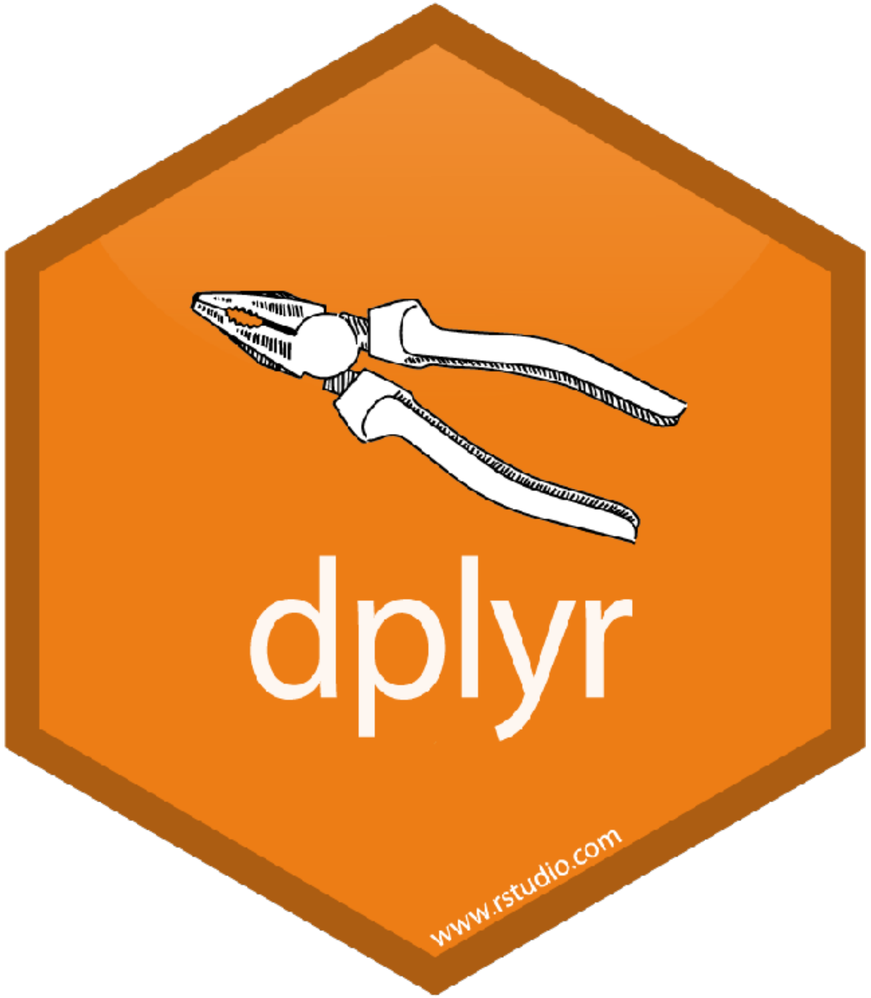
Lecture 8

Week 4, Nov 17th 2020
(4.3)

Lecture 8

- Data transformation
  - `dplyr` package
- Tidy data

- Importing data
- Exporting images
- Labels
- Scales
- Choropleth map



Data Transformation

`dplyr` + tidyr

dplyr

- You are going to learn the 7 key dplyr functions that allow you to solve the vast majority of your data manipulation challenges:

Function	Description	Equivalent SQL
`select()`	selecting columns	SELECT
`filter()`	filtering rows / subsetting	WHERE
`group_by()`	grouping data	GROUP BY
`summarise()`	summarising / aggregating data	-
`arrange()`	sorting data	ORDER BY
`join()`	joining data tables	JOIN
`mutate()`	creating new columns	COLUMN ALIAS

Basic principle

- All **`dplyr`** functions work similarly:
  - The first argument is a data frame
  - Subsequent argument describe what to do
  - Output is a new data frame

Prerequisites

- Download and load packages:

```r
library(nycflights13)
library(tidyverse)
```

```r
flights
#> # A tibble: 336,776 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013     1     1      517            515         2      830
#> 2  2013     1     1      533            529         4      850
#> 3  2013     1     1      542            540         2      923
#> 4  2013     1     1      544            545        -1     1004
#> 5  2013     1     1      554            600        -6      812
#> 6  2013     1     1      554            558        -4      740
#> # … with 3.368e+05 more rows, and 12 more variables: sched_arr_time <int>,
#> #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#> #   minute <dbl>, time_hour <dttm>
```

Your turn!

1. Are there vignettes for the **`dplyr`** package?
2. Can you find additional documentation explaining the **`flights`** dataset?
3. What variables are in the **`flights`** dataset? How many rows of data are in the **`flights`** dataset?

Solution

1. Are there vignettes for the dplyr package?

```r
# Lists topics
vignette(package = "dplyr")
# Select a topic
vignette(package = "dplyr", topic = "dplyr")
```

2. Can you find additional documentation explaining the `flights` dataset?

```r
?flights
```

3. What variables are in the `flights` dataset? How many rows of data are in the `flights` dataset?

```r
str(flights)

dim(flights)
```

`filter()`

pick observations by their values

Basic filtering

```
filter(flights, month == 1)
# A tibble: 27,004 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2      830            819        11
2   2013     1     1      533            529         4      850            830        20
3   2013     1     1      542            540         2      923            850        33
4   2013     1     1      544            545        -1     1004           1022       -18
5   2013     1     1      554            600        -6      812            837       -25
6   2013     1     1      554            558        -4      740            728        12
7   2013     1     1      555            600        -5      913            854        19
8   2013     1     1      557            600        -3      709            723       -14
9   2013     1     1      557            600        -3      838            846        -8
10  2013     1     1      558            600        -2      753            745         8
# ... with 26,994 more rows, and 10 more variables: carrier <chr>, flight <int>,
```

Basic filtering

```
filter(flights, month == 1, day == 1)
# A tibble: 842 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2      830            819        11
2   2013     1     1      533            529         4      850            830        20
3   2013     1     1      542            540         2      923            850        33
4   2013     1     1      544            545        -1     1004           1022       -18
5   2013     1     1      554            600        -6      812            837       -25
6   2013     1     1      554            558        -4      740            728        12
7   2013     1     1      555            600        -5      913            854        19
8   2013     1     1      557            600        -3      709            723       -14
9   2013     1     1      557            600        -3      838            846        -8
10  2013     1     1      558            600        -2      753            745         8
# ... with 832 more rows, and 10 more variables: carrier <chr>, flight <int>, tailnum <chr>,
```

Basic filtering

```
filter(flights, month == 1, day == 1, dep_delay > 0)
# A tibble: 352 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2      830            819        11
2   2013     1     1      533            529         4      850            830        20
3   2013     1     1      542            540         2      923            850        33
4   2013     1     1      601            600         1      844            850        -6
5   2013     1     1      608            600         8      807            735        32
6   2013     1     1      611            600        11      945            931        14
7   2013     1     1      613            610         3      925            921         4
8   2013     1     1      623            610        13      920            915         5
9   2013     1     1      632            608        24      740            728        12
10  2013     1     1      644            636         8      931            940        -9
# ... with 342 more rows, and 10 more variables: carrier <chr>, flight <int>, tailnum <chr>,
```

Save a new data frame

```r
# Save data frame using assignment operator "<-"
jan1 <- filter(flights, month == 1, day == 1)

# Save and print filtered data frame by wrapping entire code with ()
(dec25 <- filter(flights, month == 12, day == 25))
```

Logical tests

```
12 == 12

12 <= c(12, 11)

12 %in% c(12, 11, 8)

x <- c(12, NA, 11, NA, 8)
is.na(x)
```

`<`	Less than
`>`	Greater than
`==`	Equal to
`<=`	Less than or equal to
`>=`	Greater than or equal to
`!=`	Not equal to
`%in%`	Group membership
`is.na`	Is NA
`!is.na`	Is not NA

Comparison

- What will these operations produce?

```
1) filter(flights, month == 12)
2) filter(flights, month != 12)
3) filter(flights, month %in% c(11, 12))
4) filter(flights, arr_delay <= 120)
5) filter(flights, !(arr_delay <= 120))
6) filter(flights, is.na(tailnum))
```

Menti Quiz

Multiple logical tests

```
12 == 12 & 12 < 14

12 == 12 & 12 < 10

12 == 12 | 12 < 10

any(12 == 12, 12 < 10)

all(12 == 12, 12 < 10)
```

`?base::Logical`

`&`	boolean and
`|`	boolean or
`xor`	exclusively x or y
`!`	not
`any`	any true
`all`	all true

Multiple comparisons

```r
# Using comma is same as using &
filter(flights, month == 12, day == 25)
filter(flights, month == 12 & day == 25)

# Use %in% as a shortcut for |
filter(flights, month == 11 | month == 12)
filter(flights, month %in% c(11, 12))

# Are the outputs the same?
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

`arrange()`

Reorder the rows

Ordering your data

```
arrange(flights, dep_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1   2013    12     7     2040           2123       -43       40           2352        48      B6     97
2   2013     2     3     2022           2055       -33     2240           2338       -58      DL   1715
3   2013    11    10     1408           1440       -32     1549           1559       -10      EV   5713
4   2013     1    11     1900           1930       -30     2233           2243       -10      DL   1435
5   2013     1    29     1703           1730       -27     1947           1957       -10      F9    837
6   2013     8     9      729            755       -26     1002            955         7      MQ   3478
7   2013    10    23     1907           1932       -25     2143           2143         0      EV   4361
8   2013     3    30     2030           2055       -25     2213           2250       -37      MQ   4573
9   2013     3     2     1431           1455       -24     1601           1631       -30      9E   3318
10  2013     5     5      934            958       -24     1225           1309       -44      B6    375
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Ordering your data

```
arrange(flights, dep_delay, arr_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>   <int>          <int>     <dbl>   <chr>  <int>
1   2013    12     7     2040           2123       -43      40           2352        48      B6     97
2   2013     2     3     2022           2055       -33    2240           2338       -58      DL   1715
3   2013    11    10     1408           1440       -32    1549           1559       -10      EV   5713
4   2013     1    11     1900           1930       -30    2233           2243       -10      DL   1435
5   2013     1    29     1703           1730       -27    1947           1957       -10      F9    837
6   2013     8     9      729            755       -26    1002            955         7      MQ   3478
7   2013     3    30     2030           2055       -25    2213           2250       -37      MQ   4573
8   2013    10    23     1907           1932       -25    2143           2143         0      EV   4361
9   2013     5     5      934            958       -24    1225           1309       -44      B6    375
10  2013     9    18     1631           1655       -24    1812           1845       -33      AA   2223
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Ordering your data

```
arrange(flights, desc(dep_delay))
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>   <int>          <int>     <dbl>   <chr>  <int>
1   2013     1     9      641            900      1301    1242           1530      1272      HA     51
2   2013     6    15     1432           1935      1137    1607           2120      1127      MQ   3535
3   2013     1    10     1121           1635      1126    1239           1810      1109      MQ   3695
4   2013     9    20     1139           1845      1014    1457           2210      1007      AA    177
5   2013     7    22      845           1600      1005    1044           1815       989      MQ   3075
6   2013     4    10     1100           1900       960    1342           2211       931      DL   2391
7   2013     3    17     2321            810       911     135           1020       915      DL   2119
8   2013     6    27      959           1900       899    1236           2226       850      DL   2007
9   2013     7    22     2257            759       898     121           1026       895      DL   2047
10  2013    12     5      756           1700       896    1058           2020       878      AA    172
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Missing data order

- Note: missing values are always sorted at the end:

```
df <- tibble(x = c(5, 2, 5, NA))
# A tibble: 4 × 1
      x
  <dbl>
1     5
2     2
3     5
4    NA
```

```
arrange(df, x)
# A tibble: 4 × 1
      x
  <dbl>
1     2
2     5
3     5
4    NA
```

```
arrange(df, desc(x))
# A tibble: 4 × 1
      x
  <dbl>
1     5
2     5
3     2
4    NA
```

`select()`

Pick variables by their names

Selecting variables

```
select(flights, year, month, day)
# A tibble: 336,776 × 3
    year month   day
   <int> <int> <int>
1   2013     1     1
2   2013     1     1
3   2013     1     1
4   2013     1     1
5   2013     1     1
6   2013     1     1
7   2013     1     1
8   2013     1     1
9   2013     1     1
10  2013     1     1
# ... with 336,766 more rows
```

```
select(flights, year:day)
# A tibble: 336,776 × 3
    year month   day
   <int> <int> <int>
1   2013     1     1
2   2013     1     1
3   2013     1     1
4   2013     1     1
5   2013     1     1
6   2013     1     1
7   2013     1     1
8   2013     1     1
9   2013     1     1
10  2013     1     1
# ... with 336,766 more rows
```

Deselecting variable

- Deselect with “-” sign

```
select(flights, -(year:day))
# A tibble: 336,776 × 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
      <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1       517            515         2      830            819        11      UA   1545
2       533            529         4      850            830        20      UA   1714
3       542            540         2      923            850        33      AA   1141
4       544            545        -1     1004           1022       -18      B6    725
5       554            600        -6      812            837       -25      DL    461
6       554            558        -4      740            728        12      UA   1696
7       555            600        -5      913            854        19      B6    507
8       557            600        -3      709            723       -14      EV   5708
9       557            600        -3      838            846        -8      B6     79
10      558            600        -2      753            745         8      AA    301
```

Useful select functions

`-`	Select everything but
`:`	Select range
`contains()`	Select columns whose name contains a character string
`ends_with()`	Select columns whose name ends with a string
`everything()`	Select every column
`matches()`	Select columns whose name matches a regular expression
`num_range()`	Select columns named x1, x2, x3, x4, x5
`one_of()`	Select columns whose names are in a group of names
`starts_with()`	Select columns whose name starts with a character string

select by name patterns

```
select(flights, ends_with("time"))
# A tibble: 336,776 × 5
   dep_time sched_dep_time arr_time sched_arr_time air_time
      <int>          <int>    <int>          <int>    <dbl>
1       517            515      830            819      227
2       533            529      850            830      227
3       542            540      923            850      160
4       544            545     1004           1022      183
5       554            600      812            837      116
6       554            558      740            728      150
7       555            600      913            854      158
8       557            600      709            723       53
9       557            600      838            846      140
10      558            600      753            745      138
# ... with 336,766 more rows
```

Multiple name patterns

```
select(flights, c(carrier, ends_with("time"), contains("delay")))
# A tibble: 336,776 × 8
   carrier dep_time sched_dep_time arr_time sched_arr_time air_time dep_delay arr_delay
     <chr>    <int>          <int>    <int>          <int>    <dbl>     <dbl>     <dbl>
1       UA      517            515      830            819      227         2        11
2       UA      533            529      850            830      227         4        20
3       AA      542            540      923            850      160         2        33
4       B6      544            545     1004           1022      183        -1       -18
5       DL      554            600      812            837      116        -6       -25
6       UA      554            558      740            728      150        -4        12
7       B6      555            600      913            854      158        -5        19
8       EV      557            600      709            723       53        -3       -14
9       B6      557            600      838            846      140        -3        -8
10      AA      558            600      753            745      138        -2         8
# ... with 336,766 more rows
```

Reorder variables

```
select(flights, time_hour, air_time, everything())
# A tibble: 336,776 × 19
             time_hour air_time  year month   day dep_time sched_dep_time dep_delay arr_time
                <dttm>    <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013-01-01 05:00:00      227  2013     1     1      517            515         2      830
2  2013-01-01 05:00:00      227  2013     1     1      533            529         4      850
3  2013-01-01 05:00:00      160  2013     1     1      542            540         2      923
4  2013-01-01 05:00:00      183  2013     1     1      544            545        -1     1004
5  2013-01-01 06:00:00      116  2013     1     1      554            600        -6      812
6  2013-01-01 05:00:00      150  2013     1     1      554            558        -4      740
7  2013-01-01 06:00:00      158  2013     1     1      555            600        -5      913
8  2013-01-01 06:00:00       53  2013     1     1      557            600        -3      709
9  2013-01-01 06:00:00      140  2013     1     1      557            600        -3      838
10 2013-01-01 06:00:00      138  2013     1     1      558            600        -2      753
# ... with 336,766 more rows, and 10 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
```

Renaming variables

```
rename(flights, newName = dep_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time  newName arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>    <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515        2      830            819        11
2   2013     1     1      533            529        4      850            830        20
3   2013     1     1      542            540        2      923            850        33
4   2013     1     1      544            545       -1     1004           1022       -18
5   2013     1     1      554            600       -6      812            837       -25
6   2013     1     1      554            558       -4      740            728        12
7   2013     1     1      555            600       -5      913            854        19
8   2013     1     1      557            600       -3      709            723       -14
9   2013     1     1      557            600       -3      838            846        -8
10  2013     1     1      558            600       -2      753            745         8
# ... with 336,766 more rows, and 10 more variables: carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
```

Your turn!

1) What happens if you include the name of a variable multiple times in a `select()` call?

2) What does the `any_of()` function do? Why might it be helpful in conjunction with this vector?

```
vars <- c(“year", "month", "day", "dep_delay", "arr_delay")
```

3) Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

```
select(flights, contains("TIME"))
```

Solution

```
# 1. what happens if you call a variable multiple times in select()
select(flights, month, month)
# A tibble: 336,776 × 1
   month
   <int>
1      1
2      1
3      1
4      1
5      1
6      1
7      1
```

Solution

```
# 2. what does any_of() do?
# Matches variable names in a character vector.

vars <- c("year", "month", "day", "dep_delay", "arr_delay")
select(flights, any_of(vars))

## # A tibble: 336,776 x 5
##     year month   day dep_delay arr_delay
##    <int> <int> <int>     <dbl>     <dbl>
##  1  2013     1     1         2        11
##  2  2013     1     1         4        20
##  3  2013     1     1         2        33
##  4  2013     1     1        -1       -18
##  5  2013     1     1        -6       -25
```

Solution

```r
#3 Does the result of running the following code surprise you? How do the
select helpers deal with case by default? How can you change that default?

# The default helper functions are insensitive to case. This can be changes
by setting `ignore.case=FALSE`.

select(flights, contains("TIME"))
select(flights, contains("TIME", ignore.case = FALSE))
```

`mutate()`

create new variables

create new variables

```r
# Smaller dataset for demo
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)


mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

Useful tips

```
# You can refer to columns that you've just created
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

```
# If you only want to keep the new variables
transmute(flights,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

Useful creation functions

- There are a wide variety of functions you can use with **`mutate()`**

Functions	Description
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank, etc`	ranking
`between`	are values between a and b?
`ntile`	bin values into buckets

arithmetic

```
transmute(flights,
  normalized_delay =
    dep_delay/(mean(dep_delay, na.rm = TRUE)))
```

Functions	Description
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank, etc`	ranking
`between`	are values between a and b?
`ntile`	bin values into buckets

Modular arithmetic

```r
transmute(flights,
  dep_time,
  hour = dep_time %/% 100,
  minute = dep_time %% 100
)
```

```
%/% (Integer division)
%% (Remainder)
```

```
#> # A tibble: 336,776 x 3
#>   dep_time  hour minute
#>      <int> <dbl>  <dbl>
#> 1      517     5     17
#> 2      533     5     33
#> 3      542     5     42
#> 4      544     5     44
#> 5      554     5     54
```

Functions	Description
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank, etc`	ranking
`between`	are values between a and b?
`ntile`	bin values into buckets

non-linear transformations

```
transmute(flights,
  log_air_time = log2(air_time),
  exp_delay = exp(dep_delay))
```

Functions	Description
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank, etc`	ranking
`between`	are values between a and b?
`ntile`	bin values into buckets

offsets & cumulative aggregates

```r
transmute(flights,
  dep_delay = dep_delay,
  lag_delay = lag(dep_delay),
  sum_delay = cumsum(dep_delay))
```

```
## # A tibble: 336,776 x 3
##    dep_delay lag_delay sum_delay
##        <dbl>     <dbl>     <dbl>
##  1         2        NA         2
##  2         4         2         6
##  3         2         4         8
##  4        -1         2         7
##  5        -6        -1         1
```

Functions	Description
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank, etc`	ranking
`between`	are values between a and b?
`ntile`	bin values into buckets

Ranking

```r
y <- c(1, 2, 2, NA, 3, 4)

min_rank(y)
#> [1]  1  2  2 NA  4  5

row_number(y)
#> [1]  1  2  3 NA  4  5

dense_rank(y)
#> [1]  1  2  2 NA  3  4

percent_rank(y)
#> [1] 0.00 0.25 0.25   NA 0.75 1.00

cume_dist(y)
#> [1] 0.2 0.6 0.6  NA 0.8 1.0
```

Functions	Description
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank, etc`	ranking
`between`	are values between a and b?
`ntile`	bin values into buckets

ntile

```r
transmute(flights,
  arr_delay = arr_delay,
  bucket = ntile(arr_delay, 10))
```

```
## # A tibble: 336,776 x 2
##    arr_delay bucket
##        <dbl>  <int>
##  1        11      8
##  2        20      8
##  3        33      9
##  4       -18      3
##  5       -25      2
##  6        12      8
##  7        19      8
##  8       -14      3
##  9        -8      5
## 10         8      7
## # ... with 336,766 more rows
```

Functions	Description
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank, etc`	ranking
`between`	are values between a and b?
`ntile`	bin values into buckets

Your turn!

1) Create a new variable `distance_km` that converts distance in miles to kilometres
2) Create a `time_per_km` variable based on `air_time` and `distance_km`.

Solution

```
transmute(flights,
          distance_km = distance * 1.60934,
          time_per_km = air_time / distance_km)
```

`summarise()` & `group_by()`

Create grouped summaries

summarise()

- **summarise()** collapses a data frame into a single row

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))

#> # A tibble: 1 x 1
#>   delay
#>   <dbl>
#> 1  12.6
```

Why is this important?
Try without this argument.

- **summarise()** is not really useful without pairing it with **group_by()**

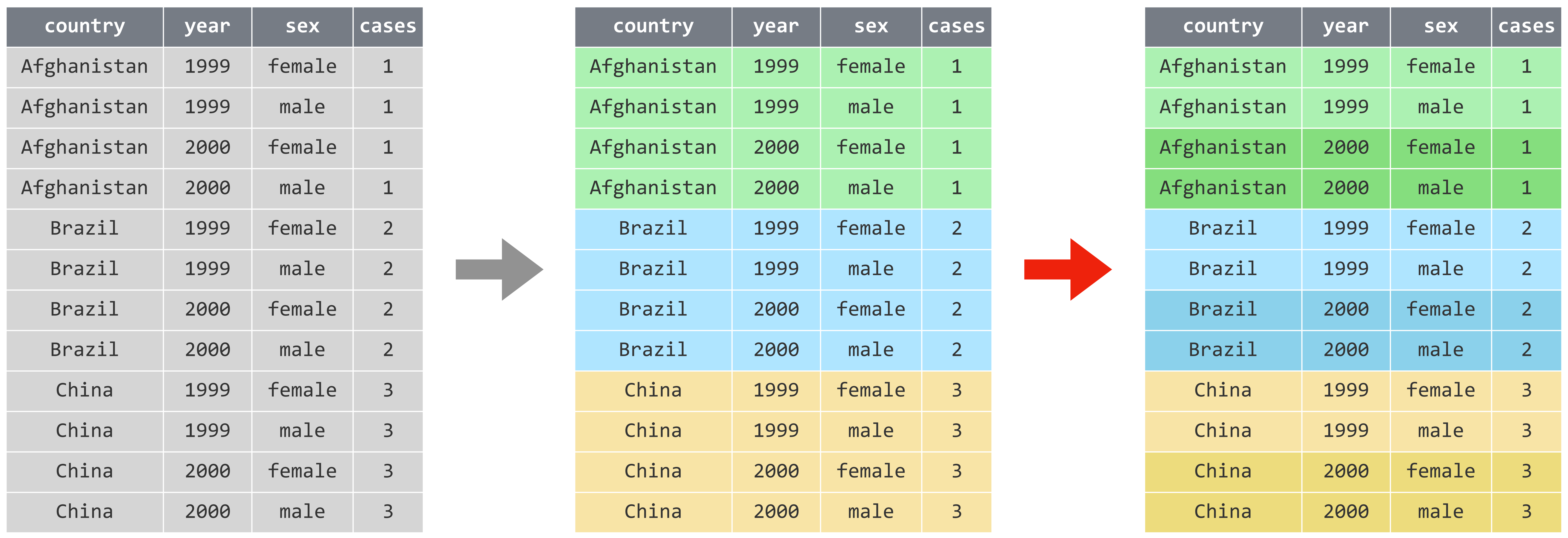
group_by()

country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

```
group_by(data, country)
```

group_by()



country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

`group_by(data, country)`

`group_by(data, country, year)`

group_by()

country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

group_by(data, country)

country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

group_by(data, country, year)

country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

ungroup(data)

country	year	sex	cases
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

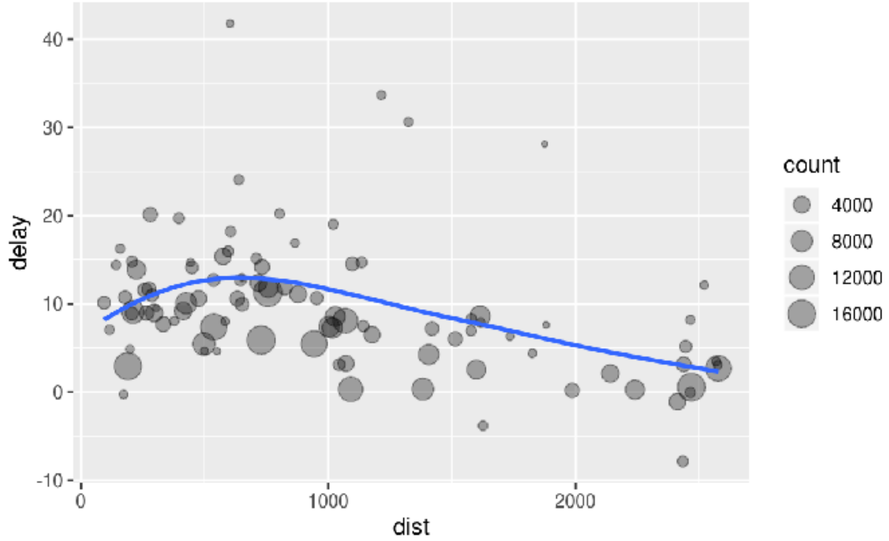
Pipe operator

Chaining functions together

Example

- Imagine that we want to explore the relationship between the distance and average delay for each location. Using what you know about dplyr, you might write code like this:

```
by_dest <- group_by(flights, dest)

delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)

delay <- filter(delay, count > 20, dest != “HNL")

ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```



Example

- Imagine that we want to explore the relationship between the distance and average delay for each location. Using what you know about dplyr, you might write code like this:

```
by_dest <- group_by(flights, dest)
```

grouping by destination

```
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
```

summarising count, distance and arrival delay

```
delay <- filter(delay, count > 20, dest != “HNL")
```

filtering out low counts and Honolulu

```
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```

creating a plot

Example

```r
by_dest <- group_by(flights, dest)

delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)

delay <- filter(delay, count > 20, dest != “HNL")
```

```r
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
```

Command/CTRL + Shift + M

Pipe operator

```r
# without pipe to subset "suv" cars
filter(mpg, class == "suv")

# equivalent form with pipe
mpg %>% filter(class  == "suv")
```

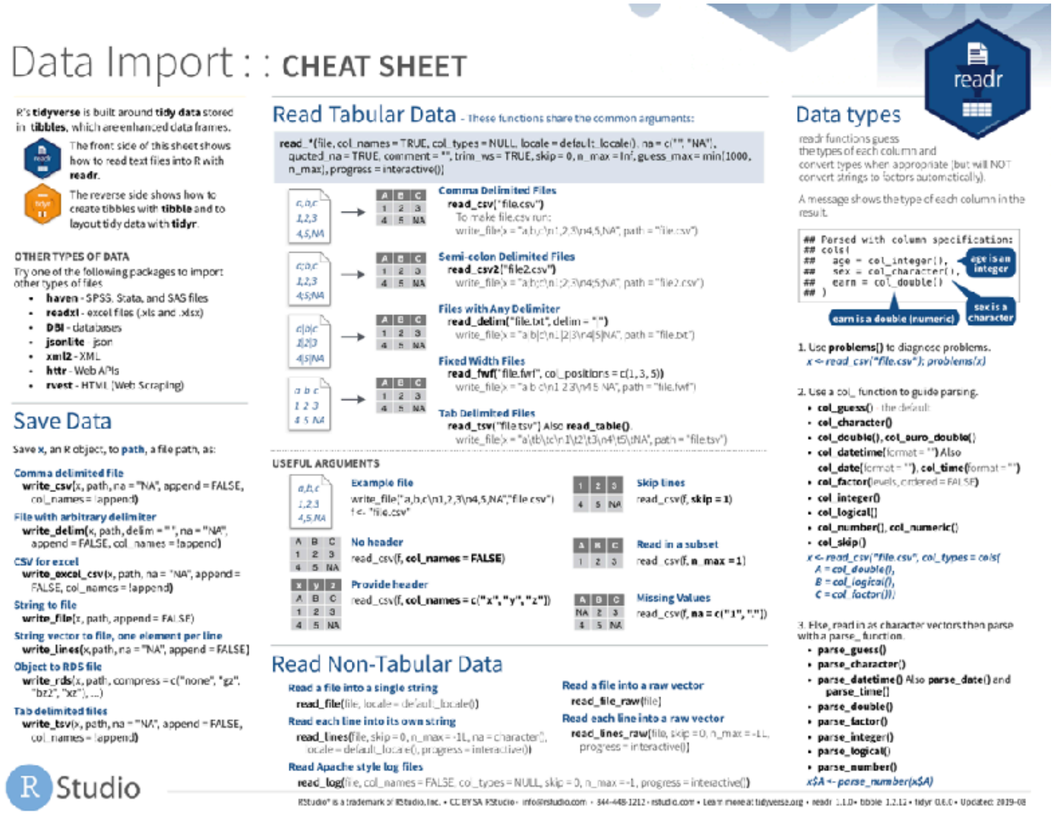
Command/CTRL + Shift + M

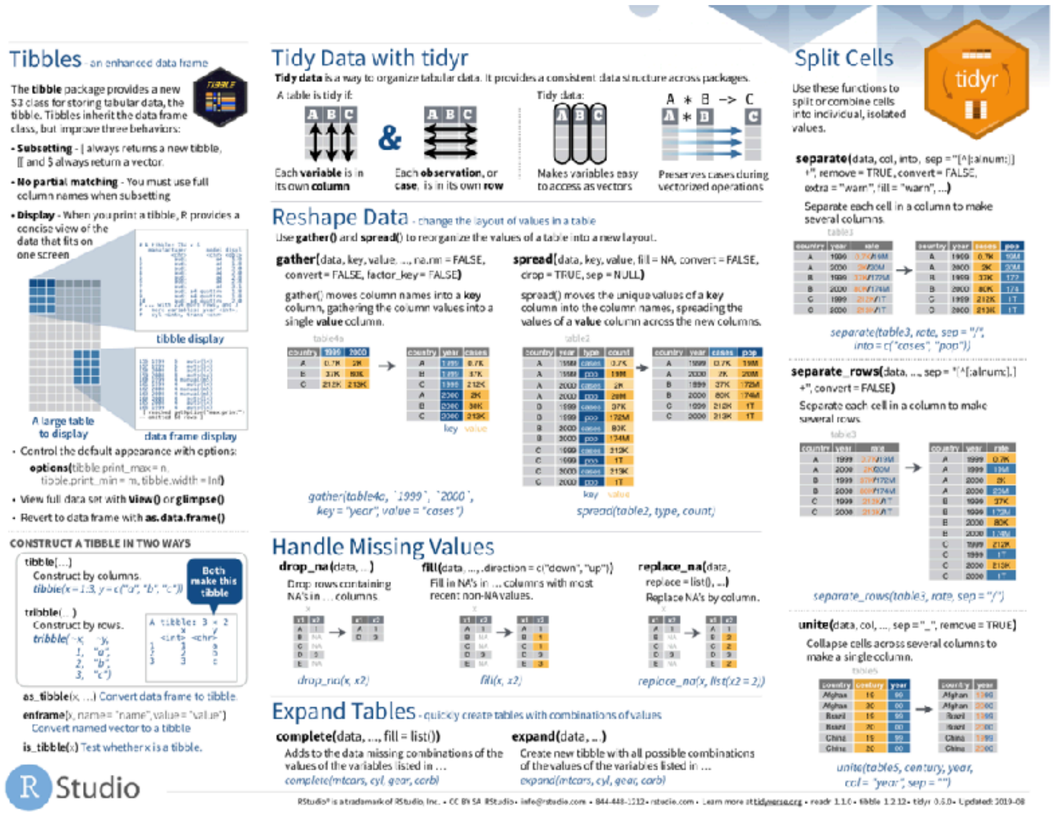
“Tidy datasets are all alike,
but every messy dataset is messy in its own way.”
– Hadley Wickham

Tidy data

`data structure`

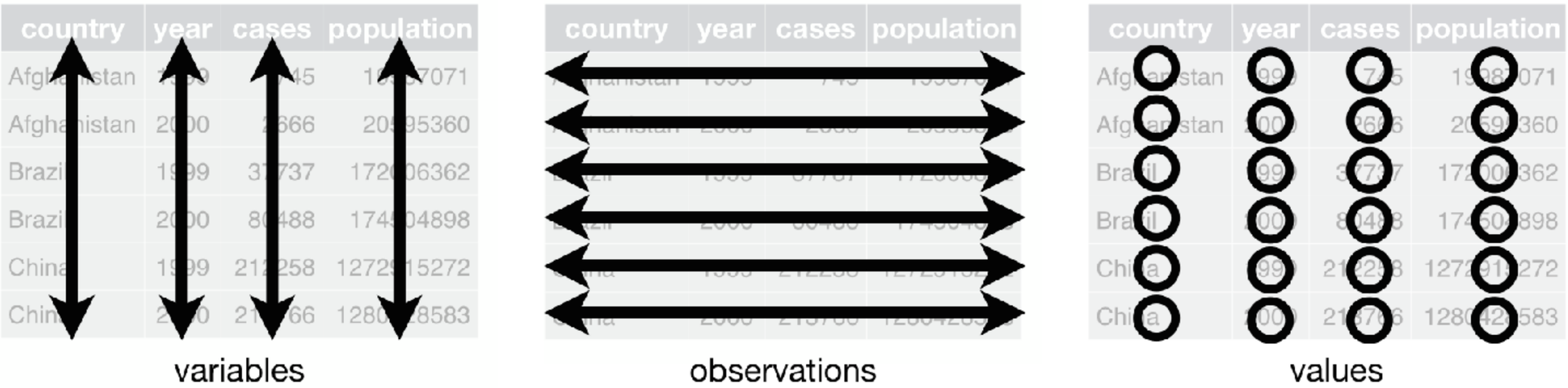
Data Import Cheetsheet





3 rules of tidy data

- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell



https://r4ds.had.co.nz/tidy-data.html

Your turn!

Is this a tidy table?

Table 1:

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Your turn!

Is this a tidy table?

Table 2:

country	year	type	count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

Your turn!

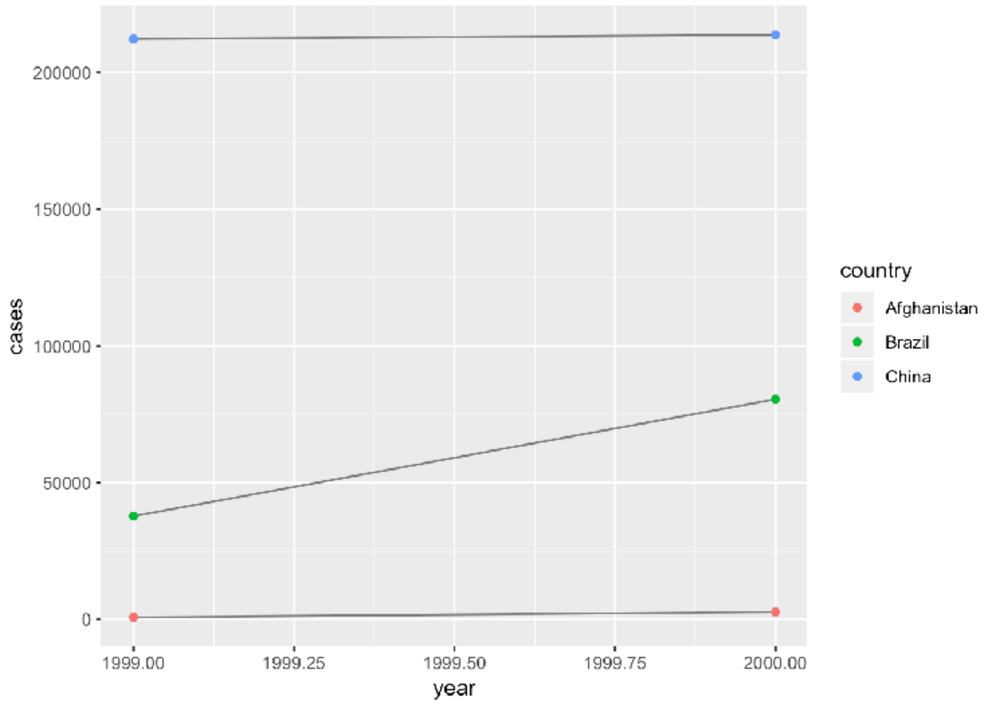
Is this a tidy table?

Table 3:

country	year	rate
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

Tidy data example

```r
# using a sample dataset from tidyr  (tidyr::table1)
# check the table View(tidyr::table1)
ggplot(data  = table1, mapping = aes(year, cases)) +
  geom_line(mapping = aes(group = country), colour = "grey50") +
  geom_point(mapping = aes(colour = country))
```



4 key functions

Function	Old name	Description
pivot_longer()	gather()	transforms data from wide to long
pivot_wider()	spread()	transforms data from long to wide
separate()		splits a single column into multiple columns
unite()		combines multiple columns into a single column

variables

observations

values

`pivot_longer()`

Transform data from wide to long

pivot_longer() example

```
table4a %>%
  pivot_longer(cols = c('1999', '2000'), names_to = "year", values_to = "cases")
```

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

names_to

values_to

pivot_longer()

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

Your turn!

1. Download the data (**bomber_wide.rds**) in the data folder
2. Import the .rds with `read_rds()` function
3. Reshape this data from wide to long

Solution

```
# One example
df <-  read_rds("Lecture_8_data/bomber_wide.rds")
df %>%
  pivot_longer(cols = `1996`:`2014`,
               names_to = "year",
               values_to = "value"
```

`pivot_wider()`

Transform data from long to wide

pivot_wider() example

```
table2 %>%
  pivot_wider(names_from = type, values_from = count)
```

		names_from	values_from
country	year	type	count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

pivot_wider()

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Your turn!

1. Download the data (**bomber_long.rds**) in the data folder
2. Import the .rds with `read_rds()` function
3. Reshape this data from long to wide

Solution

```
df <-  read_rds("Lecture_8_data/bomber_long.rds")
df %>%
  pivot_wider( names_from = Output,
               values_from = Value)
```

`separate()`

split a single column into multiple columns

separate() example

```
table3 %>%
  separate(rate, into = c("cases", "population"), sep="/", convert = TRUE)
```

country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

table3

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

separate() example

```r
table3 %>%
  separate(year, into = c("century", "year"), sep = 2)

#> # A tibble: 6 x 4
#>   country     century year  rate
#>   <chr>       <chr>   <chr> <chr>
#> 1 Afghanistan 19      99    745/19987071
#> 2 Afghanistan 20      00    2666/20595360
#> 3 Brazil      19      99    37737/172006362
#> 4 Brazil      20      00    80488/174504898
#> 5 China       19      99    212258/1272915272
#> 6 China       20      00    213766/1280428583
```

Your turn!

1. Download the data (**bomber_combined.rds**) in the data folder
2. Import the .rds with **`read_rds()`** function
3. Separate the AC variable into “Type” and “MD”

Solution

```r
read_rds("data/bomber_combined.rds") %>%
  separate(AC, into = c("Type", "MD"), sep = " ")
```

```
     Type   MD   FY      Cost    FH   Gallons
1  Bomber  B-1 1996  72753781 26914  88594449
2  Bomber  B-1 1997  71297263 25219  85484074
3  Bomber  B-1 1998  84026805 24205  85259038
4  Bomber  B-1 1999  71848336 23306  79323816
5  Bomber  B-1 2000  58439777 25013  86230284
6  Bomber  B-1 2001  94946077 25059  86892432
```

`unite()`

combine multiple columns into a single column

unite() example

```
table5 %>%
  unite(new, century, year, sep = "")
```

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

country	century	year	rate
Afghanistan	19	99	745 / 19987071
Afghanistan	20	0	2666 / 20595360
Brazil	19	99	37737 / 172006362
Brazil	20	0	80488 / 174504898
China	19	99	212258 / 1272915272
China	20	0	213766 / 1280428583

Your turn!

1. Download the data (bomber_prefix.rds) in the data folder
2. Import the .rds with `read_rds()` function
3. Unite the prefix and number columns into a “MD” variable with “-“ as separator

Solution

```
read_rds("data/bomber_combined.rds") %>%
  unite(MD, prefix, number, sep = “-”)
```

```
      Type   MD   FY  Output     Value
1   Bomber  B-1 1996      FH     26914
2   Bomber  B-1 1997      FH     25219
3   Bomber  B-1 1998      FH     24205
4   Bomber  B-1 1999      FH     23306
5   Bomber  B-1 2000      FH     25013
```

Your turn!

1. Download the data (**bomber_mess.rds**) in the data folder
2. Import the .rds with `read_rds()` function
3. Clean this data up so it looks like:

```
# A tibble: 57 × 6
     Type    MD    FY      Cost    FH   Gallons
*   <chr> <chr> <chr>     <int> <int>     <int>
1  Bomber   B-1  1996  72753781 26914  88594449
2  Bomber   B-1  1997  71297263 25219  85484074
3  Bomber   B-1  1998  84026805 24205  85259038
4  Bomber   B-1  1999  71848336 23306  79323816
5  Bomber   B-1  2000  58439777 25013  86230284
6  Bomber   B-1  2001  94946077 25059  86892432
7  Bomber   B-1  2002  96458536 26581  89198262
8  Bomber   B-1  2003  68650070 21491  74485788
9  Bomber   B-1  2004 101895634 28118 101397707
10 Bomber   B-1  2005 124816690 21859  78410415
# ... with 47 more rows
```

Solution

```r
read_rds("data/bomber_mess.rds") %>%
  unite(col = MD, prefix:number, sep = "-") %>%
  separate(Metric, into = c("FY", "Output")) %>%
  spread(Output, Value) %>%
  as_tibble()
```

```
# A tibble: 57 × 6
     Type    MD    FY      Cost    FH   Gallons
*   <chr> <chr> <chr>     <int> <int>     <int>
1  Bomber   B-1  1996  72753781 26914  88594449
2  Bomber   B-1  1997  71297263 25219  85484074
3  Bomber   B-1  1998  84026805 24205  85259038
4  Bomber   B-1  1999  71848336 23306  79323816
5  Bomber   B-1  2000  58439777 25013  86230284
6  Bomber   B-1  2001  94946077 25059  86892432
7  Bomber   B-1  2002  96458536 26581  89198262
8  Bomber   B-1  2003  68650070 21491  74485788
9  Bomber   B-1  2004 101895634 28118 101397707
10 Bomber   B-1  2005 124816690 21859  78410415
# ... with 47 more rows
```

Data Wrangling

Practice

Your turn!

Let's apply **`dplyr`** and **`tidyr`** functions you have learned.

- **`tidyr::who`** is a dataset of TB cases browken down by year, country, age, gender and diagnosis methods
- Examine the table and its documentation, and clean the data into a tidy data
- Hint:
  1. Gather columns and remove missing values
  2. Separate the values like "new_sp_m014" into columns
  3. Separate the gender and age
  4. Drop redundant columns using **`dplyr::select()`**
- Create a graph with the tidy data

Solution

1. Pivot lonter new_sp_m014:newrel_f65 columns and remove missing values

```
who1 <- who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
  )
```

```
## # A tibble: 76,046 x 6
##    country     iso2  iso3   year key         cases
##  * <chr>       <chr> <chr> <int> <chr>       <int>
##  1 Afghanistan AF    AFG    1997 new_sp_m014     0
##  2 Afghanistan AF    AFG    1998 new_sp_m014    30
##  3 Afghanistan AF    AFG    1999 new_sp_m014     8
##  4 Afghanistan AF    AFG    2000 new_sp_m014    52
##  5 Afghanistan AF    AFG    2001 new_sp_m014   129
```

Solution

2. Clean up the key column, e.g. new_sp_m014
   - `new` encodes new cases of TB
   - next 3 letters describe the type of TB:
     - `rel` = relapse
     - `ep` = extraplumonary TB
     - `sn` = could not be diagnosed by a plumonary smear (smear negagive)
     - `sp` = smear positive
   - `m` = males, `f` = females
   - age group: `014` = 1 - 14 years old age group

```
## # A tibble: 76,046 x 6
##    country     iso2  iso3   year key         cases
##  * <chr>       <chr> <chr> <int> <chr>       <int>
##  1 Afghanistan AF    AFG    1997 new_sp_m014     0
##  2 Afghanistan AF    AFG    1998 new_sp_m014    30
```

Solution

2. But, there is an inconsistency in naming the key.
   - eg. `new_rel` and `newrel`
   - use `stringr::str_replace()` function to replace `newrel` with `new_rel`

```
who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
```

```
  country     iso2  iso3   year key         cases
  <chr>       <chr> <chr> <int> <chr>       <int>
1 Afghanistan AF    AFG    1997 new_sp_m014     0
2 Afghanistan AF    AFG    1998 new_sp_m014    30
3 Afghanistan AF    AFG    1999 new_sp_m014     8
4 Afghanistan AF    AFG    2000 new_sp_m014    52
5 Afghanistan AF    AFG    2001 new_sp_m014   129
```

Solution

2. Now use `separate()` to split key into 3 columns

```
who3  <- who2 %>%
  separate(key, into = c("new", "type", "sex-age"), sep ="_")
```

```
  key                  ##    country     iso2  iso3   year new   type  `sex-age` cases
  <chr>                ##    <chr>       <chr> <chr> <int> <chr> <chr> <chr>     <int>
1 new_sp_m014          ##  1 Afghanistan AF    AFG    1997 new   sp    m014          0
2 new_sp_m1524         ##  2 Afghanistan AF    AFG    1998 new   sp    m014         30
3 new_sp_m2534         ##  3 Afghanistan AF    AFG    1999 new   sp    m014          8
4 new_sp_m3544         ##  4 Afghanistan AF    AFG    2000 new   sp    m014         52
5 new_sp_m4554         ##  5 Afghanistan AF    AFG    2001 new   sp    m014        129
```

Solution

3. Separate `sex-age` into 2 columns

```
who4 <- who3 %>%
  separate(`sex-age`, into = c("sex", "age"), sep = 1)
```

```
`sex-age`
  <chr>
 1 m014
 2 m1524
 3 m2534
 4 m3544
 5 m4554
```

```
##    country     iso2  iso3   year new   type  sex   age   cases
##    <chr>       <chr> <chr> <int> <chr> <chr> <chr> <chr> <int>
##  1 Afghanistan AF    AFG    1997 new   sp    m     014       0
##  2 Afghanistan AF    AFG    1998 new   sp    m     014      30
##  3 Afghanistan AF    AFG    1999 new   sp    m     014       8
##  4 Afghanistan AF    AFG    2000 new   sp    m     014      52
##  5 Afghanistan AF    AFG    2001 new   sp    m     014     129
```

Solution

4. Drop redundant columns:

```
who5 <- who4 %>%
  select(-new, -iso2, -iso3)
```

```
  country      year var   sex   age   value
  <chr>       <int> <chr> <chr> <chr> <int>
1 Afghanistan  1997 sp    m     014       0
2 Afghanistan  1998 sp    m     014      30
3 Afghanistan  1999 sp    m     014       8
4 Afghanistan  2000 sp    m     014      52
5 Afghanistan  2001 sp    m     014     129
```

Solution

You can build up a complete piped operations as shown below:

```
who %>%
  pivot_longer(cols = new_sp_m014:newrel_f65, names_to=“key”,
               values_to =“cases”, values_drop_na = TRUE) %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel")) %>%
  separate(key, c("new", "var", "sexage")) %>%
  separate(sexage, c("sex", "age"), sep = 1) %>%
  select(-new, -iso2, -iso3)
```

Importing data

`readr` package

Tabular data

- to import tabular data, use functions from **`readr`** package (**`tidyverse`**)
  - It is much faster than base R function such as `read.csv()`
  - It loads as **`tibble`** instead of R's traditional `data.frame`

Function	Description
`read_csv()`	comma separated (csv) files
`read_tsv()`	tab separated files
`read_delim()`	general delimited files
`read_fwf()`	fixed width files
`read_table()`	tabular files where columns are separated by white-space

example

```r
car_data <- read_csv(file = "data/mtcars.csv")
```

```
## Parsed with column specification:
## cols(
##   mpg = col_double(),
##   cyl = col_integer(),
##   disp = col_double(),
##   hp = col_integer(),
##   drat = col_double(),
##   wt = col_double(),
##   qsec = col_double(),
##   vs = col_integer(),
##   am = col_integer(),
##   gear = col_integer(),
##   carb = col_integer()
## )
```

example

```r
car_cata <- read_csv(file = "data/mtcars.csv", col_types =
  cols(
    mpg = col_double(),
    cyl = col_integer(),
    disp = col_double(),
    hp = col_integer(),
    drat = col_double(),
    vs = col_integer(),
    wt = col_double(),
    qsec = col_double(),
    am = col_integer(),
    gear = col_integer(),
    carb = col_integer()
  )
)
```

Excel file

- to load *.xls* or *.xlsx* files, use **`read_excel()`** function in the **`readxl`** package.
  - You can specify the sheet of of an Excel spreadsheet.
  - Read its documentation for more details.
  - Installed as a part of `tidyverse`, but not a core `tidyverse` package

```r
# Need to load the package explicitly
library(readxl)

# Loading the first sheet of an Excel file
data <- read_excel("data/datasets.xlsx", 1)
```

Exporting images

ggsave

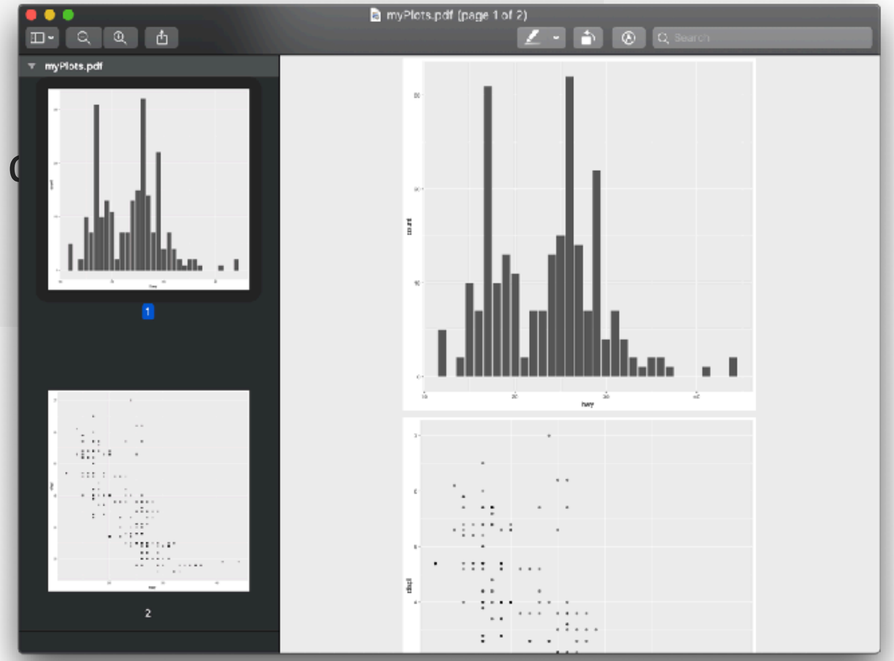
- call **`ggsave()`** after calling **`ggplot()`** to save the last ggplot object:

```
ggplot(mtcars, mapping = aes(x = wt, y = mpg))+
  geom_point(shape = 1)
# save PDF
ggsave("myplot.pdf", width = 8, height = 8, units = "cm", useDingbats = F)
# save PNG
ggsave("myplot.png", width = 8, height = 8, units = "cm")
```

create a PDF

- call **ggplot()** functions between **pdf()** and **dev.off()**
- Each plot on a sperate page

```
pdf("myPlots.pdf", width = 8, height = 8)

# plot your graphs here
ggplot(data = mpg) + geom_bar(aes(x = hwy))
ggplot(data = mpg) + geom_point(aes(x = hwy, y = c

dev.off()
```



Axes/graph labels

ggplot2

Labels

ggtitle(“Title”)	Add a main title above the plot
xlab(“x-axis”)	Change the label on the x-axis
ylab(“y-axis”)	Change the label on the y-axis
labs(…)	**title** = “main title above the plot”, **subtitle** = “subtitle below title”, **caption** = “caption below plot”, **x** = “x-axis”, **y** = “y-axis”
annotate()	annotate(geom=“text”, x = 10, y = 10, label=“A”)

Scales

ggplot2

Scales

- Scales map data values to the visual values of an aesthetic.
- To change a mapping, add a new scale:

aesthetic to adjust

types of scale

```
ggplot(data = mpg) +
  geom_bar( mapping = aes(x = drv, fill = drv)) +
  scale_fill_manual(
    values = c("skyblue", "royalblue", "navy"),
    limits = c("4", "f", "r"),
    breaks = c("4", "f", "r"),
    name = "drv",
    labels = c("4 (4wd)", "f (fwd)", "r (rwd)")
  )
```

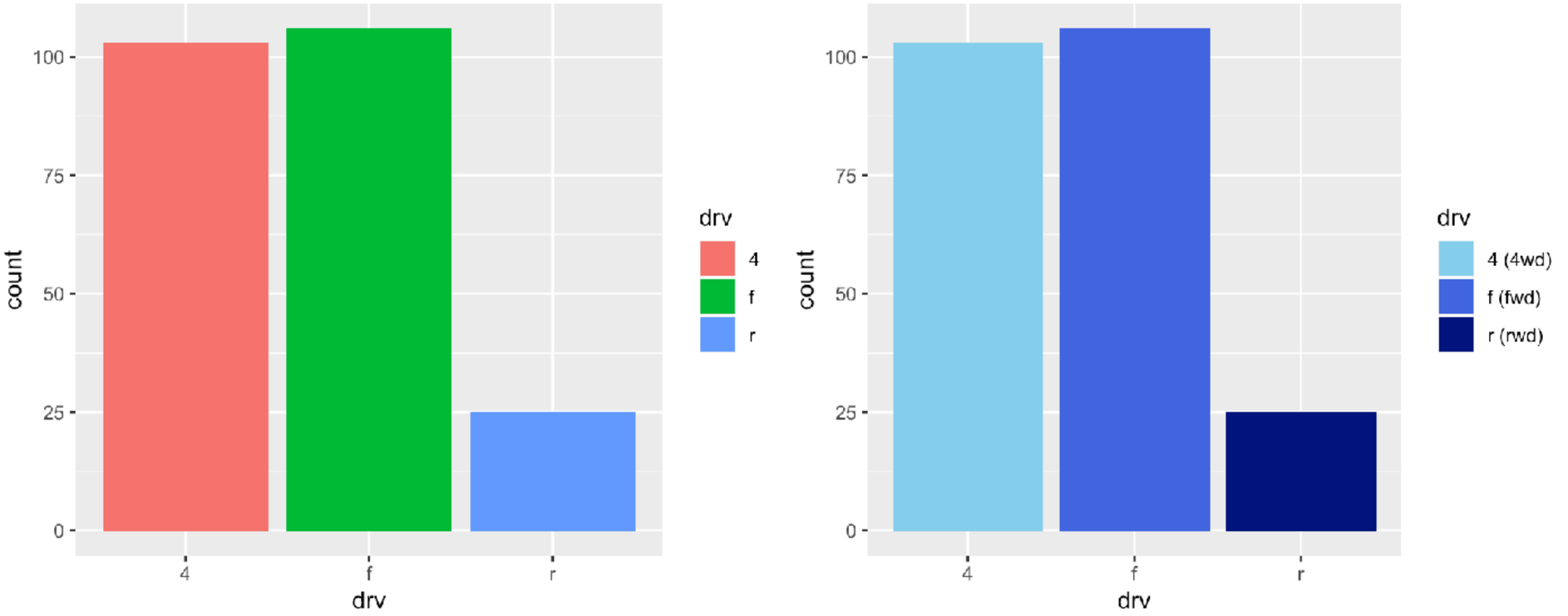
scale-specific arguments

range of values to include in mapping

breaks to use in legend/axis

Title to use in legend/axis

labels to use in legend/axis




```r
p1 <- ggplot(data = mpg)+
  geom_bar( mapping = aes(x = drv, fill = drv))

p2 <- ggplot(data = mpg)+
  geom_bar( mapping = aes(x = drv, fill = drv)) +
  scale_fill_manual(
    values = c("skyblue", "royalblue", "navy"),
    limits = c("4", "f", "r"),
    breaks = c("4", "f", "r"),
    name = "drv",
    labels = c("4 (4wd)", "f (fwd)", "r (rwd)")
  )
grid.arrange(p1, p2, ncol = 2)
```

General purpose scales

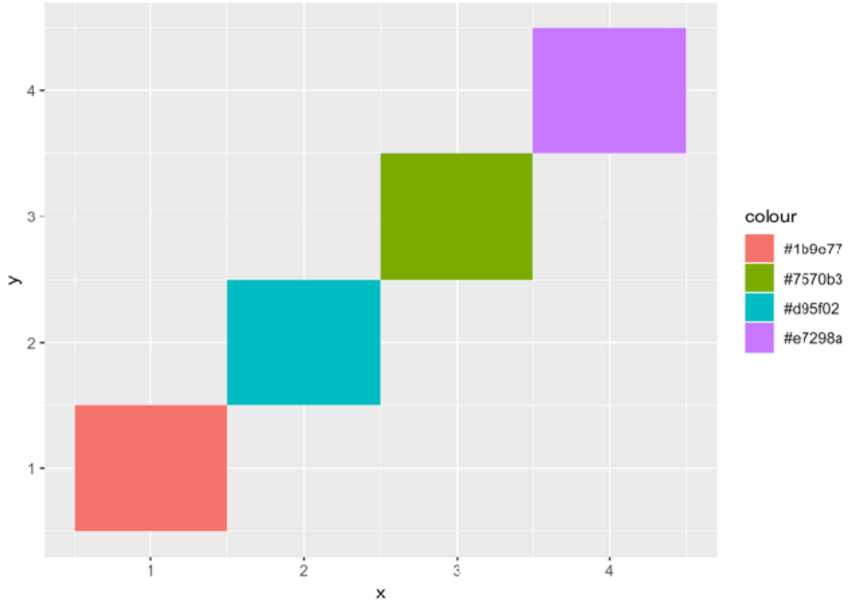
`scale_*_continuous()`	map continuous values to visual scale
`scale_*_discrete()`	map discrete values to visual scale
`scale_*_identity()`	use data values as visual ones
`scale_*_manual(values = c())`	map discrete values to manually chosen visual ones
`scale_*_date(date_labels = “%m/%d”, date_breaks = “2 weeks”)`	treat data values as dates (class = `Date`)
`scale_*datetime()`	treat data x values as date times. (class = `POSIXct`)

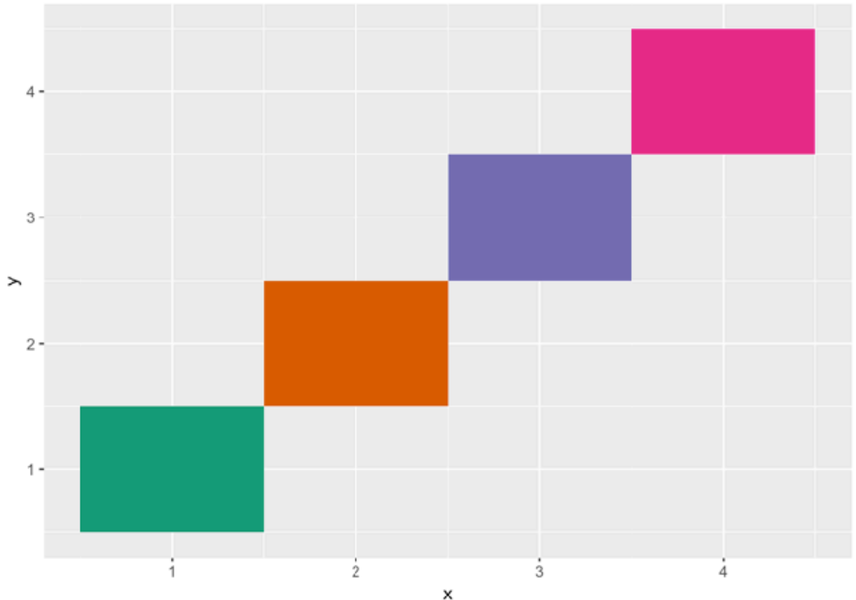
`* = alpha, color, fill, linetype, shape, size, x, y`

scale_*_identity()

```r
df <- data.frame(
  x = 1:4,
  y = 1:4,
  colour = c('#1b9e77','#d95f02','#7570b3','#e7298a')
)
ggplot(df, aes(x, y)) + geom_tile(aes(fill = colour))

ggplot(df, aes(x, y)) + geom_tile(aes(fill = colour)) +
  scale_fill_identity()
```

```
  x y  colour
1 1 1 #1b9e77
2 2 2 #d95f02
3 3 3 #7570b3
4 4 4 #e7298a
```



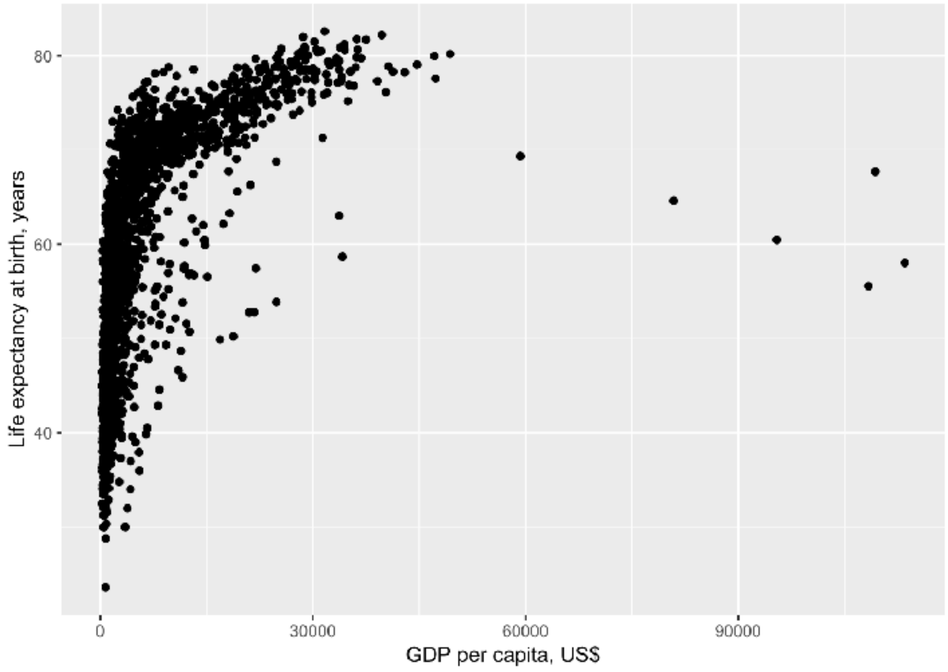


X and Y location scales

`scale_x_log10()`	plot x on log10 scale
`scale_x_reverse()`	reverse direction of x-axis
`scale_x_sqrt()`	plot x on square root scale

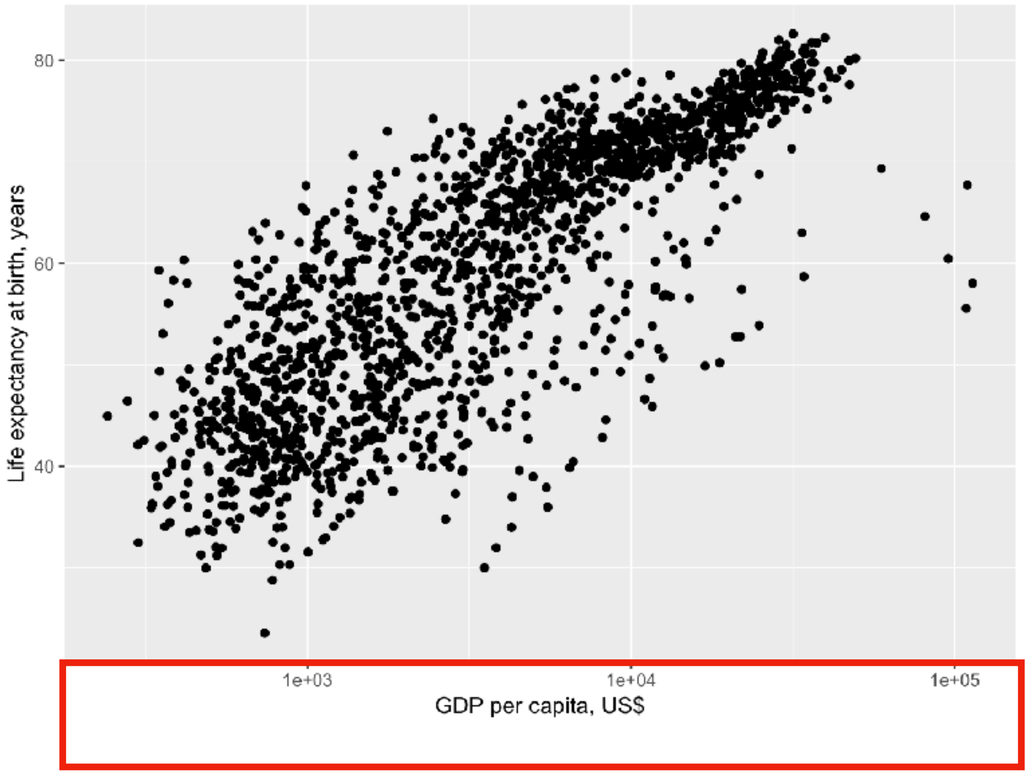
gapminder

```r
# install.packages("gapminder")
library(gapminder)

# looking up ?gapminder to see the documentation

# scatter plot of GDP per capita and life expectancy
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



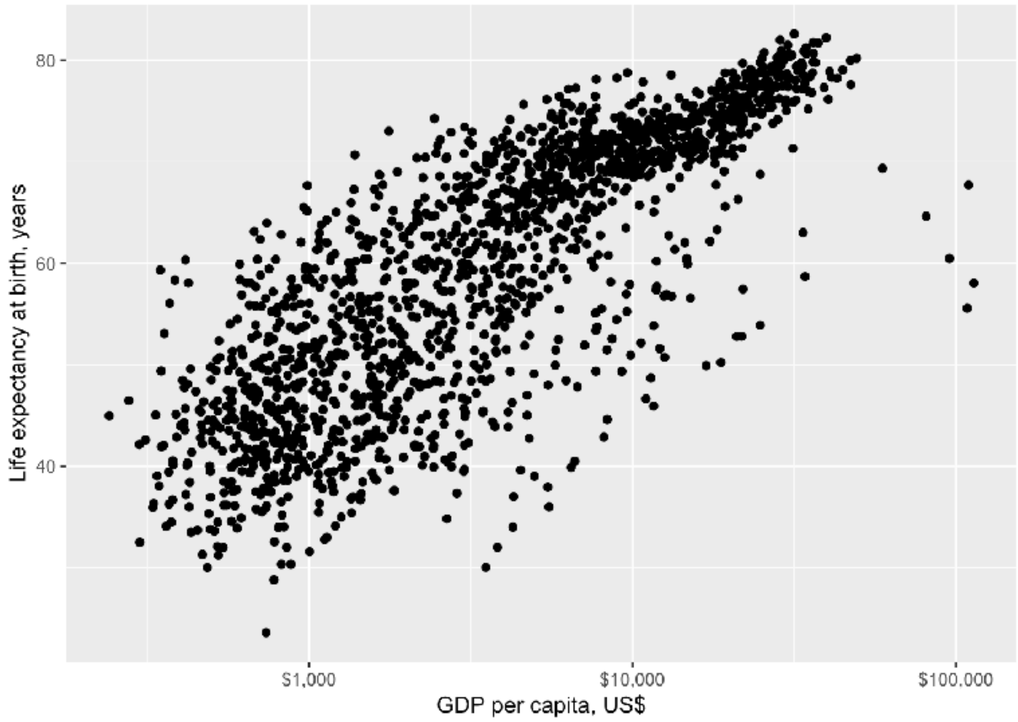
gapminder

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  scale_x_log10() +
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



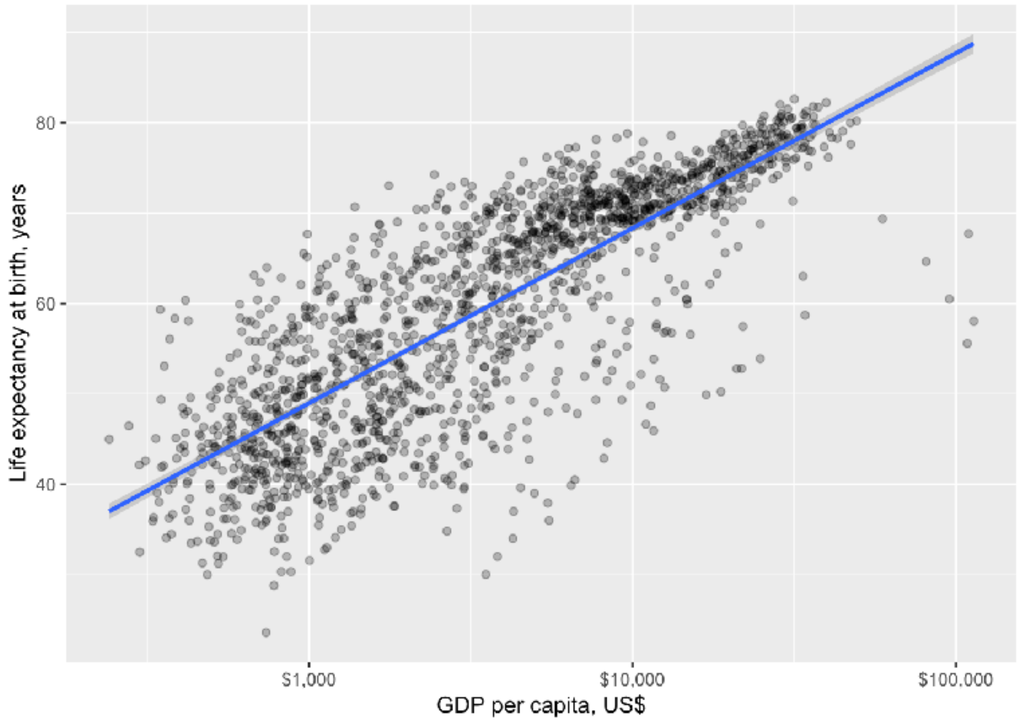
gapminder

```
# handy function scale::dollar, scale::comma by passing a function
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  scale_x_log10(labels  = scales::dollar) +
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



gapminder

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point(alpha = 0.3) +
  geom_smooth(method  = "gam") +
  scale_x_log10(labels  = scales::dollar) +
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



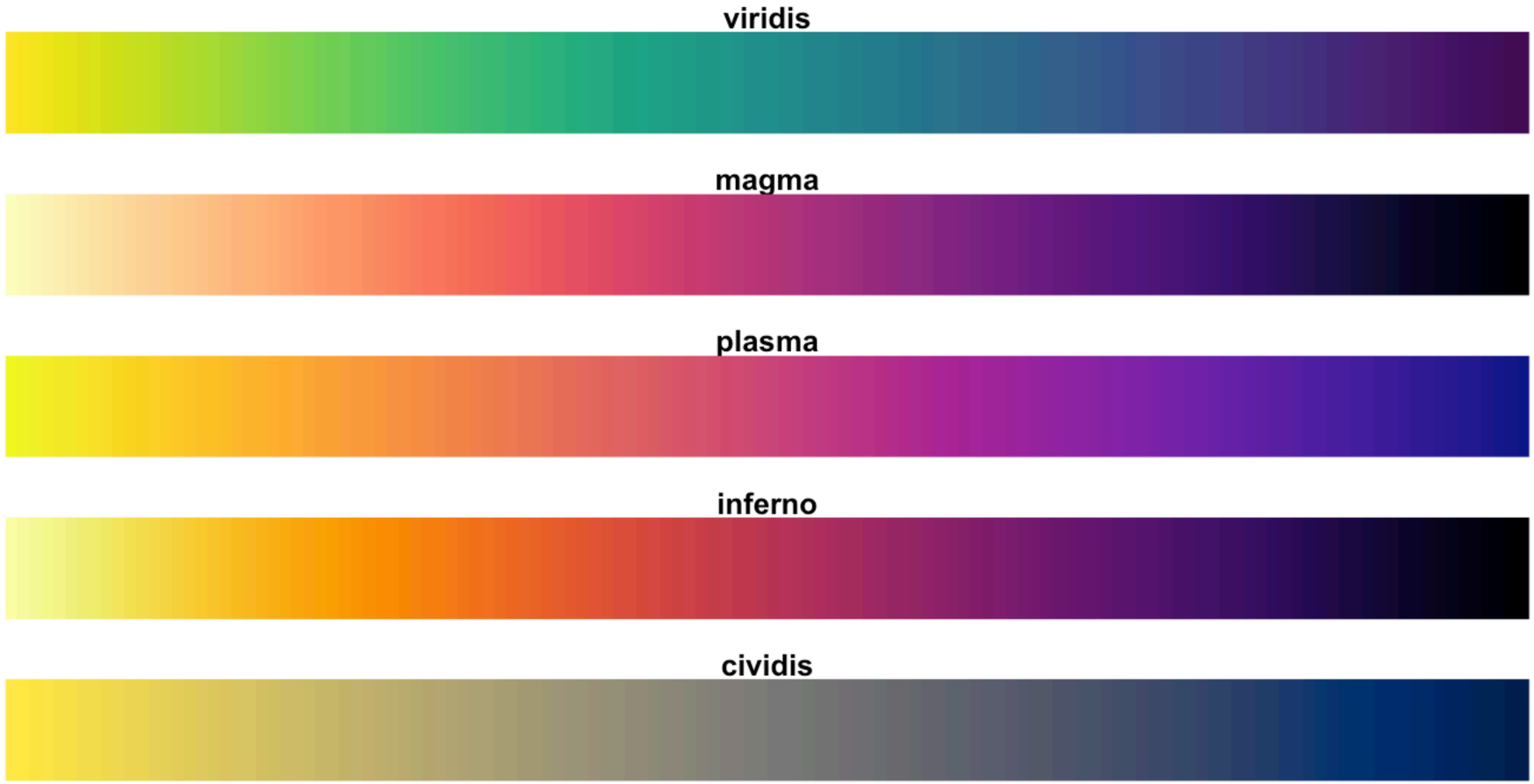
Colour scales

ggplot2

Colour scale

- Is your data **`discrete`** or **`continuous`**?
- Colourblind-friendly palette
  - Package: **`viridis`** (comes with ggplot2 v3.0.0)
    - https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html
  - Package: **`RColorBrewer`**
    - display.brewer.all()
    - display.brewer.pal(8, "Blues")

viridis



Comparisons

Base R: rainbow.colours

Base R: heat.colours

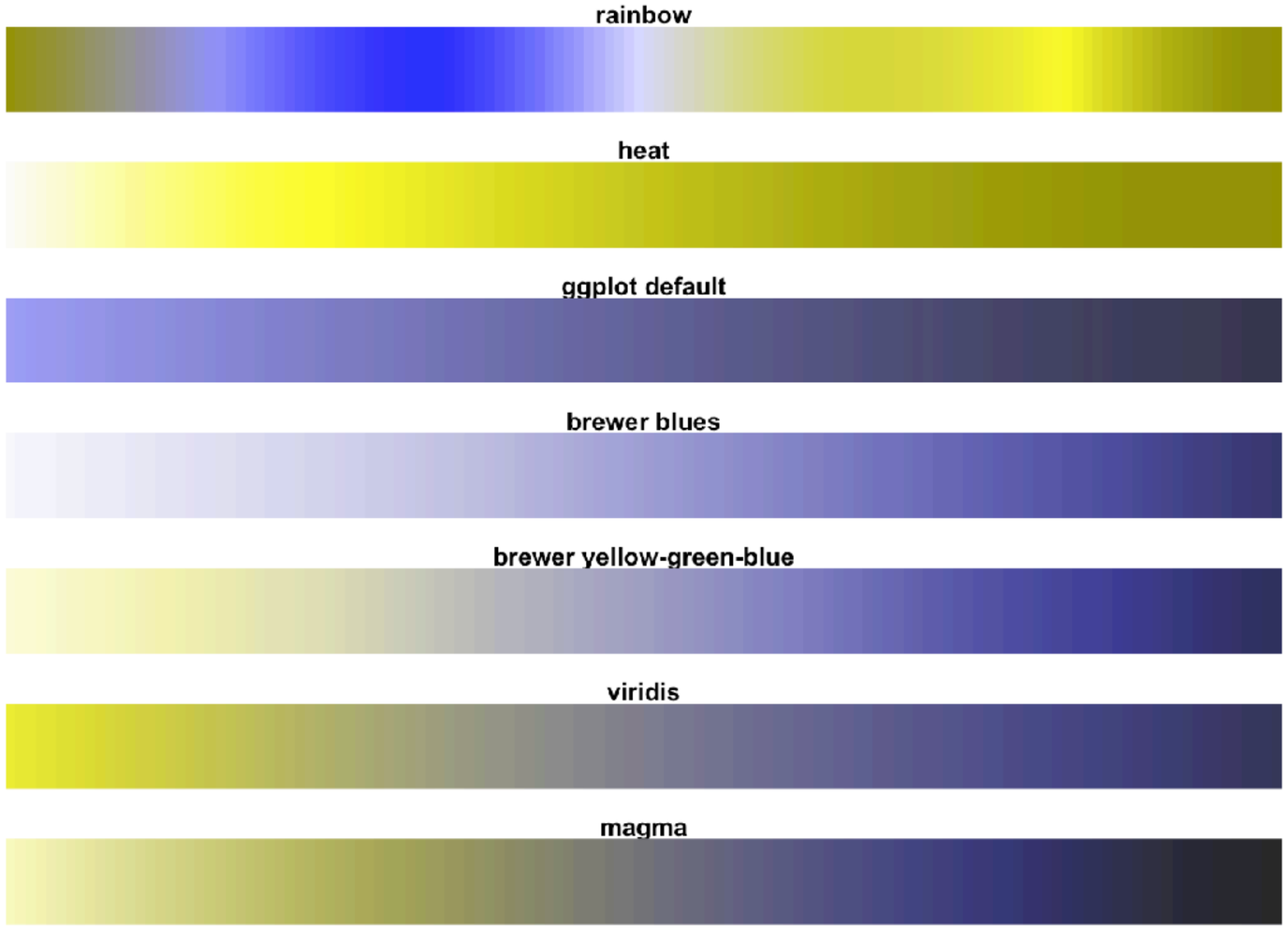
ggplot2 default

colorbrewer blues

colorbrewer yellow-green-blue

Green-blind (Deuteranopia)

Comparisons

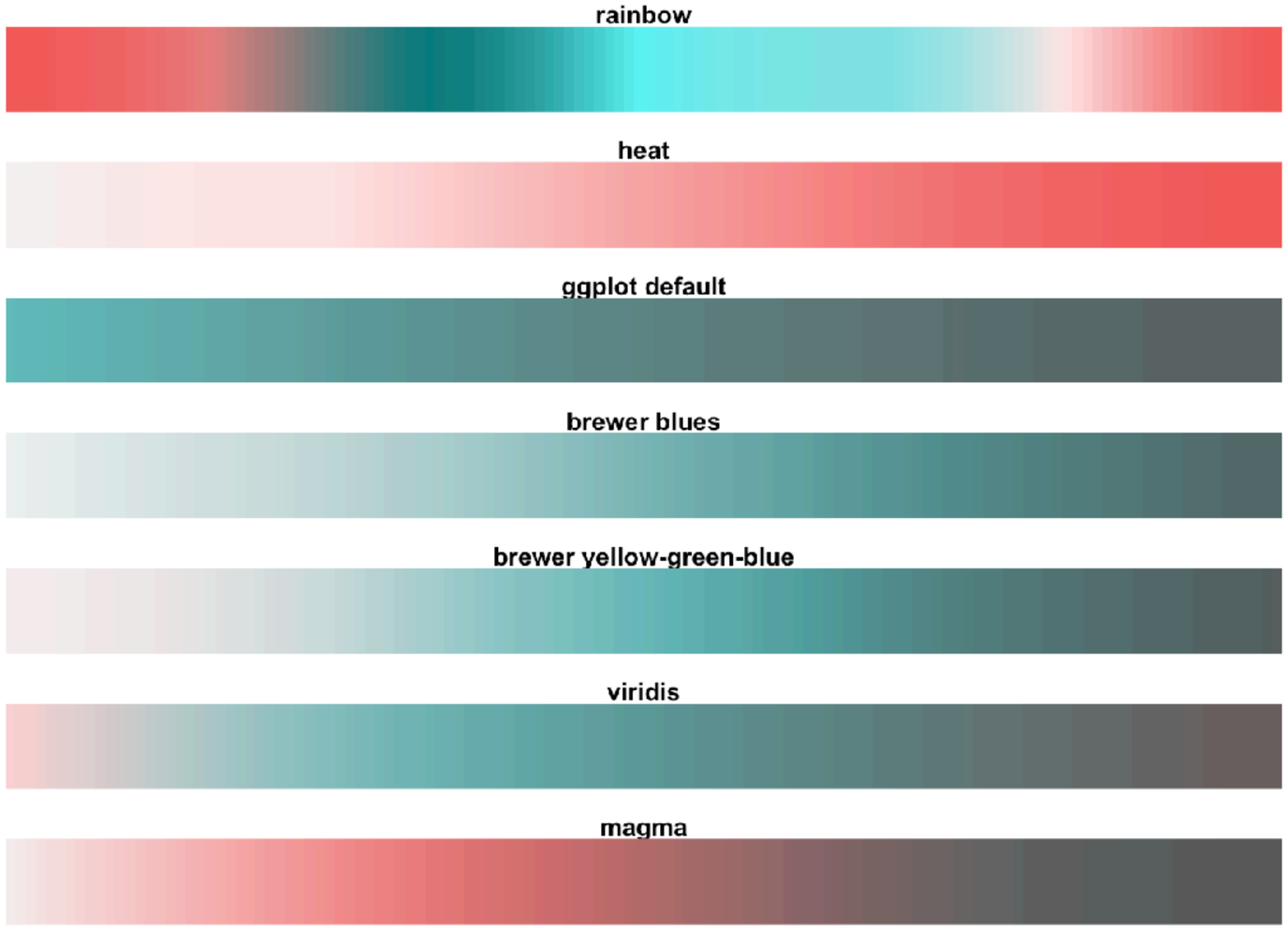


Red-blind (Protanopia)

Comparisons

Blue-blind (Tritanopia)

Comparisons



Desaturated

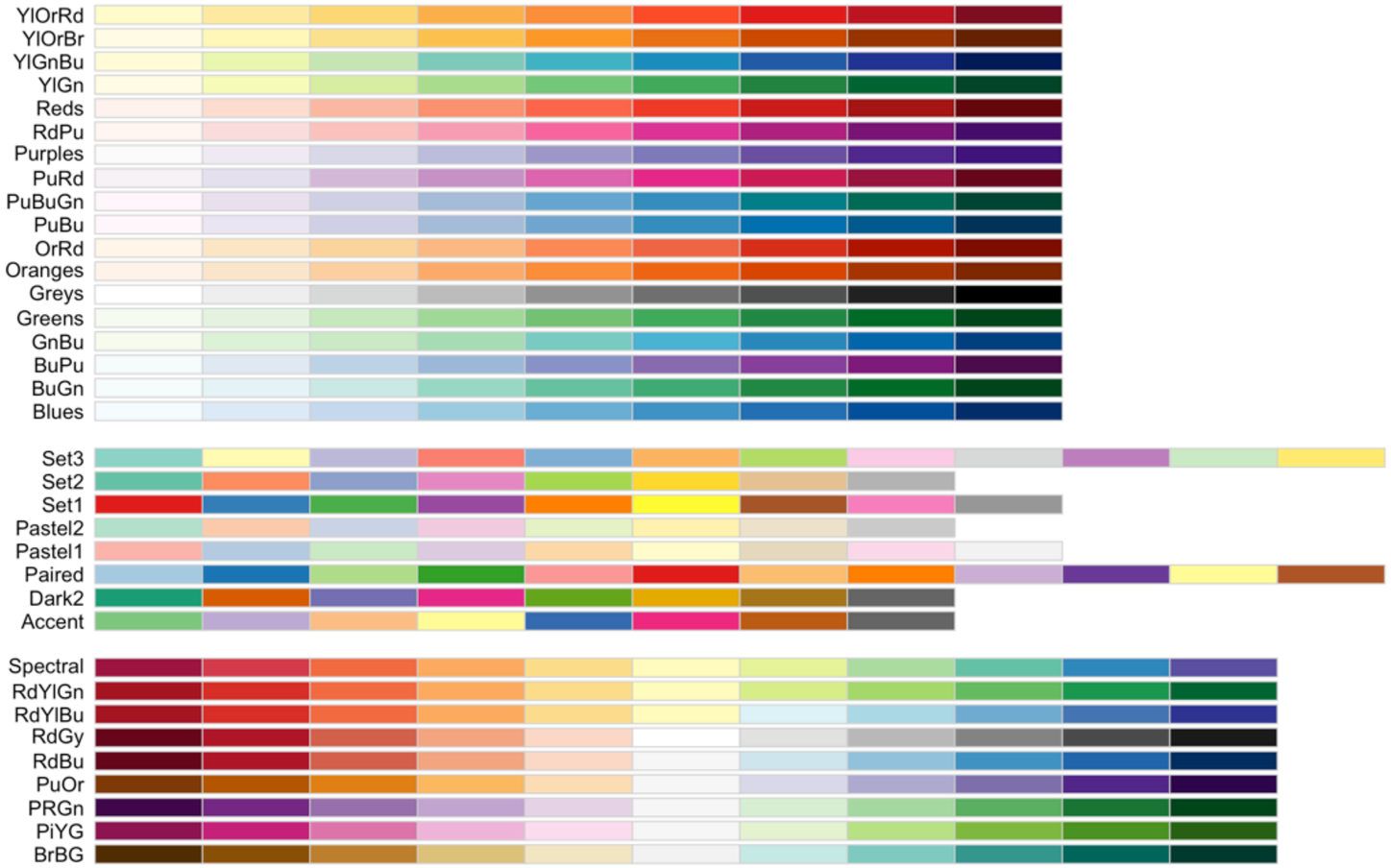
Comparisons

Fill & colour scale - Discrete

`scale_*_discrete()`	Colours evenly spaced around the colour wheel (same as hue)
`scale_*_hue()`	Colours evenly spread around the colour wheel
`scale_*_grey()`	Greyscale palette
`scale_*_viridis_d()`	Viridis palettes
`scale_*_brewer()`	ColorBrewer palettes
`scale_*_manual()`	Manually specified colours

ColorBrewer

```
display.brewer.all()
```



Fill & colour scale - Continuous

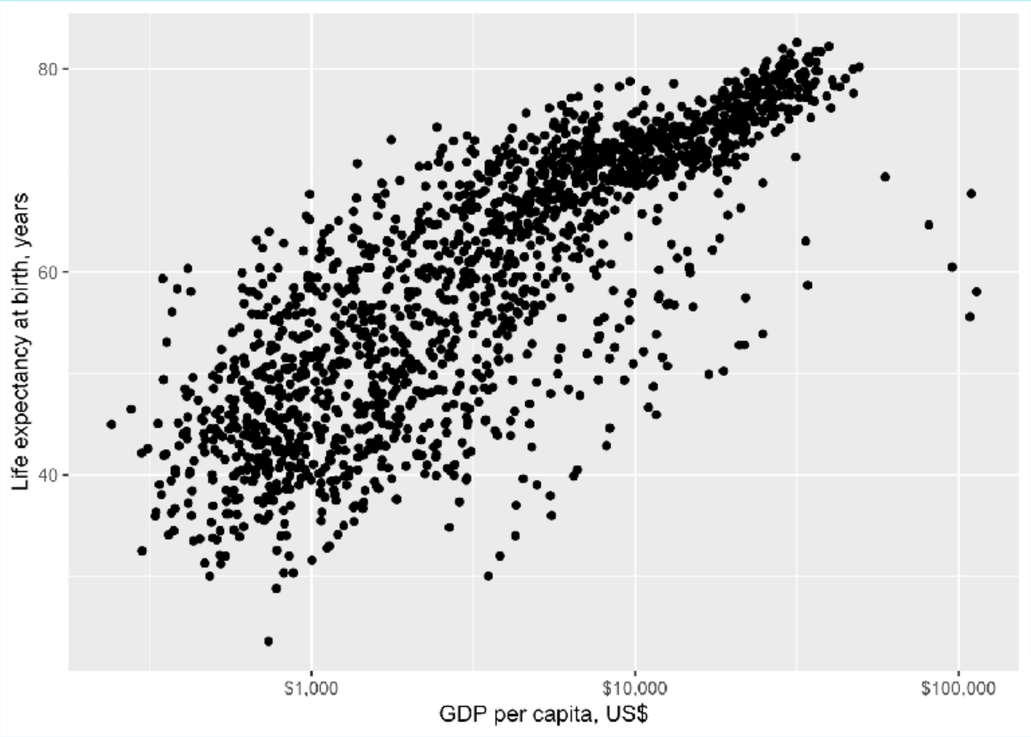
`scale_*_gradient()`	Two-colour gradient
`scale_*_gradient2()`	Gradient with a middle colour and tow colours that diverge from it
`scale_*_gradientn()`	Gradient with n colour, equally spaced
`scale_*_viridis_c()`	Viridis palettes

- Do you really need a continuous scale?
  - Can you bin the data to use discrete scale?

Your turn!

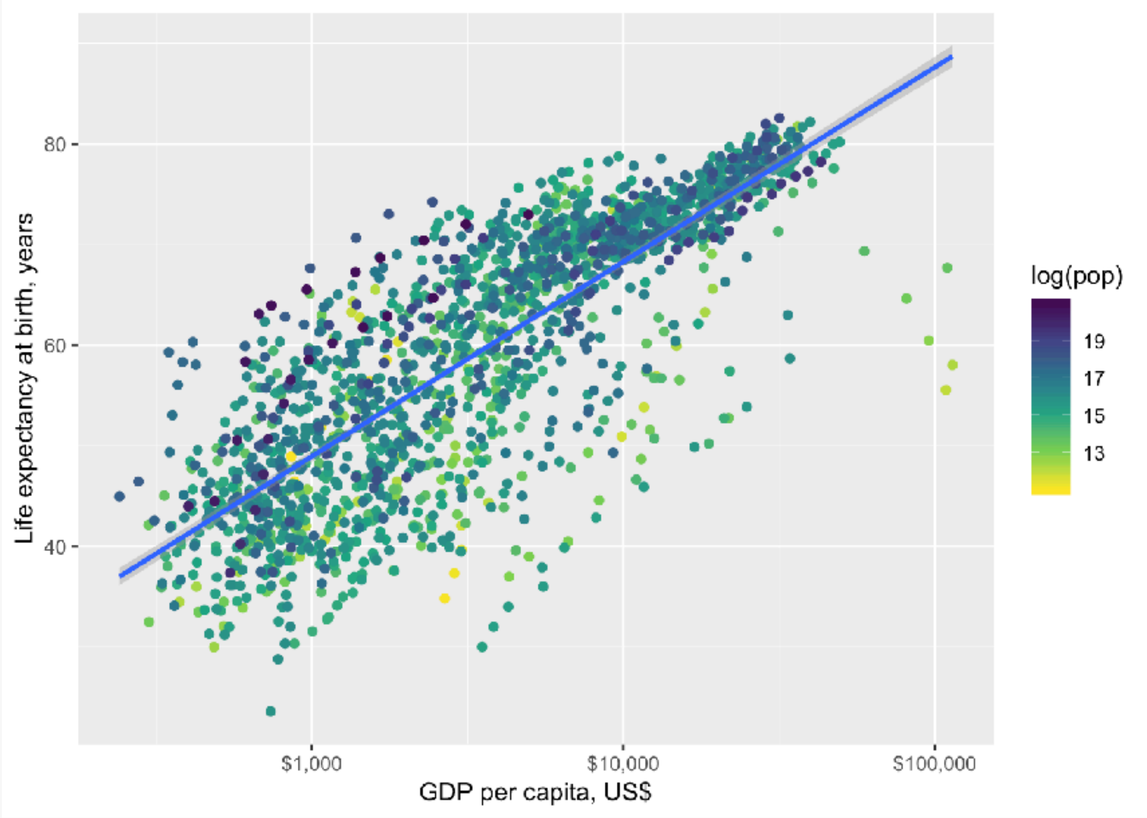
1) Modify the code to map **pop** variable (population) to colour aesthetics.
2) Did you map colour using a linear scale or log scale?

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
 scale_x_log10(labels  = scales::dollar) +
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



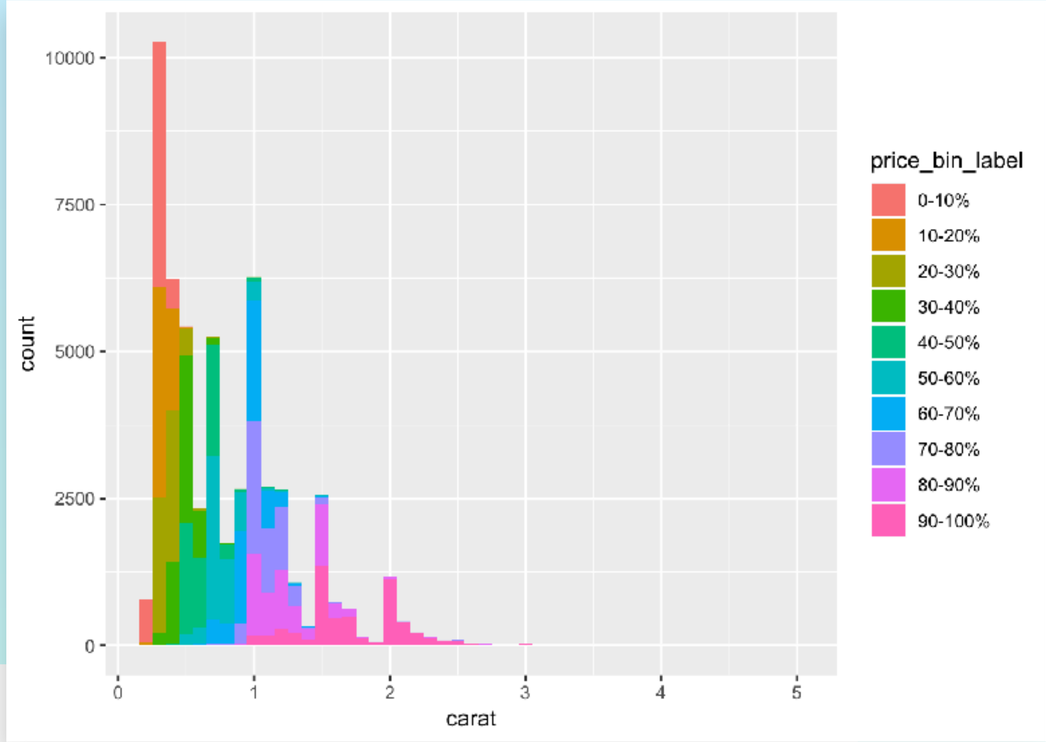
Solution

```
ggplot(data = gapminder %>% arrange(pop), mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point(mapping = aes(colour = log(pop))) +
  geom_smooth(method  = "gam") +
  scale_x_log10(labels  = scales::dollar) +
  scale_color_viridis_c(direction = -1)+
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



Your turn!

1) Apply **`scale_*_viridis_d()`** to the following code:
2) Can you use other viridis colour palettes?

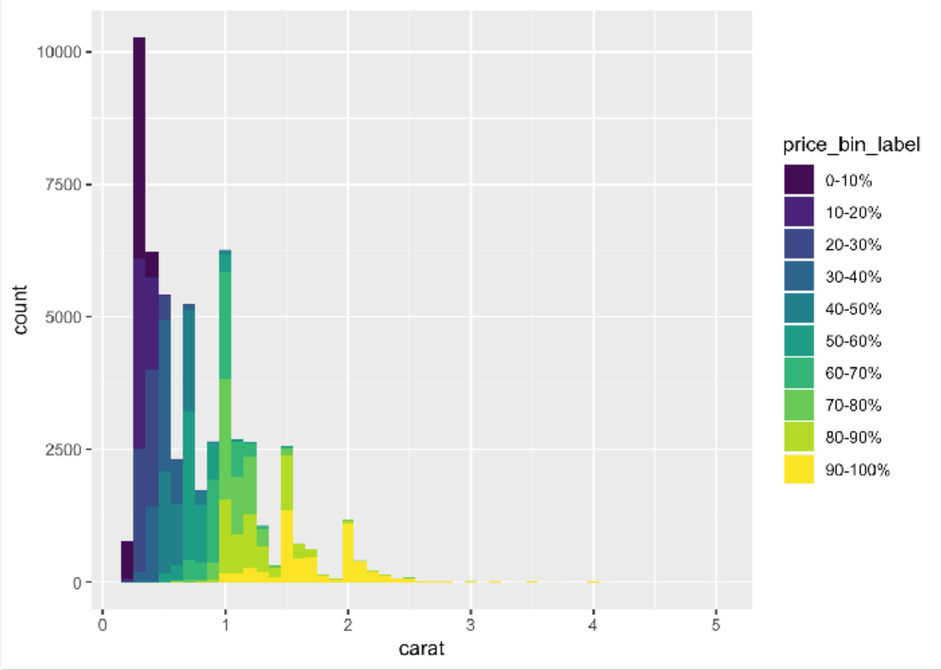


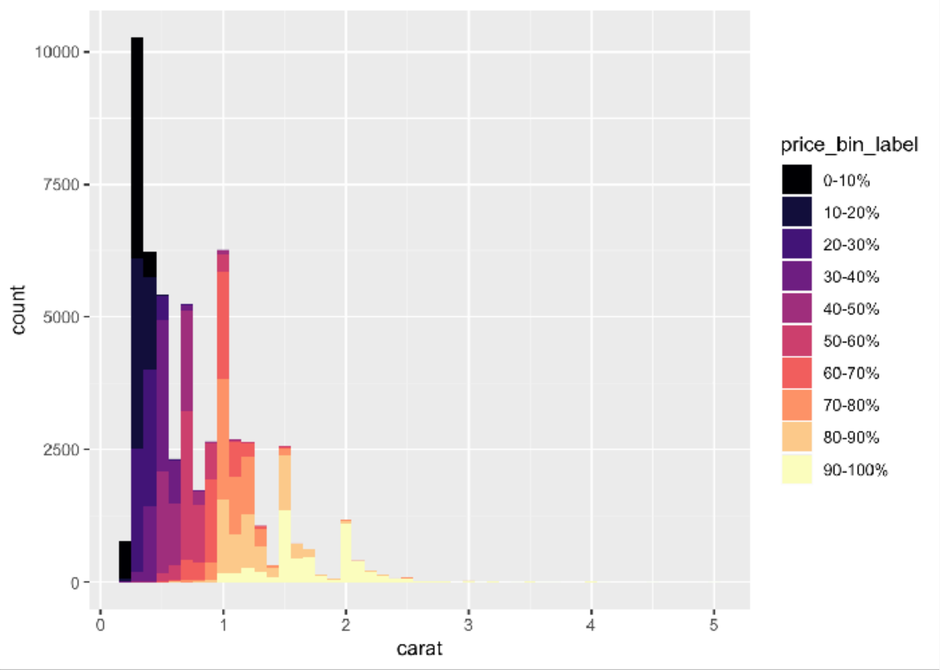

```r
# create a new variable by binning by price
df <- diamonds %>%
  mutate(price_bin = ntile(price, 10),
         price_bin_label = factor(price_bin, levels = c(1:10),
                                  labels = c("0-10%", "10-20%", "20-30%", "30-40%", "40-50%",
                                             "50-60%", "60-70%", "70-80%", "80-90%", "90-100%")))
ggplot(data = df) +
  geom_histogram(mapping = aes(x = carat, fill = price_bin_label), binwidth = 0.1)
```

Solution

```
ggplot(data = df) +
  geom_histogram(mapping = aes(x = carat, fill = price_bin_label), binwidth = 0.1)+
  scale_fill_viridis_d()
```

```
ggplot(data = df) +
  geom_histogram(mapping = aes(x = carat, fill = price_bin_label), binwidth = 0.1)+
  scale_fill_viridis_d(option="magma")
```

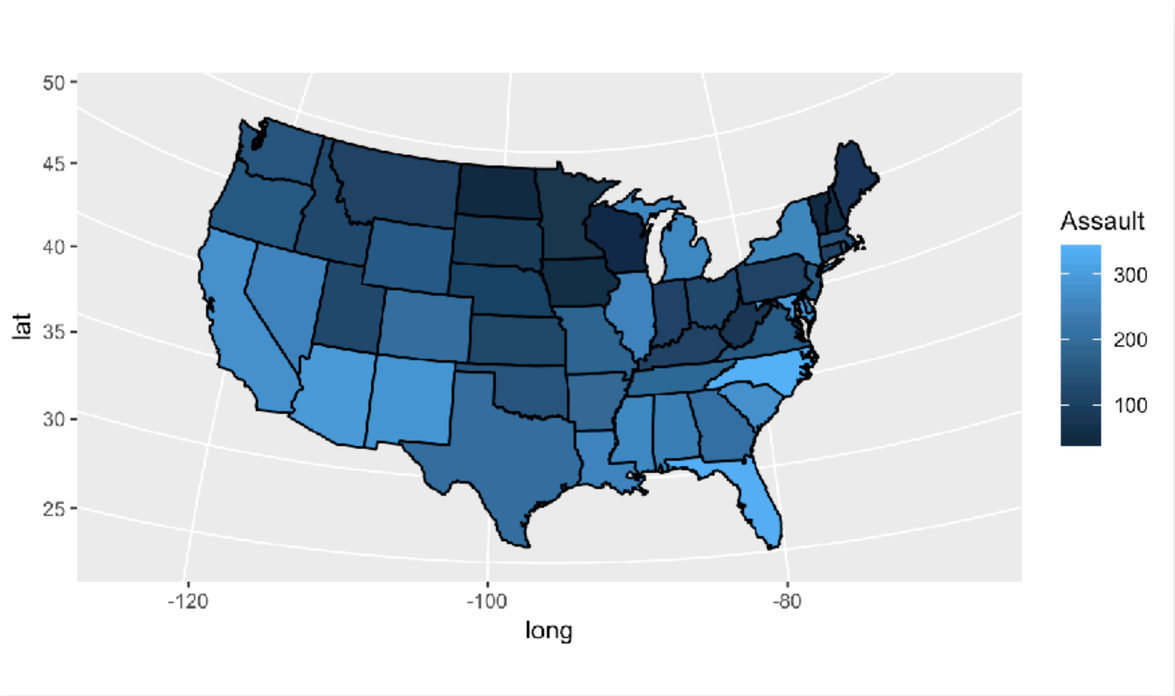




Choropleth

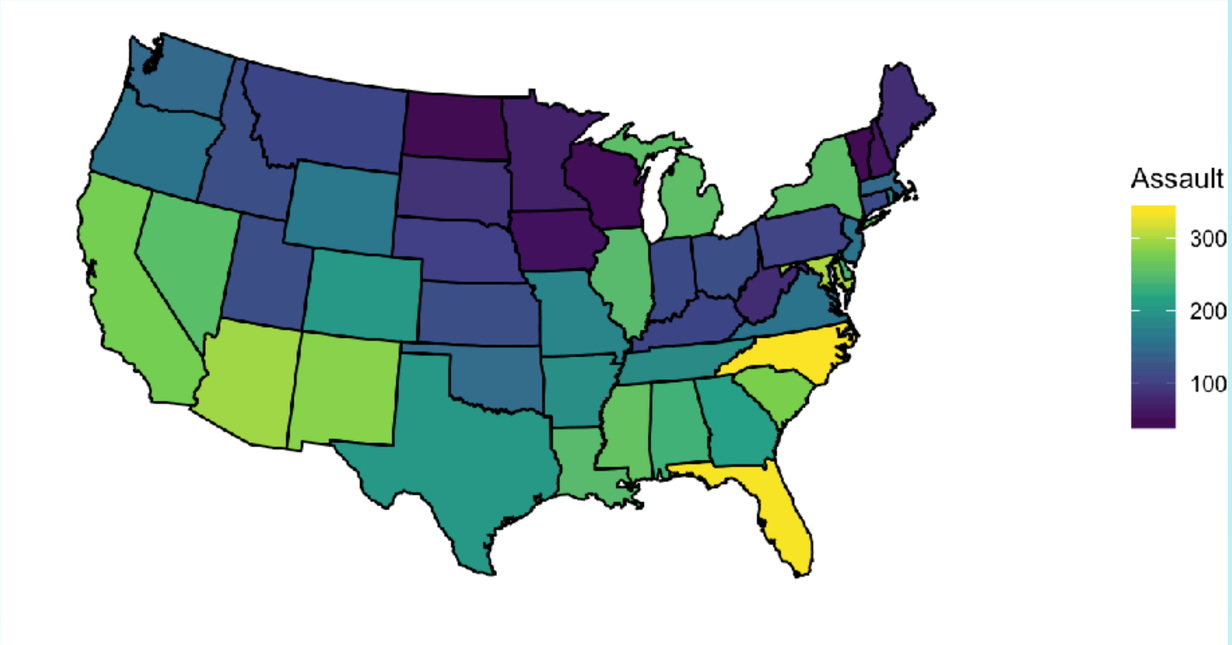
Choropleth Map

```r
# using USArrests datasets
crimes <- data.frame(USArrests) %>%
  mutate(state = tolower(rownames(USArrests)))

# using maps library
library(maps)
library(ggplot2)
states_map <- map_data("state")
# merge datasets together
crime_map <- states_map %>%
  left_join(crimes, by = c("region" = "state")) %>%
  arrange(group, order)

ggplot(data = crime_map, aes(x=long, y=lat, group = group, fill = Assault))+
  geom_polygon(colour = “black")+
  coord_map("polyconic")
```

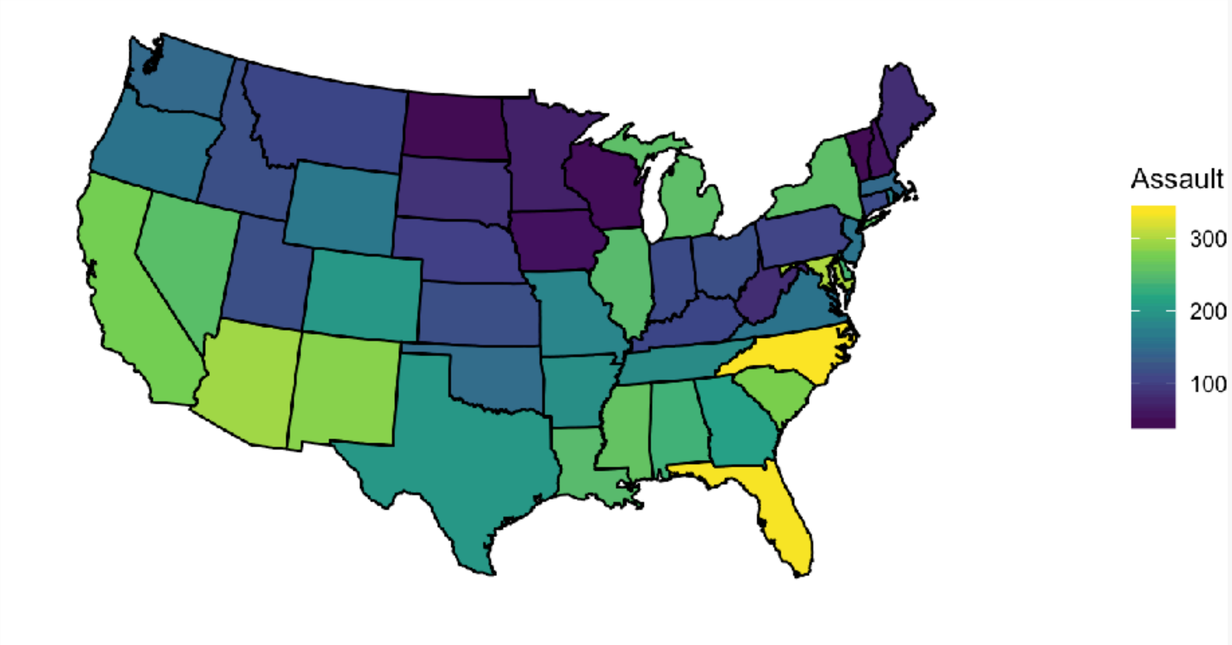


Your turn!

1) Apply `scale_fill_viridis_c()`



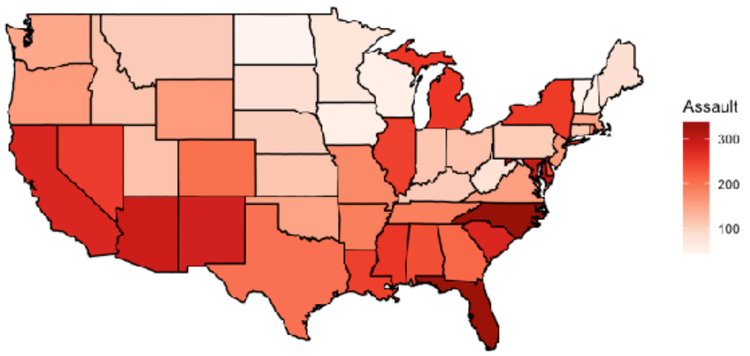
Solution



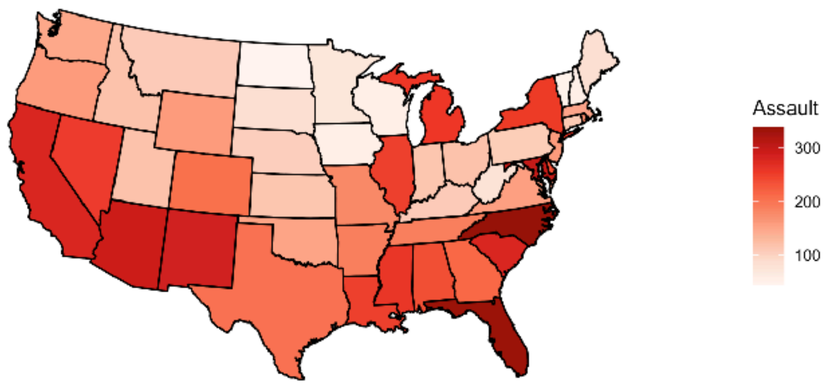

```
ggplot(data = crime_map, aes(x=long, y=lat, group = group, fill = Assault))+
  geom_polygon(colour = "black")+
  coord_map("polyconic")+
  scale_fill_viridis_c()+
  theme_void()
```

Choice of Projections

```
library(RColorBrewer)

ggplot(data = crime_map, aes(x=long, y=lat, group = group,
fill = Assault))+
  geom_polygon(colour = "black")+
  scale_fill_gradientn(colors = brewer.pal(8,"Reds")) +
  coord_map(projection = "mercator")+
  theme_void()
```




```
ggplot(data = crime_map, aes(x=long, y=lat, group = group,
fill = Assault))+
  geom_polygon(colour = "black")+
  scale_fill_gradientn(colors = brewer.pal(8,"Reds")) +
  coord_map("albers", lat0=30, lat1=40) +
  theme_void()
```



Map of EU countries

```r
# Map of Europe
# Get map data for world
world_map <- map_data("world")

# Check $regions
sort(unique(world_map$region))

##   [1] "Afghanistan"
##   [2] "Albania"
##   [3] "Algeria"
##   [4] "American Samoa"
##   [5] "Andorra"
##   [6] "Angola"
##   [7] "Anguilla"
##   [8] "Antarctica"
##   [9] "Antigua"
##  [10] "Argentina"
```

Map of EU countries

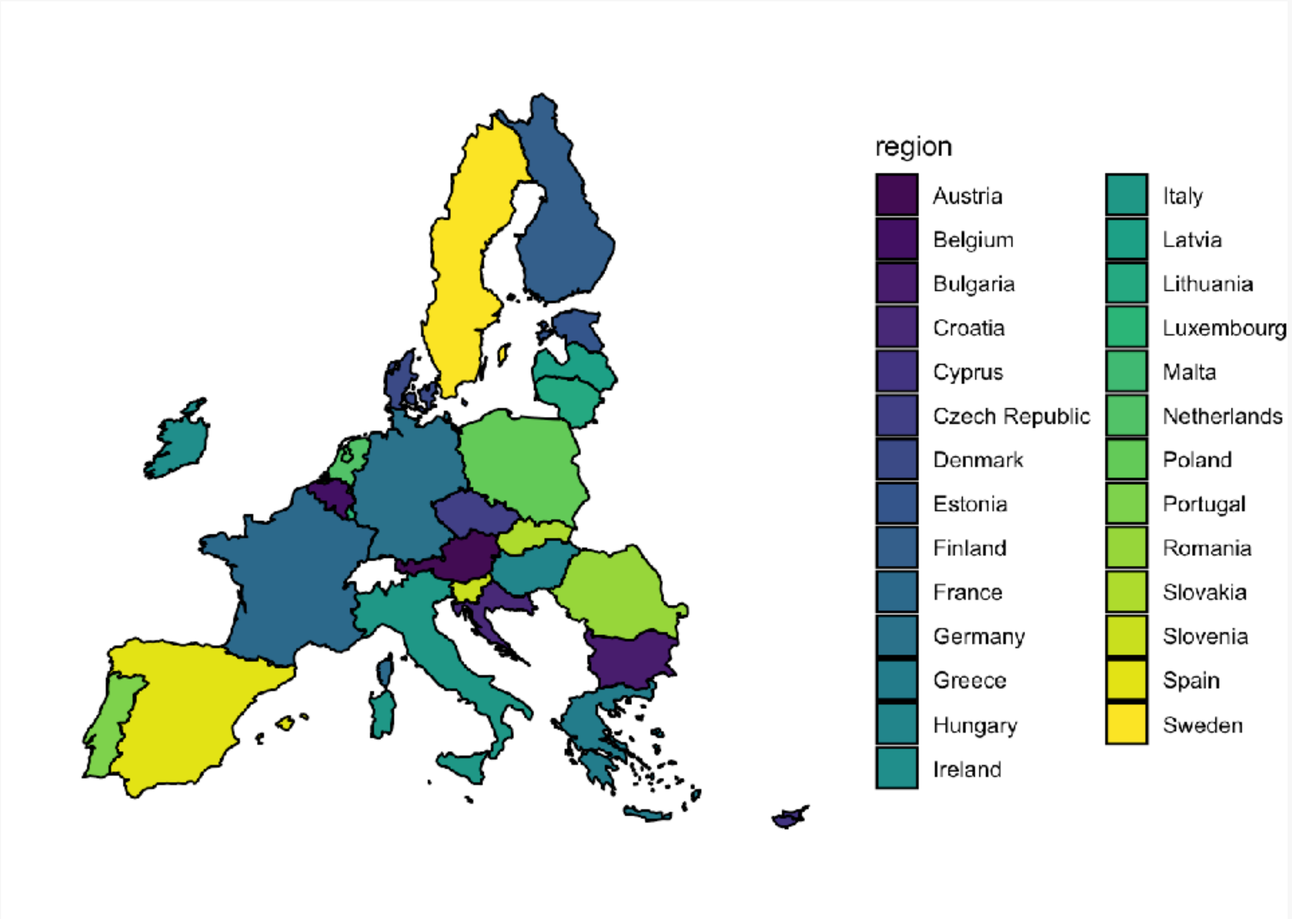
```
# A vector of countries
europeanUnion <- c("Austria","Belgium","Bulgaria","Croatia","Cyprus",
                   "Czech Republic","Denmark","Estonia","Finland","France",
                   "Germany","Greece","Hungary","Ireland","Italy","Latvia",
                   "Lithuania","Luxembourg","Malta","Netherlands","Poland",
                   "Portugal","Romania","Slovakia","Slovenia","Spain",
                   "Sweden")
# “UK”
# get the map data
eu_map <- map_data("world", region = europeanUnion)
```

Your turn!

1) Create a map of EU countries with **`eu_map`** data.

Solution

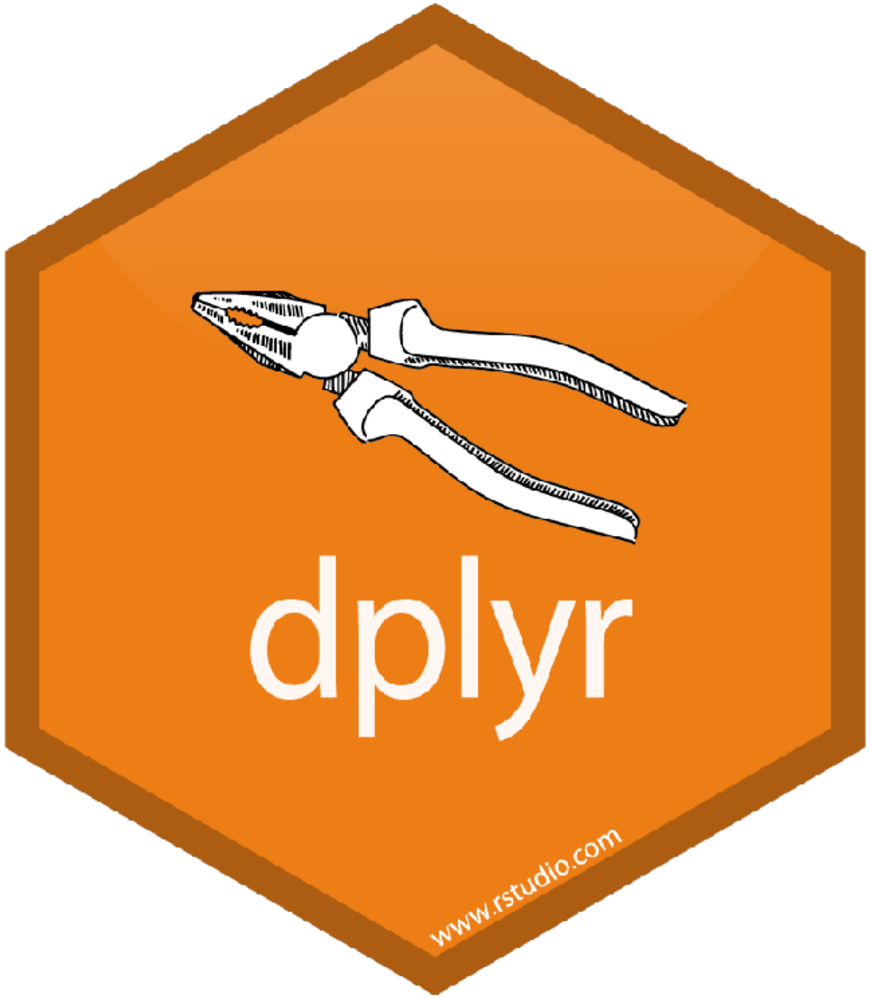
```
ggplot(eu_map, mapping = aes(x = long, y = lat, group = group, fill = region)) +
  geom_polygon(colour = "black")+
  scale_fill_viridis_d() +
  coord_map("sinusoidal")+
  theme_void()
```



Lecture 8 - Summary

- Data transformation
  - `dplyr` package
- Tidy data

- Importing data
- Exporting images
- Labels
- Scales
- Choropleth map



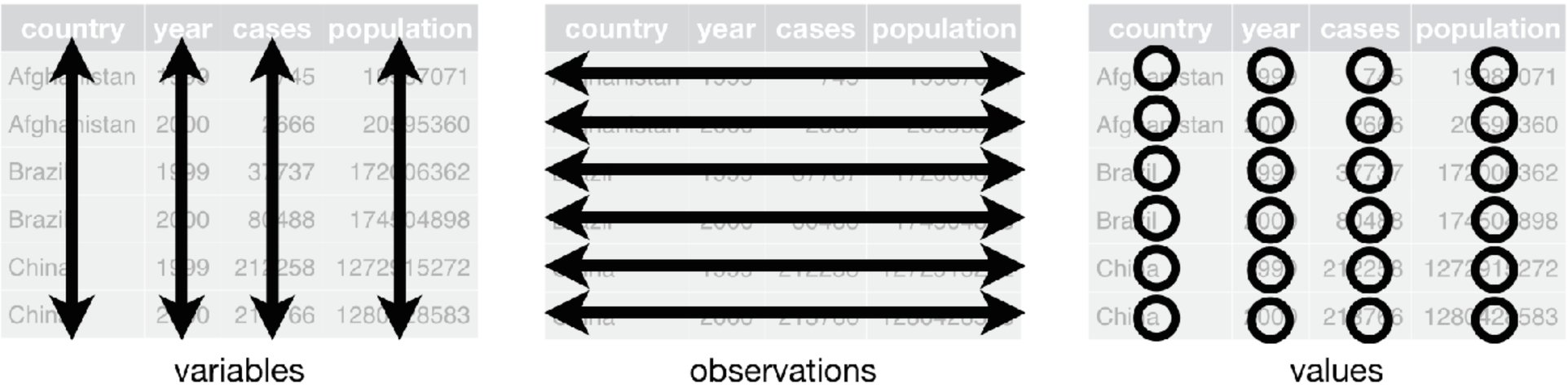
Recap

- Data transformation with dplyr

Function	Description	Equivalent SQL
`select()`	selecting columns	SELECT
`filter()`	filtering rows / subsetting	WHERE
`group_by()`	grouping data	GROUP BY
`summarise()`	summarising / aggregating data	-
`arrange()`	sorting data	ORDER BY
`join()`	joining data tables	JOIN
`mutate()`	creating new columns	COLUMN ALIAS

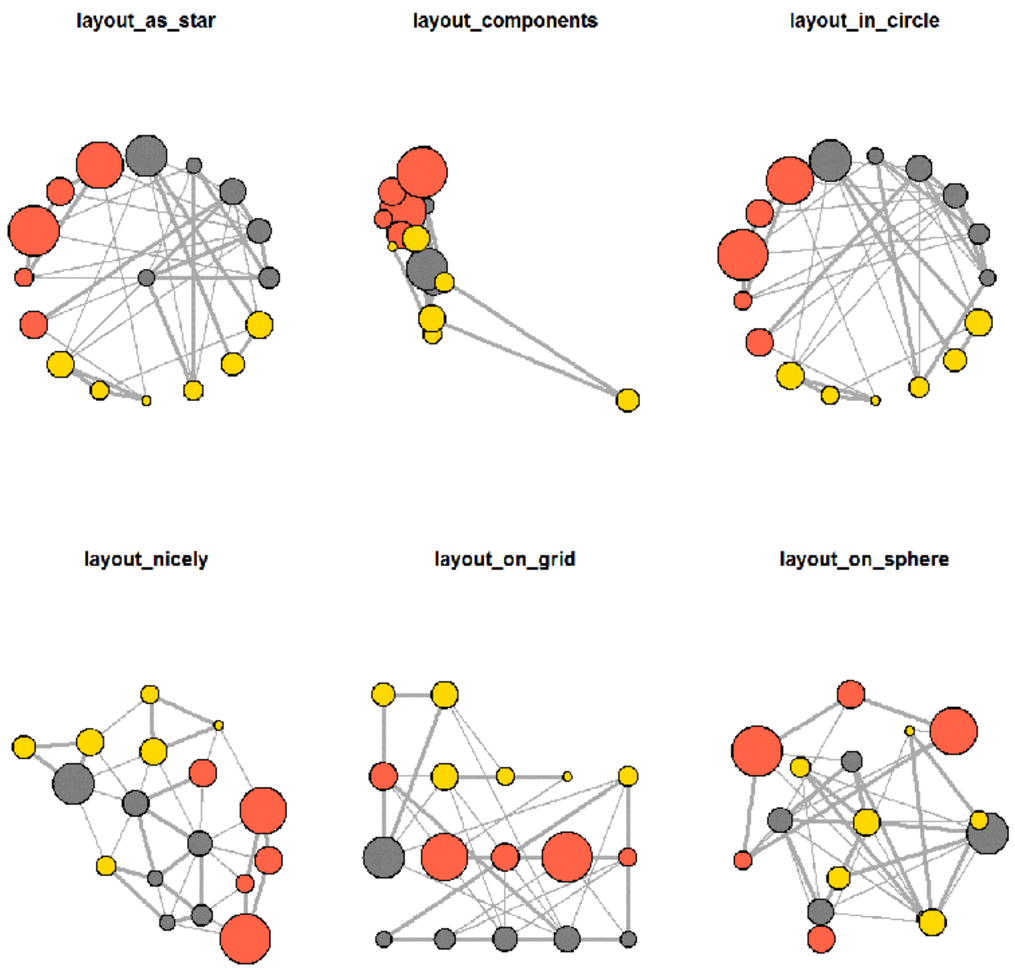
Recap - tidy data

- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell



Lecture 9 - preview

- Network visualisation
- Exploratory data analyses



In-class exercise

- Instruction:
  - Go to Insendi and download the markdown:

  - Work together with your classmates in the breakout room
  - If you have a question, send a message to the instructor
    - You may be pulled out of breakout room if there is a common question
    - Also, check the forum to see answers to FAQs
  - Submit the HTML output indivually, via Insendi by **the next day 7am (UK time)!**
  - **You should now be able to resubmit within the deadline**