Training Workshop on the basics of SEM using R

Intro to R and basic data wrangling

Overview

- R objects
- R packages
- Reading data into R
- Basic data wrangling with tidyverse
  - `select()`
  - `filter()`
  - `mutate()`
  - `rename()`
  - `arrange()`
  - `group_by()` and `summarize()`
  - `%>%` pipe operator

It's normal to struggle but it gets better and exciting!

Illustration adapted from Allison Horst

R Objects

- You can consider R objects as "*saving information*"
- e.g., text, number, matrix, vector, dataframe.
- In other words everything in R is an object.

R Objects

- Objects are assigned a value using `<-`

```
a1 <- 10
print(a1)
```

▶ Run

```
a2 <- 20
a2
```

▶ Run

```
a3 <- c(10, 20, 30)
a3
```

▶ Run

```
a1 * a2
```

▶ Run

```
st_name <- "christopher"
st_age <- 23
st_sex <- "male"

student <- c(st_name, st_age, st_sex)
student
```

▶ Run

R Objects

- Object names can be anything here!
- I personally use lower-case style.
- Check-out the recommended good practices in R (e.g., naming objects, writing codes)by the tidyverse style guide

R packages

- Collection of functions that load into your working environment.
- It contain code that other R users have prepared for the community.
- Installing packages

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

- Loading packages

```
library(tidyverse)
```

Importing data

- SPSS, Stata, SAS files: haven package
- Excel files: readxl package
- CSV files: readr package

Reading data into R

SPSS, Stata & SAS using haven package

```r
library(haven)
```

```r
# SPSS
read_sav("path/data.sav")
```

```r
# Stata
read_dta("path/data.dta")
```

```r
# SAS
read_sas("path/data.sas7bdat")
```

Reading data into R

Excel files using readxl package

```r
library(readxl)
read_excel("path/dataset.xls")
```

```
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Speci
          <dbl>       <dbl>        <dbl>       <dbl> <chr>
 1          5.1         3.5          1.4         0.2 setos
 2          4.9         3            1.4         0.2 setos
 3          4.7         3.2          1.3         0.2 setos
 4          4.6         3.1          1.5         0.2 setos
 5          5           3.6          1.4         0.2 setos
 6          5.4         3.9          1.7         0.4 setos
 7          4.6         3.4          1.4         0.3 setos
 8          5           3.4          1.5         0.2 setos
 9          4.4         2.9          1.4         0.2 setos
10          4.9         3.1          1.5         0.1 setos
# ... with 140 more rows
```

Reading data into R

CSV files using readr package

```r
install.packages("readr")
library(readr)
```

```r
# comma separated (CSV) files
read_csv("path/data.csv")
```

```r
# tab separated files
read_tsv("path/data.tsv")
```

```r
# general delimited files
read_delim("path/data.delim")
```

Reading data into R

Tips on naming (file) names

- machine readable
    - avoid spaces, punctuation, accented, characters, case sensitivity
- human readable
    - easy to search for files later
    - easy narrow files lists based on names
    - easy to extract into from file names
- J. Bryan "How to name files"

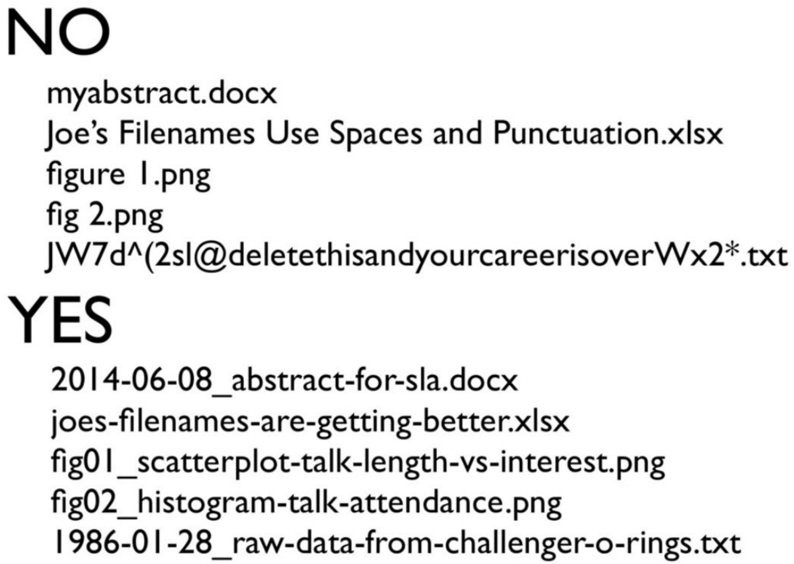


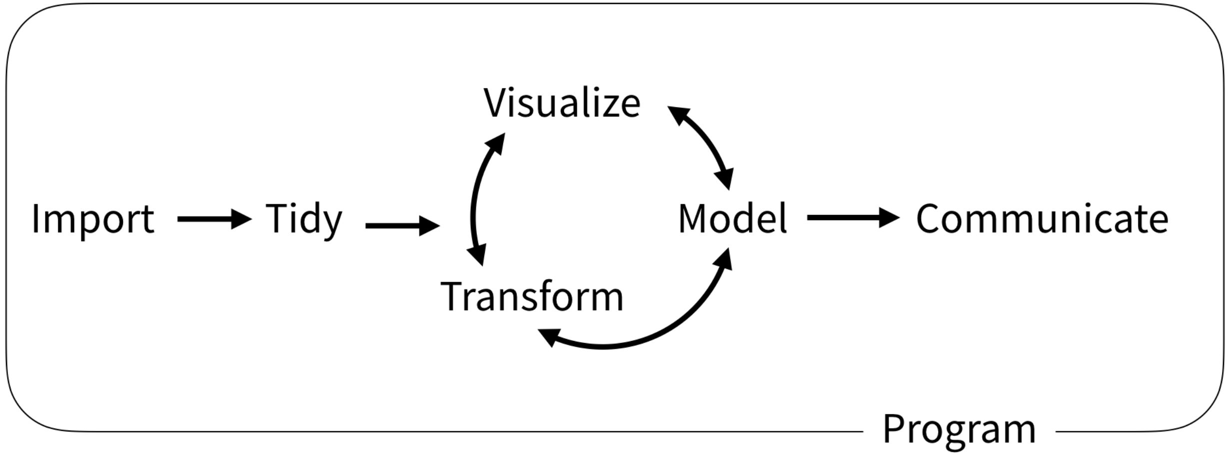
Illustration adapted from Jennifer Bryan

Tidyverse

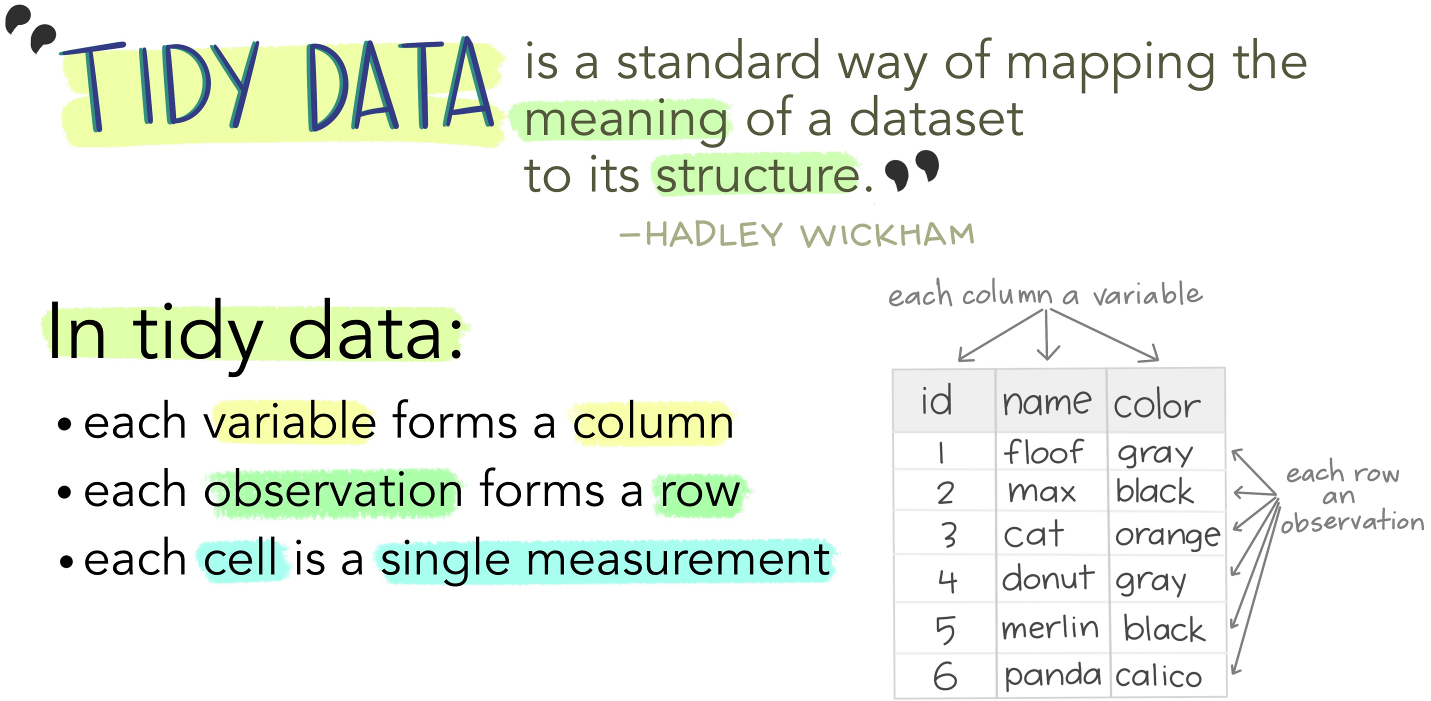
What is a tidyverse?

A collection of R packages designed for data science.

All packages share an underlying philosophy, grammar, and data structure.



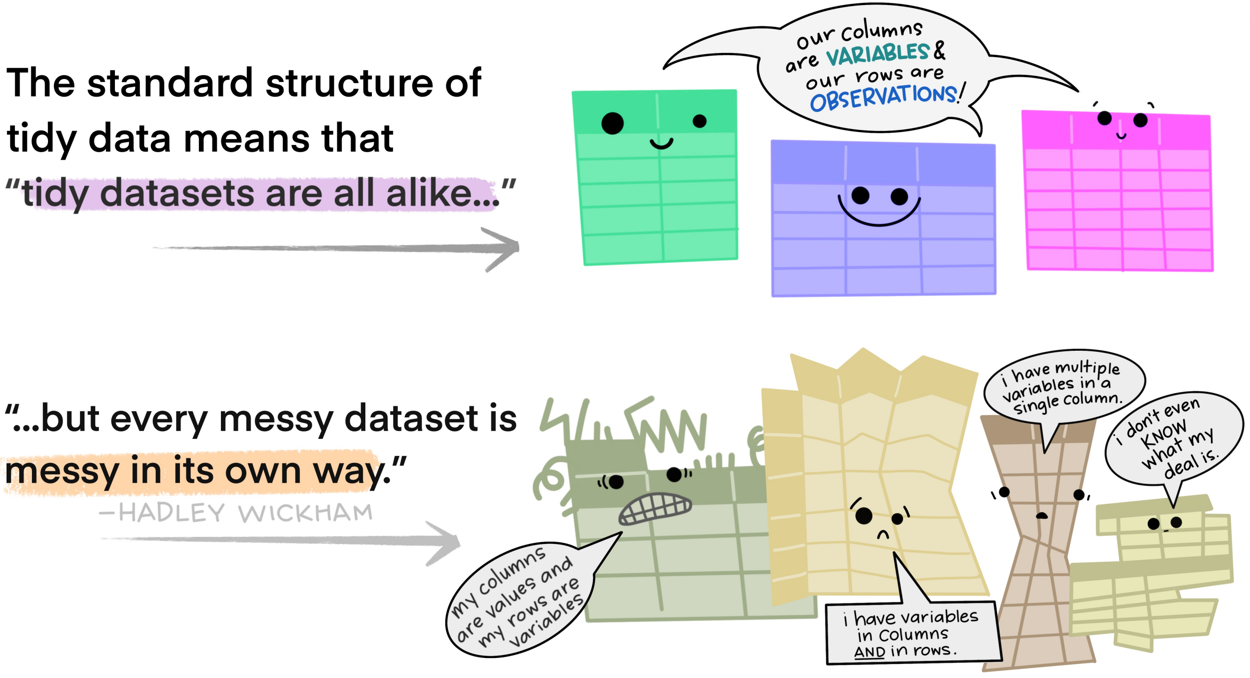
Tidyverse :: tidy data



Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10

Artist: Allison Horst

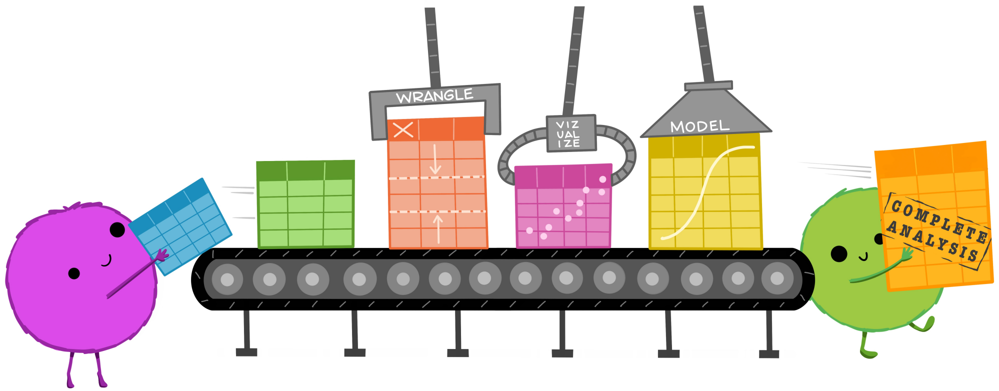
Tidyverse :: tidy data



Artist: Allison Horst

Tidyverse :: tidy data

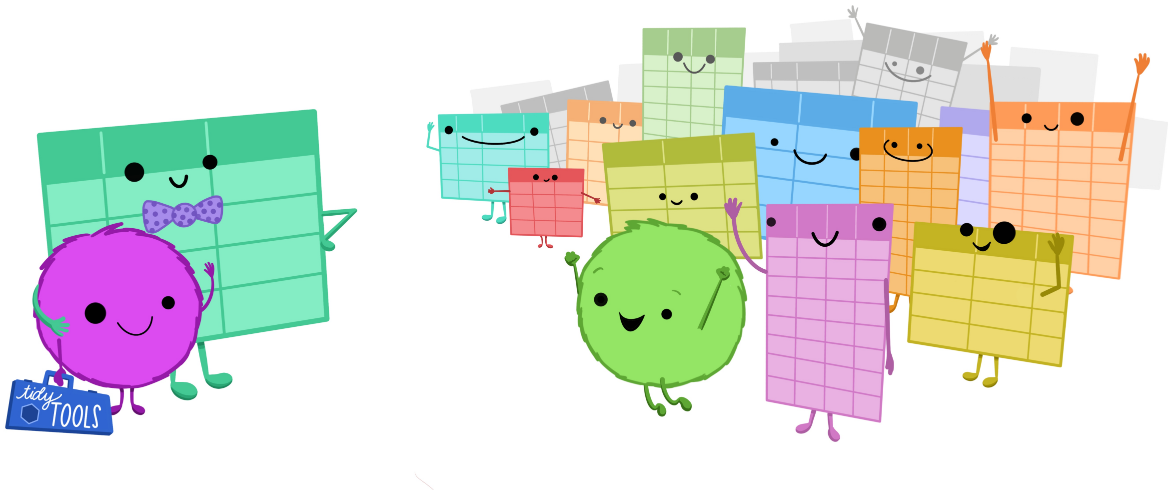
Tidy data makes it easier for reproducibility and reuse



Artist: Allison Horst

Tidyverse :: tidy data

Yehey! Tidy Data for the win!



Artist: Allison Horst

Data wrangling using `dplyr`

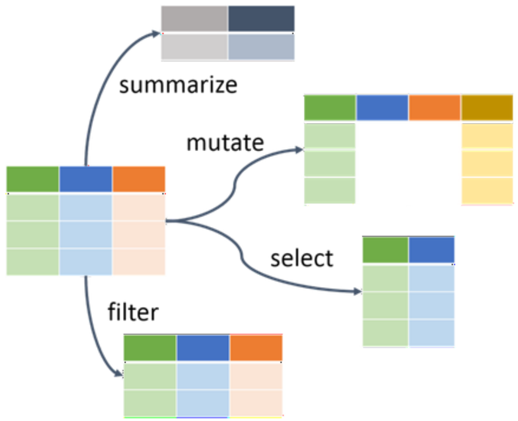
Artist: Allison Horst

`dplyr`

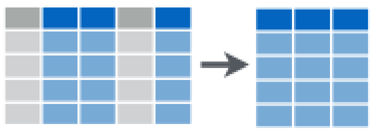
Overview

- `select()` picks variables based on their names
- `mutate()` adds new variables
- `filter()` picks cases based on their values
- `summarise()` reduces multiple values down to a single summary
- `arrange()` change the ordering of the rows

see `dplyr` cheatsheets



select()



```
data
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop
   <fct>       <fct>     <int>   <dbl>    <int>
 1 Afghanistan Asia       1952    28.8  8425333
 2 Afghanistan Asia       1957    30.3  9240934
 3 Afghanistan Asia       1962    32.0 10267083
 4 Afghanistan Asia       1967    34.0 11537966
 5 Afghanistan Asia       1972    36.1 13079460
 6 Afghanistan Asia       1977    38.4 14880372
 7 Afghanistan Asia       1982    39.9 12881816
 8 Afghanistan Asia       1987    40.8 13867957
 9 Afghanistan Asia       1992    41.7 16317921
10 Afghanistan Asia       1997    41.8 22227415
# ... with 1,694 more rows
```

```
select(data, continent, country, pop)
```

```
# A tibble: 1,704 x 3
   continent country          pop
   <fct>     <fct>          <int>
 1 Asia      Afghanistan  8425333
 2 Asia      Afghanistan  9240934
 3 Asia      Afghanistan 10267083
 4 Asia      Afghanistan 11537966
 5 Asia      Afghanistan 13079460
 6 Asia      Afghanistan 14880372
 7 Asia      Afghanistan 12881816
 8 Asia      Afghanistan 13867957
 9 Asia      Afghanistan 16317921
10 Asia      Afghanistan 22227415
# ... with 1,694 more rows
```

select()

We can also **remove** variables with a **–** (minus)

```
data
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop
   <fct>       <fct>     <int>   <dbl>    <int>
 1 Afghanistan Asia       1952    28.8  8425333
 2 Afghanistan Asia       1957    30.3  9240934
 3 Afghanistan Asia       1962    32.0 10267083
 4 Afghanistan Asia       1967    34.0 11537966
 5 Afghanistan Asia       1972    36.1 13079460
 6 Afghanistan Asia       1977    38.4 14880372
 7 Afghanistan Asia       1982    39.9 12881816
 8 Afghanistan Asia       1987    40.8 13867957
 9 Afghanistan Asia       1992    41.7 16317921
10 Afghanistan Asia       1997    41.8 22227415
# ... with 1,694 more rows
```

```
select(data, -year, -pop)
```

```
# A tibble: 1,704 x 4
   country     continent lifeExp gdpPercap
   <fct>       <fct>       <dbl>     <dbl>
 1 Afghanistan Asia         28.8      779.
 2 Afghanistan Asia         30.3      821.
 3 Afghanistan Asia         32.0      853.
 4 Afghanistan Asia         34.0      836.
 5 Afghanistan Asia         36.1      740.
 6 Afghanistan Asia         38.4      786.
 7 Afghanistan Asia         39.9      978.
 8 Afghanistan Asia         40.8      852.
 9 Afghanistan Asia         41.7      649.
10 Afghanistan Asia         41.8      635.
# ... with 1,694 more rows
```

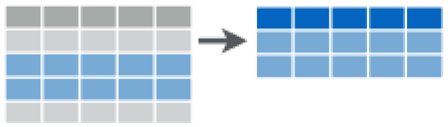
`select()`

Selection helpers

These *selection helpers* match variables according to a given pattern.

- `starts_with()` starts with a prefix
- `ends_with()` ends with a suffix
- `contains()` contains a literal string
- `matches()` matches regular expression

filter()



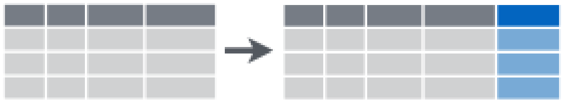
```
data
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop
   <fct>       <fct>     <int>   <dbl>    <int>
 1 Afghanistan Asia       1952    28.8  8425333
 2 Afghanistan Asia       1957    30.3  9240934
 3 Afghanistan Asia       1962    32.0 10267083
 4 Afghanistan Asia       1967    34.0 11537966
 5 Afghanistan Asia       1972    36.1 13079460
 6 Afghanistan Asia       1977    38.4 14880372
 7 Afghanistan Asia       1982    39.9 12881816
 8 Afghanistan Asia       1987    40.8 13867957
 9 Afghanistan Asia       1992    41.7 16317921
10 Afghanistan Asia       1997    41.8 22227415
# ... with 1,694 more rows
```

```
filter(data, country == "Philippines")
```

```
# A tibble: 12 x 6
   country     continent  year lifeExp      pop
   <fct>       <fct>     <int>   <dbl>    <int>
 1 Philippines Asia       1952    47.8 22438691
 2 Philippines Asia       1957    51.3 26072194
 3 Philippines Asia       1962    54.8 30325264
 4 Philippines Asia       1967    56.4 35356600
 5 Philippines Asia       1972    58.1 40850141
 6 Philippines Asia       1977    60.1 46850962
 7 Philippines Asia       1982    62.1 53456774
 8 Philippines Asia       1987    64.2 60017788
 9 Philippines Asia       1992    66.5 67185766
10 Philippines Asia       1997    68.6 75012988
11 Philippines Asia       2002    70.3 82995088
12 Philippines Asia       2007    71.7 91077287
```

`mutate()`



The `mutate` function will take a statement similar to this:

- `variable_name = do_some_calculation`
- `variable_name` will be attached at the end of the dataset.

mutate()

Let's calculate the gdp

```
data
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop
   <fct>       <fct>     <int>   <dbl>    <int>
 1 Afghanistan Asia       1952    28.8  8425333
 2 Afghanistan Asia       1957    30.3  9240934
 3 Afghanistan Asia       1962    32.0 10267083
 4 Afghanistan Asia       1967    34.0 11537966
 5 Afghanistan Asia       1972    36.1 13079460
 6 Afghanistan Asia       1977    38.4 14880372
 7 Afghanistan Asia       1982    39.9 12881816
 8 Afghanistan Asia       1987    40.8 13867957
 9 Afghanistan Asia       1992    41.7 16317921
10 Afghanistan Asia       1997    41.8 22227415
# ... with 1,694 more rows
```

```
mutate(data, GDP = gdpPercap * pop)
```

```
# A tibble: 1,704 x 7
   country     continent  year lifeExp      pop
   <fct>       <fct>     <int>   <dbl>    <int>
 1 Afghanistan Asia       1952    28.8  8425333
 2 Afghanistan Asia       1957    30.3  9240934
 3 Afghanistan Asia       1962    32.0 10267083
 4 Afghanistan Asia       1967    34.0 11537966
 5 Afghanistan Asia       1972    36.1 13079460
 6 Afghanistan Asia       1977    38.4 14880372
 7 Afghanistan Asia       1982    39.9 12881816
 8 Afghanistan Asia       1987    40.8 13867957
 9 Afghanistan Asia       1992    41.7 16317921
10 Afghanistan Asia       1997    41.8 22227415
# ... with 1,694 more rows
```

rename()

Changes the variable name while keeping all else intact.

- new_variable_name = old_variable_name

```
data
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop
   <fct>       <fct>     <int>   <dbl>    <int>
 1 Afghanistan Asia       1952    28.8  8425333
 2 Afghanistan Asia       1957    30.3  9240934
 3 Afghanistan Asia       1962    32.0 10267083
 4 Afghanistan Asia       1967    34.0 11537966
 5 Afghanistan Asia       1972    36.1 13079460
 6 Afghanistan Asia       1977    38.4 14880372
 7 Afghanistan Asia       1982    39.9 12881816
 8 Afghanistan Asia       1987    40.8 13867957
 9 Afghanistan Asia       1992    41.7 16317921
10 Afghanistan Asia       1997    41.8 22227415
# ... with 1,694 more rows
```

```
rename(data, population = pop)
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp populatio
   <fct>       <fct>     <int>   <dbl>      <int
 1 Afghanistan Asia       1952    28.8    842533
 2 Afghanistan Asia       1957    30.3    924093
 3 Afghanistan Asia       1962    32.0   1026708
 4 Afghanistan Asia       1967    34.0   1153796
 5 Afghanistan Asia       1972    36.1   1307946
 6 Afghanistan Asia       1977    38.4   1488037
 7 Afghanistan Asia       1982    39.9   1288181
 8 Afghanistan Asia       1987    40.8   1386795
 9 Afghanistan Asia       1992    41.7   1631792
10 Afghanistan Asia       1997    41.8   2222741
# ... with 1,694 more rows
```

arrange()

You can order data by variable to show the highest or lowest values first.

consider `lifeExp` default is lowest first

```
data
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop
   <fct>       <fct>     <int>   <dbl>    <int>
 1 Afghanistan Asia       1952    28.8  8425333
 2 Afghanistan Asia       1957    30.3  9240934
 3 Afghanistan Asia       1962    32.0 10267083
 4 Afghanistan Asia       1967    34.0 11537966
 5 Afghanistan Asia       1972    36.1 13079460
 6 Afghanistan Asia       1977    38.4 14880372
 7 Afghanistan Asia       1982    39.9 12881816
 8 Afghanistan Asia       1987    40.8 13867957
 9 Afghanistan Asia       1992    41.7 16317921
10 Afghanistan Asia       1997    41.8 22227415
# ... with 1,694 more rows
```

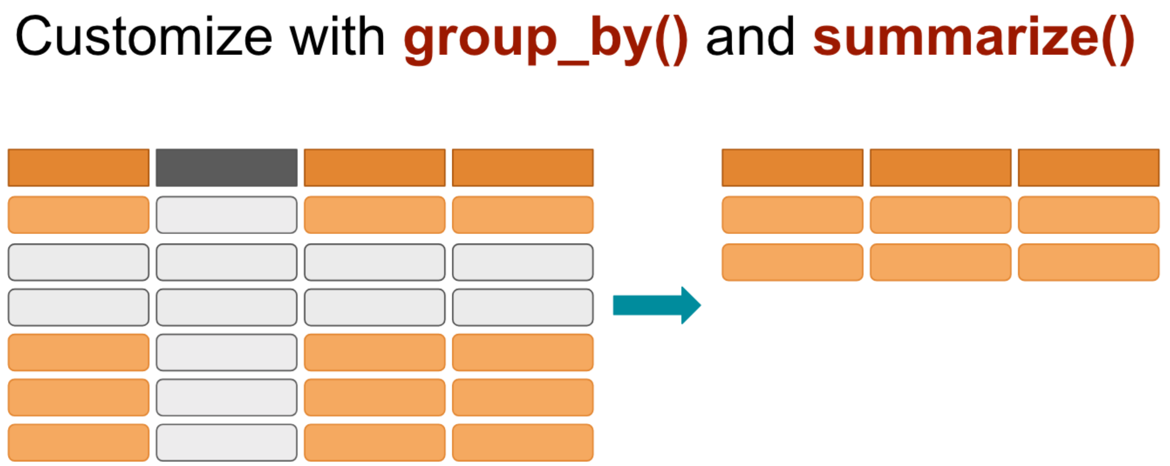
`desc()` sort `lifeExp` from highest to lowest

```
arrange(data, desc(lifeExp))
```

```
# A tibble: 1,704 x 6
   country          continent  year lifeExp
   <fct>            <fct>     <int>   <dbl>
 1 Japan            Asia       2007    82.6 1274
 2 Hong Kong, China Asia       2007    82.2   69
 3 Japan            Asia       2002    82   1270
 4 Iceland          Europe     2007    81.8    3
 5 Switzerland      Europe     2007    81.7   75
 6 Hong Kong, China Asia       2002    81.5   67
 7 Australia        Oceania    2007    81.2  204
 8 Spain            Europe     2007    80.9  404
 9 Sweden           Europe     2007    80.9   90
10 Israel           Asia       2007    80.7   64
# ... with 1,694 more rows
```

group_by and summarise()

- Use when you want to aggregate your data (by groups).
- Sometimes we want to calculate group statistics.



group_by and summarise()

Suppose we want to know the average population by continent.

```
data
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp
   <fct>       <fct>     <int>   <dbl>
 1 Afghanistan Asia       1952    28.8
 2 Afghanistan Asia       1957    30.3
 3 Afghanistan Asia       1962    32.0
 4 Afghanistan Asia       1967    34.0
 5 Afghanistan Asia       1972    36.1
 6 Afghanistan Asia       1977    38.4
 7 Afghanistan Asia       1982    39.9
 8 Afghanistan Asia       1987    40.8
 9 Afghanistan Asia       1992    41.7
10 Afghanistan Asia       1997    41.8
# ... with 1,694 more rows
```

```
grouped_by_continent <- group_by(data, continent)
summarise(grouped_by_continent, avg_pop = mean(pop))
```

```
# A tibble: 5 x 2
  continent   avg_pop
  <fct>         <dbl>
1 Africa     9916003.
2 Americas  24504795.
3 Asia      77038722.
4 Europe    17169765.
5 Oceania    8874672.
```

group_by and summarise()

Suppose we want to know the average population by continent.

```
data
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp
   <fct>       <fct>     <int>   <dbl>
 1 Afghanistan Asia       1952    28.8
 2 Afghanistan Asia       1957    30.3
 3 Afghanistan Asia       1962    32.0
 4 Afghanistan Asia       1967    34.0
 5 Afghanistan Asia       1972    36.1
 6 Afghanistan Asia       1977    38.4
 7 Afghanistan Asia       1982    39.9
 8 Afghanistan Asia       1987    40.8
 9 Afghanistan Asia       1992    41.7
10 Afghanistan Asia       1997    41.8
# ... with 1,694 more rows
```

```
grouped_by_continent <- group_by(data, continent)
summarised_data <- summarise(grouped_by_continent, avg_pop
arrange(summarised_data, desc(avg_pop))
```

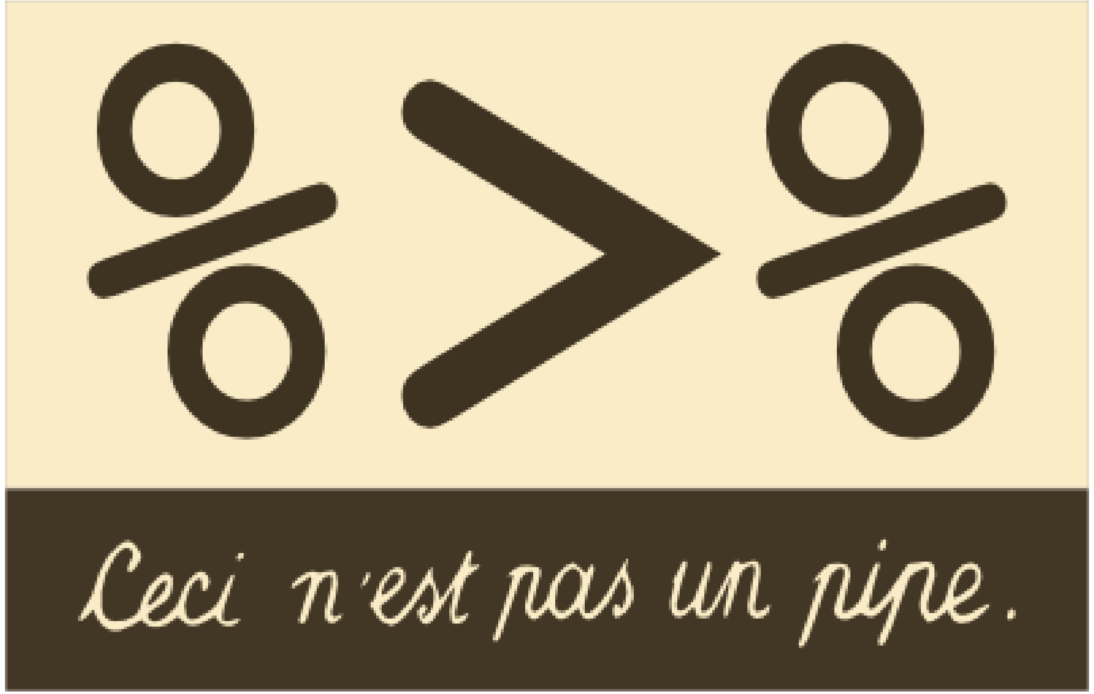
```
# A tibble: 5 x 2
  continent    avg_pop
  <fct>          <dbl>
1 Asia       77038722.
2 Americas   24504795.
3 Europe     17169765.
4 Africa      9916003.
5 Oceania     8874672.
```

Too many codes!

It's hard to follow!

It's hard to keep track of the codes!

%>% pipe operator



The %>% operator

The `%>%` helps your write code in a way that is easier to read and understand.

Calculating population by continent **without %>%**

```
grouped_by_continent <- group_by(data, continent
summarised_data <- summarise(grouped_by_continen
arrange(summarised_data, desc(avg_pop))
```

```
# A tibble: 5 x 2
  continent   avg_pop
  <fct>         <dbl>
1 Asia      77038722.
2 Americas  24504795.
3 Europe    17169765.
4 Africa     9916003.
5 Oceania    8874672.
```

Calculating population by continent **with %>%**

```
data %>%
  group_by(continent) %>%
  summarise(avg_pop = mean(pop)) %>%
  arrange(desc(avg_pop))
```

```
# A tibble: 5 x 2
  continent   avg_pop
  <fct>         <dbl>
1 Asia      77038722.
2 Americas  24504795.
3 Europe    17169765.
4 Africa     9916003.
5 Oceania    8874672.
```

The %>% operator

Suppose we want to know the evarage life expectancy of Asian countries per year.

Calculating population by continent **without %>%**

```
filtered_by_asia <- filter(data, continent == "A
grouped_by_country_year <- group_by(filtered_by_
summarise(grouped_by_country_year, avg_lifeExp =
```

```
# A tibble: 396 x 3
# Groups:   country [33]
   country      year avg_lifeExp
   <fct>       <int>       <dbl>
 1 Afghanistan  1952        28.8
 2 Afghanistan  1957        30.3
 3 Afghanistan  1962        32.0
 4 Afghanistan  1967        34.0
 5 Afghanistan  1972        36.1
 6 Afghanistan  1977        38.4
 7 Afghanistan  1982        39.9
 8 Afghanistan  1987        40.8
```

Calculating population by continent **with %>%**

```
data %>%
  filter(continent == "Asia") %>%
  group_by(country, year) %>%
  summarise(avg_lifeExp = mean(lifeExp))
```

```
# A tibble: 396 x 3
# Groups:   country [33]
   country      year avg_lifeExp
   <fct>       <int>       <dbl>
 1 Afghanistan  1952        28.8
 2 Afghanistan  1957        30.3
 3 Afghanistan  1962        32.0
 4 Afghanistan  1967        34.0
 5 Afghanistan  1972        36.1
 6 Afghanistan  1977        38.4
 7 Afghanistan  1982        39.9
```

The %>% operator

Calculating population by continent **without %>%**

```
filtered_by_asia <- filter(data, continent == "A
grouped_by_country <- group_by(filtered_by_asia,
summarised_by_country <- summarise(grouped_by_co
arrange(summarised_by_country, desc(avg_lifeExp)
```

```
# A tibble: 33 x 2
   country            avg_lifeExp
   <fct>                    <dbl>
 1 Japan                     74.8
 2 Israel                    73.6
 3 Hong Kong, China          73.5
 4 Singapore                 71.2
 5 Taiwan                    70.3
 6 Kuwait                    68.9
 7 Sri Lanka                 66.5
 8 Lebanon                   65.9
 9 Bahrain                   65.6
10 Korea, Rep.               65.0
```

Calculating population by continent **with %>%**

```
data %>%
  filter(continent == "Asia") %>%
  group_by(country) %>%
  summarise(avg_lifeExp = mean(lifeExp)) %>%
  arrange(desc(avg_lifeExp))
```

```
# A tibble: 33 x 2
   country            avg_lifeExp
   <fct>                    <dbl>
 1 Japan                     74.8
 2 Israel                    73.6
 3 Hong Kong, China          73.5
 4 Singapore                 71.2
 5 Taiwan                    70.3
 6 Kuwait                    68.9
 7 Sri Lanka                 66.5
 8 Lebanon                   65.9
 9 Bahrain                   65.6
```

Let's practice!

Thank you!

Some slide content were heavily adapted from Fabio Votta

Slides created via the R packages:

xaringan by Yihui

xaringanthemer and xaringanExtra by Garrick