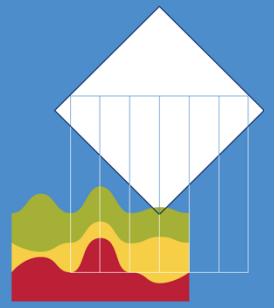
Introduction to the tidyverse

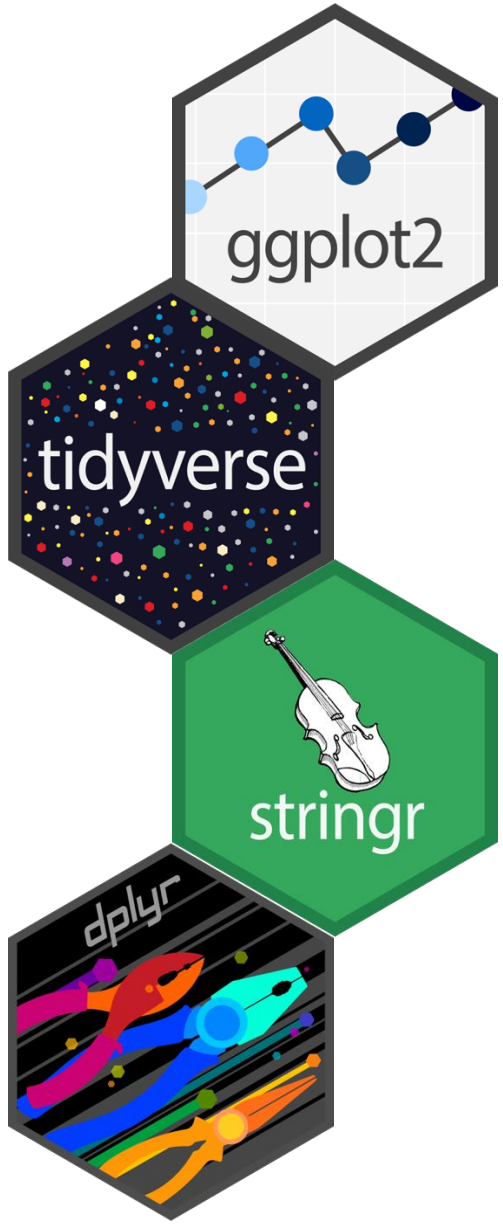
Maria Christodoulou

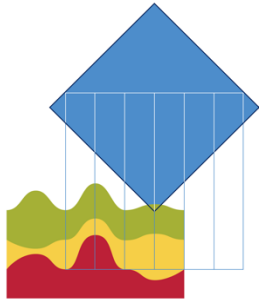
Mariagrazia Zottoli



What is the `tidyverse`?

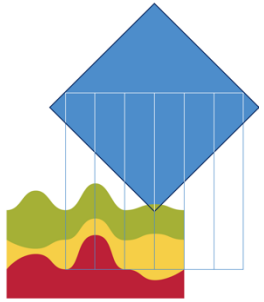
- `tidyverse` is a collection of packages developed primarily by Hadley Wickham
- All packages are consistently written and behave predictably
- The tools provided are intuitive and cover most data manipulation needs





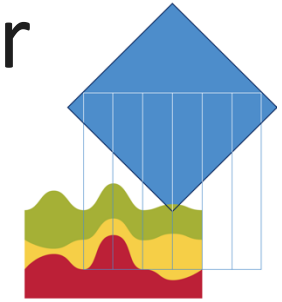
What is data wrangling?

- It is the process of managing, formatting, and transforming raw data prior to analysis
- Consists of three steps:
  - importing data
  - tidying data
  - transforming data
- Often the process between tidying and transforming is iterative



When are you likely to need it?

- Data wrangling is a necessary step any time you deal with a new dataset
- This is often because the way we record data and the way we need to format data for particular analyses differ greatly
- Data wrangling does not have to be complicated or intimidating



The `starwars` dataset

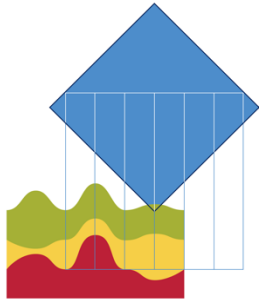
- In `dplyr` library
- Contains details of Star Wars characters
- 87 rows and 14 variables

```
> library(tidyverse)
> starwars
# A tibble: 87 × 14
   name     height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>     <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
 1 Luke S…     172    77 blond   fair    blue       19   male  mascu… Tatooi… Human
 2 C-3PO       167    75 NA      gold    yellow    112   none  mascu… Tatooi… Droid
 3 R2-D2        96    32 NA      white,… red        33   none  mascu… Naboo   Droid
 4 Darth …     202   136 none    white   yellow     41.9 male  mascu… Tatooi… Human
 5 Leia O…     150    49 brown   light   brown      19   fema… femin… Aldera… Human
 6 Owen L…     178   120 brown,… light   blue       52   male  mascu… Tatooi… Human
 7 Beru W…     165    75 brown   light   blue       47   fema… femin… Tatooi… Human
 8 R5-D4        97    32 NA      white,… red        NA   none  mascu… Tatooi… Droid
```

The five verbs of `dplyr` – `filter()`

- `filter()` lets you view only the observations that fulfil certain requirements
- It does NOT delete all observations that do not fulfil the conditions
- Standard syntax `filter(data, condition)`
- Returning to the starwars data set, say we want to view only characters above 100 cm height (sorry Ewoks…)

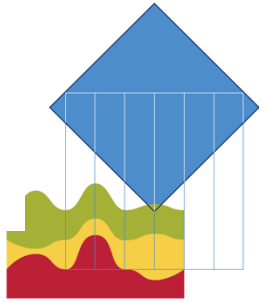
```
filter(starwars, height>100)
```



The five verbs of `dplyr` – `filter()`

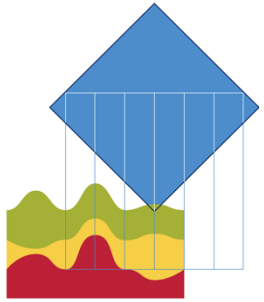
```
filter(starwars, height>100)
```

```
> filter(starwars,height>100)
# A tibble: 74 × 14
   name    height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>    <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
 1 Luke S…    172    77 blond   fair    blue       19   male  mascu… Tatooi… Human
 2 C-3PO      167    75 NA      gold    yellow    112   none  mascu… Tatooi… Droid
 3 Darth …    202   136 none    white   yellow     41.9 male  mascu… Tatooi… Human
 4 Leia O…    150    49 brown   light   brown      19   fema… femin… Aldera… Human
 5 Owen L…    178   120 brown,… light   blue       52   male  mascu… Tatooi… Human
 6 Beru W…    165    75 brown   light   blue       47   fema… femin… Tatooi… Human
 7 Biggs …    183    84 black   light   brown      24   male  mascu… Tatooi… Human
 8 Obi-Wa…    182    77 auburn… fair    blue-g…    57   male  mascu… Stewjon Human
 9 Anakin…    188    84 blond   fair    blue       41.9 male  mascu… Tatooi… Human
10 Wilhuf…    180    NA auburn… fair    blue       64   male  mascu… Eriadu  Human
# … with 64 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



The five verbs of `dplyr` – `filter()`

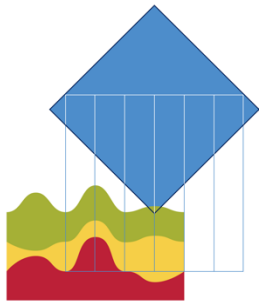
- Reminder – logical conditions include: >, <, ==, !=, >=,<=
- How would you filter entries with mass less than or equal to 100?



The five verbs of `dplyr` – `filter()`

```
filter(starwars, mass<=100)
```

```
> filter(starwars,mass<=100)
# A tibble: 49 × 14
   name     height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>     <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr> <chr>   <chr>
 1 Luke S…     172    77 blond   fair    blue       19   male  mascu… Tatooi… Human
 2 C-3PO       167    75 NA      gold    yellow    112   none  mascu… Tatooi… Droid
 3 R2-D2        96    32 NA      white,… red        33   none  mascu… Naboo   Droid
 4 Leia O…     150    49 brown   light   brown      19   fema… femin… Aldera… Human
 5 Beru W…     165    75 brown   light   blue       47   fema… femin… Tatooi… Human
 6 R5-D4        97    32 NA      white,… red        NA   none  mascu… Tatooi… Droid
 7 Biggs …     183    84 black   light   brown      24   male  mascu… Tatooi… Human
 8 Obi-Wa…     182    77 auburn… fair    blue-g…    57   male  mascu… Stewjon Human
 9 Anakin…     188    84 blond   fair    blue       41.9 male  mascu… Tatooi… Human
10 Han So…     180    80 brown   fair    brown      29   male  mascu… Corell… Human
# … with 39 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



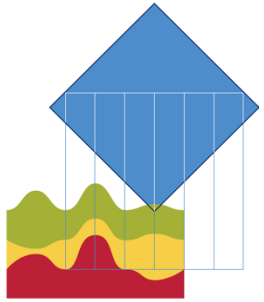
The five verbs of `dplyr` – `filter()`

- Reminder - logical conditions include: >, <, ==, !=, >=,<=

- How would you filter entries with mass less than or equal to100?

```
filter(starwars, mass<=100)
```

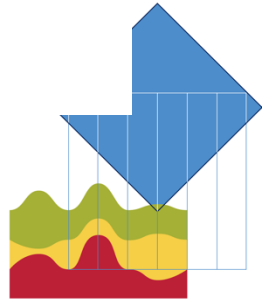
- How about entries with red eyes?



The five verbs of `dplyr` – `filter()`

```
filter(starwars, eye_color=="red")
```

```
> filter(starwars,eye_color=="red")
# A tibble: 5 × 14
  name      height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
  <chr>      <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
1 R2-D2         96    32 NA      white,… red          33 none  mascu… Naboo   Droid
2 R5-D4         97    32 NA      white,… red          NA none  mascu… Tatooi… Droid
3 IG-88        200   140 none    metal   red          15 none  mascu… NA      Droid
4 Bossk        190   113 none    green   red          53 male  mascu… Trando… Trando…
5 Nute Gu…     191    90 none    mottle… red          NA male  mascu… Cato N… Neimod…
# … with 3 more variables: films <list>, vehicles <list>, starships <list>, and
#   abbreviated variable names ¹hair_color, ²skin_color, ³eye_color, ⁴birth_year,
#   ⁵homeworld
# ℹ Use `colnames()` to see all variable names
```



The five verbs of `dplyr – filter()`

- Reminder - logical conditions include: >, <, ==, !=, >=,<=

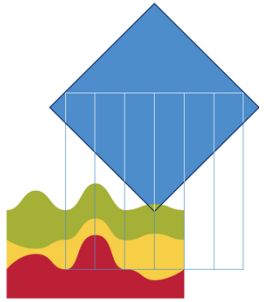
- How would you filter entries with mass less than or equal to100?

```
filter(starwars, mass<=100)
```

- How about entries with red eyes?

```
filter(starwars, eye_color=="red")
```

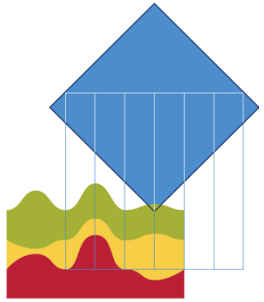
- How would you view all species expect from droids?



The five verbs of `dplyr` – `filter()`

```
filter(starwars, species!="Droid")
```

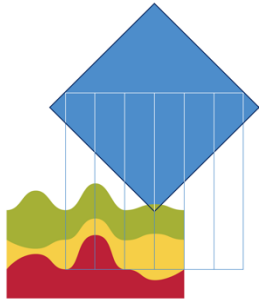
```
> filter(starwars, species!="Droid")
# A tibble: 77 × 14
   name    height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>    <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
 1 Luke S…    172    77 blond   fair    blue       19   male  mascu… Tatooi… Human
 2 Darth …    202   136 none    white   yellow     41.9 male  mascu… Tatooi… Human
 3 Leia O…    150    49 brown   light   brown      19   fema… femin… Aldera… Human
 4 Owen L…    178   120 brown,… light   blue       52   male  mascu… Tatooi… Human
 5 Beru W…    165    75 brown   light   blue       47   fema… femin… Tatooi… Human
 6 Biggs …    183    84 black   light   brown      24   male  mascu… Tatooi… Human
 7 Obi-Wa…    182    77 auburn… fair    blue-g…    57   male  mascu… Stewjon Human
 8 Anakin…    188    84 blond   fair    blue       41.9 male  mascu… Tatooi… Human
 9 Wilhuf…    180    NA auburn… fair    blue       64   male  mascu… Eriadu  Human
10 Chewba…    228   112 brown   unknown blue      200   male  mascu… Kashyy… Wookiee
# … with 67 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



The five verbs of `dplyr` – `filter()`

- What if you wanted to fulfil more than one condition?

- For example: what if you wanted to see all droids with red eyes?



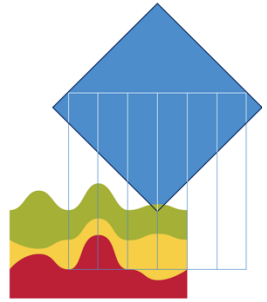
The five verbs of `dplyr` – `filter()`

- Solution 1: Break it into two steps
- How would you do that?
- Step 1:

```
filter(starwars, eye_color=="red")
```

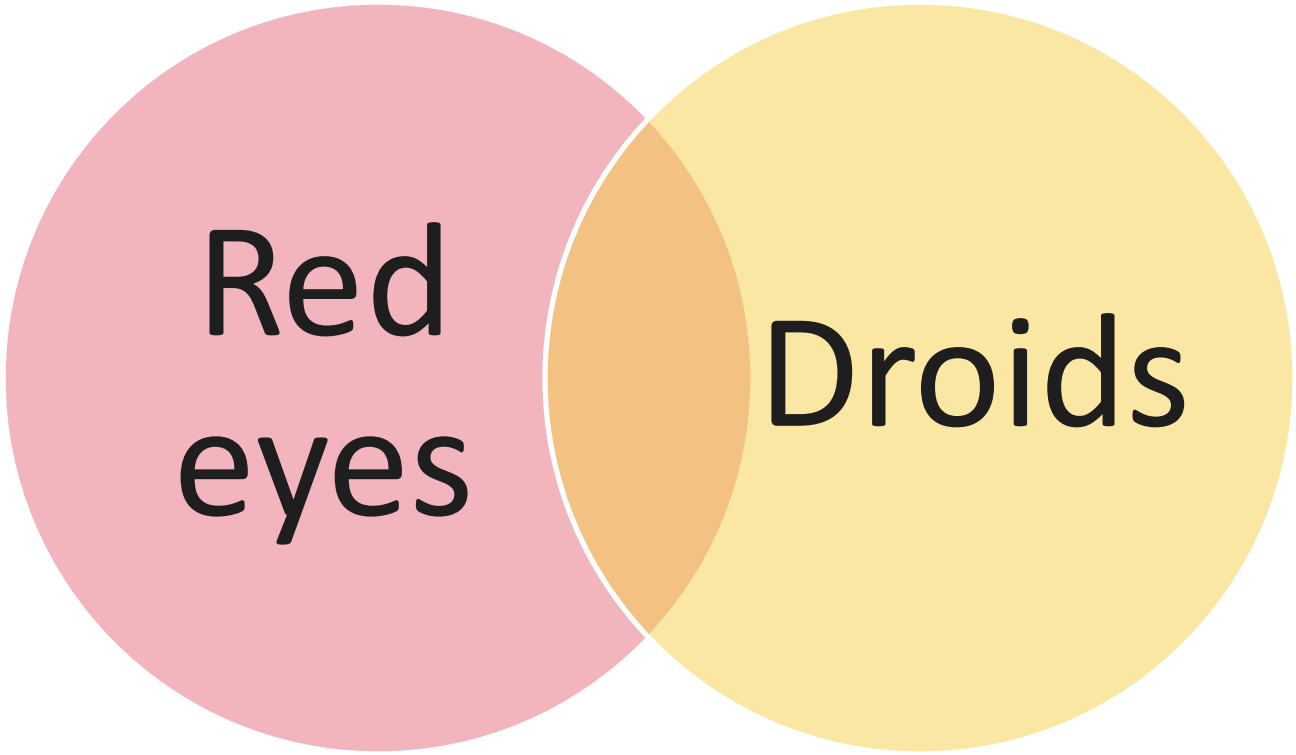
- Step 2:

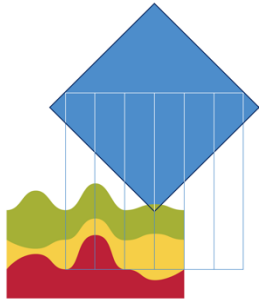
```
filter(filter(starwars, eye_color=="red"),
                species =="Droid")
```



The five verbs of `dplyr` – `filter()`

- Solution 2: Using set operations such as & and |

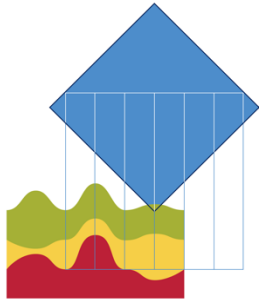




The five verbs of `dplyr – filter()`

Try the following lines of code

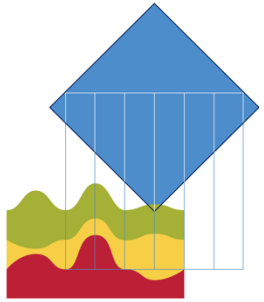
```
filter(starwars, eye_color=="red" &
         species =="Droid")
filter(starwars, eye_color=="red" |
          species =="Droid")
filter(starwars, eye_color=="red" &
         species !="Droid")
filter(starwars, eye_color=="red" |
         species !="Droid")
```



The five verbs of `dplyr` – `filter()`

```
filter(starwars, eye_color=="red" &
         species =="Droid")
```

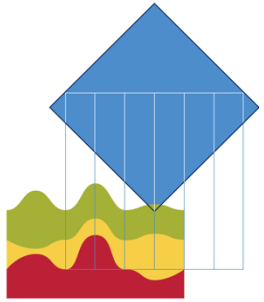
```
> filter(starwars, eye_color=="red" &
+          species =="Droid")
# A tibble: 3 × 14
  name  height  mass hair_color skin_…¹ eye_c…² birth…³ sex   gender homew…⁴ species
  <chr>  <int> <dbl> <chr>      <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
1 R2-D2     96    32 NA         white,… red          33 none  mascu… Naboo   Droid
2 R5-D4     97    32 NA         white,… red          NA none  mascu… Tatooi… Droid
3 IG-88    200   140 none       metal   red          15 none  mascu… NA      Droid
# … with 3 more variables: films <list>, vehicles <list>, starships <list>, and
#   abbreviated variable names ¹skin_color, ²eye_color, ³birth_year, ⁴homeworld
# i Use `colnames()` to see all variable names
```



The five verbs of `dplyr` – `filter()`

```
filter(starwars, eye_color=="red" |
                 species =="Droid")
```

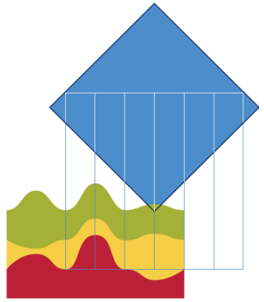
```
> filter(starwars, eye_color=="red" |
+          species =="Droid")
# A tibble: 8 × 14
  name     height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
  <chr>     <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
1 C-3PO       167    75 NA      gold    yellow      112 none  mascu… Tatooi… Droid
2 R2-D2        96    32 NA      white,… red          33 none  mascu… Naboo   Droid
3 R5-D4        97    32 NA      white,… red          NA none  mascu… Tatooi… Droid
4 IG-88       200   140 none    metal   red          15 none  mascu… NA      Droid
5 Bossk       190   113 none    green   red          53 male  mascu… Trando… Trando…
6 Nute Gu…    191    90 none    mottle… red          NA male  mascu… Cato N… Neimod…
7 R4-P17       96    NA none    silver… red, b…      NA none  femin… NA      Droid
8 BB8          NA    NA none    none    black        NA none  mascu… NA      Droid
# … with 3 more variables: films <list>, vehicles <list>, starships <list>, and
#   abbreviated variable names ¹hair_color, ²skin_color, ³eye_color, ⁴birth_year,
#   ⁵homeworld
# ℹ Use `colnames()` to see all variable names
```



The five verbs of `dplyr` – `filter()`

```
filter(starwars, eye_color=="red" &
                 species !="Droid")
```

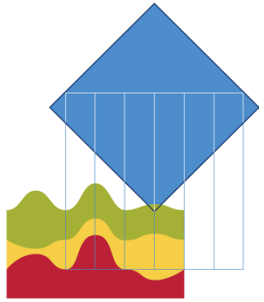
```
> filter(starwars, eye_color=="red" &
+          species !="Droid")
# A tibble: 2 × 14
  name     height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
  <chr>     <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
1 Bossk       190   113 none    green   red          53 male  mascu… Trando… Trando…
2 Nute Gu…    191    90 none    mottle… red          NA male  mascu… Cato N… Neimod…
# … with 3 more variables: films <list>, vehicles <list>, starships <list>, and
#   abbreviated variable names ¹hair_color, ²skin_color, ³eye_color, ⁴birth_year,
#   ⁵homeworld
# ℹ Use `colnames()` to see all variable names
```



The five verbs of `dplyr` – `filter()`

```
filter(starwars, eye_color=="red" |
         species !="Droid")
```

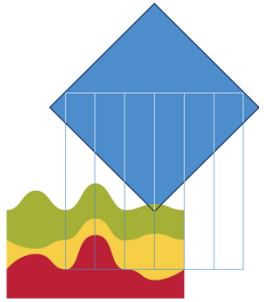
```
> filter(starwars, eye_color=="red" |
+          species !="Droid")
# A tibble: 80 × 14
   name    height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>    <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
 1 Luke S…    172    77 blond   fair    blue       19   male  mascu… Tatooi… Human
 2 R2-D2       96    32 NA      white,… red        33   none  mascu… Naboo   Droid
 3 Darth …    202   136 none    white   yellow     41.9 male  mascu… Tatooi… Human
 4 Leia O…    150    49 brown   light   brown      19   fema… femin… Aldera… Human
 5 Owen L…    178   120 brown,… light   blue       52   male  mascu… Tatooi… Human
 6 Beru W…    165    75 brown   light   blue       47   fema… femin… Tatooi… Human
 7 R5-D4       97    32 NA      white,… red        NA   none  mascu… Tatooi… Droid
 8 Biggs …    183    84 black   light   brown      24   male  mascu… Tatooi… Human
 9 Obi-Wa…    182    77 auburn… fair    blue-g…    57   male  mascu… Stewjon Human
10 Anakin…    188    84 blond   fair    blue       41.9 male  mascu… Tatooi… Human
# … with 70 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



The five verbs of `dplyr` – `select()`

- `select()` allows you to choose the variables you want to view or manipulate
- Whereas `filter()` works on rows, `select()` works on columns
- Standard syntax : `select(data, variables)`
- For the `starwars` dataset let us say we want to only view the character name and the eye colour:

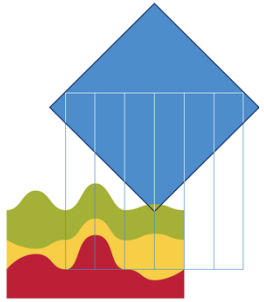
```
select(starwars, name, eye_color)
```



The five verbs of `dplyr` – `select()`

```
select(starwars, name, eye_color)
```

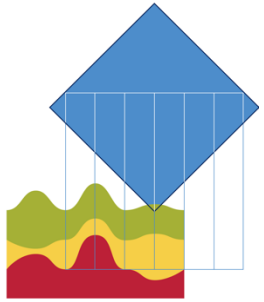
```
> select(starwars, name, eye_color)
# A tibble: 87 × 2
   name               eye_color
   <chr>              <chr>
 1 Luke Skywalker     blue
 2 C-3PO              yellow
 3 R2-D2              red
 4 Darth Vader        yellow
 5 Leia Organa        brown
 6 Owen Lars          blue
 7 Beru Whitesun lars blue
 8 R5-D4              red
 9 Biggs Darklighter  brown
10 Obi-Wan Kenobi     blue-gray
# … with 77 more rows
# ℹ Use `print(n = ...)` to see more rows
>
```



The five verbs of `dplyr – select()`

- You can also select by variable position if you know the column numbers you are after:

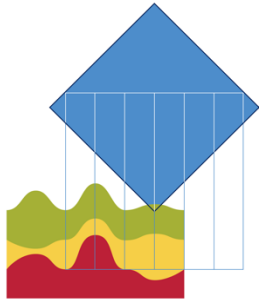
```
> starwars
# A tibble: 87 × 14
   name    height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>    <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr> <chr>   <chr>
 1 Luke S…    172    77 blond   fair    blue       19   male  mascu… Tatooi… Human
 2 C-3PO      167    75 NA      gold    yellow    112   none  mascu… Tatooi… Droid
 3 R2-D2       96    32 NA      white,… red        33   none  mascu… Naboo   Droid
 4 Darth …    202   136 none    white   yellow     41.9 male  mascu… Tatooi… Human
 5 Leia O…    150    49 brown   light   brown      19   fema… femin… Aldera… Human
 6 Owen L…    178   120 brown,… light   blue       52   male  mascu… Tatooi… Human
 7 Beru W…    165    75 brown   light   blue       47   fema… femin… Tatooi… Human
 8 R5-D4       97    32 NA      white,… red        NA   none  mascu… Tatooi… Droid
 9 Biggs …    183    84 black   light   brown      24   male  mascu… Tatooi… Human
10 Obi-Wa…    182    77 auburn… fair    blue-g…    57   male  mascu… Stewjon Human
# … with 77 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



The five verbs of `dplyr` – `select()`

```
select(starwars, 1,6)
```

```
> select(starwars, 1,6)
# A tibble: 87 × 2
   name               eye_color
   <chr>              <chr>
 1 Luke Skywalker     blue
 2 C-3PO              yellow
 3 R2-D2              red
 4 Darth Vader        yellow
 5 Leia Organa        brown
 6 Owen Lars          blue
 7 Beru Whitesun lars blue
 8 R5-D4              red
 9 Biggs Darklighter  brown
10 Obi-Wan Kenobi     blue-gray
# … with 77 more rows
# ℹ Use `print(n = ...)` to see more rows
```

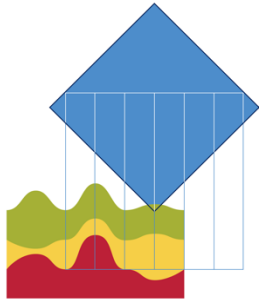


The five verbs of `dplyr – select()`

- You can also select by variable position if you know the column numbers you are after:

```
select(starwars, 1,6)
```

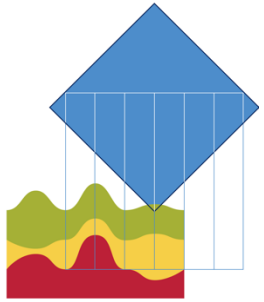
- How would you select the 2nd, 3rd, and 5th columns?



The five verbs of `dplyr` – `select()`

```
select(starwars, 2,3,5)
```

```
> select(starwars, 2,3,5)
# A tibble: 87 × 3
   height  mass skin_color
    <int> <dbl> <chr>
 1    172    77 fair
 2    167    75 gold
 3     96    32 white, blue
 4    202   136 white
 5    150    49 light
 6    178   120 light
 7    165    75 light
 8     97    32 white, red
 9    183    84 light
10    182    77 fair
# … with 77 more rows
# ℹ Use `print(n = ...)` to see more rows
```



The five verbs of `dplyr – select()`

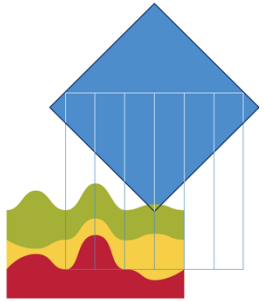
- You can also select by variable position if you know the column numbers you are after:

```
select(starwars, 1,6)
```

- How would you select the 2nd, 3rd, and 5th columns?

```
select(starwars, 2,3,5)
```

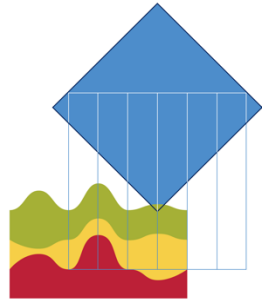
- How would you select the first five columns?



The five verbs of `dplyr` – `select()`

```
select(starwars, 1:5)
```

```
> select(starwars, 1:5)
# A tibble: 87 × 5
   name               height  mass hair_color    skin_color
   <chr>               <int> <dbl> <chr>         <chr>
 1 Luke Skywalker        172    77 blond         fair
 2 C-3PO                 167    75 NA            gold
 3 R2-D2                  96    32 NA            white, blue
 4 Darth Vader           202   136 none          white
 5 Leia Organa           150    49 brown         light
 6 Owen Lars             178   120 brown, grey   light
 7 Beru Whitesun lars    165    75 brown         light
 8 R5-D4                  97    32 NA            white, red
 9 Biggs Darklighter     183    84 black         light
10 Obi-Wan Kenobi        182    77 auburn, white fair
# … with 77 more rows
# ℹ Use `print(n = ...)` to see more rows
```



The five verbs of `dplyr – select()`

- You can also select by variable position if you know the column numbers you are after:

```
select(starwars, 1,6)
```

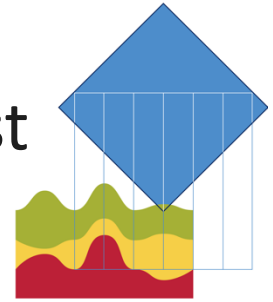
- How would you select the 2nd, 3rd, and 5th columns?

```
select(starwars, 2,3,5)
```

- How would you select the first five columns?

```
select(starwars, 1:5)
```

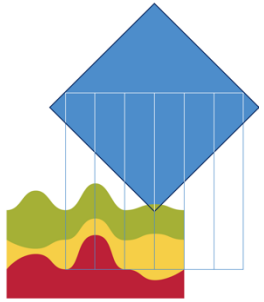
- Trickier one: How would you select all columns except for the first one?



The five verbs of `dplyr` – `select()`

```
select(starwars,-1)
```

```
> select(starwars,-1)
# A tibble: 87 × 13
   height  mass hair_co…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species films
    <int> <dbl> <chr>     <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>   <lis>
 1    172    77 blond     fair    blue       19   male  mascu… Tatooi… Human   <chr>
 2    167    75 NA        gold    yellow    112   none  mascu… Tatooi… Droid   <chr>
 3     96    32 NA        white,… red        33   none  mascu… Naboo   Droid   <chr>
 4    202   136 none      white   yellow     41.9 male  mascu… Tatooi… Human   <chr>
 5    150    49 brown     light   brown      19   fema… femin… Aldera… Human   <chr>
 6    178   120 brown, g… light   blue       52   male  mascu… Tatooi… Human   <chr>
 7    165    75 brown     light   blue       47   fema… femin… Tatooi… Human   <chr>
 8     97    32 NA        white,… red        NA   none  mascu… Tatooi… Droid   <chr>
 9    183    84 black     light   brown      24   male  mascu… Tatooi… Human   <chr>
10    182    77 auburn, … fair    blue-g…    57   male  mascu… Stewjon Human   <chr>
# … with 77 more rows, 2 more variables: vehicles <list>, starships <list>, and
#   abbreviated variable names ¹hair_color, ²skin_color, ³eye_color, ⁴birth_year,
#   ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

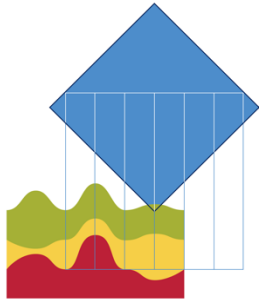


The five verbs of `dplyr – select()`

- `select()` also contains some helper functions to make your life easier

- To select all variables beginning with the letter "`h`", use the helper `starts_with()`

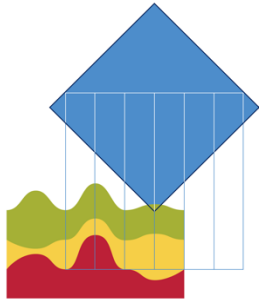
```
select(starwars, starts_with("h"))
```



The five verbs of `dplyr` – `select()`

```
select(starwars, starts_with("h"))
```

```
> select(starwars, starts_with("h"))
# A tibble: 87 × 3
   height hair_color    homeworld
    <int> <chr>         <chr>
 1    172 blond         Tatooine
 2    167 NA            Tatooine
 3     96 NA            Naboo
 4    202 none          Tatooine
 5    150 brown         Alderaan
 6    178 brown, grey   Tatooine
 7    165 brown         Tatooine
 8     97 NA            Tatooine
 9    183 black         Tatooine
10    182 auburn, white Stewjon
# … with 77 more rows
# ℹ Use `print(n = ...)` to see more rows
```



The five verbs of `dplyr – select()`

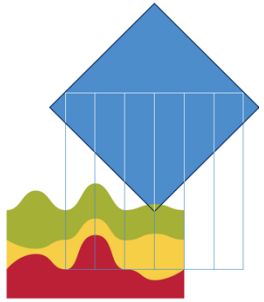
- `select()` also contains some helper functions to make your life easier

- To select all variables beginning with the letter "`h`", use the helper `starts_with()`

```
select(starwars, starts_with("h"))
```

- To select all variables ending with "`color`", use the helper `ends_with()`

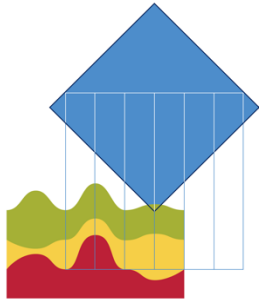
```
select(starwars, ends_with("color"))
```



The five verbs of `dplyr` – `select()`

```
select(starwars, ends_with("color"))
```

```
> select(starwars, ends_with("color"))
# A tibble: 87 × 3
   hair_color    skin_color  eye_color
   <chr>         <chr>       <chr>
 1 blond         fair        blue
 2 NA            gold        yellow
 3 NA            white, blue red
 4 none          white       yellow
 5 brown         light       brown
 6 brown, grey   light       blue
 7 brown         light       blue
 8 NA            white, red  red
 9 black         light       brown
10 auburn, white fair        blue-gray
# … with 77 more rows
# ℹ Use `print(n = ...)` to see more rows
```

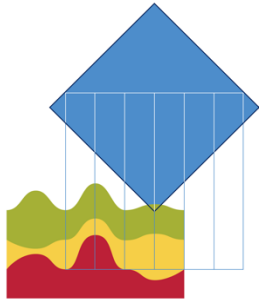


The five verbs of `dplyr – select()`

- `select()` also contains some helper functions to make your life easier

- To select all variables that contain an underscore (“_”), use the helper `contains()`

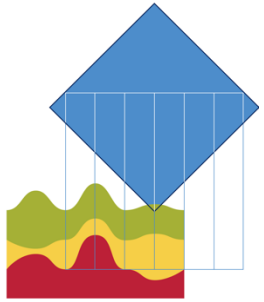
```
select(starwars, contains("_"))
```



The five verbs of `dplyr` – `select()`

```
select(starwars, contains("_"))
```

```
> select(starwars, contains("_"))
# A tibble: 87 × 4
   hair_color    skin_color  eye_color birth_year
   <chr>         <chr>       <chr>           <dbl>
 1 blond         fair        blue             19
 2 NA            gold        yellow          112
 3 NA            white, blue red              33
 4 none          white       yellow           41.9
 5 brown         light       brown            19
 6 brown, grey   light       blue             52
 7 brown         light       blue             47
 8 NA            white, red  red              NA
 9 black         light       brown            24
10 auburn, white fair        blue-gray        57
# … with 77 more rows
# ℹ Use `print(n = ...)` to see more rows
```



The five verbs of `dplyr – select()`

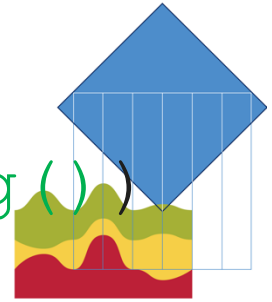
- `select()` also contains some helper functions to make your life easier

- To select all variables that contain an underscore ("_"), use the helper `contains()`

```
select(starwars, contains("_"))
```

- You can use select to re-order variables using the helper `everything()`

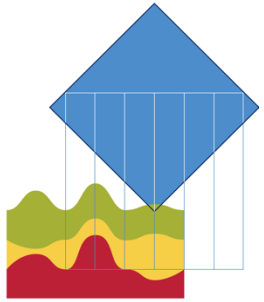
```
select(starwars, name, birth_year, everything())
```



The five verbs of `dplyr` – `select()`

```
select(starwars, name, birth_year,
       everything())
```

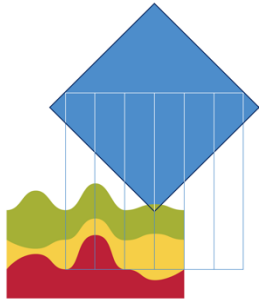
```
> select(starwars, name, birth_year, everything())
# A tibble: 87 × 14
   name    birth…¹ height  mass hair_…² skin_…³ eye_c…⁴ sex   gender homew…⁵ species
   <chr>     <dbl>  <int> <dbl> <chr>   <chr>   <chr>   <chr> <chr> <chr>   <chr>
 1 Luke S…    19      172    77 blond   fair    blue    male  mascu… Tatooi… Human
 2 C-3PO     112      167    75 NA      gold    yellow  none  mascu… Tatooi… Droid
 3 R2-D2      33       96    32 NA      white,… red     none  mascu… Naboo   Droid
 4 Darth …    41.9    202   136 none    white   yellow  male  mascu… Tatooi… Human
 5 Leia O…    19      150    49 brown   light   brown   fema… femin… Aldera… Human
 6 Owen L…    52      178   120 brown,… light   blue    male  mascu… Tatooi… Human
 7 Beru W…    47      165    75 brown   light   blue    fema… femin… Tatooi… Human
 8 R5-D4      NA       97    32 NA      white,… red     none  mascu… Tatooi… Droid
 9 Biggs …    24      183    84 black   light   brown   male  mascu… Tatooi… Human
10 Obi-Wa…    57      182    77 auburn… fair    blue-g… male  mascu… Stewjon Human
# … with 77 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹birth_year, ²hair_color,
#   ³skin_color, ⁴eye_color, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



How to pattern-match in `filter()`:

```
filter(starwars,str_detect(name, pattern =
                           "ei"))
```

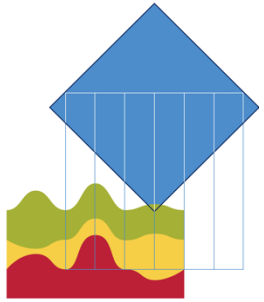
```
> filter(starwars,str_detect(name, pattern = "ei"))
# A tibble: 1 × 14
  name      height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
  <chr>      <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
1 Leia Or…     150    49 brown   light   brown        19 fema… femin… Aldera… Human
# … with 3 more variables: films <list>, vehicles <list>, starships <list>, and
#   abbreviated variable names ¹hair_color, ²skin_color, ³eye_color, ⁴birth_year,
#   ⁵homeworld
# ℹ Use `colnames()` to see all variable names
```



The five verbs of `dplyr – mutate()`

- `mutate()` can be used to create new variables from existing ones
- The new variable will appear at the end of the tibble
- Standard syntax:

```
mutate(data, new_variable_name=calculation)
```

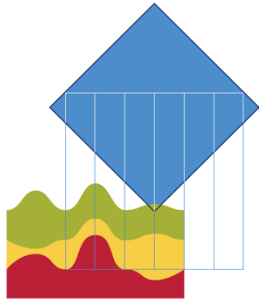


The five verbs of `dplyr – mutate()`

- Say for example you wanted to calculate the weight for all the starwars individuals on the surface of the earth

- To do this you need to multiply their mass (in kg) with 9.8 m/s^2

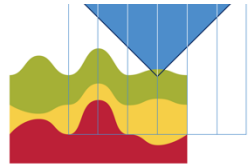
```
mutate(starwars, weight=mass*9.8)
```



The five verbs of `dplyr` – `mutate()`

```
mutate(starwars, weight=mass*9.8)
```

```
> mutate(starwars, weight=mass*9.8)
# A tibble: 87 × 15
   name               height  mass hair_color    skin_color  eye_color birth_year sex    gender  homew…¹ species films vehic…² stars…³ weight
   <chr>               <int> <dbl> <chr>         <chr>       <chr>           <dbl> <chr>  <chr>   <chr>   <chr>   <lis> <list>  <list>    <dbl>
 1 Luke Skywalker        172    77 blond         fair        blue             19   male   mascul… Tatooi… Human   <chr> <chr>   <chr>     755.
 2 C-3PO                 167    75 NA            gold        yellow          112   none   mascul… Tatooi… Droid   <chr> <chr>   <chr>     735
 3 R2-D2                  96    32 NA            white, blue red              33   none   mascul… Naboo   Droid   <chr> <chr>   <chr>     314.
 4 Darth Vader           202   136 none          white       yellow           41.9 male   mascul… Tatooi… Human   <chr> <chr>   <chr>    1333.
 5 Leia Organa           150    49 brown         light       brown            19   female femini… Aldera… Human   <chr> <chr>   <chr>     480.
 6 Owen Lars             178   120 brown, grey   light       blue             52   male   mascul… Tatooi… Human   <chr> <chr>   <chr>    1176
 7 Beru Whitesun lars    165    75 brown         light       blue             47   female femini… Tatooi… Human   <chr> <chr>   <chr>     735
 8 R5-D4                  97    32 NA            white, red  red              NA   none   mascul… Tatooi… Droid   <chr> <chr>   <chr>     314.
 9 Biggs Darklighter     183    84 black         light       brown            24   male   mascul… Tatooi… Human   <chr> <chr>   <chr>     823.
10 Obi-Wan Kenobi        182    77 auburn, white fair        blue-gray        57   male   mascul… Stewjon Human   <chr> <chr>   <chr>     755.
# … with 77 more rows, and abbreviated variable names ¹homeworld, ²vehicles, ³starships
# ℹ Use `print(n = ...)` to see more rows
```



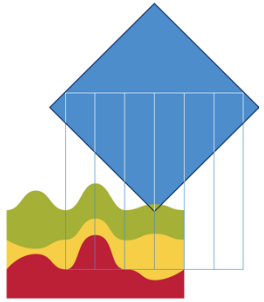
The five verbs of `dplyr` – `mutate()`

- You could also use a conditional statement for the new variable using the `ifelse()` helper function:

```
ifelse(condition, outcome if condition is TRUE,
       outcome if condition is FALSE)
```

- For example, if you wanted to create a variable separating all characters into Droids and Not Droids:

```
mutate(starwars,
is_it_a_droid=ifelse(species=="Droid", "YES",
                     "NO"))
```



The five verbs of dplyr – mutate()

```
mutate(starwars, is_it_a_droid=
ifelse(species=="Droid", "YES", "NO"))
```

```
> mutate(starwars, is_it_a_droid= ifelse(species=="Droid", "YES", "NO"))
# A tibble: 87 × 15
   name               height  mass hair_color    skin_color  eye_color birth…¹ sex   gender homew…² species films vehic…³ stars…⁴ is_it…⁵
   <chr>               <int> <dbl> <chr>         <chr>       <chr>       <dbl> <chr> <chr>  <chr>   <chr>   <lis> <list>  <list>  <chr>
 1 Luke Skywalker        172    77 blond         fair        blue         19   male  mascu… Tatooi… Human   <chr> <chr>   <chr>   NO
 2 C-3PO                 167    75 NA            gold        yellow      112   none  mascu… Tatooi… Droid   <chr> <chr>   <chr>   YES
 3 R2-D2                  96    32 NA            white, blue red          33   none  mascu… Naboo   Droid   <chr> <chr>   <chr>   YES
 4 Darth Vader           202   136 none          white       yellow       41.9 male  mascu… Tatooi… Human   <chr> <chr>   <chr>   NO
 5 Leia Organa           150    49 brown         light       brown        19   fema… femin… Aldera… Human   <chr> <chr>   <chr>   NO
 6 Owen Lars             178   120 brown, grey   light       blue         52   male  mascu… Tatooi… Human   <chr> <chr>   <chr>   NO
 7 Beru Whitesun lars    165    75 brown         light       blue         47   fema… femin… Tatooi… Human   <chr> <chr>   <chr>   NO
 8 R5-D4                  97    32 NA            white, red  red          NA   none  mascu… Tatooi… Droid   <chr> <chr>   <chr>   YES
 9 Biggs Darklighter     183    84 black         light       brown        24   male  mascu… Tatooi… Human   <chr> <chr>   <chr>   NO
10 Obi-Wan Kenobi        182    77 auburn, white fair        blue-gray    57   male  mascu… Stewjon Human   <chr> <chr>   <chr>   NO
# … with 77 more rows, and abbreviated variable names ¹birth_year, ²homeworld, ³vehicles, ⁴starships, ⁵is_it_a_droid
# ℹ Use `print(n = ...)` to see more rows
```

The five verbs of `dplyr` – `summarise()`

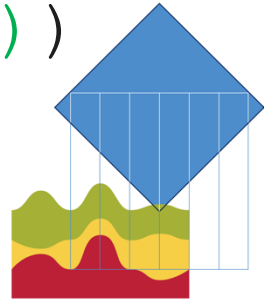
- `summarise()` can be used to apply standard functions, such as the mean or standard deviation, on your dataset

- Standard syntax:

```
summarise(data, function(variable))
```

- For example, to work out the mean height for the whole dataset

```
summarise(starwars, mean(height, na.rm=TRUE))
```

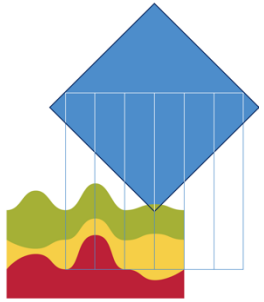


The five verbs of `dplyr` – `summarise()`

```
summarise(starwars, mean(height, na.rm=TRUE))
```

```
> summarise(starwars, mean(height, na.rm=TRUE))
# A tibble: 1 × 1
  `mean(height, na.rm = TRUE)`
                         <dbl>
1                         174.

> summarise(starwars, mean(height))
# A tibble: 1 × 1
  `mean(height)`
           <dbl>
1             NA
```



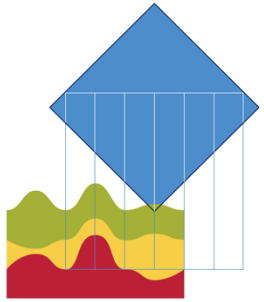
The five verbs of `dplyr` – `summarise()`

- `summarise()` becomes much more interesting and useful when combined with the `group_by()` function
- `group_by()` groups together items on which you want to apply specific functions

```
group_by(data, variable to group by)
```

- For example, if you wanted to work out the mean height for each species in the `starwars` dataset:

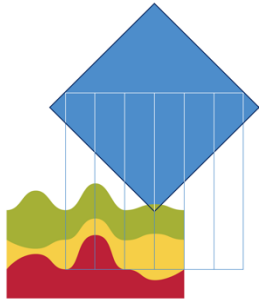
```
summarise(group_by(starwars, species),
          mean(height, na.rm=TRUE))
```



The five verbs of `dplyr` – `summarise()`

```
summarise(group_by(starwars, species),
          mean(height, na.rm=TRUE))
```

```
> summarise(group_by(starwars, species), mean(height, na.rm=TRUE))
# A tibble: 38 × 2
   species    `mean(height, na.rm = TRUE)`
   <chr>                             <dbl>
 1 Aleena                             79
 2 Besalisk                          198
 3 Cerean                            198
 4 Chagrian                          196
 5 Clawdite                          168
 6 Droid                             131.
 7 Dug                               112
 8 Ewok                               88
 9 Geonosian                         183
10 Gungan                            209.
# … with 28 more rows
# ℹ Use `print(n = ...)` to see more rows
```



The five verbs of `dplyr` – `arrange()`

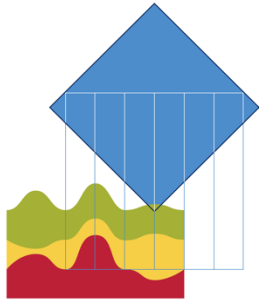
- `arrange()` sorts your dataset either numerically or alphabetically based on the selected variable

- Standard syntax:

```
arrange(data, variable)
```

- For example, arrange `starwars` by height, or eye colour

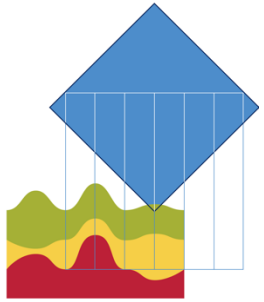
```
arrange(starwars, height)
arrange(starwars, eye_color)
```



The five verbs of `dplyr` – `arrange()`

```
arrange(starwars, height)
```

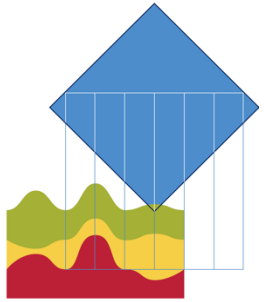
```
> arrange(starwars, height)
# A tibble: 87 × 14
   name    height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>    <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
 1 Yoda        66    17 white   green   brown       896 male  mascu… NA      Yoda's…
 2 Ratts …     79    15 none    grey, … unknown      NA male  mascu… Aleen … Aleena
 3 Wicket…     88    20 brown   brown   brown         8 male  mascu… Endor   Ewok
 4 Dud Bo…     94    45 none    blue, … yellow       NA male  mascu… Vulpter Vulpte…
 5 R2-D2       96    32 NA      white,… red          33 none  mascu… Naboo   Droid
 6 R4-P17      96    NA none    silver… red, b…      NA none  femin… NA      Droid
 7 R5-D4       97    32 NA      white,… red          NA none  mascu… Tatooi… Droid
 8 Sebulba    112    40 none    grey, … orange       NA male  mascu… Malast… Dug
 9 Gasgano    122    NA none    white,… black        NA male  mascu… Troiken Xexto
10 Watto      137    NA black   blue, … yellow       NA male  mascu… Toydar… Toydar…
# … with 77 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



The five verbs of `dplyr` – `arrange()`

```
arrange(starwars, eye_color)
```

```
> arrange(starwars, eye_color)
# A tibble: 87 × 14
   name     height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>     <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
 1 Greedo      173    74 NA      green   black        44 male  mascu… Rodia   Rodian
 2 Nien N…     160    68 none    grey    black        NA male  mascu… Sullust Sullus…
 3 Gasgano     122    NA none    white,… black        NA male  mascu… Troiken Xexto
 4 Kit Fi…     196    87 none    green   black        NA male  mascu… Glee A… Nautol…
 5 Plo Ko…     188    80 none    orange  black        22 male  mascu… Dorin   Kel Dor
 6 Lama Su     229    88 none    grey    black        NA male  mascu… Kamino  Kamino…
 7 Taun We     213    NA none    grey    black        NA fema… femin… Kamino  Kamino…
 8 Shaak …     178    57 none    red, b… black        NA fema… femin… Shili   Togruta
 9 Tion M…     206    80 none    grey    black        NA male  mascu… Utapau  Pau'an
10 BB8          NA    NA none    none    black        NA none  mascu… NA      Droid
# … with 77 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

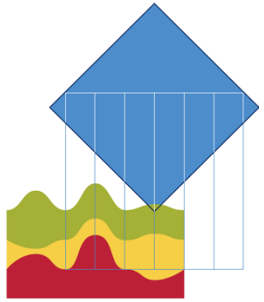


The five verbs of `dplyr` – `arrange()`

- Normally it goes from smallest value to greatest but you can reverse that using the helper `desc()`

```
arrange(starwars, desc(height))
```

```
arrange(starwars, desc(skin_color))
```



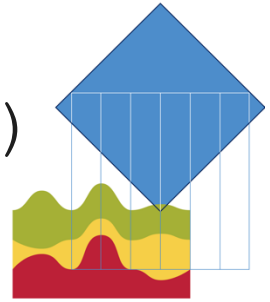
The five verbs of `dplyr – arrange()`

- You can also arrange using multiple columns, but be careful: order matters!

```
arrange(starwars,skin_color,eye_color,mass)
```

is NOT equal to

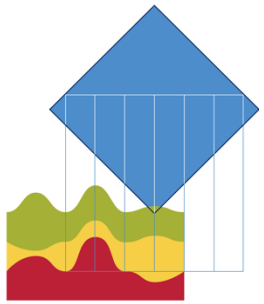
```
arrange(starwars,eye_color,skin_color,mass)
```



The five verbs of `dplyr` – `arrange()`

```
arrange(starwars,skin_color,eye_color,mass)
```

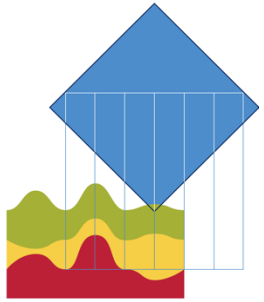
```
> arrange(starwars,skin_color,eye_color,mass)
# A tibble: 87 × 14
   name    height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>    <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
 1 Mas Am…    196    NA none    blue    blue         NA male  mascu… Champa… Chagri…
 2 Ayla S…    178    55 none    blue    hazel        48 fema… femin… Ryloth  Twi'lek
 3 Dud Bo…     94    45 none    blue, … yellow       NA male  mascu… Vulpter Vulpte…
 4 Watto      137    NA black   blue, … yellow       NA male  mascu… Toydar… Toydar…
 5 Tarfful    234   136 brown   brown   blue         NA male  mascu… Kashyy… Wookiee
 6 Wicket…     88    20 brown   brown   brown         8 male  mascu… Endor   Ewok
 7 Eeth K…    171    NA black   brown   brown        NA male  mascu… Iridon… Zabrak
 8 Dexter…    198   102 none    brown   yellow       NA male  mascu… Ojom    Besali…
 9 Ackbar     180    83 none    brown … orange       41 male  mascu… Mon Ca… Mon Ca…
10 Grievo…    216   159 none    brown,… green,…      NA male  mascu… Kalee   Kaleesh
# … with 77 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



The five verbs of `dplyr` – `arrange()`

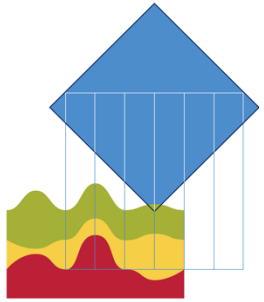
```
arrange(starwars,eye_color,skin_color,mass)
```

```
> arrange(starwars,eye_color,skin_color,mass)
# A tibble: 87 × 14
   name    height  mass hair_…¹ skin_…² eye_c…³ birth…⁴ sex   gender homew…⁵ species
   <chr>    <int> <dbl> <chr>   <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
 1 Greedo     173    74 NA      green   black        44 male  mascu… Rodia   Rodian
 2 Kit Fi…    196    87 none    green   black        NA male  mascu… Glee A… Nautol…
 3 Nien N…    160    68 none    grey    black        NA male  mascu… Sullust Sullus…
 4 Tion M…    206    80 none    grey    black        NA male  mascu… Utapau  Pau'an
 5 Lama Su    229    88 none    grey    black        NA male  mascu… Kamino  Kamino…
 6 Taun We    213    NA none    grey    black        NA fema… femin… Kamino  Kamino…
 7 BB8         NA    NA none    none    black        NA none  mascu… NA      Droid
 8 Plo Ko…    188    80 none    orange  black        22 male  mascu… Dorin   Kel Dor
 9 Shaak …    178    57 none    red, b… black        NA fema… femin… Shili   Togruta
10 Gasgano    122    NA none    white,… black        NA male  mascu… Troiken Xexto
# … with 77 more rows, 3 more variables: films <list>, vehicles <list>,
#   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,
#   ³eye_color, ⁴birth_year, ⁵homeworld
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```



Piping

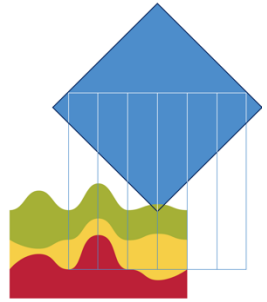
- Initially introduced through the magrittr package, the main pipe is now part of base R
- The main pipe is this composite symbol: %>%
- It essentially feeds the output from the left of the pipe to the input on the right



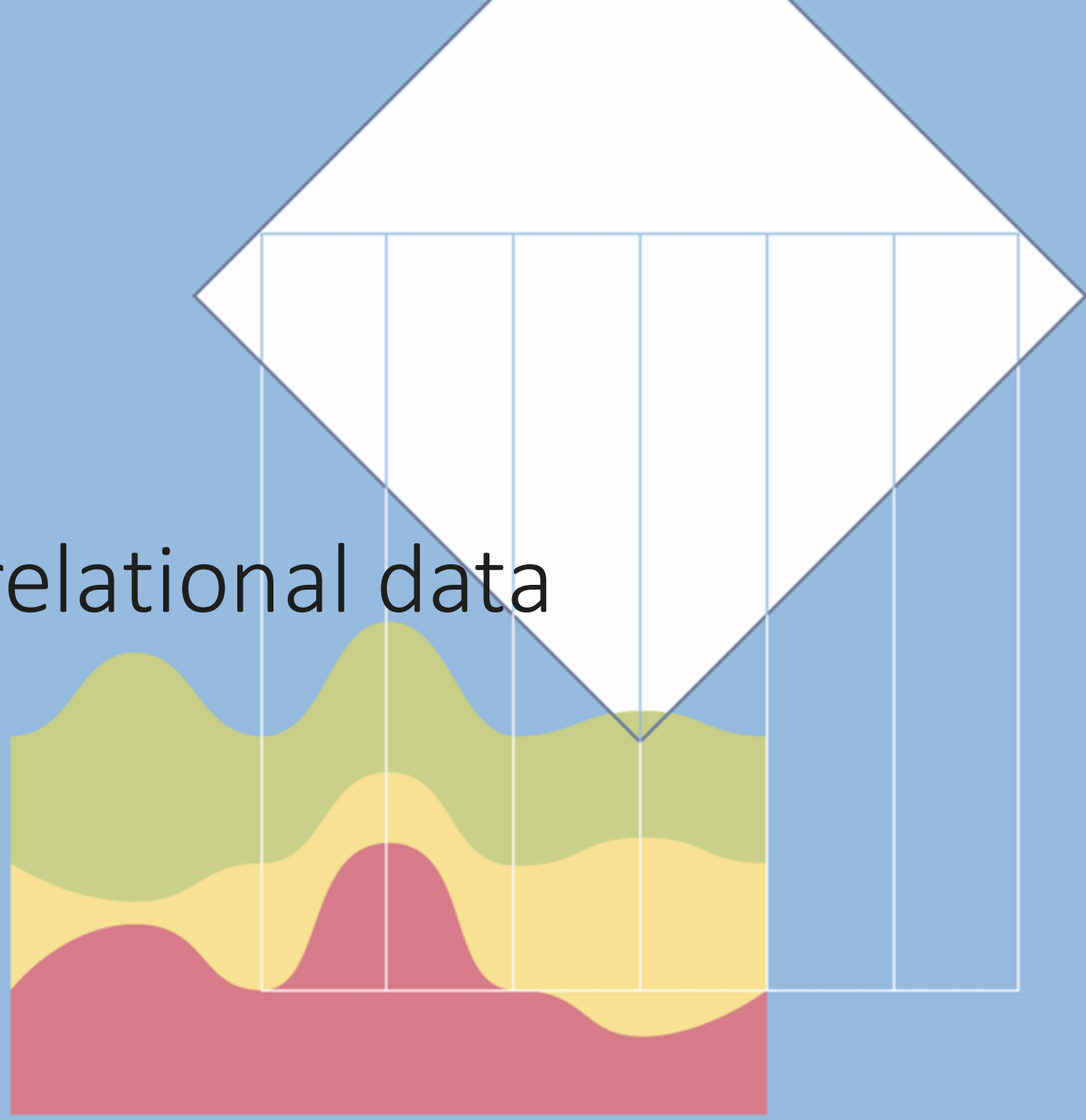
Piping

```
starwars %>%
    filter(eye_color=="red") %>%
    filter(species =="Droid")
```


```
> starwars %>%
+   filter(eye_color=="red") %>%
+   filter(species =="Droid")
# A tibble: 3 × 14
  name  height  mass hair_color skin_…¹ eye_c…² birth…³ sex   gender homew…⁴ species
  <chr>  <int> <dbl> <chr>      <chr>   <chr>     <dbl> <chr> <chr>  <chr>   <chr>
1 R2-D2     96    32 NA         white,… red          33 none  mascu… Naboo   Droid
2 R5-D4     97    32 NA         white,… red          NA none  mascu… Tatooi… Droid
3 IG-88    200   140 none       metal   red          15 none  mascu… NA      Droid
# … with 3 more variables: films <list>, vehicles <list>, starships <list>, and
#   abbreviated variable names ¹skin_color, ²eye_color, ³birth_year, ⁴homeworld
# ℹ Use `colnames()` to see all variable names
```



Tidy and relational data

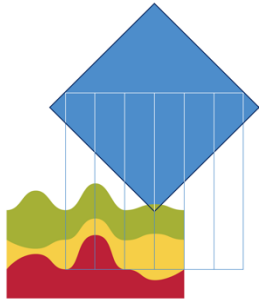


OXFORD UNIVERSITY STATISTICAL CONSULTING

Tidy data – what are they?

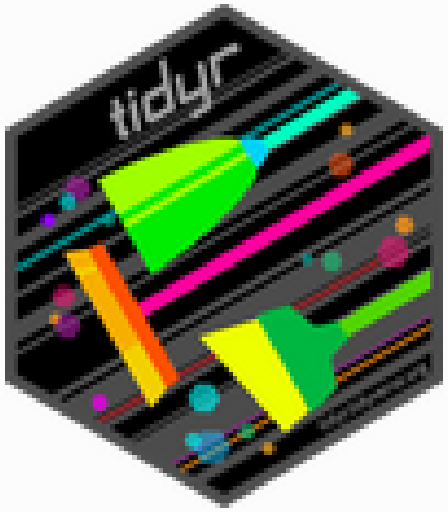
“Tidy datasets are all alike, but every messy dataset is messy in its own way.”

Hadley Wickham



Tidy data – what are they?

- Tidy data are easier to manipulate through the tidyverse
- They are also much easier to plot using ggplot2
- The idea behind tidy data is to have one column per variable, one row per observation, and one value per cell
- Often the rule of “one column per variable” is the one that breaks down



Mmmmmeeeeeeesssssss……

- Dataset billboard in tidyr
- Song rankings top 100 for 2000
- Weeks 1 to 76 are presented as separate entries

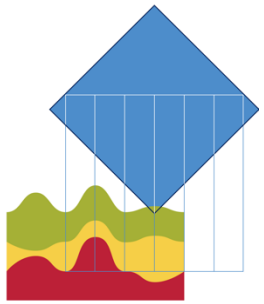
Filter | Cols: « < 1 - 50 > »

	artist	track	date.entered	wk1	wk2	wk3	wk4	wk5	wk6	wk7	wk8	wk9	wk10	wk11	wk12	wk13
1	2 Pac	Baby Don't Cry (Keep...	2000-02-26	87	82	72	77	87	94	99	NA	NA	NA	NA	NA	NA
2	2Ge+her	The Hardest Part Of ...	2000-09-02	91	87	92	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
3	3 Doors Down	Kryptonite	2000-04-08	81	70	68	67	66	57	54	53	51	51	51	51	47
4	3 Doors Down	Loser	2000-10-21	76	76	72	69	67	65	55	59	62	61	61	59	61
5	504 Boyz	Wobble Wobble	2000-04-15	57	34	25	17	17	31	36	49	53	57	64	70	75
6	98^0	Give Me Just One Nig...	2000-08-19	51	39	34	26	26	19	2	2	3	6	7	22	29
7	A*Teens	Dancing Queen	2000-07-08	97	97	96	95	100	NA	NA	NA	NA	NA	NA	NA	NA
8	Aaliyah	I Don't Wanna	2000-01-29	84	62	51	41	38	35	35	38	38	36	37	37	38
9	Aaliyah	Try Again	2000-03-18	59	53	38	28	21	18	16	14	12	10	9	8	6

Let’s get tidying with pivot_longer

```
library(tidyverse)
View(billboard)
billboard %>%
  pivot_longer(wk1:wk76, names_to = "Week", values_to = "Chart position")
```

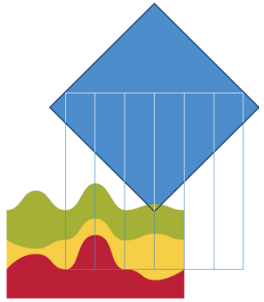
	artist	track	date.entered	Week	Chart position
1	2 Pac	Baby Don't Cry (Keep...	2000-02-26	wk1	87
2	2 Pac	Baby Don't Cry (Keep...	2000-02-26	wk2	82
3	2 Pac	Baby Don't Cry (Keep...	2000-02-26	wk3	72
4	2 Pac	Baby Don't Cry (Keep...	2000-02-26	wk4	77
5	2 Pac	Baby Don't Cry (Keep...	2000-02-26	wk5	87
6	2 Pac	Baby Don't Cry (Keep...	2000-02-26	wk6	94
7	2 Pac	Baby Don't Cry (Keep...	2000-02-26	wk7	99



A little bit of data wrangling…

```
billboard %>%
  pivot_longer(wk1:wk76, names_to = "Week", values_to = "Chart position") %>%
  mutate(Week=str_sub(Week, start = 3, end=-1)) %>%
  View()
```

	artist	track	date.entered	Week	Chart position
1	2 Pac	Baby Don't Cry (Keep...	2000-02-26	1	87
2	2 Pac	Baby Don't Cry (Keep...	2000-02-26	2	82
3	2 Pac	Baby Don't Cry (Keep...	2000-02-26	3	72
4	2 Pac	Baby Don't Cry (Keep...	2000-02-26	4	77
5	2 Pac	Baby Don't Cry (Keep...	2000-02-26	5	87
6	2 Pac	Baby Don't Cry (Keep...	2000-02-26	6	94
7	2 Pac	Baby Don't Cry (Keep...	2000-02-26	7	99
8	2 Pac	Baby Don't Cry (Keep...	2000-02-26	8	*NA*
9	2 Pac	Baby Don't Cry (Keep...	2000-02-26	9	*NA*
10	2 Pac	Baby Don't Cry (Keep...	2000-02-26	10	*NA*



Turn back please!

- To reverse this, we can use the function pivot_wider()

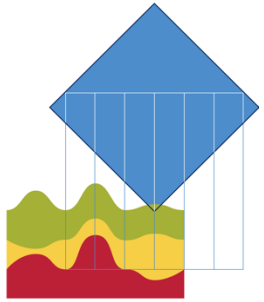
```
billboard %>%
  pivot_longer(wk1:wk76, names_to = "Week", values_to = "Chart position") %>%
  mutate(Week=str_sub(Week, start = 3, end=-1)) %>%
  mutate(Week=str_c("wk",Week)) %>%
  pivot_wider(names_from = Week, values_from = `Chart position`)
```

Filter | Cols: « < 1 - 50 > »

	artist	track	date.entered	wk1	wk2	wk3	wk4	wk5	wk6	wk7
1	2 Pac	Baby Don't Cry (Keep...	2000-02-26	87	82	72	77	87	94	99
2	2Ge+her	The Hardest Part Of ...	2000-09-02	91	87	92	*NA*	*NA*	*NA*	*NA*
3	3 Doors Down	Kryptonite	2000-04-08	81	70	68	67	66	57	54
4	3 Doors Down	Loser	2000-10-21	76	76	72	69	67	65	55

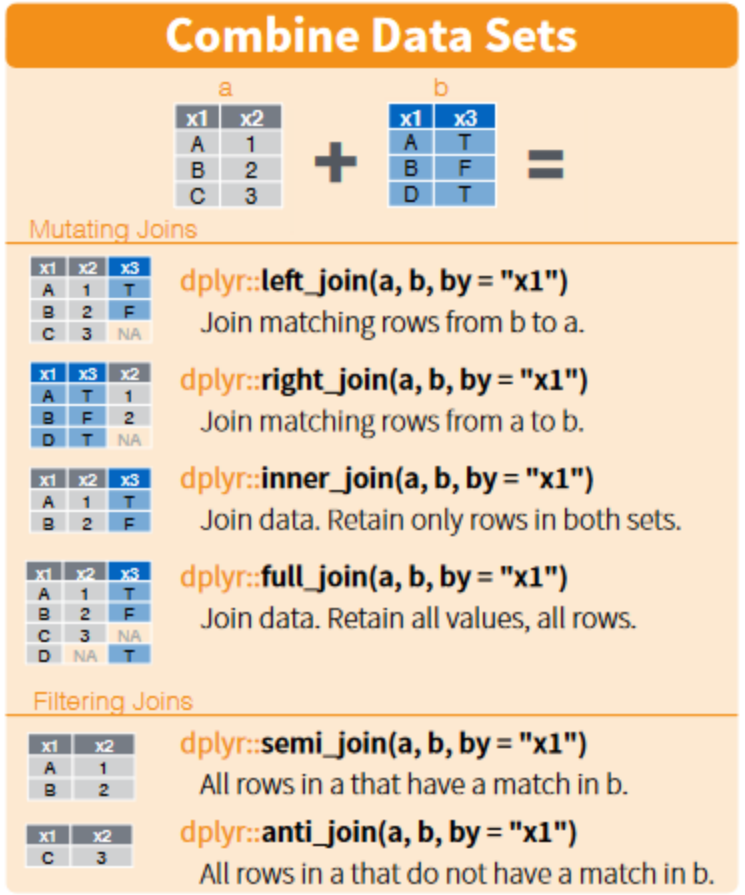
Relational data

- Often you will have data stored in multiple tibbles
- If the entries between each tibble are related in some way then these are collectively referred to as “relational data”
- In those situations, the data wrangling question becomes:
  - “How do you collect information from multiple tibbles in one place?”



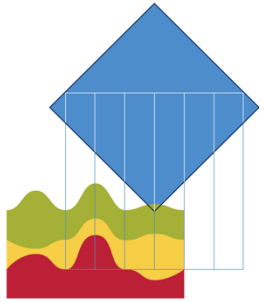
Relational data

- The tidyverse has a small and powerful collection of functions for relational data
- The tools available are split into joins and set operations – here we will be focusing on joins
- Joins do exactly what their name suggests - they join tibbles



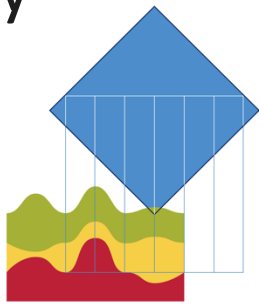
Relational data

- Joins can be split into
  - filtering joins (think dplyr filter()) and
  - mutating joins (think dplyr mutate())
- Filtering joins filter data from one tibble based on whether the samples are present in another tibble
- Mutating joins add variables to one tibble after matching observations with another



Filtering joins

- Filtering joins filter data from one tibble based on whether the samples are present in the other tibble.
- The two filtering joins we will be studying are semi_join and anti_join()
  - semi_join(x, y) keeps all observations in x that have a match in y
  - anti_join(x, y) drops all observations in x that have a match in y



Filtering joins – example

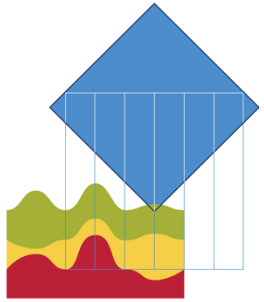
- band_instruments contains instrument information for three musicians

Filter

	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

- band_members contains the band name for three musicians

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles



Filtering joins – example

- semi_join(x, y) keeps all observations in x that have a match in y

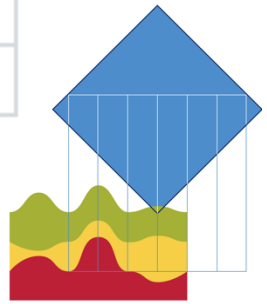
```
semi_join(band_instruments,band_members, by="name") %>%View()
```

Filter

	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles

	name	plays
1	John	guitar
2	Paul	bass



Filtering joins – example

- semi_join(x, y) keeps all observations in x that have a match in y

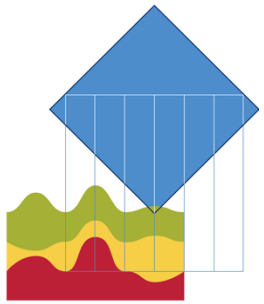
```
semi_join(band_members, band_instruments, by="name") %>%View()
```

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles

Filter

	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

	name	band
1	John	Beatles
2	Paul	Beatles



Filtering joins – example

- anti_join(x, y) drops all observations in x that have a match in y

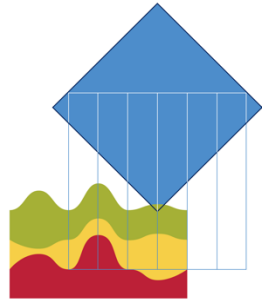
```
anti_join(band_instruments,band_members, by="name") %>%View()
```

Filter

	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles

	name	plays
1	Keith	guitar



Filtering joins – example

- anti_join(x, y) drops all observations in x that have a match in y

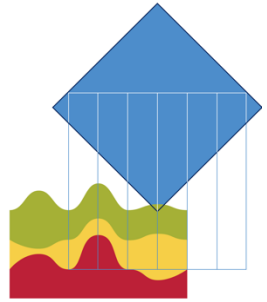
```
anti_join(band_members, band_instruments, by="name") %>%View()
```

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles

Filter

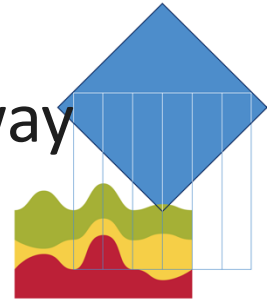
	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

	name	band
1	Mick	Stones



Mutating joins

- Mutating joins add variables from one tibble after matching observations to another.
- The two mutating joins we will be studying are inner_join() and full_join()
  - inner_join(x,y) returns all rows from x that match y, and all columns from x and y
  - full_join(x,y) returns all rows and all columns from both x and y
- There are also left_join and right_join() which work in a similar way



Mutating joins – example

- inner_join(x,y) returns all rows from x that match y, and all columns from x and y

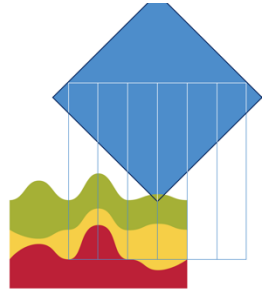
```
inner_join(band_instruments,band_members, by="name") %>%View()
```

Filter

	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles

	name	plays	band
1	John	guitar	Beatles
2	Paul	bass	Beatles



Mutating joins – example

- inner_join(x,y) returns all rows from x that match y, and all columns from x and y

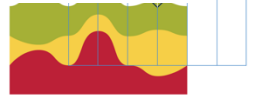
```
inner_join(band_members, band_instruments, by="name") %>%View()
```

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles

Filter

	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

	name	band	plays
1	John	Beatles	guitar
2	Paul	Beatles	bass



Mutating joins – example

- full_join(x,y) returns all rows and all columns from both x and y

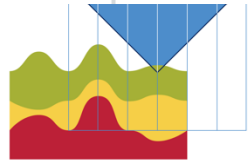
```
full_join(band_instruments,band_members, by="name") %>%View()
```

Filter

	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles

	name	plays	band
1	John	guitar	Beatles
2	Paul	bass	Beatles
3	Keith	guitar	*NA*
4	Mick	*NA*	Stones



Mutating joins – example

- full_join(x,y) returns all rows and all columns from both x and y

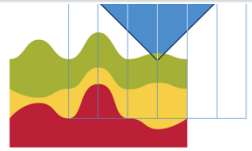
```
full_join(band_members, band_instruments, by="name") %>%View()
```

	name	band
1	Mick	Stones
2	John	Beatles
3	Paul	Beatles

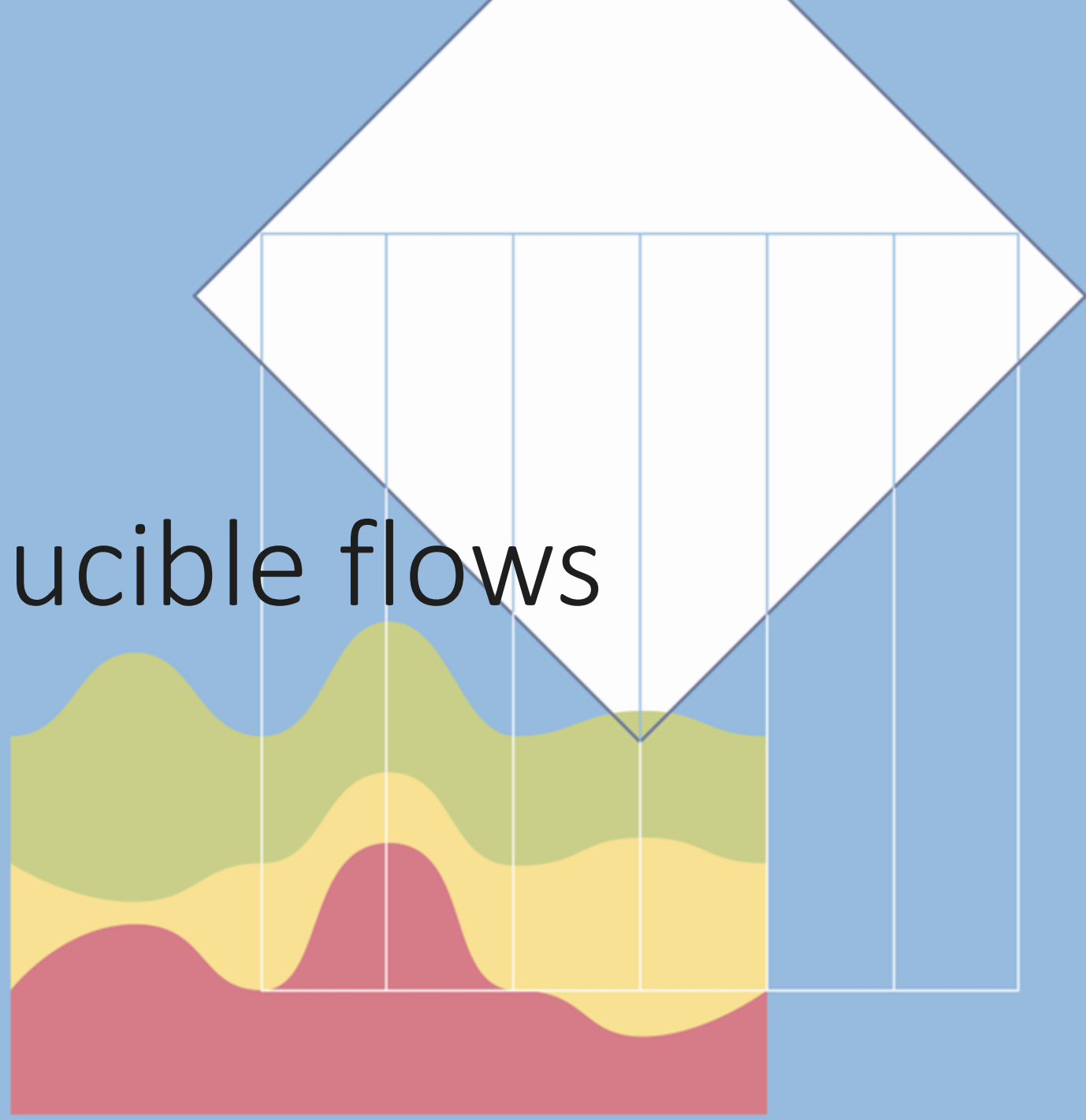
Filter

	name	plays
1	John	guitar
2	Paul	bass
3	Keith	guitar

	name	band	plays
1	Mick	Stones	*NA*
2	John	Beatles	guitar
3	Paul	Beatles	bass
4	Keith	*NA*	guitar



Reproducible flows



OXFORD UNIVERSITY STATISTICAL CONSULTING

Reproducibility: Why does it matter?

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › news feature › article

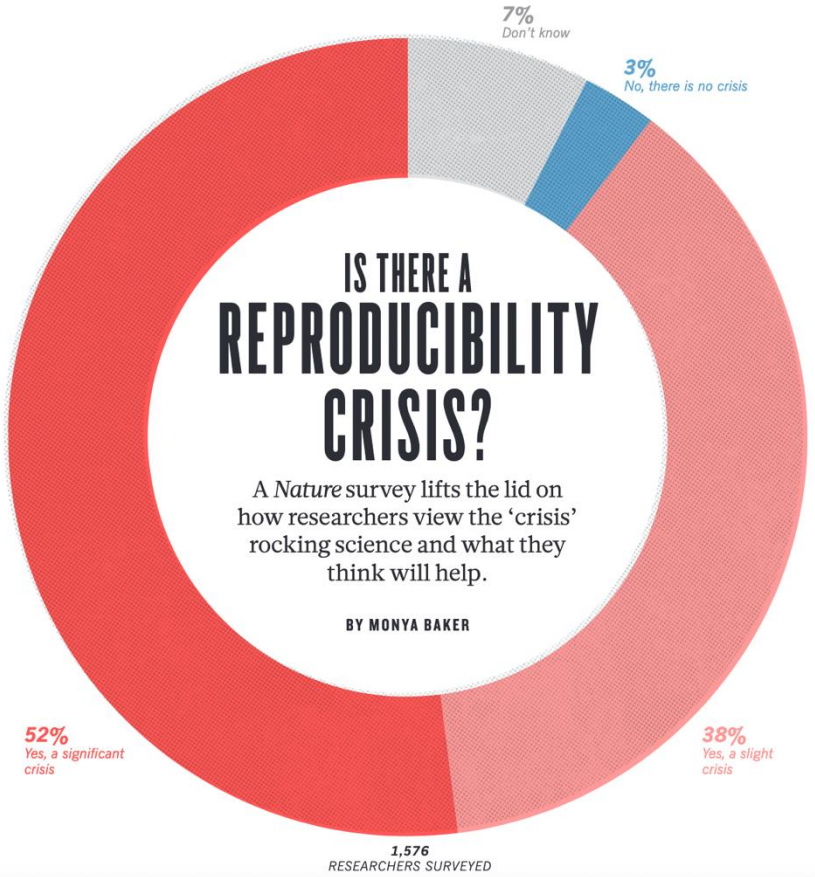
Published: 25 May 2016

1,500 scientists lift the lid on reproducibility

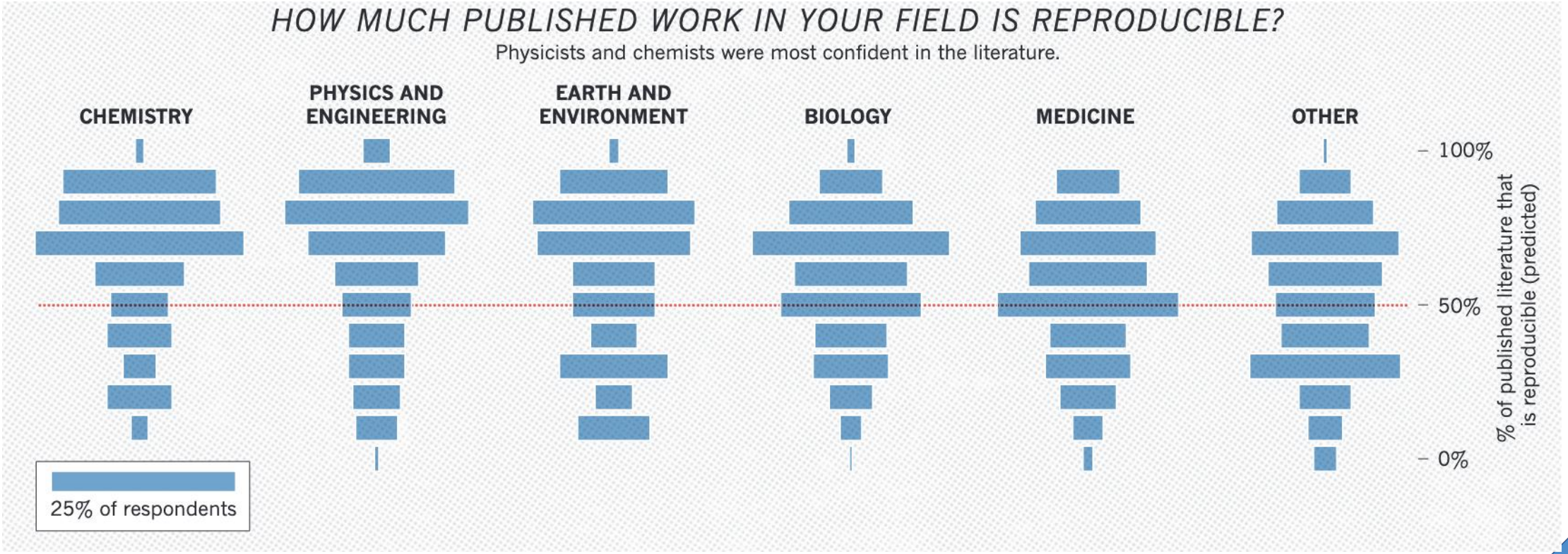
Monya Baker

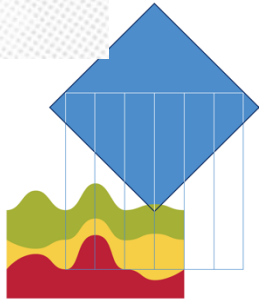
Nature **533**, 452–454 (2016) | Cite this article

106k Accesses | **1808** Citations | **4973** Altmetric | Metrics



Reproducibility: Why does it matter?

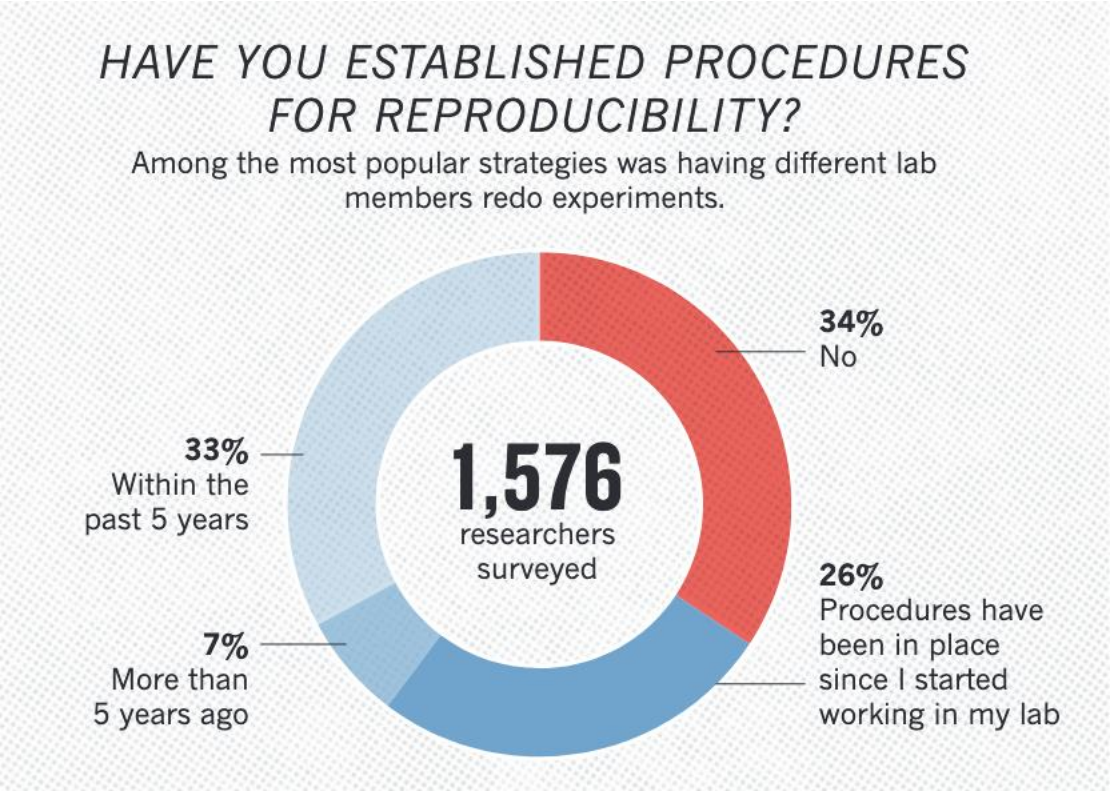




Reproducibility: Why does it matter?

WHAT CAN BE DONE?

Respondents were asked to rate 11 different approaches to improving reproducibility in science, and all got ringing endorsements. Nearly 90% — more than 1,000 people — ticked "More robust experimental design" "better statistics" and "better mentorship". Those ranked higher than the option of providing incentives (such as funding or credit towards tenure) for reproducibility-enhancing practices. But even the lowest-ranked item — journal checklists — won a whopping 69% endorsement.

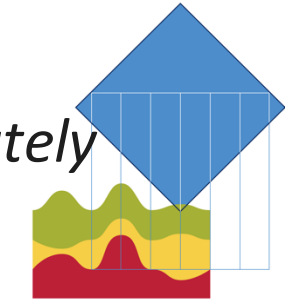


Reproducibility: Data handling and analysis

- Crucial for:
  - Due diligence
  - Reviewing
  - Collaboration
  - Further investigation
  - Helping out your most important and fussy collaborator:

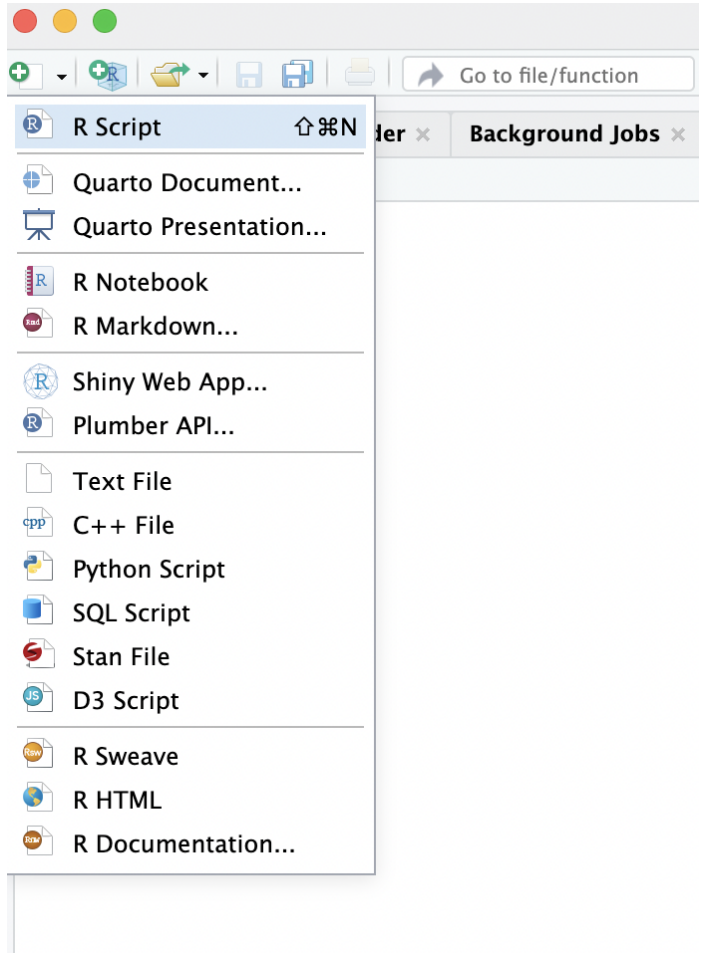
Future you…

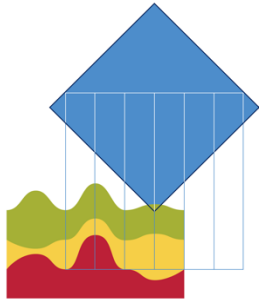
Be kind to your future selves, they are bound to have forgotten absolutely everything



Reproducible flow in R: Starting small

- A script is a text editor that you can write and run code in
- Importantly, you can save it for future use and add comments



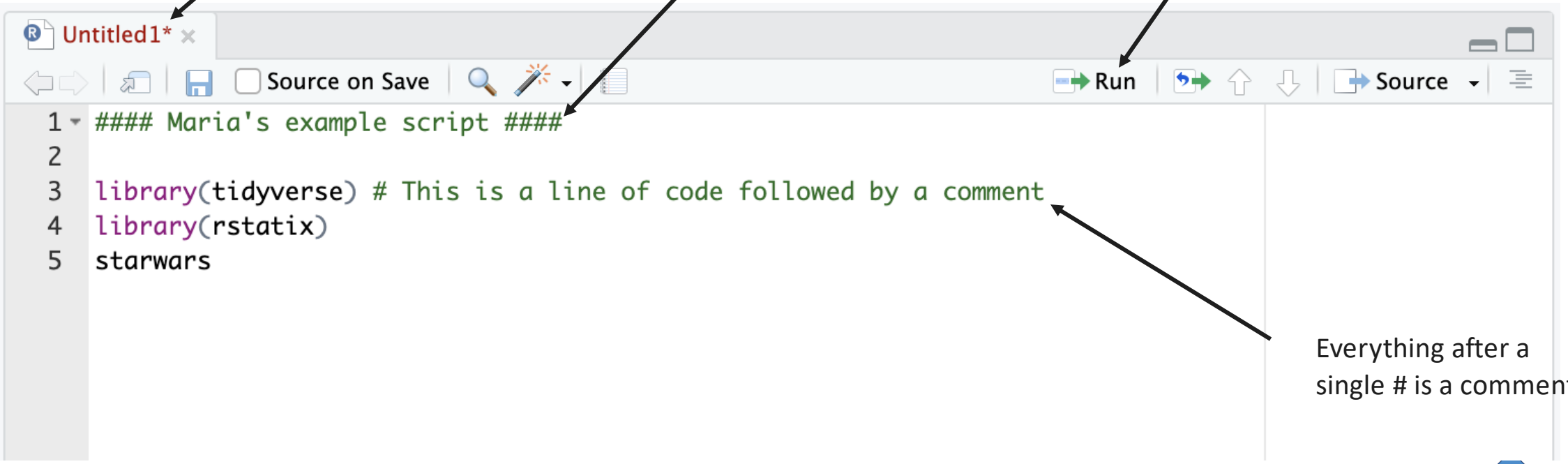


Scripts

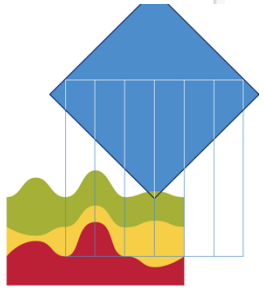
Red name and star means the script has not been saved

Four (or more) # symbols on either side create sections

This button can run either the entire script or a selection



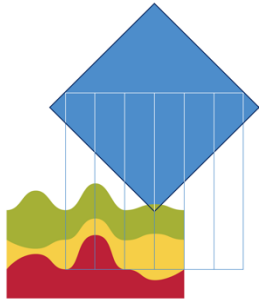
Everything after a single # is a comment



Scripts

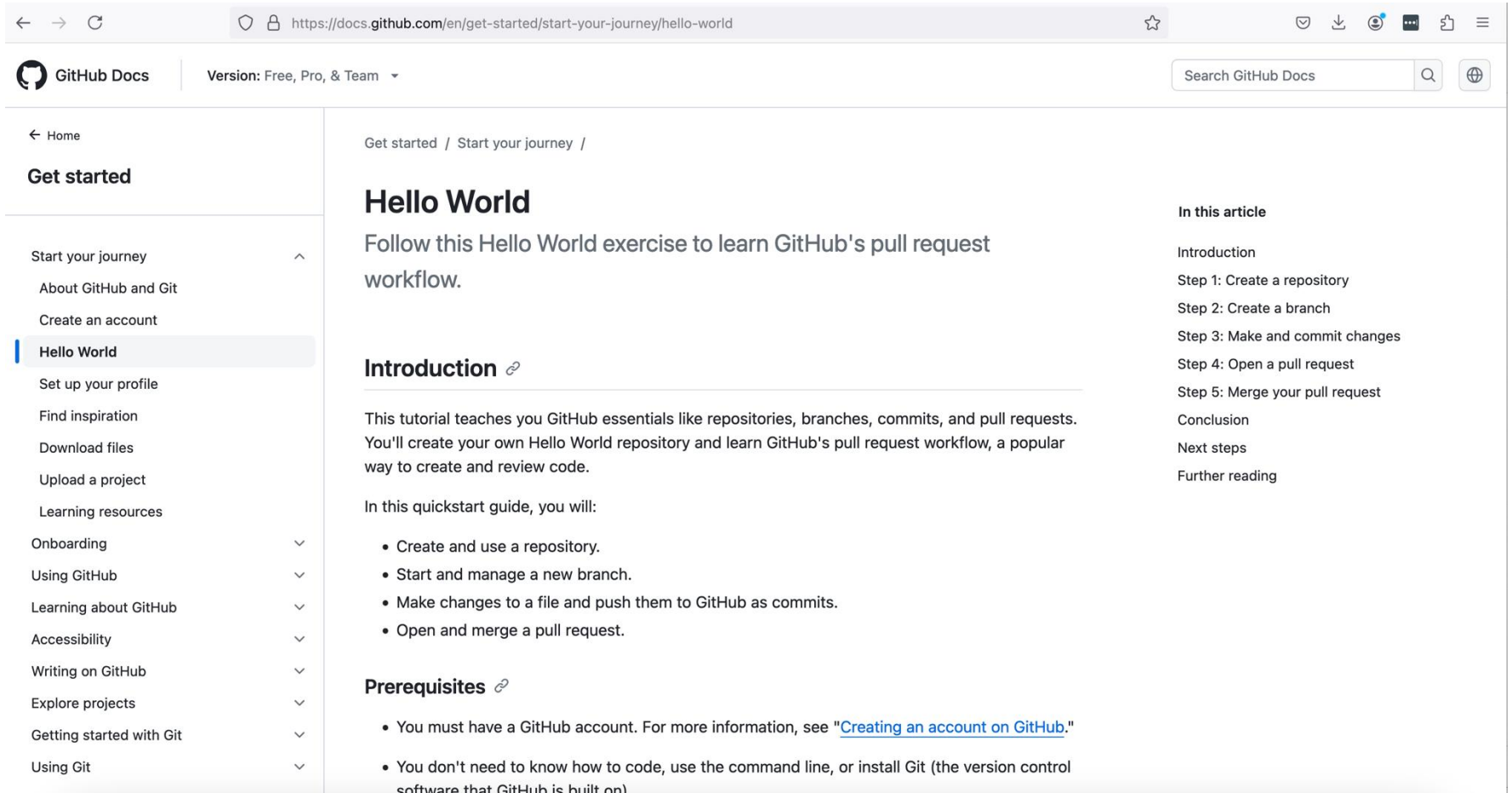
Some good practice tips:

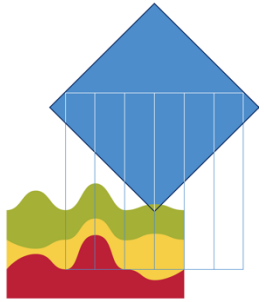
- Regularly (and I mean VERY regularly) save them
- Backup in various locations
- Comment as soon as possible - do not underestimate how quickly you forget why you did things
- Comment with sufficient detail
- Use sections for easier navigation



Or…

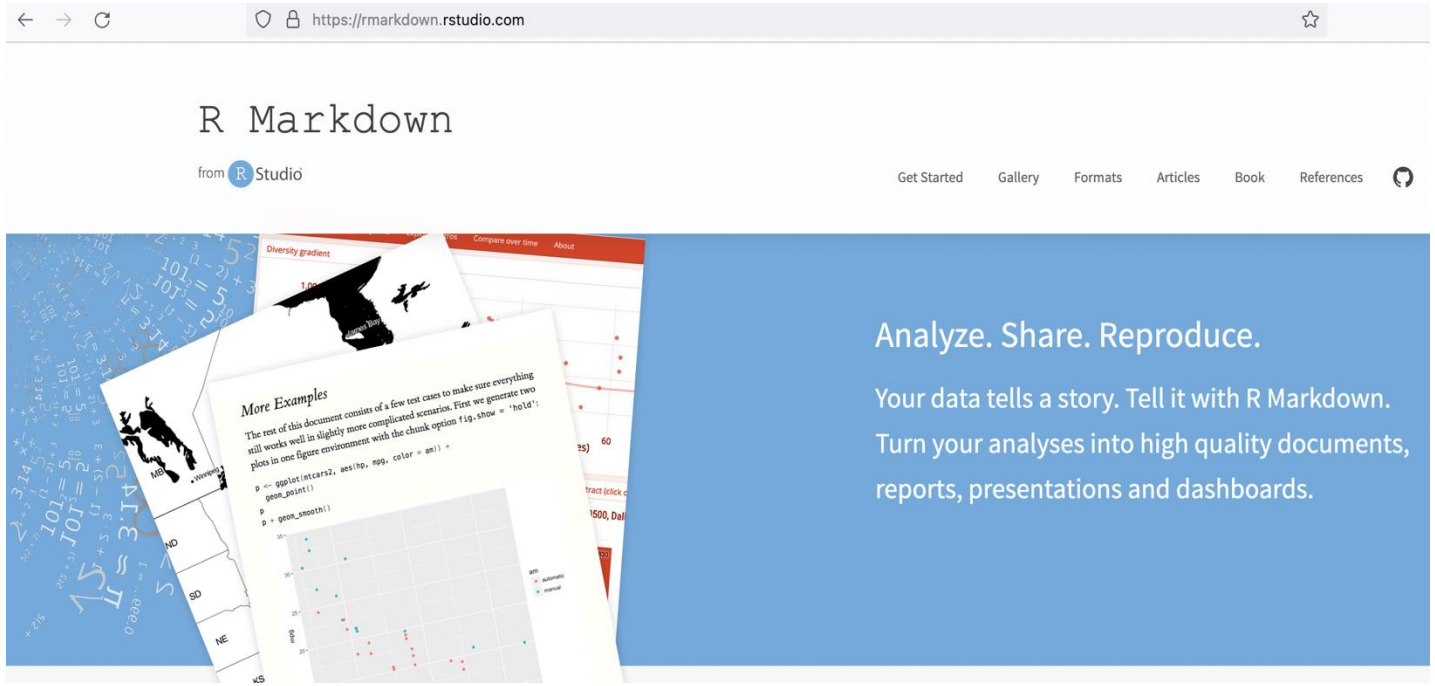
- Use github!!

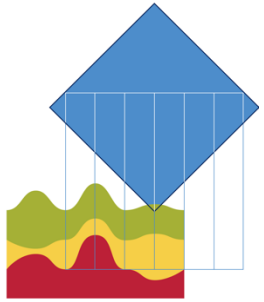




Rmarkdown

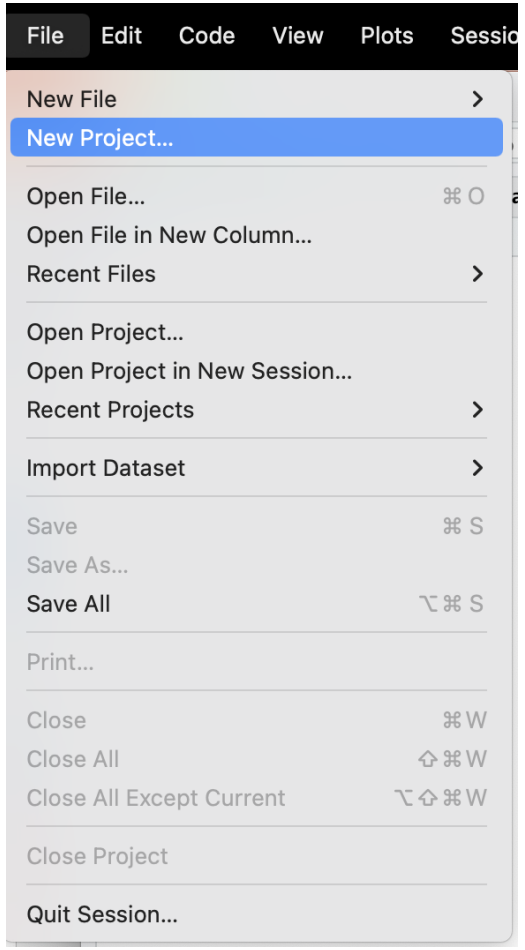
- A combination between scripts and report writing
- You can use them as an advanced version of script
- You can also produce beautiful reports, slides or even typeset



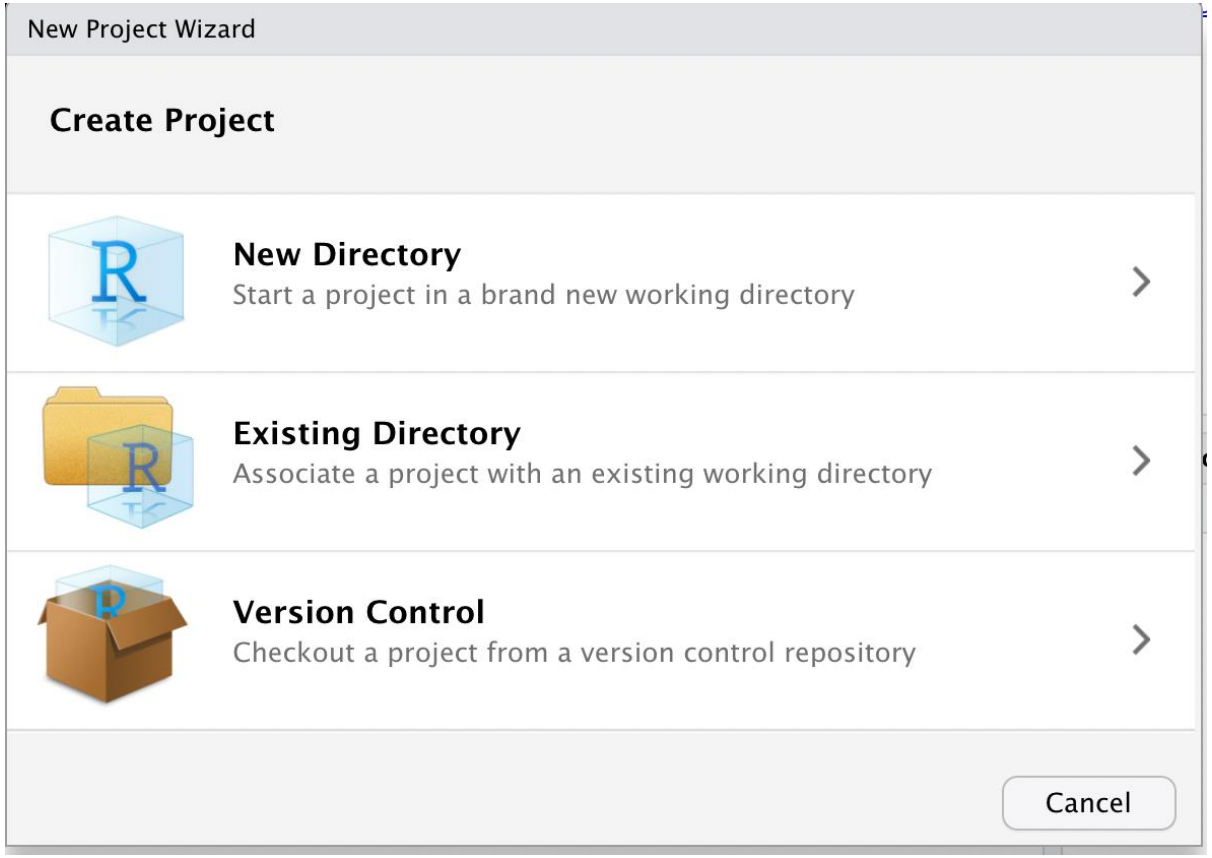


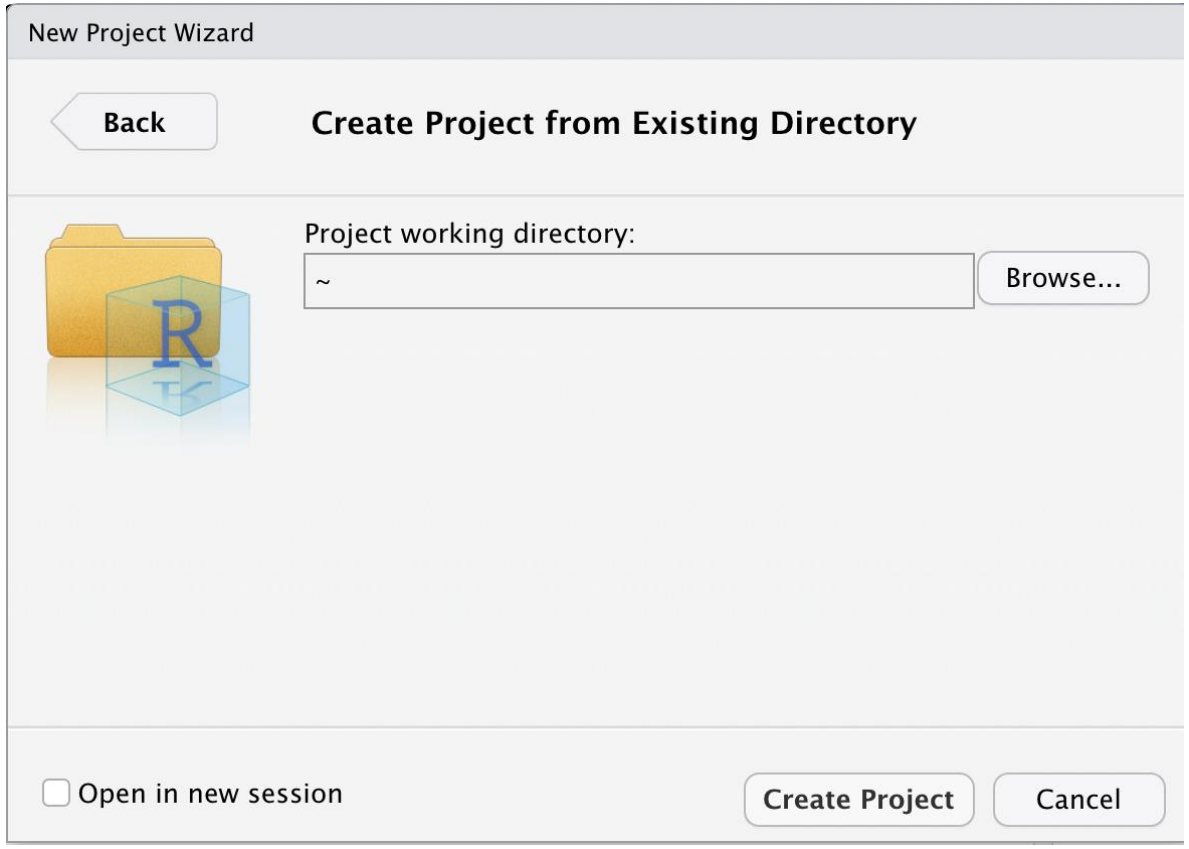
R projects

- A project is a way to contain all the information you need for a complete analysis in single directory
- This has multiple benefits
  - All paths are set to the project directory
  - Working within a project means you always have relative directory paths
  - Data and results are all kept together, ensuring reproducibility
  - You can move a project to a different computer and will run in the exact same way

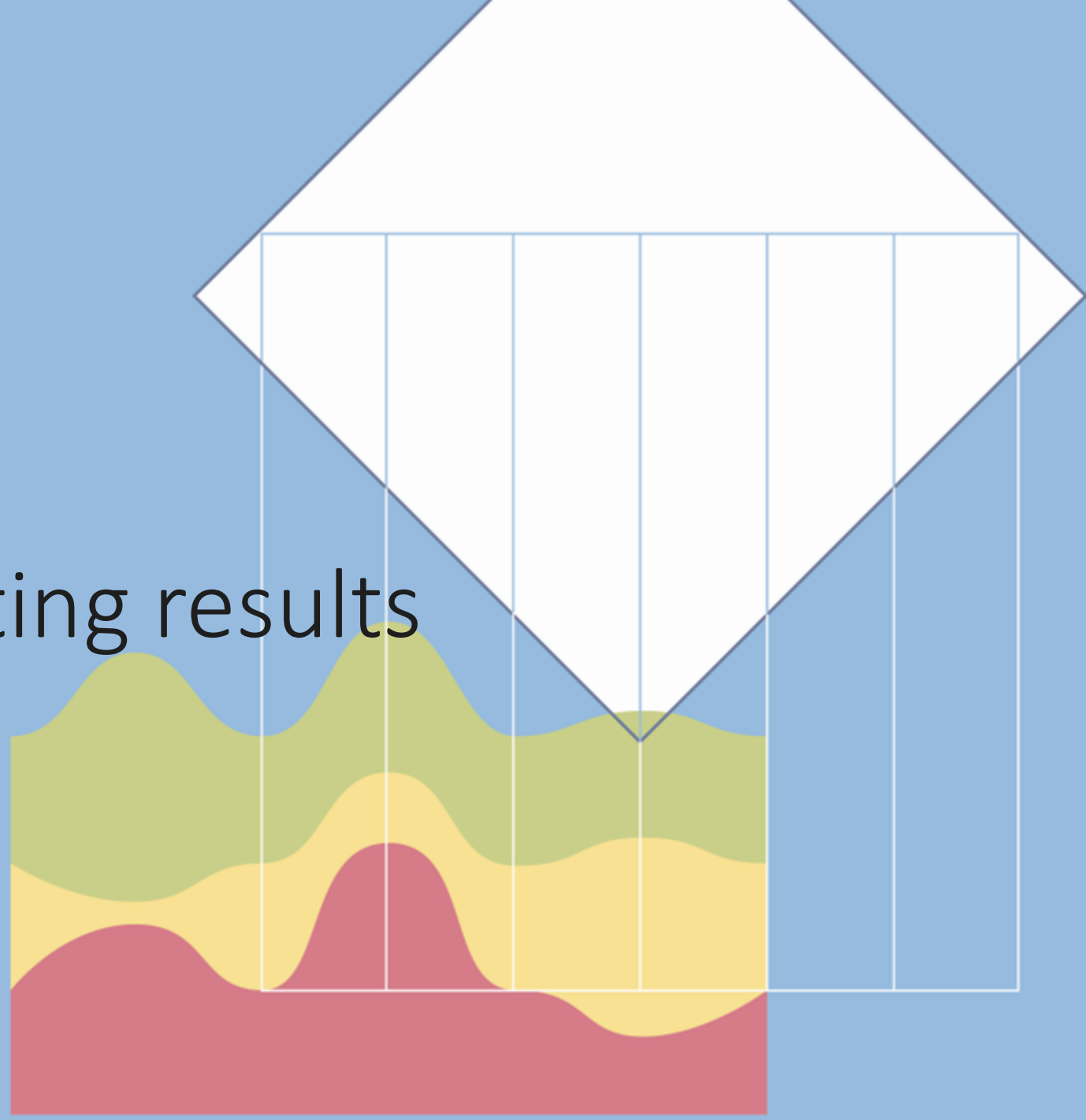


R projects


New Project Wizard
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel


New Project Wizard
Back
Create Project from Existing Directory
Project working directory:
~
Browse...
Open in new session
Create Project
Cancel

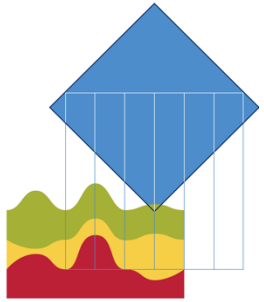
Reporting results



OXFORD UNIVERSITY STATISTICAL CONSULTING

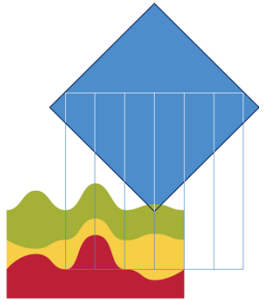
What is Rmarkdown and why should you care?

- Rmarkdown is a tool that allows you to keep your code, analysis output, and written report in one place
- Keeping your code and report in one place means you can troubleshoot in the future
- It also means you can share your work with collaborators
- Ensures reproducibility
- It can create some really beautiful output



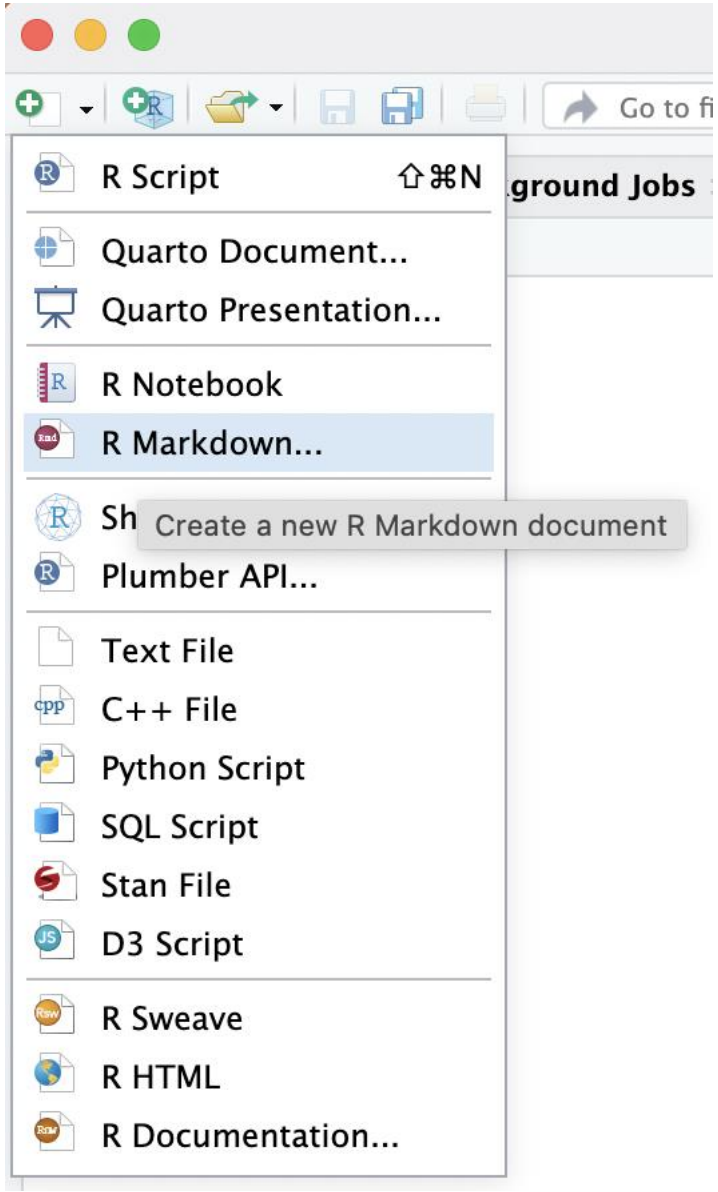
How I use Rmarkdown

- As part of my consultancy work all of my reports are Rmarkdown files
- I do all of my analysis in Rmarkdown files instead of scripts even if they are just for me – Rmarkdown is my lab-book
- This is because I want everything to be in one place rather than scattered in multiple scripts
- I…did not write my thesis in Rmarkdown…



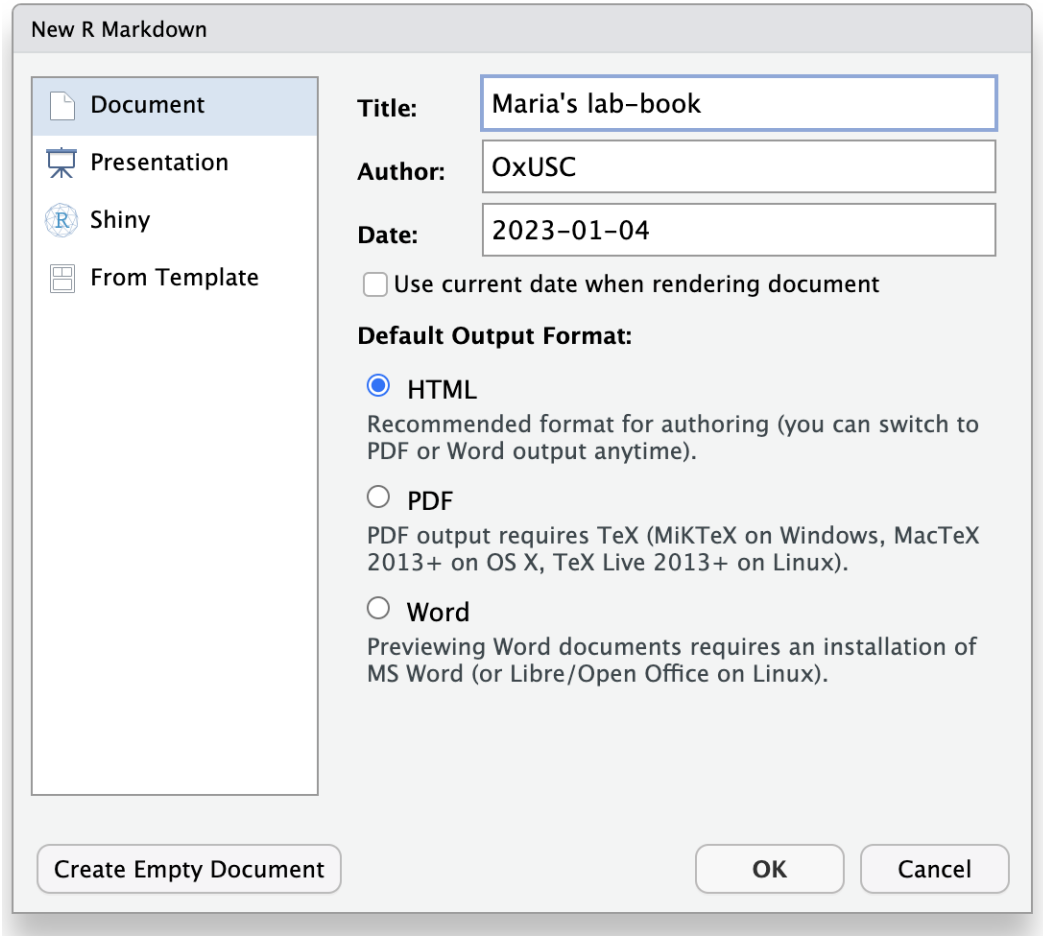
Getting started

- For html reports, all you need is Rstudio or if you are not using Rstudio then installing the rmarkdown package
- You can simply open Rstudio and select a new Rmarkdown file



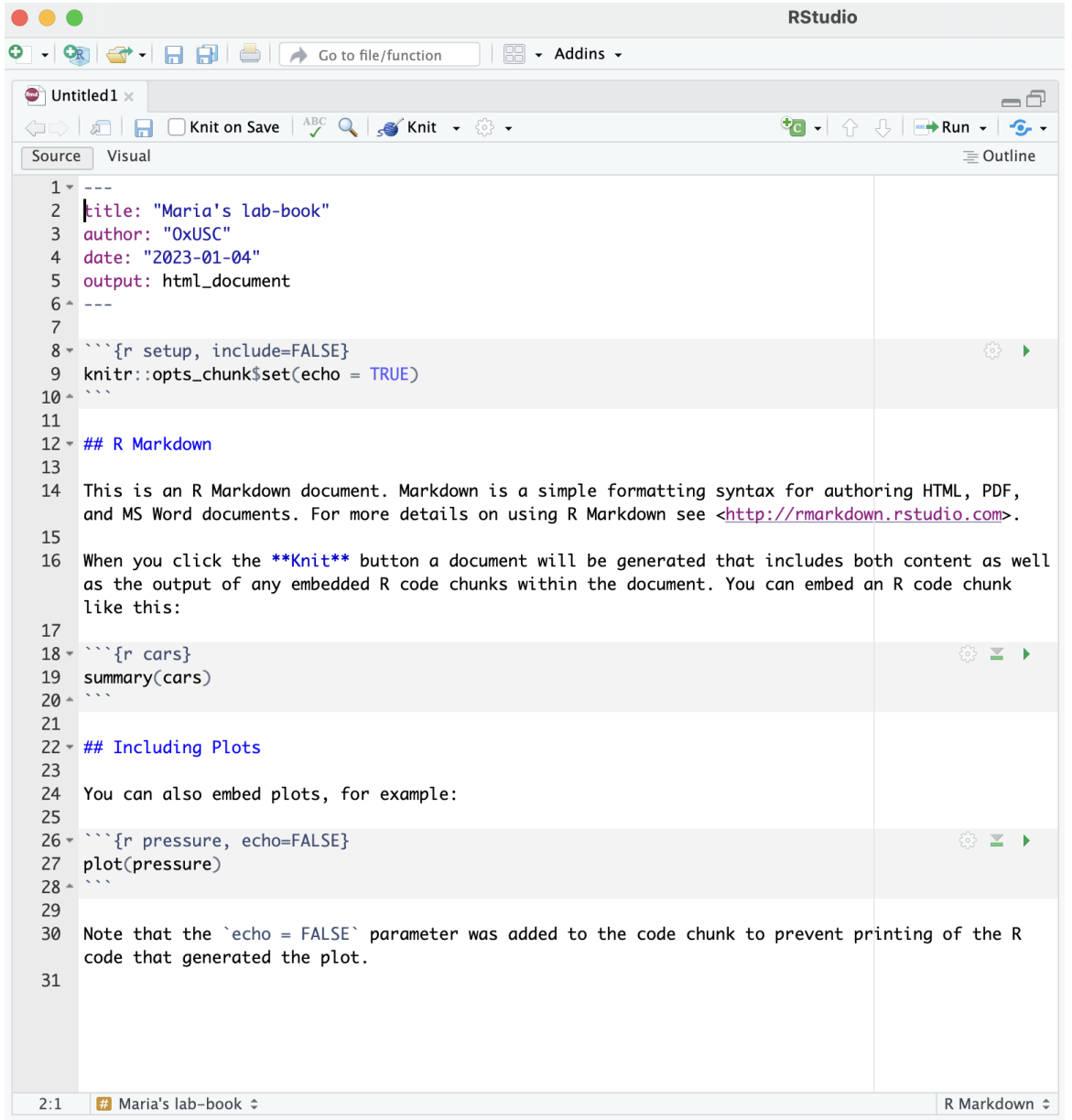
Getting started

- Once you select the type of file, you can make some choices as to what you want to create
- Here I am choosing an html document
- I am essentially selecting my metadata for this document

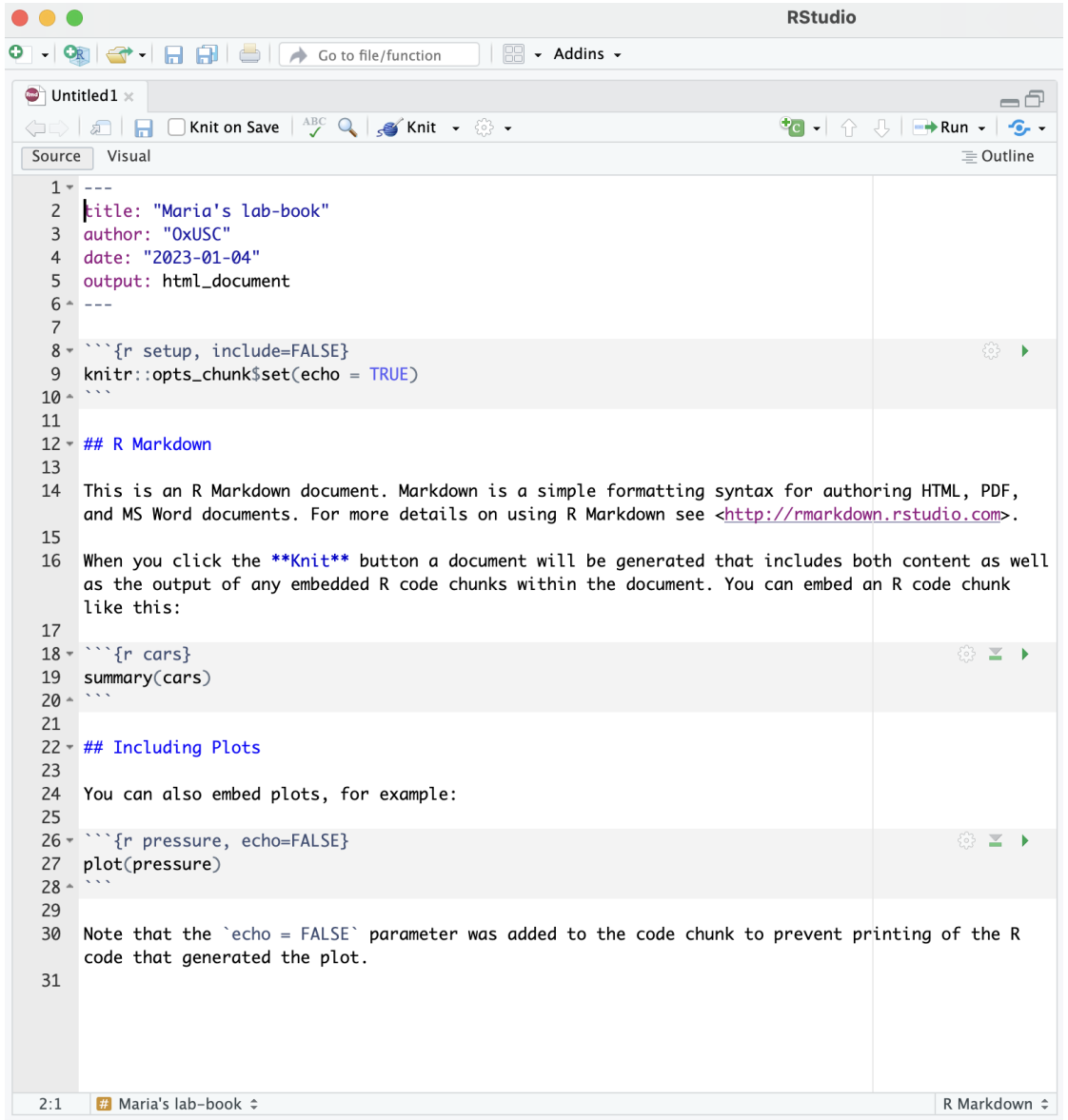


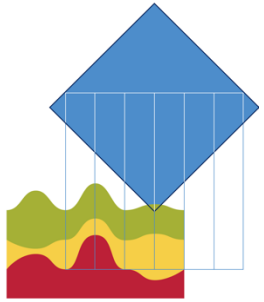
Getting started

- The default gives you some examples as to what you can do



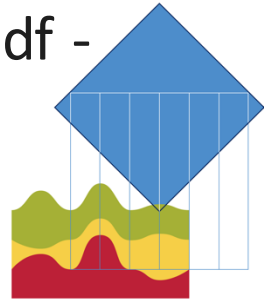
Getting started





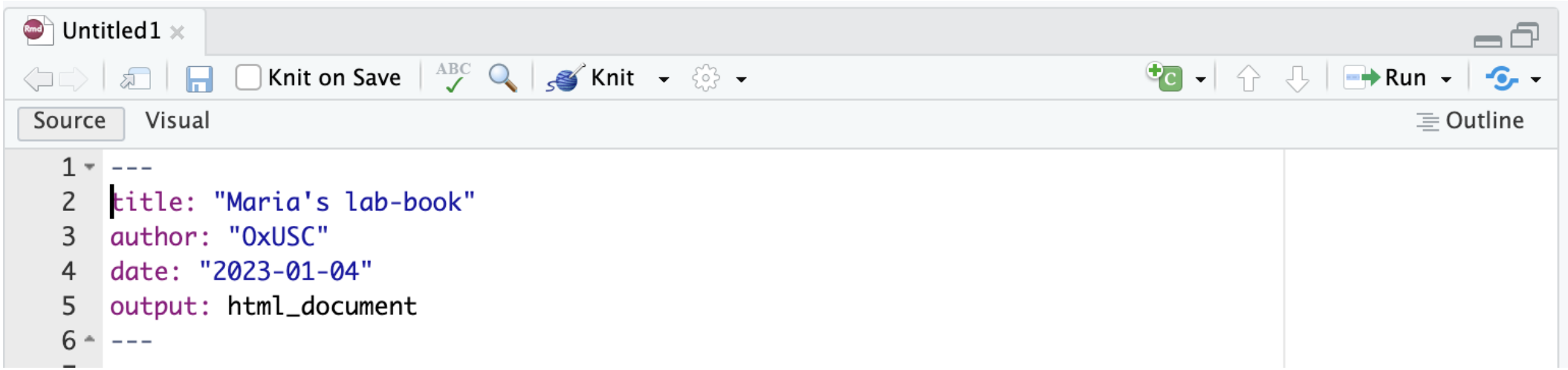
Behind the scenes

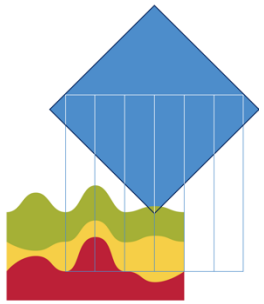
- The compilation of a document from an Rmarkdown file is a multistep process
- First up the function knit() from the knitr package runs all the code and prepares an intermediary .md document
- This intermediary .md document is processed by a program called pandoc and that takes the metadata and creates the specified output
    - If you have selected pdf then this will interact with LaTeX to create the pdf - you need TeX installed…



Metadata, text, and code chunks

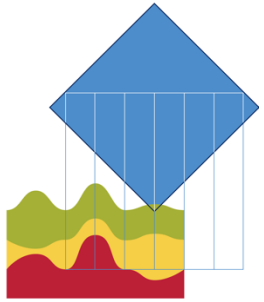
- Metadata give markdown information about what you expect your output to be and some details for it
- It’s the bit that pandoc reads to know what output to produce





Metadata, text, and code chunks

- The default Rmarkdown comes populated with some text and code chunks
- This is to give you basic idea of what everything does
- First up – try to compile your Rmarkdown file into a document



Knitting a document

MariasLabBook.html | Open in Browser | Find | Publish

Maria's lab-book

OxUSC

2023-01-04

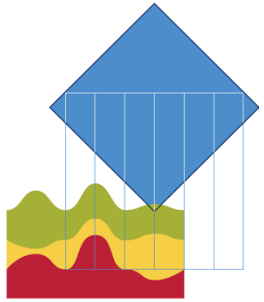
R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

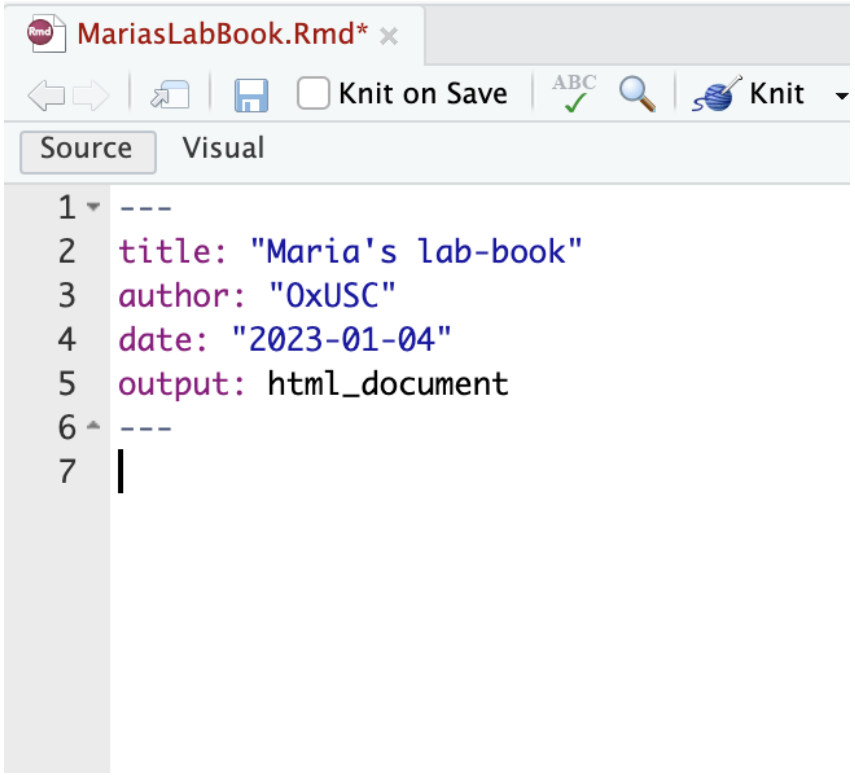
```
summary(cars)
```

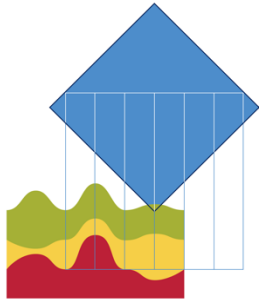
```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```



Metadata, text, and code chunks

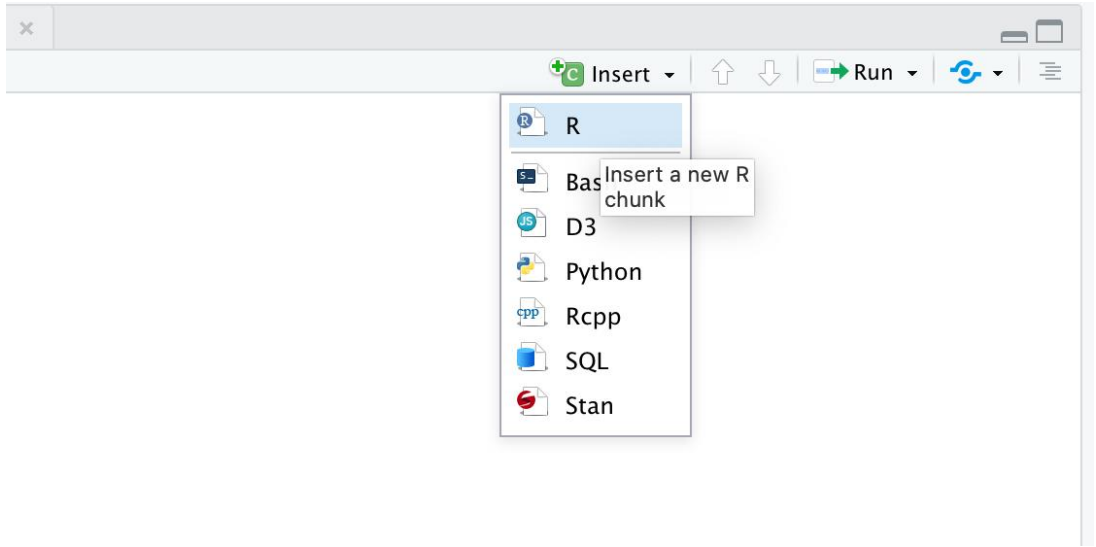
- Let's start with an example - converting the R script from one of the beginner practicals
- Let us first remove everything but the metadata

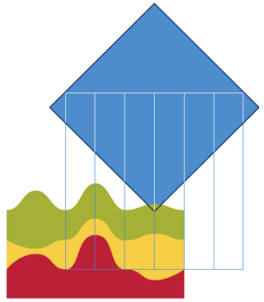




Metadata, text, and code chunks

- First up let us start with a set up chunk
- In there I like to add some things that have to happen for my code to run
  - e.g any packages to load or any global options





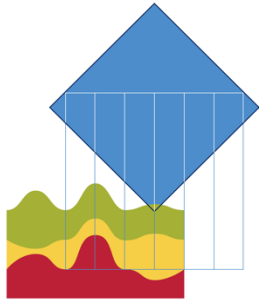
Metadata, text, and code chunks

MariasLabBook.Rmd*

Knit on Save | Knit

Source | Visual

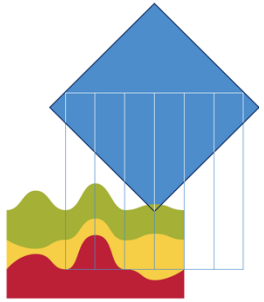
```
1  ---
2  title: "Maria's lab-book"
3  author: "OxUSC"
4  date: "2023-01-04"
5  output: html_document
6  ---
7
8  ```{r setup chunk, include=FALSE, echo=FALSE}
9
10 knitr::opts_chunk$set(include = TRUE, echo = FALSE)
11 library(tidyverse)
12
13 ```
14
15
```



Code chunks

- An r code chunk starts and ends with ```{r} and ends with ```
- After r you can add what you want your code chunk to be named for navigation - be careful, each code chunk should be individually named
- The two most common options for the code chunk are include and echo
  - echo: Whether to display the source code in the output document
  - include: Whether to include the chunk in the output document

```
8  ```{r setup chunk, include=FALSE, echo=FALSE}
9
10 knitr::opts_chunk$set(include = TRUE, echo = FALSE)
```



Text

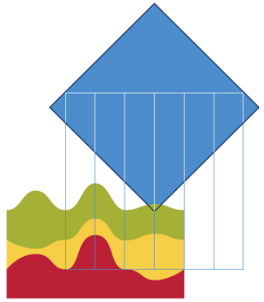
- Text includes all bits of discussion or explanation that you require
- These are all printed in the final document except if you have commented them out

MariasLabBook.Rmd*

Knit on Save | Knit

Source | Visual

```
1  ---
2  title: "Maria's lab-book"
3  author: "OxUSC"
4  date: "2023-01-04"
5  output: html_document
6  ---
7  In the chunk below I would like to setup my global options and load packages
8
9  <!-- However I would like to comment out this sentence. -->
10
```



What it looks like knitted

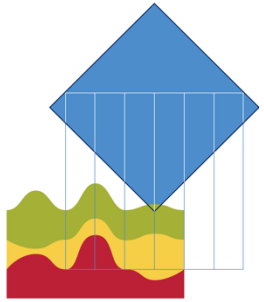
MariasLabBook.html | Open in Browser | Find

Maria's lab-book

OxUSC

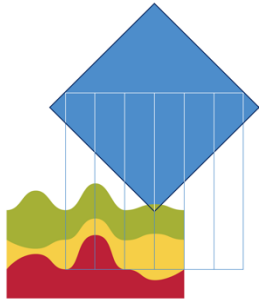
2023-01-04

In the chunk below I would like to setup my global options and load packages



Text

- Using headers for your text helps break your report into sections and helps navigation
- You can do this using the #
  - Level 1 header is #
  - Level 2 is ## etc
- Italics are *italics*
- Bold is **bold**



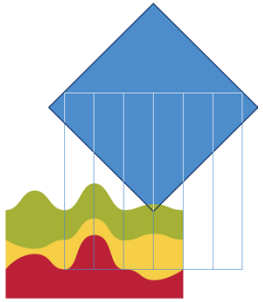
Text

MariasLabBook.Rmd* | Solutions_Practical 3 and 5_Statisti...

Knit on Save | Knit | Run

Source | Visual | Outline

```
1  ---
2  title: "Maria's lab-book"
3  author: "OxUSC"
4  date: "2023-01-04"
5  output: html_document
6  ---
7  In the chunk below I would like to setup my global options and load packages
8
9  <!-- However I would like to comment out this sentence. -->
10
11 ```{r setup chunk, include=FALSE, echo=FALSE}
12
13 knitr::opts_chunk$set(include = TRUE, echo = FALSE)
14 library(tidyverse)
15 library(tidytuesdayR)
16 library(janitor)
17 library(rstatix)
18
19 ```
20
21 #### Analysis
22
23 In this part we are doing a chi squared test on the medals won by Italy and Greece during the 2004
   Olympics. In the **chunk below** I am loading the dataset and performing a *Chi-squared test*.
```



Text

MariasLabBook.html | Open in Browser | Find

Maria's lab-book

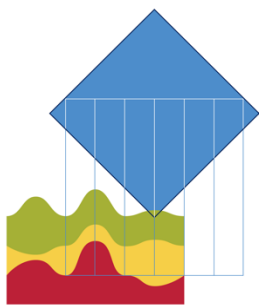
OxUSC

2023-01-04

In the chunk below I would like to setup my global options and load packages

Analysis

In this part we are doing a chi squared test on the medals won by Italy and Greece during the 2004 Olympics. In the **chunk below** I am loading the dataset and performing a *Chi-squared test*.



Reporting results

- We can use markdown to report results without copying and pasting values
- This ensures we get the correct results and don't make typos
- It also means if you change the dataset, the report will update

````
#### Analysis

In this part we are doing a chi squared test on the medals won by Italy and Greece during the 2004 Olympics. In the **chunk below** I am loading the dataset and performing a *Chi-squared test*.

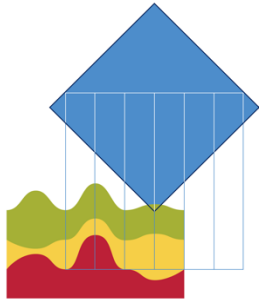
```{r chiSquaredTest, include=FALSE}
tuesdata <- tidytuesdayR::tt_load(2021, week = 31) # download the Olympics data
Olympics<-tuesdata$olympics #save the dataset under a new name

head(Olympics) #look through it

Olympics %>% filter(!is.na(medal)) %>% filter(year==2004)->AthensMedals # save the Athens medal winners
AthensMedals %>% filter(team=="Italy"|team=="Greece")->ItalyGreece2004 #filter Italian and Greek winners

ItalyGreece2004 %>%tabyl(team, medal)-> contingencyAthensITAGRE # Contingency table for chi squared test

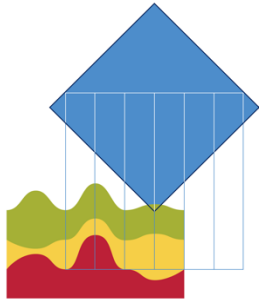
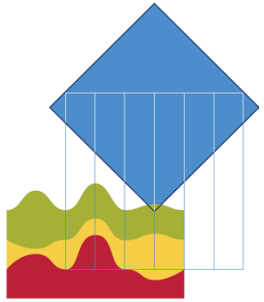

testResults<-chisq.test(contingencyAthensITAGRE) #chisquare test
```
In the analysis we conducted, we found that the chi-squared statistic was `r round(testResults$statistic, 2)` for `r testResults$parameter` degrees of freedom, giving us a p-value of `r testResults$p.value`.
````

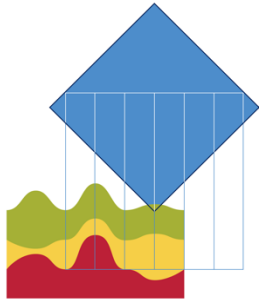
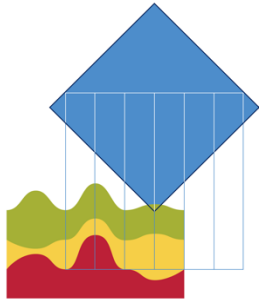


Reporting results

```
#### Analysis

In this part we are doing a chi squared test on the medals won by Italy and Greece during the 2004 Olympics. In the **chunk below** I am loading the dataset and performing a *Chi-squared test*.

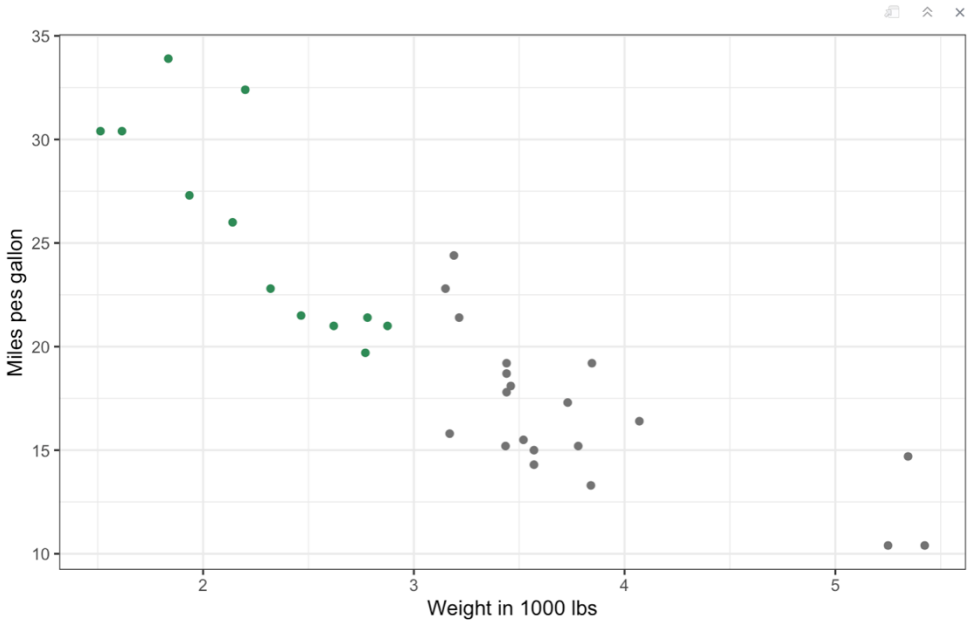
```{r chiSquaredTest, include=FALSE}
tuesdata <- tidytuesdayR::tt_load(2021, week = 31) # download the Olympics data
Olympics<-tuesdata$olympics #save the dataset under a new name

head(Olympics) #look through it

Olympics %>% filter(!is.na(medal)) %>% filter(year==2004)->AthensMedals # save the Athens medal winners
AthensMedals %>% filter(team=="Italy"|team=="Greece")->ItalyGreece2004 #filter Italian and Greek winners

ItalyGreece2004 %>%tabyl(team, medal)-> contingencyAthensITAGRE # Contingency table for chi squared test


testResults<-chisq.test(contingencyAthensITAGRE) #chisquare test
```
In the analysis we conducted, we found that the chi-squared statistic was `r round(testResults$statistic, 2)` for `r testResults$parameter` degrees of freedom, giving us a p-value of `r testResults$p.value`.
```



Reporting results

MariasLabBook.html | Open in Browser | Find

Maria's lab-book

OxUSC

2023-01-04

In the chunk below I would like to setup my global options and load packages

Analysis

In this part we are doing a chi squared test on the medals won by Italy and Greece during the 2004 Olympics. In the **chunk below** I am loading the dataset and performing a *Chi-squared test*.

In the analysis we conducted, we found that the chi-squared statistic was 5.32 for 2 degrees of freedom, giving us a p-value of 0.0699321.



Reporting results

- Summary tables are always useful for reports
- I like using the functionality from the kableExtra package
- It combines well with pipes and looks tidy in an html report

````
To explore where the differences lie between groups, we looked at the residuals.

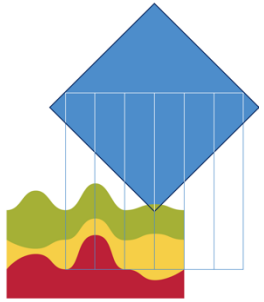
```{r residuals table}
testResults$residuals %>%
  as_tibble() %>%
  kbl(caption = "Chi-square residuals for Athens 2004 olympics medals, Italy and Greece") %>%
  kable_styling()
```
This suggests that Greece is slightly underrepresented in the Bronze category, compared to Italy.
````



Reporting results

MariasLabBook.html | Open in Browser | Find

Maria's lab-book

OxUSC

2023-01-04

In the chunk below I would like to setup my global options and load packages

Analysis

In this part we are doing a chi squared test on the medals won by Italy and Greece during the 2004 Olympics. In the **chunk below** I am loading the dataset and performing a *Chi-squared test*.

In the analysis we conducted, we found that the chi-squared statistic was 5.32 for 2 degrees of freedom, giving us a p-value of 0.0699321.

To explore where the differences lie between groups, we looked at the residuals.

Chi-square residuals for Athens 2004 olympics medals, Italy and Greece

team	Bronze	Gold	Silver
Greece	-1.6056288	0.2404695	1.2094857
Italy	0.8766159	-0.1312877	-0.6603359

This suggests that Greece is slightly underrepresented in the Bronze category, compared to Italy.



Adding graphics

````
#### Graphics

To illustrate how to add graphics to your reports, I will use one of the graphics from the
lecture.

```{r graphics}

ggplot(mtcars, aes(x=wt, y=mpg))+
  geom_point(shape=19, colour=if_else(mtcars$wt<3, "seagreen4", "grey45"))+
  xlab("Weight in 1000 lbs")+
  ylab("Miles pes gallon")+
  theme_bw()
```
````





Adding graphics

MariasLabBook.html | Open in Browser | Find

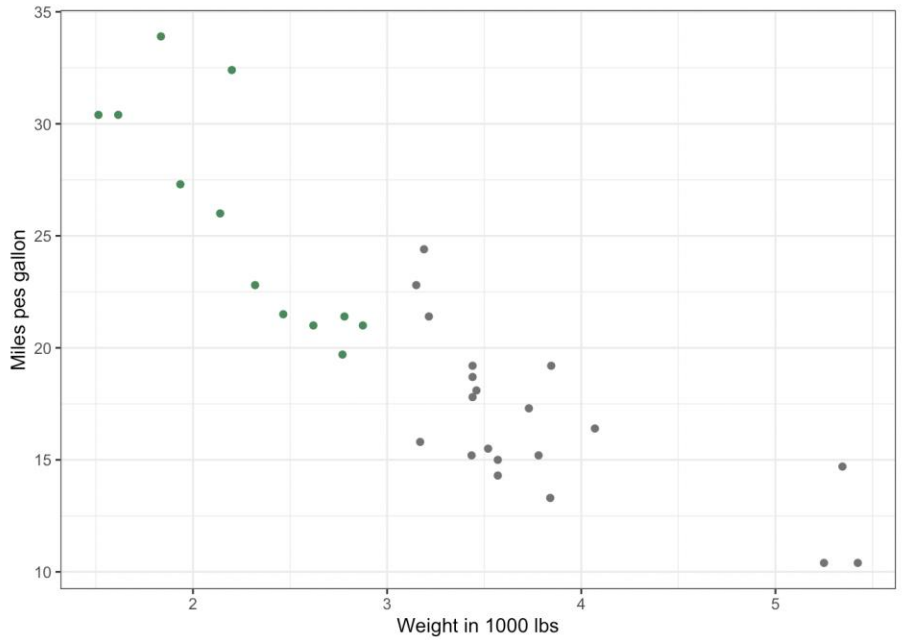
Chi-square residuals for Athens 2004 olympics medals, Italy and Greece

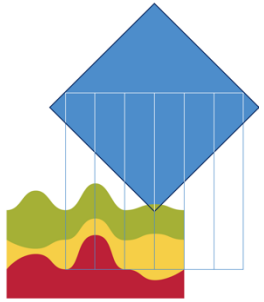
team	Bronze	Gold	Silver
Greece	-1.6056288	0.2404695	1.2094857
Italy	0.8766159	-0.1312877	-0.6603359

This suggests that Greece is slightly underrepresented in the Bronze category, compared to Italy.

Graphics

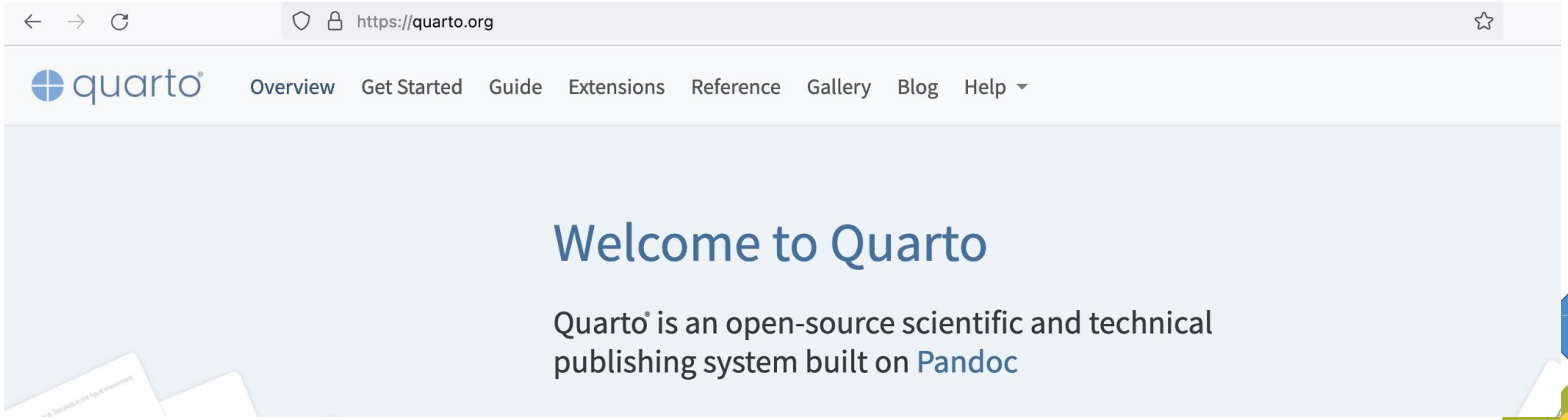
To illustrate how to add graphics to your reports, I will use one of the graphics from the lecture.

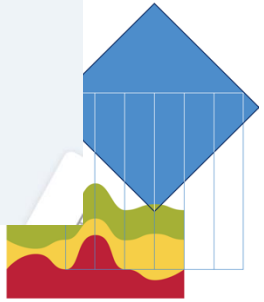




The move to quarto

- Quarto was released in late 2022 by the people who developed Rstudio
- As opposed to Rmarkdown, quarto can be used for other languages as well, such as Julia or Python





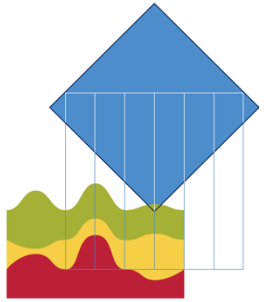
The move to quarto

- For R users, it is a next generation version of Rmarkdown, with the added bonus that it does not require additional packages for different types of content

Hello, Quarto

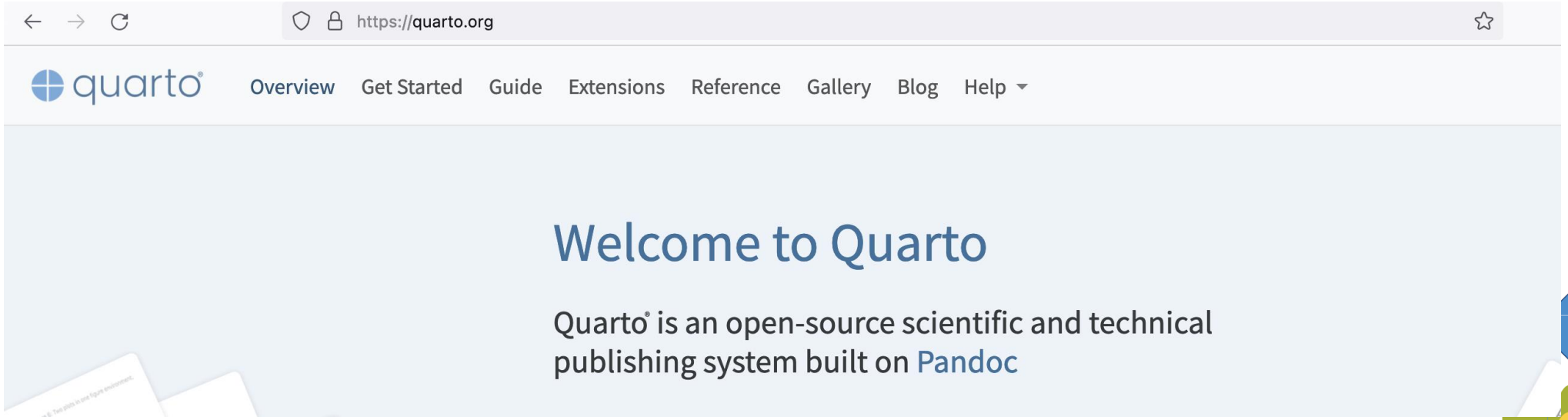
Python | R | Julia | Observable

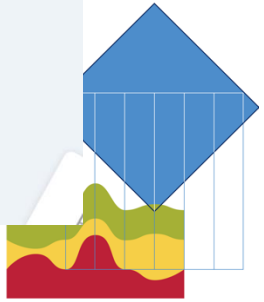
Quarto is a multi-language, next generation version of R Markdown from RStudio, with many new new features and capabilities. Like R Markdown, Quarto uses Knitr to execute R code, and is therefore able to render most existing Rmd files without modification.



The move to quarto

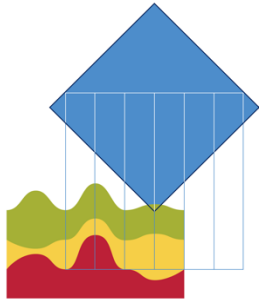
- In essence this is just an improved version of Rmarkdown
- So…why am I not teaching you this BRAND NEW piece of technology yet?





The move to quarto

- In essence this is just an improved version of Rmarkdown
- So…why am I not teaching you this BRAND NEW piece of technology yet?
- Because it’s brand new
  - Resources are not fully there yet, for beginners
  - Rmarkdown is not being removed
  - If you learn on Rmarkdown, you’ll be able to easily transition to quarto



Quarto guides and tutorials