(location_name, cause_name, val)
```

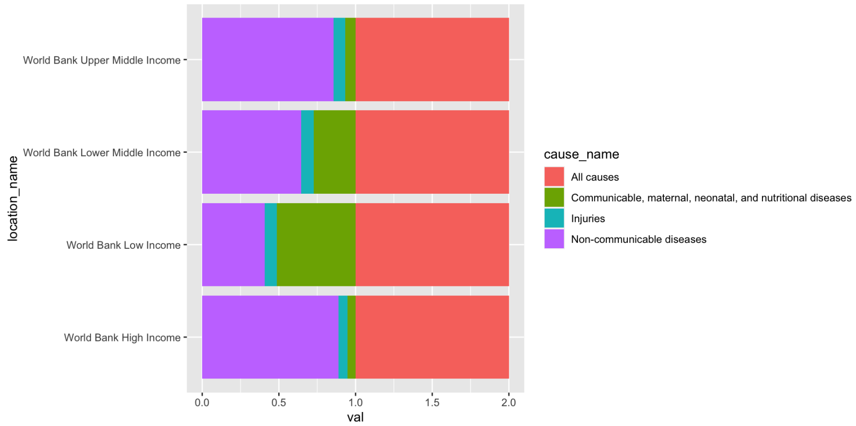
```
# A tibble: 16 × 3
   location_name                cause_name                                 val
   <chr>                        <chr>                                    <dbl>
 1 World Bank High Income       All causes                              1     
 2 World Bank High Income       Communicable, maternal, neonatal, and … 0.0528
 3 World Bank Upper Middle Income Non-communicable diseases              0.856 
 4 World Bank Upper Middle Income All causes                             1     
 5 World Bank Upper Middle Income Communicable, maternal, neonatal, and … 0.0662
 6 World Bank Low Income        All causes                              1     
 7 World Bank Upper Middle Income Injuries                               0.0780
 8 World Bank Low Income        Communicable, maternal, neonatal, and … 0.513 
 9 World Bank High Income       Injuries                                0.0579
10 World Bank Lower Middle Income Non-communicable diseases              0.645 
11 World Bank High Income       Non-communicable diseases               0.889 
12 World Bank Low Income        Injuries                                0.0812
13 World Bank Lower Middle Income All causes                             1     
14 World Bank Lower Middle Income Communicable, maternal, neonatal, and … 0.273 
15 World Bank Lower Middle Income Injuries                               0.0824
16 World Bank Low Income        Non-communicable diseases               0.406 
```

Let’s visualise this as a barchart

GBD data visualisation (II)

> How can we make this chart more meaningful?

```r
disease_burden_percent_deaths %>%
  select(location_name, cause_name, val) %>%
  ggplot() +
  aes(x = val,
      y = location_name,
      fill = cause_name) +
  geom_col()
```



Going further

We've only barely scratched the surface of wrangling with the tidyverse. These are the topics we have covered:

- Read in data files with `read_*()` functions
- Understand the difference between `tibbles` and `data.frame`
- Filter datasets with `filter()`
- Use `{stringr}` to searh/modify strings
- Use `mutate()` to modify existing columns and add new columns
- Use `group_by()` to calculate in-group values
- Use `{ggplot2}` for exploratory data analysis.

- The R for Data Sciene book is an excellent resource for reinforcing this content and movir on to more advanced topics.
- RStudio has a lot of really useful cheatsheets · https://www.rstudio.com/resources/cheatsheet
- We'll get into the technical details of using `{ggplot2}` next lecture.
- In the "survey data" week we'll introduce the principles of tidy data and several functions from `{tidyr}` for wrangling datasets.

References

1. Wickham, H. & Grolemund, G. *R for data science: Import, tidy, transform, visualize, and model data*. (O’Reilly, 2016).
2. Matloff, N. Teaching R in a Kinder, Gentler, More Effective Manner: (2022).
3. Global Health Data Exchange. Global Burden of Disease Dataset Explorer. *Institute for Health Metrics and Evaluation* (2022).
4. Savage, V. M. & West, G. B. A quantitative, theoretical framework for understanding mammalian sleep. *Proceedings of the National Academy of Sciences* **104**, 1051–1056 (2007).
5. Hans Rosling. The best stats you’ve ever seen [Video]. *The best stats you’ve ever seen* (2006).