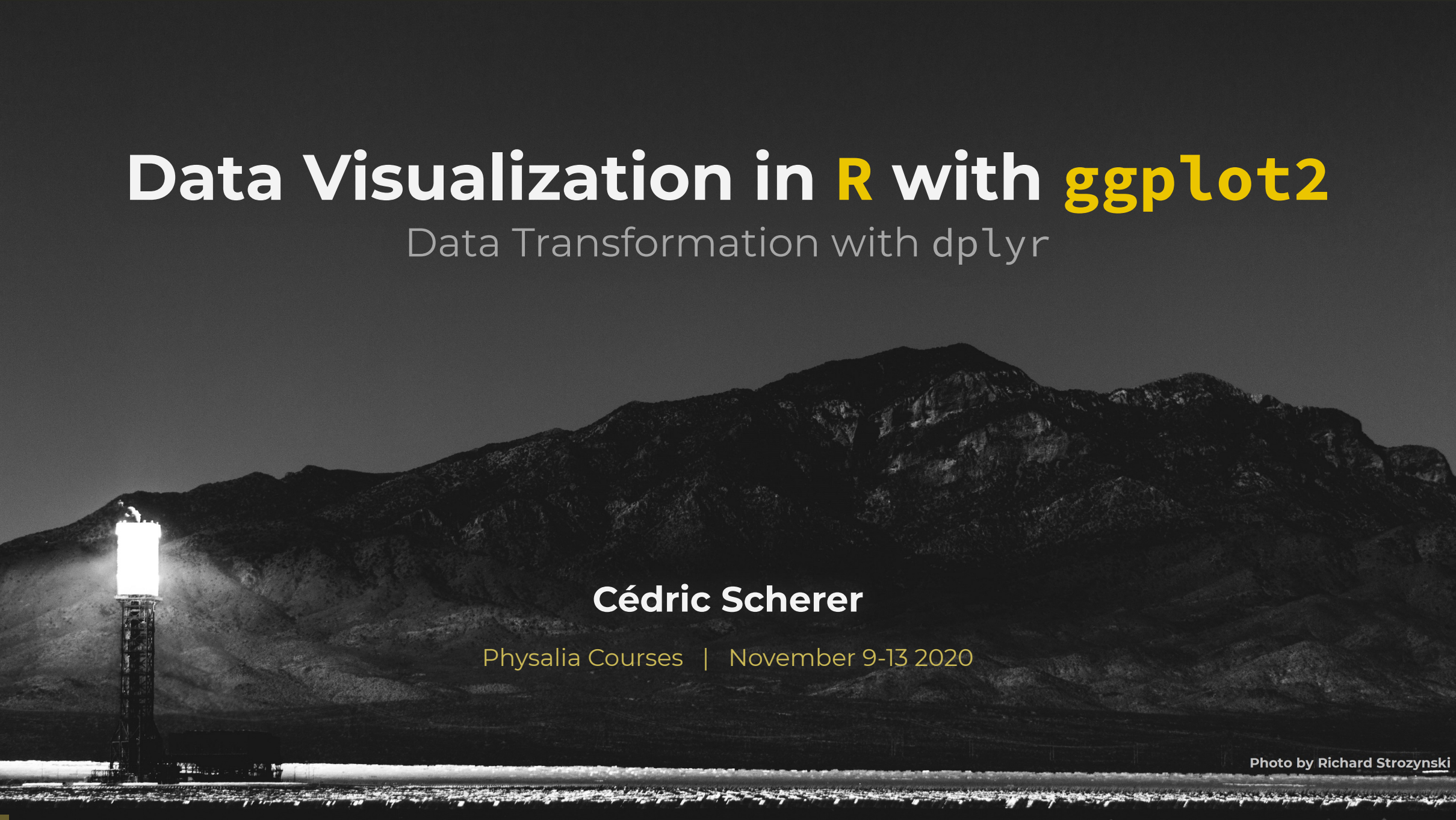
Data Visualization in R with ggplot2

Data Transformation with dplyr



Cédric Scherer

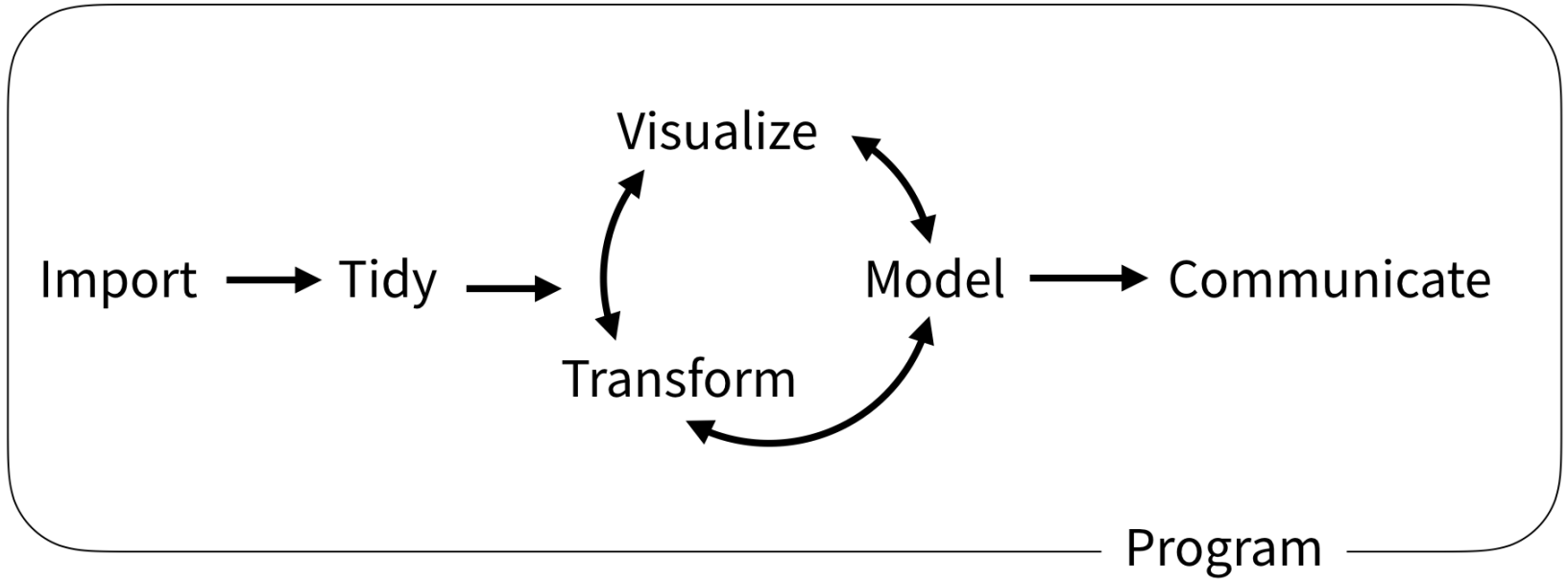
Physalia Courses | November 9-13 2020

Photo by Richard Strozynski

Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

The `tidyverse`

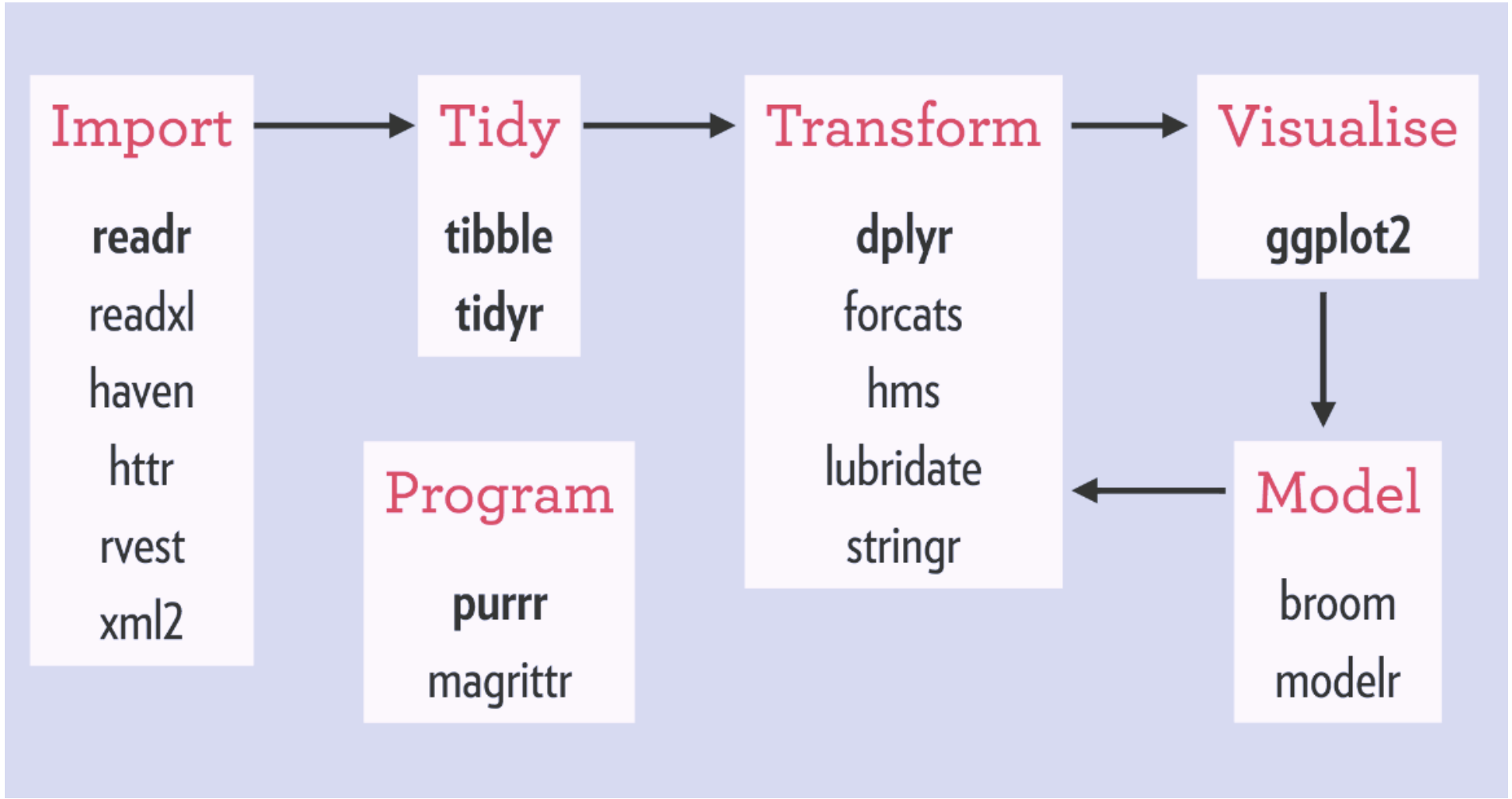
The `tidyverse` provides exemplary support for data wrangling and is the main reason for the recent popularity of `R`, especially in data-driven environments.



Source: rstudio-education.github.io/tidyverse-cookbook

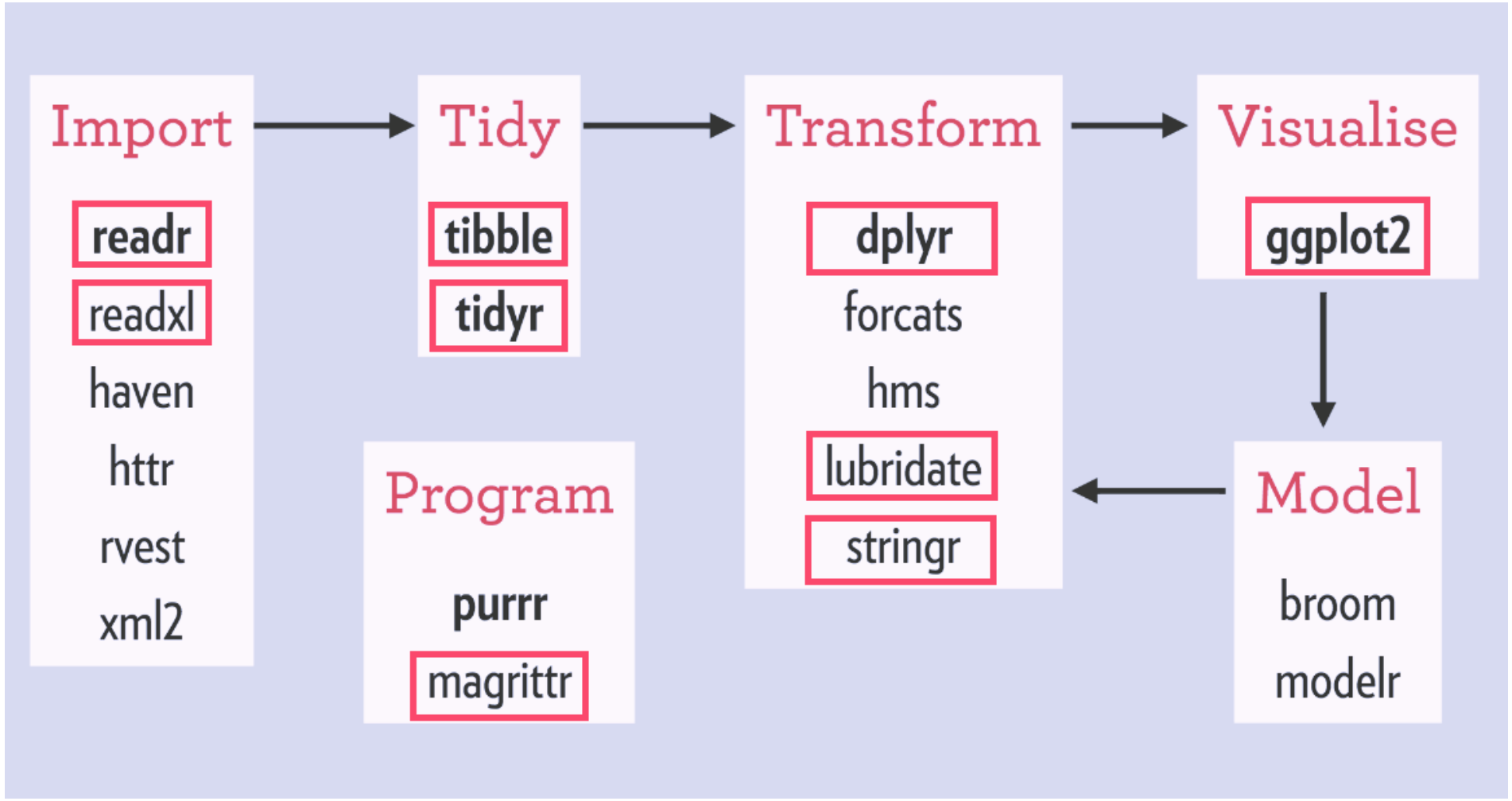
Modified from Hadley Wickham's "R for Data Science" (R4DS)

The Typical Data Science Project Flow



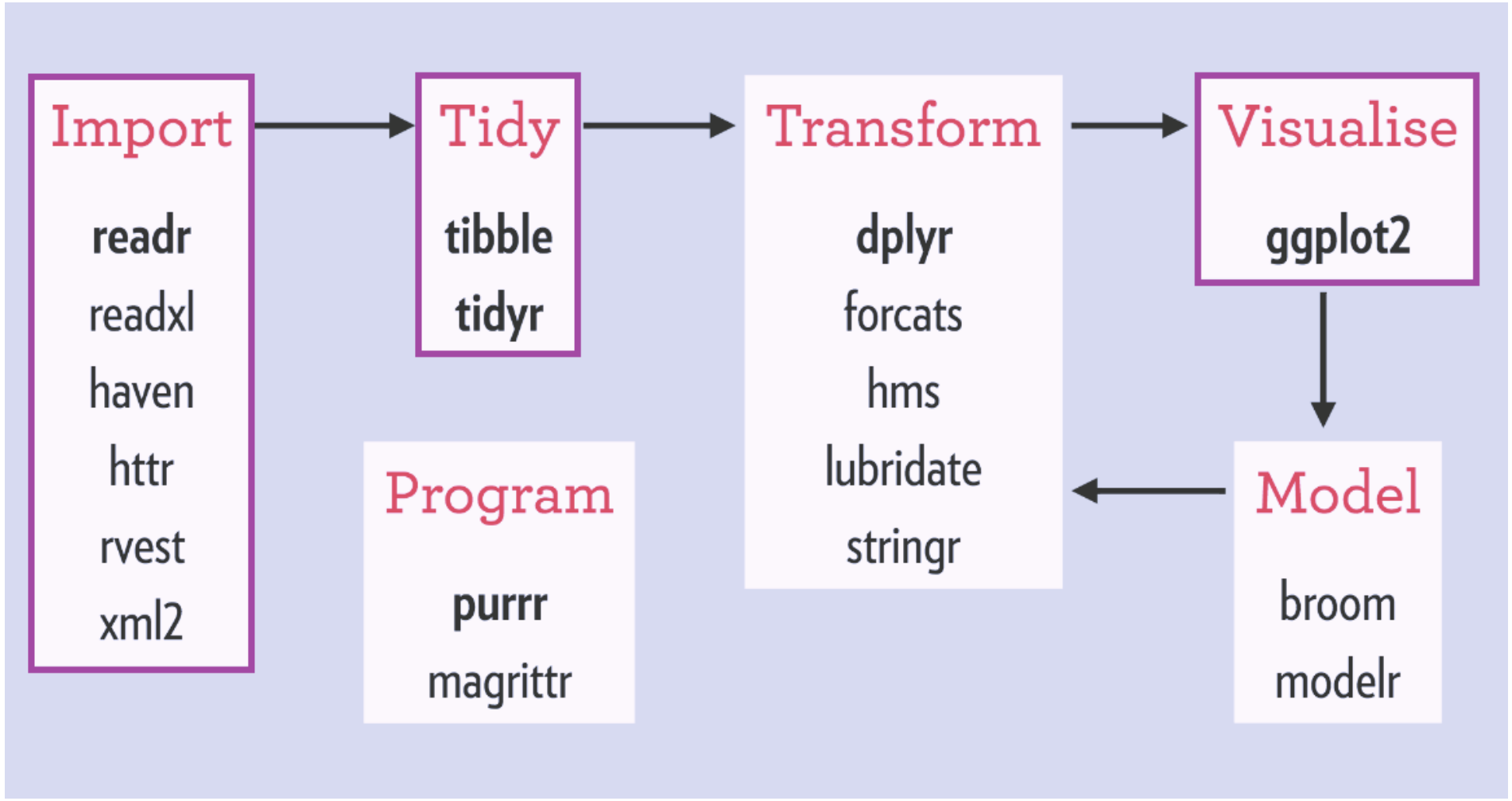
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

The Typical Data Science Project Flow



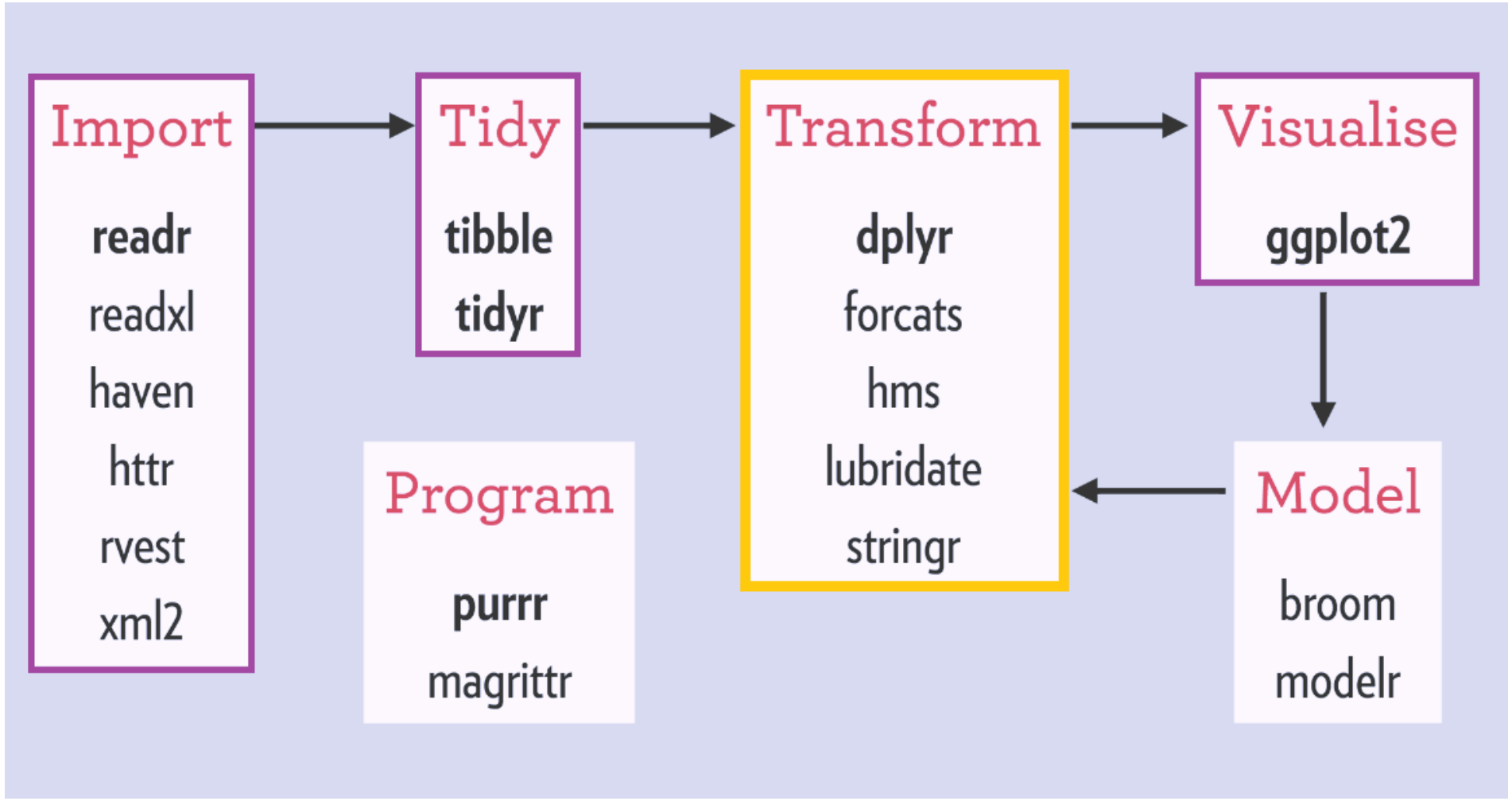
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

The Typical Data Science Project Flow



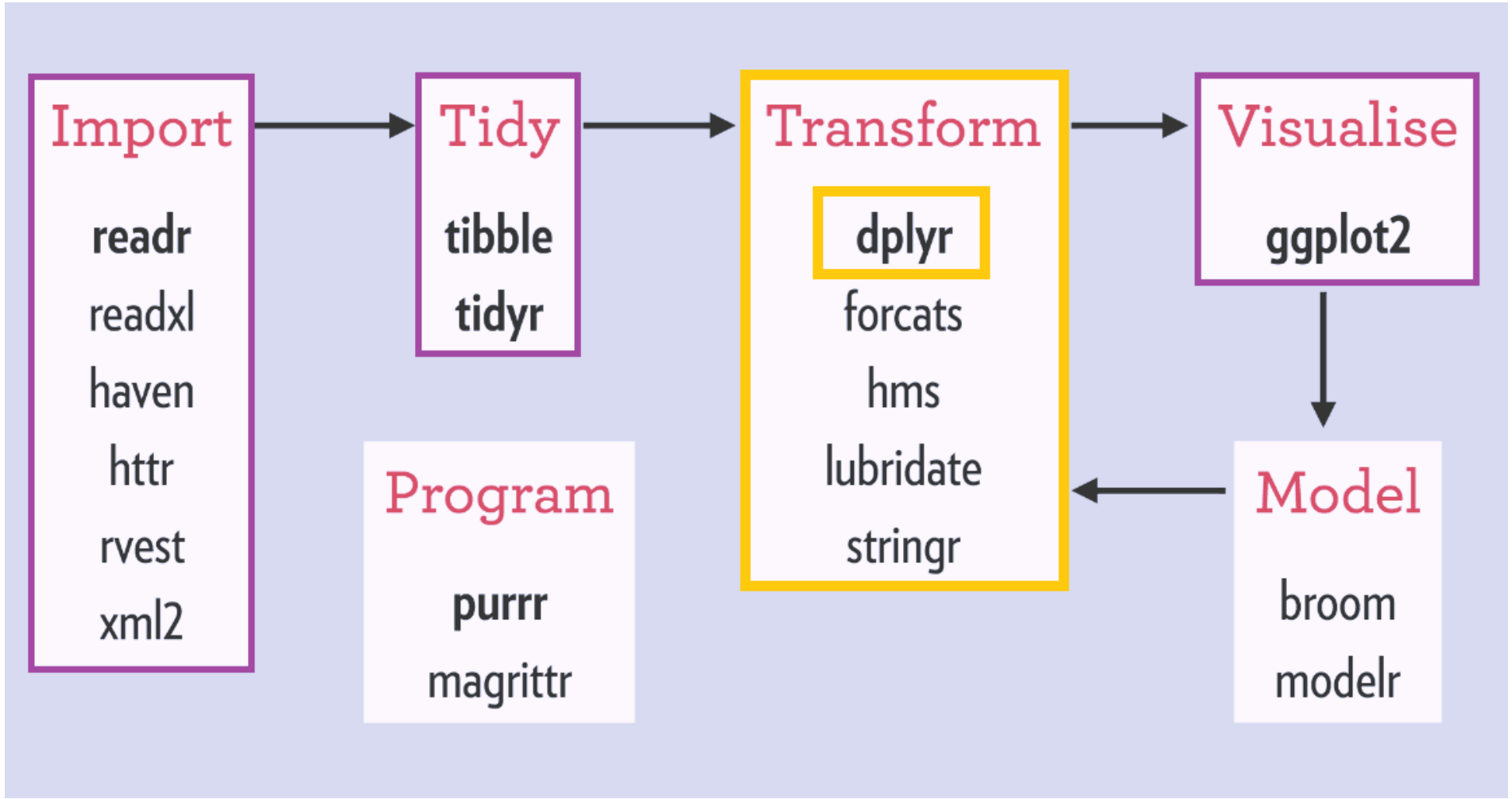
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

The Typical Data Science Project Flow



Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

The Typical Data Science Project Flow



Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

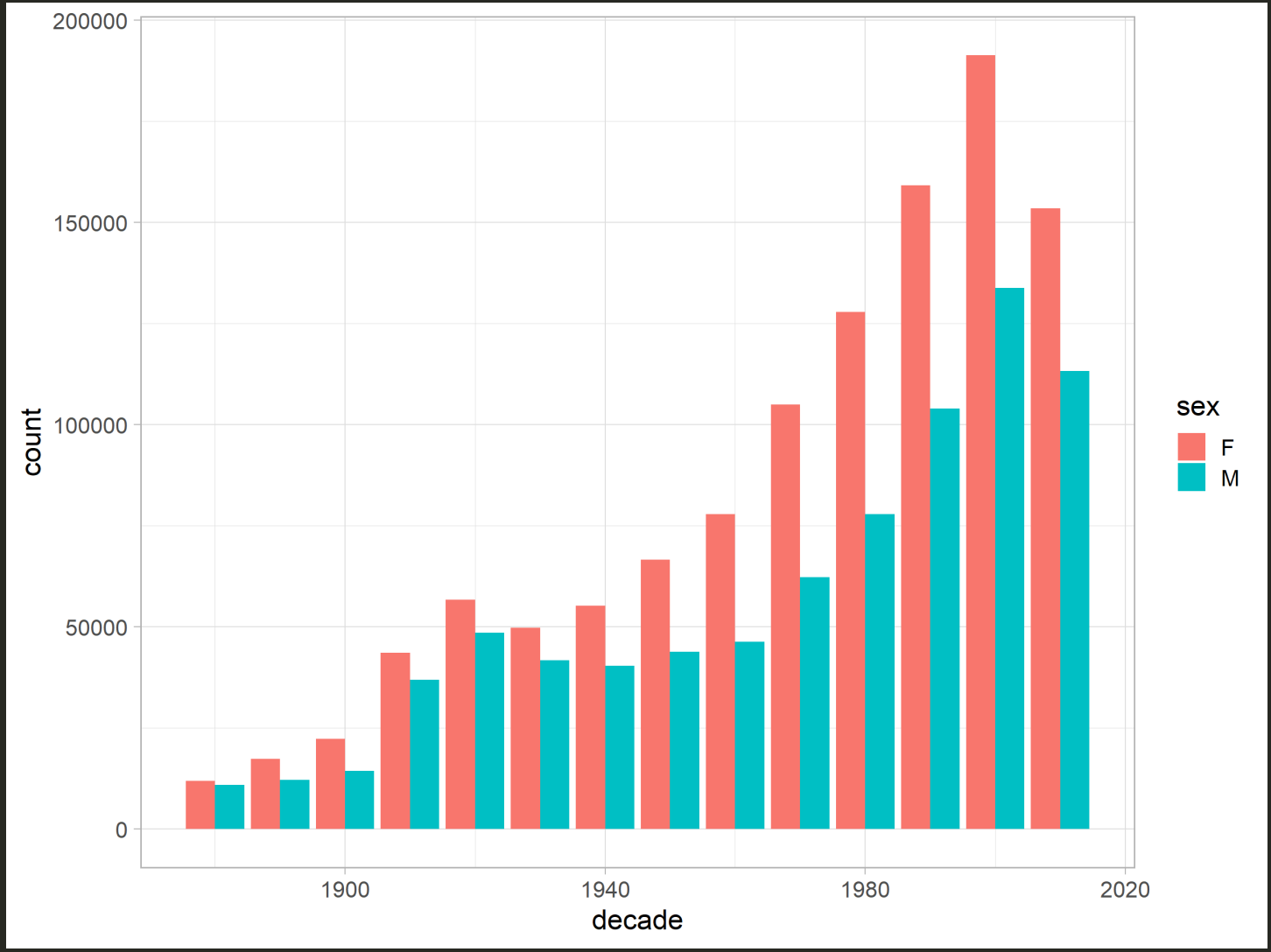
Our Example Data

- Count and proportional data of baby names used for at least five children in the US per sex and year from 1880 to 2017
- 1,924,665 rows (observations) and 4 columns (variables)
- Data source: **US Social Security Administration (SSA)**
- Available as data set `babynames` in the package `babynames`:
  `install.packages("babynames")` (includes 3 more data sets)
- See also `??babynames`

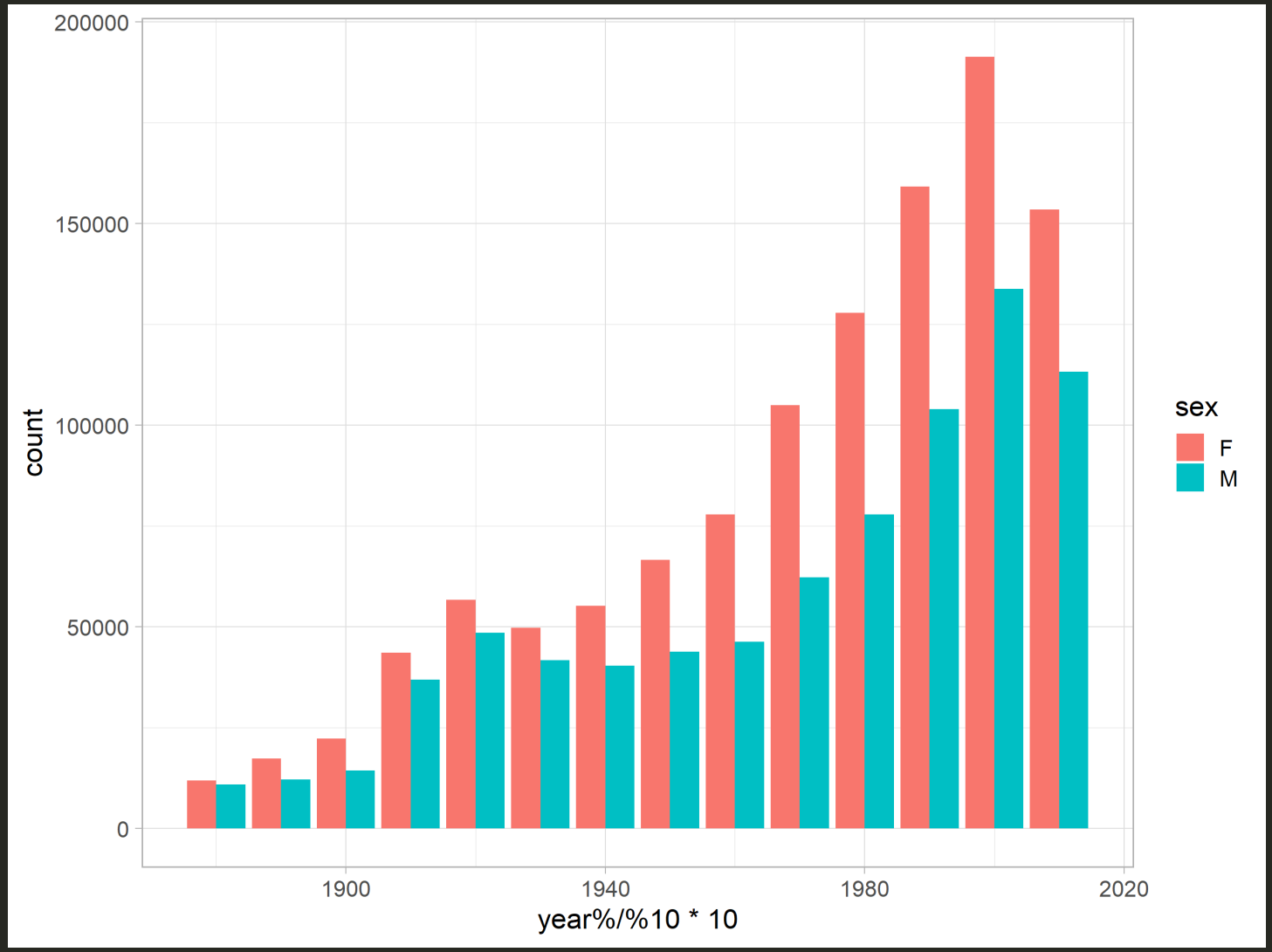
Our Example Data: babynames

```
library(babynames)

glimpse(babynames)
## Rows: 1,924,665
## Columns: 5
## $ year <dbl> 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 18...
## $ sex  <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F...
## $ name <chr> "Mary", "Anna", "Emma", "Elizabeth", "Minnie", "Margaret", "Id...
## $ n    <int> 7065, 2604, 2003, 1939, 1746, 1578, 1472, 1414, 1320, 1288, 12...
## $ prop <dbl> 0.07238359, 0.02667896, 0.02052149, 0.01986579, 0.01788843, 0....
```

Your Turn!


200000
150000
100000
50000
0
count
1900
1940
1980
2020
decade
sex
F
M

Your Turn! (Less Mean)



Excourse: Integer Division

```
1 %/% 10
## [1] 0
2 %/% 10
## [1] 0
11 %/% 10
## [1] 1
12 %/% 10
## [1] 1
```

Excourse: Integer Division

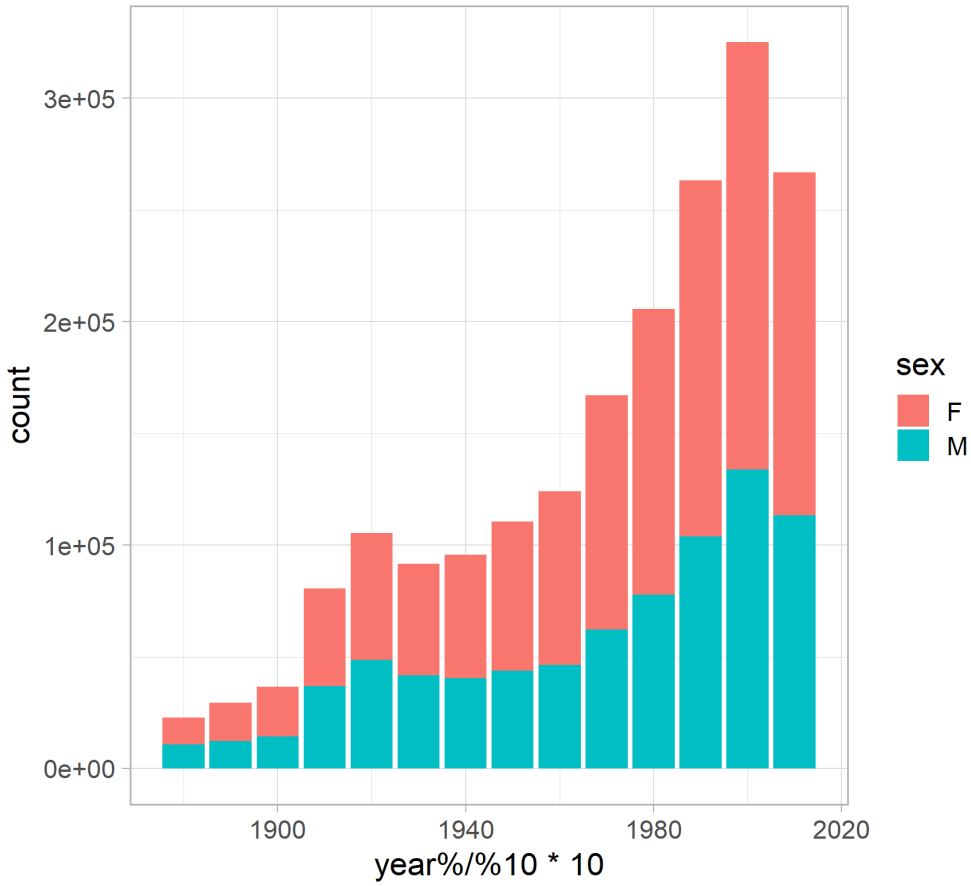
```
1 %/% 10
## [1] 0
2 %/% 10
## [1] 0
11 %/% 10
## [1] 1
12 %/% 10
## [1] 1

2019 %/% 10
## [1] 201
2020 %/% 10
## [1] 202
```

Excourse: Integer Division

```
1 %/% 10
## [1] 0
2 %/% 10
## [1] 0
11 %/% 10
## [1] 1
12 %/% 10
## [1] 1

2019 %/% 10
## [1] 201
2020 %/% 10
## [1] 202

2019 %/% 10 * 10
## [1] 2010
2020 %/% 10 * 10
## [1] 2020
```

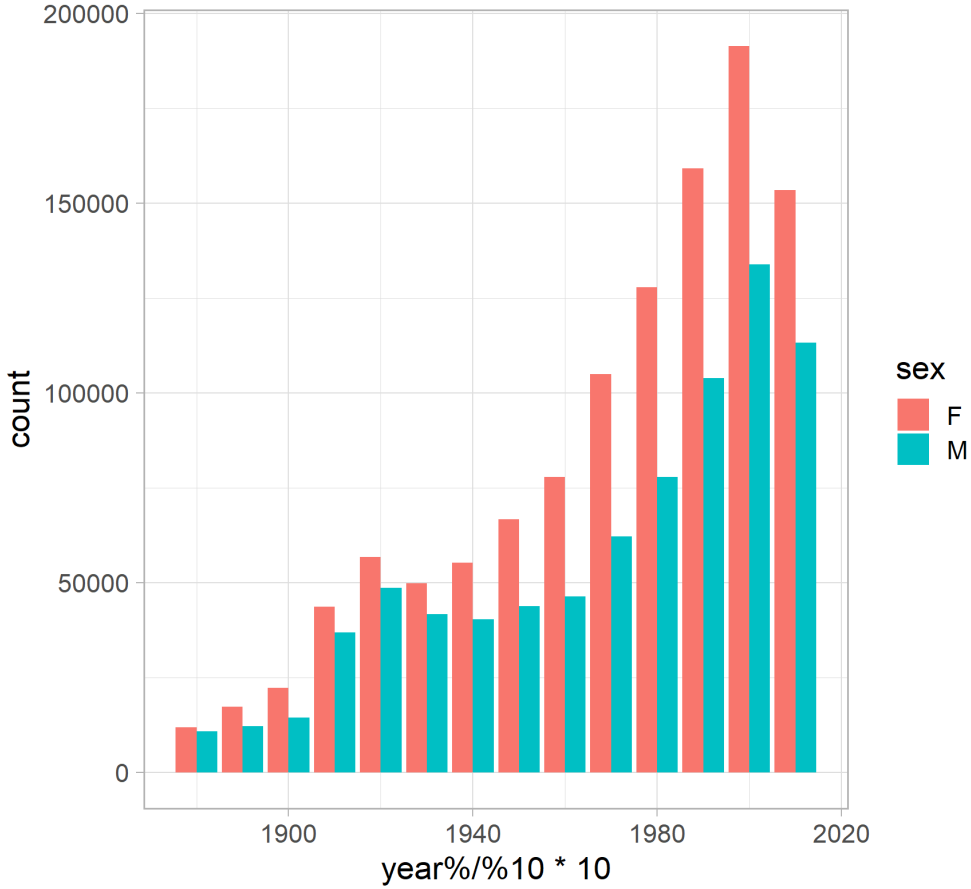
Your Turn: Dodged Bar Plots

```
ggplot(
  babynames,
  aes(
    year %/% 10 * 10,
    fill = sex
  )) +
  geom_bar(
    stat = "count"
  ) ## default: position = "stack"
```



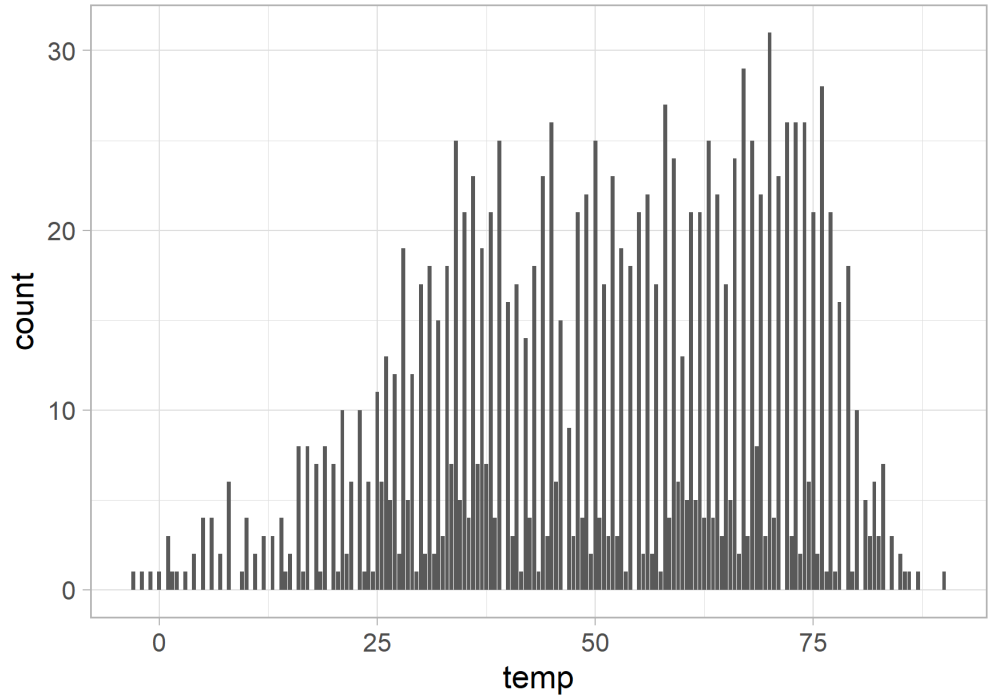
Your Turn: Dodged Bar Plots

```
ggplot(
  babynames,
  aes(
    year %/% 10 * 10,
    fill = sex
  )) +
  geom_bar(
    stat = "count",
    position = "dodge"
  )
```

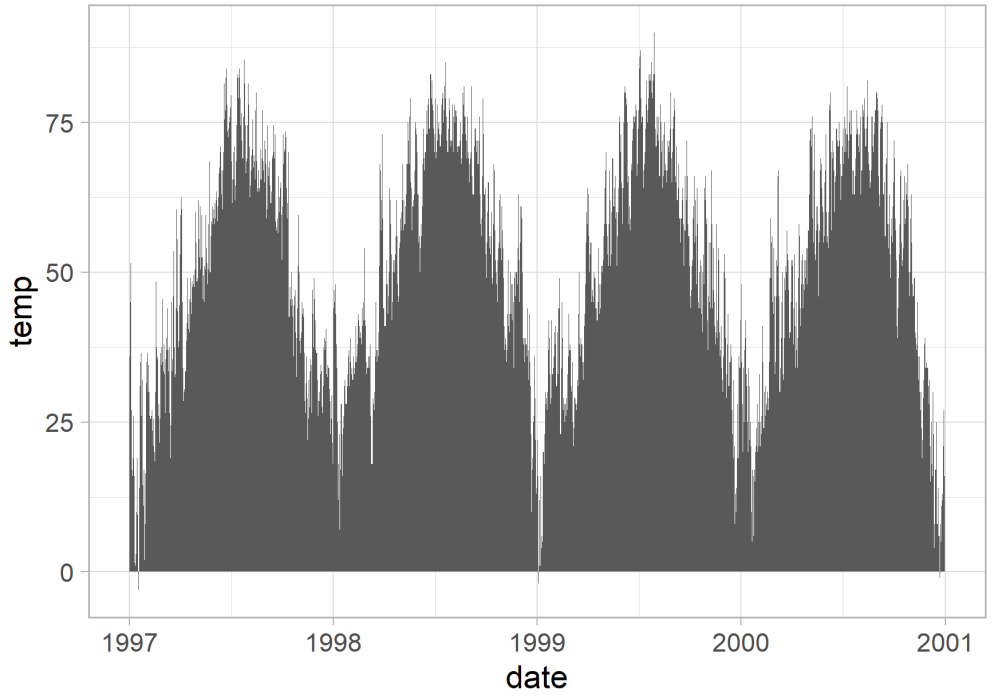


geom_bar()

```
ggplot(chic, aes(temp)) +
  geom_bar()
```

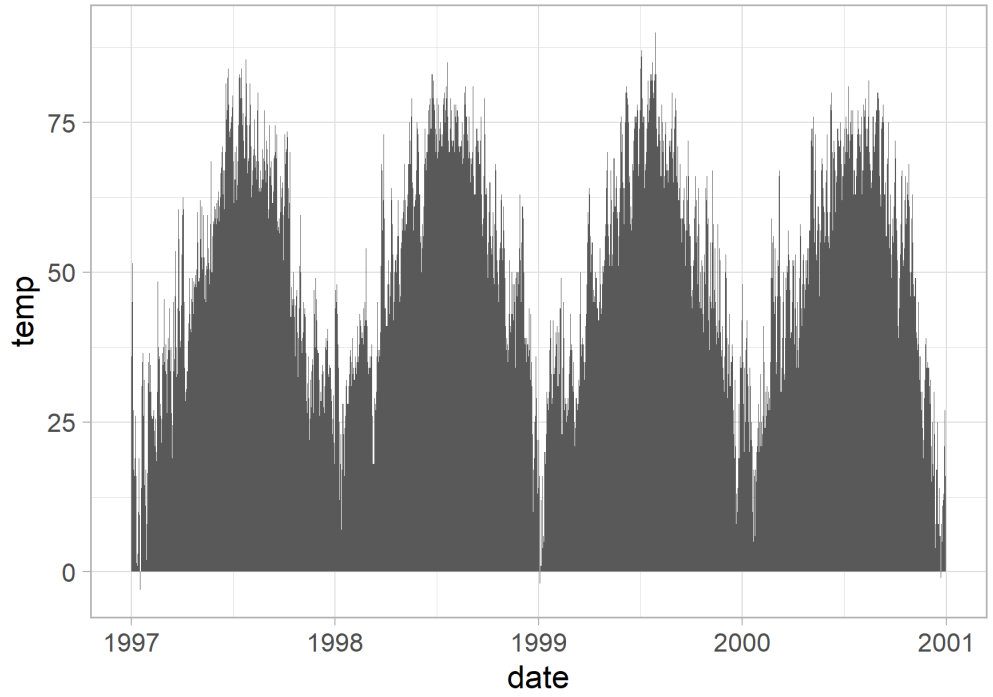



```
ggplot(chic, aes(date, temp)) +
  geom_bar(stat = "identity")
```

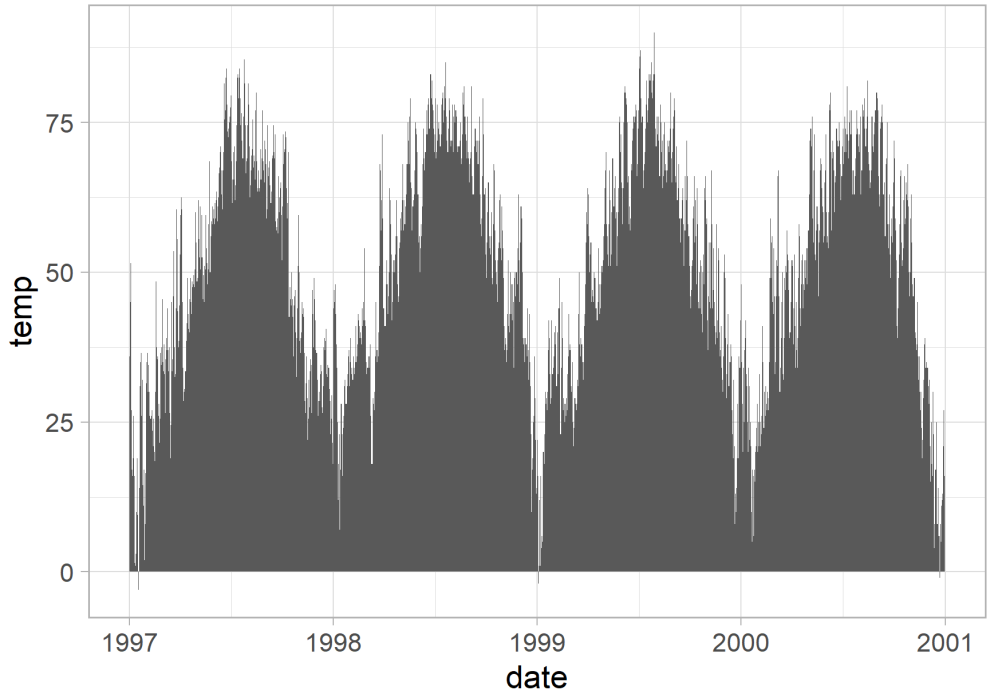


geom_bar() versus geom_col()

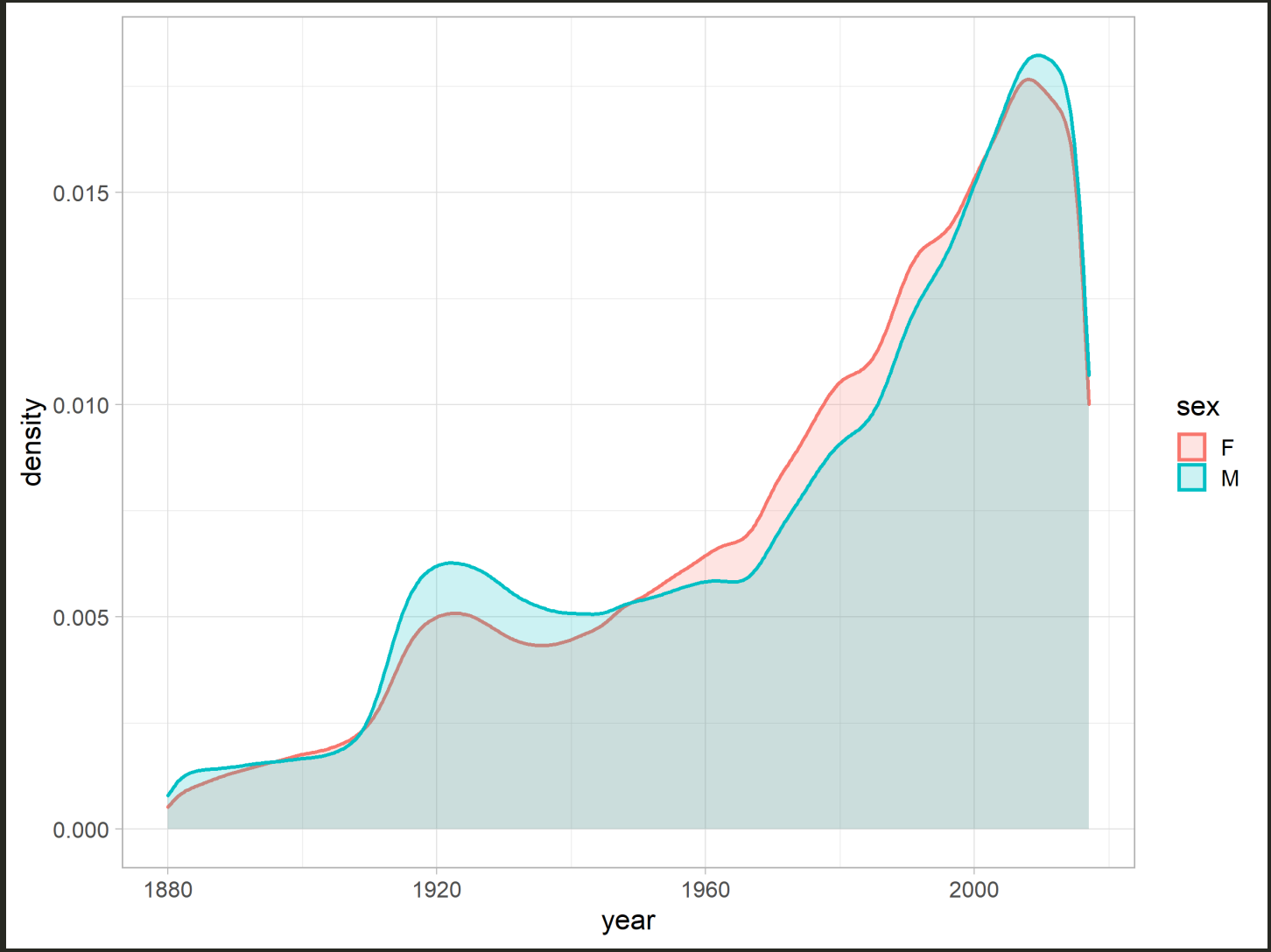
```
ggplot(chic, aes(date, temp)) +
  geom_col()
```




```
ggplot(chic, aes(date, temp)) +
  geom_bar(stat = "identity")
```

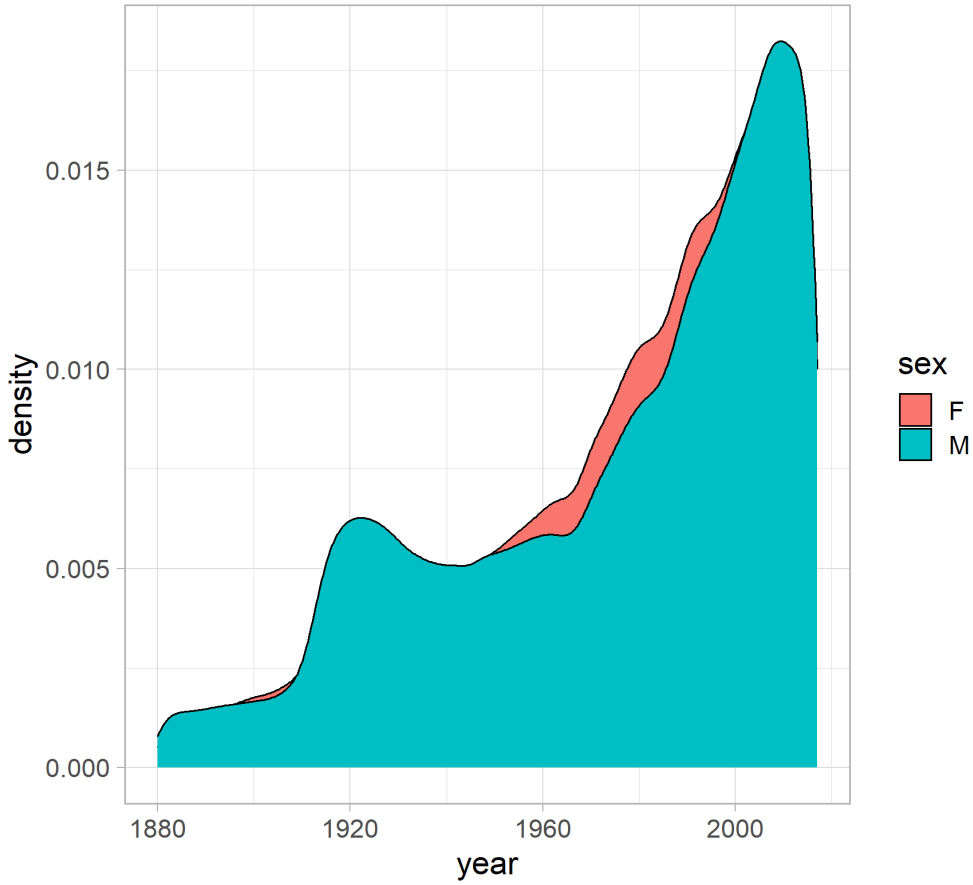


Your Turn!



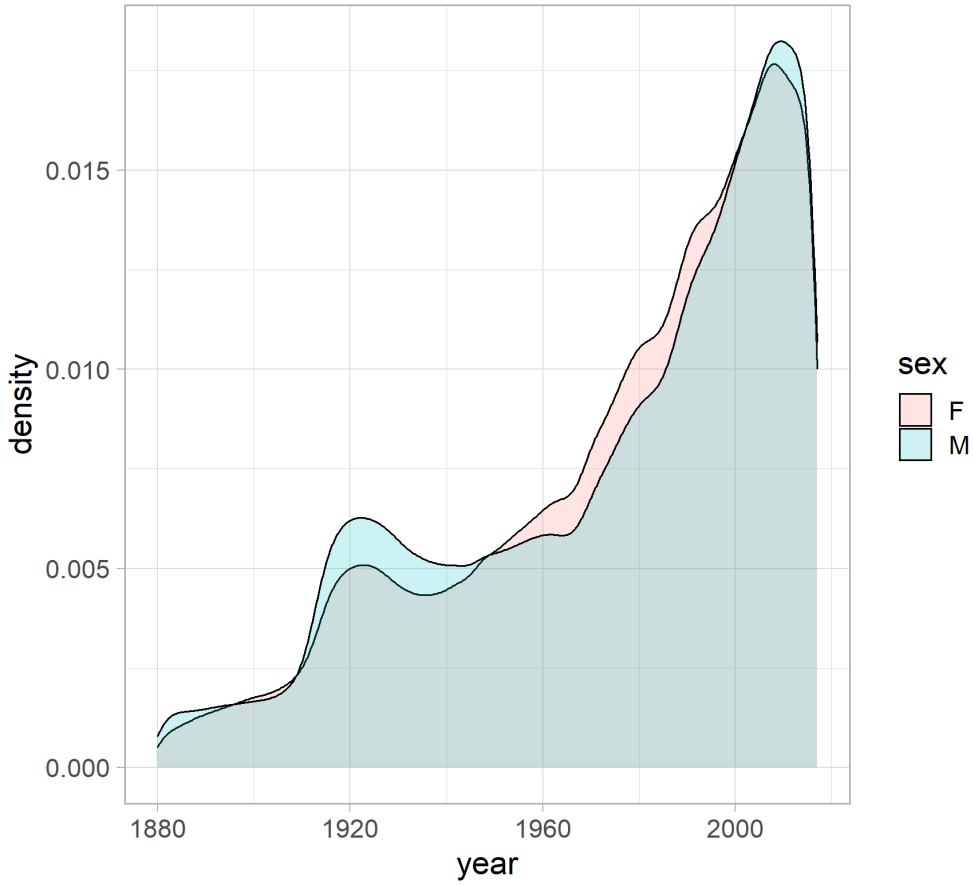
Your Turn: Density Plots

```
ggplot(
  babynames,
  aes(
    year,
    fill = sex
  )) +
  geom_density()
```



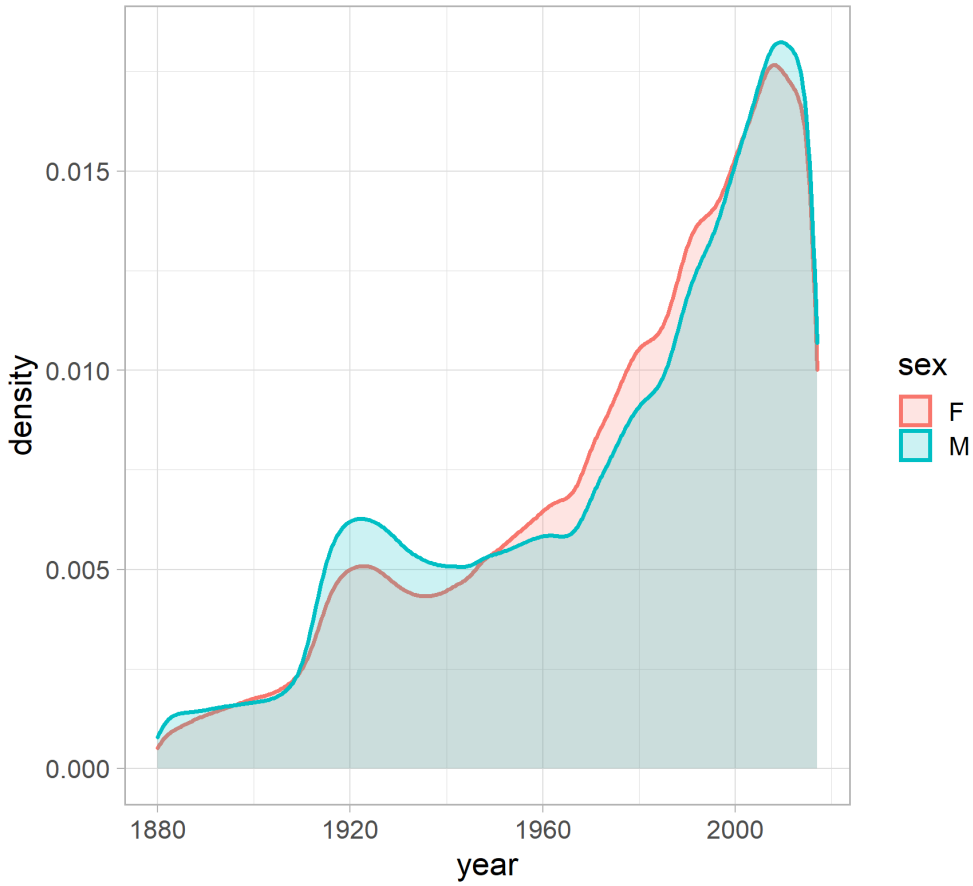
Your Turn: Density Plots

```
ggplot(
  babynames,
  aes(
    year,
    fill = sex
  )) +
  geom_density(
    alpha = .2
  )
```

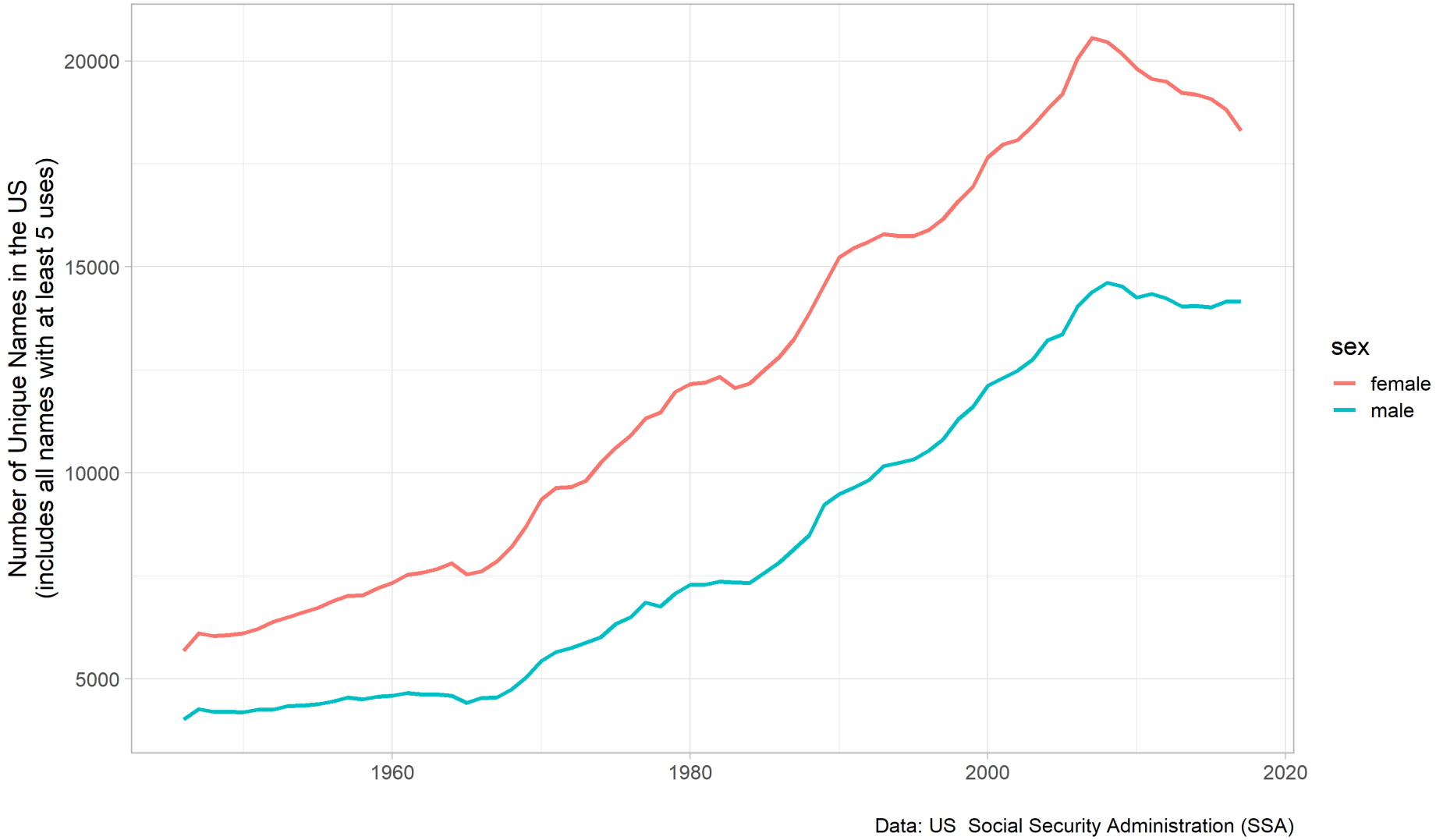


Your Turn: Density Plots

```
ggplot(
  babynames,
  aes(
    year,
    fill = sex,
    color = sex
  )) +
  geom_density(
    alpha = .2,
    size = 1
  )
```



The Rise of Unique Baby Names in the Post-WWII Era



Your Turn!

- Instead of coding, discuss what would be a way to create this plot.
    - Which geom did I use here?
    - What are details you notice but we can't do unitl now?
    - What can you already achieve with your current knowledge?
      If you like, give it a try afterwards!

The dplyr Package

Why should I care about `dplyr`?

- contains a set of convinient functions to perform common transformation and summary operations
- compared to base `R`:
  - syntax is more consistent
  - constrained options
  - always return a `data.frame` (actually, a `tibble`)
  - uses efficient data storage backends
  - can work with databases and data tables

The Main Verbs of `dplyr`

Function	Explanation
`filter()`	Pick **rows** with matching criteria
`select()`	Pick **columns** with matching criteria
`arrange()`	Reorder rows
`mutate()`	Create new variables
`summarize()` (or `summarise()`)	Sum up variables
`group_by()`	Create subsets

Consistent Syntax of `dplyr`

All functions take the same main arguments:

```
verb(data, condition)
```

The first argument is your data, subsequent arguments say what to do with the data frame, using the variable names.

filter()

Pick Rows with Matching Criteria

Pick Rows with Matching Criteria

`filter(data, condition)` allows you to select a subset of **rows**:

```
filter(babynames, year == 2000, n > 25000)
## # A tibble: 5 x 5
##    year sex   name        n   prop
##   <dbl> <chr> <chr>   <int>  <dbl>
## 1  2000 F     Emily   25953 0.0130
## 2  2000 M     Jacob   34471 0.0165
## 3  2000 M     Michael 32035 0.0153
## 4  2000 M     Matthew 28572 0.0137
## 5  2000 M     Joshua  27538 0.0132
```

Pick Rows with Matching Criteria

`filter(data, condition)` allows you to select a subset of **rows**:

```
filter(babynames, year == 2000, n > 25000)
```

Equivalent code in base `R`:

```
babynames[babynames$year == 2000 & babynames$n > 25000, ]
## # A tibble: 5 x 5
##    year sex   name        n   prop
##   <dbl> <chr> <chr>   <int>  <dbl>
## 1  2000 F     Emily   25953 0.0130
## 2  2000 M     Jacob   34471 0.0165
## 3  2000 M     Michael 32035 0.0153
## 4  2000 M     Matthew 28572 0.0137
## 5  2000 M     Joshua  27538 0.0132
```

Pick Rows with Matching Criteria

`filter(data, condition)` allows you to select a subset of **rows** → join filtering conditions with `&` and `|`:

```
filter(babynames, (name == "Cedric" | name == "Robert"))
## # A tibble: 426 x 5
##     year sex   name        n      prop
##    <dbl> <chr> <chr>   <int>     <dbl>
##  1  1880 F     Robert     12 0.000123
##  2  1880 M     Robert   2415 0.0204
##  3  1881 F     Robert      9 0.0000910
##  4  1881 M     Robert   2140 0.0198
##  5  1882 F     Robert     12 0.000104
##  6  1882 M     Robert   2500 0.0205
##  7  1883 F     Robert     11 0.0000916
##  8  1883 M     Robert   2334 0.0208
##  9  1884 F     Robert      7 0.0000509
## 10  1884 M     Robert   2468 0.0201
## # ... with 416 more rows
```

Pick Rows with Matching Criteria

`filter(data, condition)` → filter fir several using `%in%`:

```
filter(babynames, name %in% c("Cedric", "Robert"))
## # A tibble: 426 x 5
##     year sex   name        n      prop
##    <dbl> <chr> <chr>   <int>     <dbl>
##  1  1880 F     Robert     12 0.000123
##  2  1880 M     Robert   2415 0.0204
##  3  1881 F     Robert      9 0.0000910
##  4  1881 M     Robert   2140 0.0198
##  5  1882 F     Robert     12 0.000104
##  6  1882 M     Robert   2500 0.0205
##  7  1883 F     Robert     11 0.0000916
##  8  1883 M     Robert   2334 0.0208
##  9  1884 F     Robert      7 0.0000509
## 10  1884 M     Robert   2468 0.0201
## # ... with 416 more rows
```

Pick Rows with Matching Criteria

`filter(data, condition)` → join filtering conditions with `&` and `|`:

```
filter(babynames, (name == "Cedric" | name == "Robert") & year %in% 1900:1905)
## # A tibble: 15 x 5
##     year sex   name       n      prop
##    <dbl> <chr> <chr>  <int>     <dbl>
##  1  1900 F     Robert    24 0.0000755
##  2  1900 M     Robert  3821 0.0236
##  3  1901 F     Robert    16 0.0000629
##  4  1901 M     Robert  2543 0.0220
##  5  1902 F     Robert    21 0.0000749
##  6  1902 M     Robert  3180 0.0240
##  7  1903 F     Robert    13 0.0000467
##  8  1903 M     Robert  3044 0.0235
##  9  1903 M     Cedric     7 0.0000541
## 10  1904 F     Robert    13 0.0000445
## 11  1904 M     Robert  3414 0.0246
## 12  1904 M     Cedric     8 0.0000578
## 13  1905 F     Robert    21 0.0000678
## 14  1905 M     Robert  3410 0.0238
## 15  1905 M     Cedric     5 0.0000349
```

Pick Rows with Matching Criteria

`filter(data, condition)` → join filtering conditions with `&` and `|`:

```
filter(babynames, (name == "Cedric" | name == "Robert") & year %in% 1900:1905)
## # A tibble: 15 x 5
##     year sex   name       n      prop
##    <dbl> <chr> <chr>  <int>     <dbl>
##  1  1900 F     Robert    24 0.0000755
##  2  1900 M     Robert  3821 0.0236
##  3  1901 F     Robert    16 0.0000629
##  4  1901 M     Robert  2543 0.0220
##  5  1902 F     Robert    21 0.0000749
##  6  1902 M     Robert  3180 0.0240
##  7  1903 F     Robert    13 0.0000467
##  8  1903 M     Robert  3044 0.0235
##  9  1903 M     Cedric     7 0.0000541
## 10  1904 F     Robert    13 0.0000445
## 11  1904 M     Robert  3414 0.0246
## 12  1904 M     Cedric     8 0.0000578
## 13  1905 F     Robert    21 0.0000678
## 14  1905 M     Robert  3410 0.0238
## 15  1905 M     Cedric     5 0.0000349
```

Pick Rows with Matching Criteria

`filter(data, condition)` → remove rows with certain conditions wiht `!`:

```
filter(babynames, !(year %in% 1880:1900))
## # A tibble: 1,868,670 x 5
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1901 F     Mary       13136 0.0517
##  2  1901 F     Helen       5247 0.0206
##  3  1901 F     Anna        4923 0.0194
##  4  1901 F     Margaret    4424 0.0174
##  5  1901 F     Ruth        3974 0.0156
##  6  1901 F     Elizabeth   3216 0.0127
##  7  1901 F     Marie       3157 0.0124
##  8  1901 F     Florence    3131 0.0123
##  9  1901 F     Ethel       3067 0.0121
## 10  1901 F     Lillian     2681 0.0105
## # ... with 1,868,660 more rows
```

select()

Pick Columns with Matching Criteria

Pick Columns with Matching Criteria

`select(data, condition)` allows you to select a subset of **columns**:

```
select(babynames, name, year, n)
## # A tibble: 1,924,665 x 3
##    name        year     n
##    <chr>      <dbl> <int>
##  1 Mary        1880  7065
##  2 Anna        1880  2604
##  3 Emma        1880  2003
##  4 Elizabeth   1880  1939
##  5 Minnie      1880  1746
##  6 Margaret    1880  1578
##  7 Ida         1880  1472
##  8 Alice       1880  1414
##  9 Bertha      1880  1320
## 10 Sarah       1880  1288
## # ... with 1,924,655 more rows
```

Pick Columns with Matching Criteria

`select(data, condition)` allows you to select a subset of **columns**:

```
select(babynames, name, year, n)
```

Equivalent code in base `R`:

```
babynames[, c("name", "year", "n")]
## # A tibble: 1,924,665 x 3
##    name        year     n
##    <chr>      <dbl> <int>
##  1 Mary        1880  7065
##  2 Anna        1880  2604
##  3 Emma        1880  2003
##  4 Elizabeth   1880  1939
##  5 Minnie      1880  1746
##  6 Margaret    1880  1578
##  7 Ida         1880  1472
##  8 Alice       1880  1414
##  9 Bertha      1880  1320
## 10 Sarah       1880  1288
## # ... with 1,924,655 more rows
```

Pick Columns with Matching Criteria

`select(data, condition)` allows you to select a subset of **columns**:

```
select(babynames, name, year, n)
## # A tibble: 1,924,665 x 3
##    name       year     n
##    <chr>     <dbl> <int>
##  1 Mary       1880  7065
##  2 Anna       1880  2604
##  3 Emma       1880  2003
##  4 Elizabeth  1880  1939
##  5 Minnie     1880  1746
##  6 Margaret   1880  1578
##  7 Ida        1880  1472
##  8 Alice      1880  1414
##  9 Bertha     1880  1320
## 10 Sarah      1880  1288
## # ... with 1,924,655 more rows
```

```
select(babynames, -prop, -sex)
## # A tibble: 1,924,665 x 3
##     year name           n
##    <dbl> <chr>      <int>
##  1  1880 Mary        7065
##  2  1880 Anna        2604
##  3  1880 Emma        2003
##  4  1880 Elizabeth   1939
##  5  1880 Minnie      1746
##  6  1880 Margaret    1578
##  7  1880 Ida         1472
##  8  1880 Alice       1414
##  9  1880 Bertha      1320
## 10  1880 Sarah       1288
## # ... with 1,924,655 more rows
```

Pick Columns with Matching Criteria

`select(data, condition)` allows you to move **columns**:

```
select(babynames, n, prop, year, name, sex)
## # A tibble: 1,924,665 x 5
##        n   prop  year name      sex
##    <int>  <dbl> <dbl> <chr>     <chr>
##  1  7065 0.0724  1880 Mary      F
##  2  2604 0.0267  1880 Anna      F
##  3  2003 0.0205  1880 Emma      F
##  4  1939 0.0199  1880 Elizabeth F
##  5  1746 0.0179  1880 Minnie    F
##  6  1578 0.0162  1880 Margaret  F
##  7  1472 0.0151  1880 Ida       F
##  8  1414 0.0145  1880 Alice     F
##  9  1320 0.0135  1880 Bertha    F
## 10  1288 0.0132  1880 Sarah     F
## # ... with 1,924,655 more rows
```

Pick Columns with Matching Criteria

`select(data, condition)` allows you to move **columns**:

```
select(babynames, n, everything())
## # A tibble: 1,924,665 x 5
##        n  year sex   name        prop
##    <int> <dbl> <chr> <chr>      <dbl>
##  1  7065  1880 F     Mary      0.0724
##  2  2604  1880 F     Anna      0.0267
##  3  2003  1880 F     Emma      0.0205
##  4  1939  1880 F     Elizabeth 0.0199
##  5  1746  1880 F     Minnie    0.0179
##  6  1578  1880 F     Margaret  0.0162
##  7  1472  1880 F     Ida       0.0151
##  8  1414  1880 F     Alice     0.0145
##  9  1320  1880 F     Bertha    0.0135
## 10  1288  1880 F     Sarah     0.0132
## # ... with 1,924,655 more rows
```

arrange()

Reorder Rows

Reorder Rows

`arrange(data, condition)` allows you to order rows by variables:

```
arrange(babynames, prop)
## # A tibble: 1,924,665 x 5
##     year sex   name            n       prop
##    <dbl> <chr> <chr>       <int>      <dbl>
##  1  2007 M     Aaban           5 0.00000226
##  2  2007 M     Aareon          5 0.00000226
##  3  2007 M     Aaris           5 0.00000226
##  4  2007 M     Abd             5 0.00000226
##  5  2007 M     Abdulazeez      5 0.00000226
##  6  2007 M     Abdulhadi       5 0.00000226
##  7  2007 M     Abdulhamid      5 0.00000226
##  8  2007 M     Abdulkadir      5 0.00000226
##  9  2007 M     Abdulraheem     5 0.00000226
## 10  2007 M     Abdulrahim      5 0.00000226
## # ... with 1,924,655 more rows
```

Reorder Rows

`arrange(data, condition)` allows you to order rows by variables:

```
arrange(babynames, prop)
```

Equivalent code in base `R`:

```
babynames[order(babynames$prop), ]
## # A tibble: 1,924,665 x 5
##     year sex   name            n       prop
##    <dbl> <chr> <chr>       <int>      <dbl>
##  1  2007 M     Aaban           5 0.00000226
##  2  2007 M     Aareon          5 0.00000226
##  3  2007 M     Aaris           5 0.00000226
##  4  2007 M     Abd             5 0.00000226
##  5  2007 M     Abdulazeez      5 0.00000226
##  6  2007 M     Abdulhadi       5 0.00000226
##  7  2007 M     Abdulhamid      5 0.00000226
##  8  2007 M     Abdulkadir      5 0.00000226
##  9  2007 M     Abdulraheem     5 0.00000226
## 10  2007 M     Abdulrahim      5 0.00000226
## # ... with 1,924,655 more rows
```

Reorder Rows

`arrange(data, condition)` allows you to order rows by variables:

```
arrange(babynames, year, sex, -prop)
## # A tibble: 1,924,665 x 5
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1880 F     Mary        7065 0.0724
##  2  1880 F     Anna        2604 0.0267
##  3  1880 F     Emma        2003 0.0205
##  4  1880 F     Elizabeth   1939 0.0199
##  5  1880 F     Minnie      1746 0.0179
##  6  1880 F     Margaret    1578 0.0162
##  7  1880 F     Ida         1472 0.0151
##  8  1880 F     Alice       1414 0.0145
##  9  1880 F     Bertha      1320 0.0135
## 10  1880 F     Sarah       1288 0.0132
## # ... with 1,924,655 more rows
```

mutate()

Create New Variables

Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) based or not based on other variables:

```
mutate(babynames, decade = year %/% 10 * 10)
## # A tibble: 1,924,665 x 6
##     year sex   name           n   prop decade
##    <dbl> <chr> <chr>      <int>  <dbl>  <dbl>
##  1  1880 F     Mary        7065 0.0724   1880
##  2  1880 F     Anna        2604 0.0267   1880
##  3  1880 F     Emma        2003 0.0205   1880
##  4  1880 F     Elizabeth   1939 0.0199   1880
##  5  1880 F     Minnie      1746 0.0179   1880
##  6  1880 F     Margaret    1578 0.0162   1880
##  7  1880 F     Ida         1472 0.0151   1880
##  8  1880 F     Alice       1414 0.0145   1880
##  9  1880 F     Bertha      1320 0.0135   1880
## 10  1880 F     Sarah       1288 0.0132   1880
## # ... with 1,924,655 more rows
```

Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) based or not based on other variables:

```
mutate(babynames, decade = year %/% 10 * 10)
```

Equivalent code in base `R`:

```
#babynames$decade <- babynames$year %/% 10 * 10
## -> changes object `babynames`!
transform(babynames, decade = year %/% 10 * 10)
##       year sex         name    n       prop decade
## 1     1880   F         Mary 7065 0.07238359   1880
## 2     1880   F         Anna 2604 0.02667896   1880
## 3     1880   F         Emma 2003 0.02052149   1880
## 4     1880   F    Elizabeth 1939 0.01986579   1880
## 5     1880   F       Minnie 1746 0.01788843   1880
## 6     1880   F     Margaret 1578 0.01616720   1880
## 7     1880   F          Ida 1472 0.01508119   1880
## 8     1880   F        Alice 1414 0.01448696   1880
## 9     1880   F       Bertha 1320 0.01352390   1880
## 10    1880   F        Sarah 1288 0.01319605   1880
## 11    1880   F        Annie 1258 0.01288868   1880
```

Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) based or not based on other variables:

```
mutate(babynames, name = str_to_lower(name))
## # A tibble: 1,924,665 x 5
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1880 F     mary        7065 0.0724
##  2  1880 F     anna        2604 0.0267
##  3  1880 F     emma        2003 0.0205
##  4  1880 F     elizabeth   1939 0.0199
##  5  1880 F     minnie      1746 0.0179
##  6  1880 F     margaret    1578 0.0162
##  7  1880 F     ida         1472 0.0151
##  8  1880 F     alice       1414 0.0145
##  9  1880 F     bertha      1320 0.0135
## 10  1880 F     sarah       1288 0.0132
## # ... with 1,924,655 more rows
```

Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) based or not based on other variables:

```
mutate(babynames, note = "Please check!")
## # A tibble: 1,924,665 x 6
##     year sex   name           n   prop note
##    <dbl> <chr> <chr>      <int>  <dbl> <chr>
##  1  1880 F     Mary        7065 0.0724 Please check!
##  2  1880 F     Anna        2604 0.0267 Please check!
##  3  1880 F     Emma        2003 0.0205 Please check!
##  4  1880 F     Elizabeth   1939 0.0199 Please check!
##  5  1880 F     Minnie      1746 0.0179 Please check!
##  6  1880 F     Margaret    1578 0.0162 Please check!
##  7  1880 F     Ida         1472 0.0151 Please check!
##  8  1880 F     Alice       1414 0.0145 Please check!
##  9  1880 F     Bertha      1320 0.0135 Please check!
## 10  1880 F     Sarah       1288 0.0132 Please check!
## # ... with 1,924,655 more rows
```

Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) based or not based on other variables:

```
mutate(babynames, note = if_else(year < 1900, "Please check!", "Ok"))
## # A tibble: 1,924,665 x 6
##     year sex   name           n   prop note
##    <dbl> <chr> <chr>      <int>  <dbl> <chr>
##  1  1880 F     Mary        7065 0.0724 Please check!
##  2  1880 F     Anna        2604 0.0267 Please check!
##  3  1880 F     Emma        2003 0.0205 Please check!
##  4  1880 F     Elizabeth   1939 0.0199 Please check!
##  5  1880 F     Minnie      1746 0.0179 Please check!
##  6  1880 F     Margaret    1578 0.0162 Please check!
##  7  1880 F     Ida         1472 0.0151 Please check!
##  8  1880 F     Alice       1414 0.0145 Please check!
##  9  1880 F     Bertha      1320 0.0135 Please check!
## 10  1880 F     Sarah       1288 0.0132 Please check!
## # ... with 1,924,655 more rows
```

Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) based or not based on other variables:

```
mutate(babynames, id = row_number())
## # A tibble: 1,924,665 x 6
##     year sex   name           n   prop    id
##    <dbl> <chr> <chr>      <int>  <dbl> <int>
##  1  1880 F     Mary        7065 0.0724     1
##  2  1880 F     Anna        2604 0.0267     2
##  3  1880 F     Emma        2003 0.0205     3
##  4  1880 F     Elizabeth   1939 0.0199     4
##  5  1880 F     Minnie      1746 0.0179     5
##  6  1880 F     Margaret    1578 0.0162     6
##  7  1880 F     Ida         1472 0.0151     7
##  8  1880 F     Alice       1414 0.0145     8
##  9  1880 F     Bertha      1320 0.0135     9
## 10  1880 F     Sarah       1288 0.0132    10
## # ... with 1,924,655 more rows
```

Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) based or not based on other variables:

```
mpg2 <- select(mpg, model, hwy, cty) ## only for visualization
mutate(mpg2, diff = hwy - cty, perc = diff / hwy)
## # A tibble: 234 x 5
##    model        hwy   cty  diff  perc
##    <chr>      <int> <int> <int> <dbl>
##  1 a4            29    18    11 0.379
##  2 a4            29    21     8 0.276
##  3 a4            31    20    11 0.355
##  4 a4            30    21     9 0.3
##  5 a4            26    16    10 0.385
##  6 a4            26    18     8 0.308
##  7 a4            27    18     9 0.333
##  8 a4 quattro    26    18     8 0.308
##  9 a4 quattro    25    16     9 0.36
## 10 a4 quattro    28    20     8 0.286
## # ... with 224 more rows
```

Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) based or not based on other variables:

```
mpg2 <- select(mpg, model, hwy, cty) ## only for visualization
mutate(mpg2, diff = hwy - cty, perc = diff / hwy)
```

Equivalent code in base R:

```
mpg_diff <- transform(mpg2, diff = hwy - cty)
transform(mpg_diff, perc = diff / hwy)
##                       model hwy cty diff      perc
## 1                        a4  29  18   11 0.3793103
## 2                        a4  29  21    8 0.2758621
## 3                        a4  31  20   11 0.3548387
## 4                        a4  30  21    9 0.3000000
## 5                        a4  26  16   10 0.3846154
## 6                        a4  26  18    8 0.3076923
## 7                        a4  27  18    9 0.3333333
## 8                a4 quattro  26  18    8 0.3076923
## 9                a4 quattro  25  16    9 0.3600000
## 10               a4 quattro  28  20    8 0.2857143
## 11               a4 quattro  27  19    8 0.2962963
```

Create New Variables

`mutate()` in combination with `across()` allows you to conditionally create several new columns:

```
mutate(mpg2, across(is.numeric, as.character))
## # A tibble: 234 x 3
##    model      hwy   cty
##    <chr>      <chr> <chr>
##  1 a4         29    18
##  2 a4         29    21
##  3 a4         31    20
##  4 a4         30    21
##  5 a4         26    16
##  6 a4         26    18
##  7 a4         27    18
##  8 a4 quattro 26    18
##  9 a4 quattro 25    16
## 10 a4 quattro 28    20
## # ... with 224 more rows
```

summarize()

Sum Up Variables

Sum Up Variables

`summarize(data, condition)` allows you to calculate summary statistics:

```
summarize(babynames, unique_children = sum(n, na.rm = TRUE))
## # A tibble: 1 x 1
##   unique_children
##             <int>
## 1       348120517
```

Sum Up Variables

`summarize(data, condition)` allows you to calculate summary statistics:

```
summarize(babynames, unique_children = sum(n, na.rm = TRUE))
## # A tibble: 1 x 1
##   unique_children
##             <int>
## 1       348120517
```

Equivalent code in base R:

```
sum(babynames$n, na.rm = TRUE)
## [1] 348120517
```

Sum Up Variables

`summarize(data, condition)` allows you to calculate summary statistics:

```
summarize(babynames, n = n())
## # A tibble: 1 x 1
##         n
##     <int>
## 1 1924665
```

Sum Up Variables

`summarize(data, condition)` allows you to calculate summary statistics:

```
summarize(babynames, n = n(), first_year = min(year), last_year = max(year))
## # A tibble: 1 x 3
##         n first_year last_year
##     <int>      <dbl>     <dbl>
## 1 1924665       1880      2017
```

Sum Up Variables

`summarize()` in combination with `across()` allows you to calculate summary statistics for multiple variables:

```
summarize(mpg, across(is.numeric, mean))
## # A tibble: 1 x 5
##   displ  year   cyl   cty   hwy
##   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  3.47 2004.  5.89  16.9  23.4
```

group_by()

Create Subsets

Create Subsets

`group_by(data, condition)` allows you to break down your data into specified groups based on variables:

```
group_by(babynames, sex)
## # A tibble: 1,924,665 x 5
## # Groups:   sex [2]
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1880 F     Mary        7065 0.0724
##  2  1880 F     Anna        2604 0.0267
##  3  1880 F     Emma        2003 0.0205
##  4  1880 F     Elizabeth   1939 0.0199
##  5  1880 F     Minnie      1746 0.0179
##  6  1880 F     Margaret    1578 0.0162
##  7  1880 F     Ida         1472 0.0151
##  8  1880 F     Alice       1414 0.0145
##  9  1880 F     Bertha      1320 0.0135
## 10  1880 F     Sarah       1288 0.0132
## # ... with 1,924,655 more rows
```

Create Subsets

`group_by(data, condition)` allows you to break down your data into specified groups based on variables:

```
group_by(babynames, sex, year)
## # A tibble: 1,924,665 x 5
## # Groups:   sex, year [276]
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1880 F     Mary        7065 0.0724
##  2  1880 F     Anna        2604 0.0267
##  3  1880 F     Emma        2003 0.0205
##  4  1880 F     Elizabeth   1939 0.0199
##  5  1880 F     Minnie      1746 0.0179
##  6  1880 F     Margaret    1578 0.0162
##  7  1880 F     Ida         1472 0.0151
##  8  1880 F     Alice       1414 0.0145
##  9  1880 F     Bertha      1320 0.0135
## 10  1880 F     Sarah       1288 0.0132
## # ... with 1,924,655 more rows
```

Remove Subsets

... and `ungroup()` lets you remove any subsets:

```
grouped <- group_by(babynames, sex, year)
ungroup(grouped)
## # A tibble: 1,924,665 x 5
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1880 F     Mary        7065 0.0724
##  2  1880 F     Anna        2604 0.0267
##  3  1880 F     Emma        2003 0.0205
##  4  1880 F     Elizabeth   1939 0.0199
##  5  1880 F     Minnie      1746 0.0179
##  6  1880 F     Margaret    1578 0.0162
##  7  1880 F     Ida         1472 0.0151
##  8  1880 F     Alice       1414 0.0145
##  9  1880 F     Bertha      1320 0.0135
## 10  1880 F     Sarah       1288 0.0132
## # ... with 1,924,655 more rows
```


THE SUPERPOWER OF
group_by()
Verbs will be automatically applied "by group"!

`group_by()` SUPERPOWER!

`group_by()` in combination with other `dplyr` verbs allows you to calculate summary statistics for specified groups:

```
names_group <- group_by(babynames, sex)
names_group
## # A tibble: 1,924,665 x 5
## # Groups:   sex [2]
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1880 F     Mary        7065 0.0724
##  2  1880 F     Anna        2604 0.0267
##  3  1880 F     Emma        2003 0.0205
##  4  1880 F     Elizabeth   1939 0.0199
##  5  1880 F     Minnie      1746 0.0179
##  6  1880 F     Margaret    1578 0.0162
##  7  1880 F     Ida         1472 0.0151
##  8  1880 F     Alice       1414 0.0145
##  9  1880 F     Bertha      1320 0.0135
## 10  1880 F     Sarah       1288 0.0132
## # ... with 1,924,655 more rows
```

`group_by()` SUPERPOWER!

`group_by()` in combination with other `dplyr` verbs allows you to calculate summary statistics for specified groups:

```
names_group <- group_by(babynames, sex)
summarize(names_group, sum = sum(n))
## # A tibble: 2 x 2
##   sex         sum
##   <chr>     <int>
## 1 F     172371079
## 2 M     175749438
```

`group_by()` SUPERPOWER!

`group_by()` in combination with other `dplyr` verbs allows you to calculate summary statistics for specified groups:

```
names_cent <- filter(babynames, year %in% c(1890, 1900))
names_cent <- group_by(names_cent, sex, year)
names_cent
## # A tibble: 6,425 x 5
## # Groups:   sex, year [4]
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1890 F     Mary       12078 0.0599
##  2  1890 F     Anna        5233 0.0259
##  3  1890 F     Elizabeth   3112 0.0154
##  4  1890 F     Margaret    3100 0.0154
##  5  1890 F     Emma        2980 0.0148
##  6  1890 F     Florence    2744 0.0136
##  7  1890 F     Ethel       2718 0.0135
##  8  1890 F     Minnie      2650 0.0131
##  9  1890 F     Clara       2496 0.0124
## 10  1890 F     Bertha      2388 0.0118
## # ... with 6,415 more rows
```

`group_by()` SUPERPOWER!

`group_by()` in combination with other `dplyr` verbs allows you to calculate summary statistics for specified groups:

```
names_cent <- filter(babynames, year %in% c(1890, 1900))
names_cent <- group_by(names_cent, sex, year)
names_cent <- mutate(names_cent, sum = sum(n))
names_cent <- group_by(names_cent, sex, year, name)
names_cent
## # A tibble: 6,425 x 6
## # Groups:   sex, year, name [6,425]
##     year sex   name           n   prop    sum
##    <dbl> <chr> <chr>      <int>  <dbl>  <int>
##  1  1890 F     Mary       12078 0.0599 190376
##  2  1890 F     Anna        5233 0.0259 190376
##  3  1890 F     Elizabeth   3112 0.0154 190376
##  4  1890 F     Margaret    3100 0.0154 190376
##  5  1890 F     Emma        2980 0.0148 190376
##  6  1890 F     Florence    2744 0.0136 190376
##  7  1890 F     Ethel       2718 0.0135 190376
##  8  1890 F     Minnie      2650 0.0131 190376
##  9  1890 F     Clara       2496 0.0124 190376
## 10  1890 F     Bertha      2388 0.0118 190376
## # ... with 6,415 more rows
```

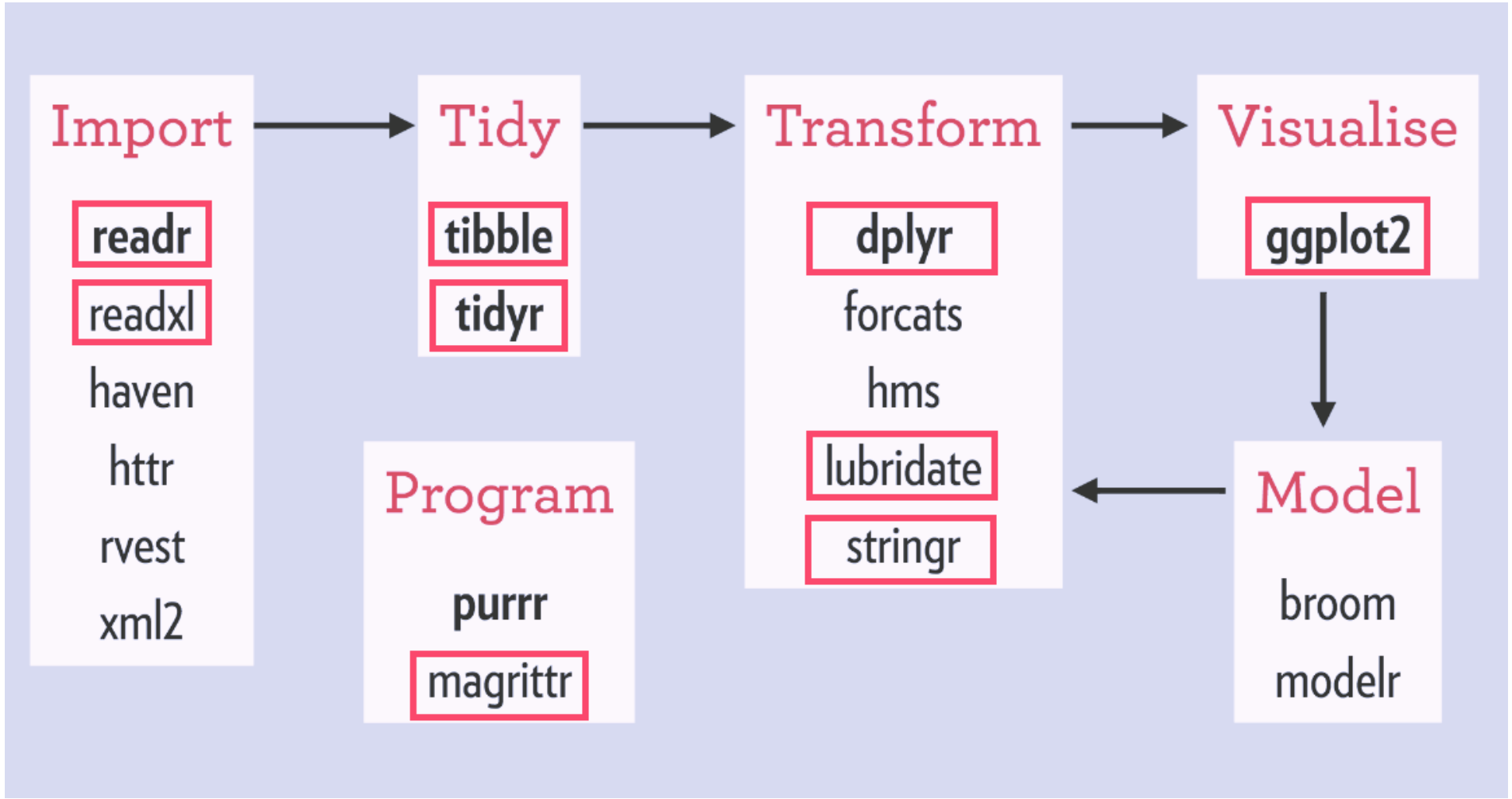
`group_by()` SUPERPOWER!

`group_by()` in combination with other `dplyr` verbs allows you to calculate summary statistics for specified groups:

```
names_cent <- filter(babynames, year %in% c(1890, 1900))
names_cent <- group_by(names_cent, sex, year)
names_cent <- mutate(names_cent, sum = sum(n))
names_cent <- group_by(names_cent, sex, year, name)
names_sum <- summarize(names_cent, prop = prop, prop_check = sum(n) / unique(sum))
arrange(names_sum, desc(sex), -prop_check)
## # A tibble: 6,425 x 5
## # Groups:   sex, year [4]
##    sex    year name      prop prop_check
##    <chr> <dbl> <chr>    <dbl>      <dbl>
##  1 M      1890 John    0.0710     0.0766
##  2 M      1890 William 0.0626     0.0675
##  3 M      1900 John    0.0606     0.0653
##  4 M      1900 William 0.0529     0.0570
##  5 M      1900 James   0.0447     0.0481
##  6 M      1890 James   0.0426     0.0459
##  7 M      1890 George  0.0372     0.0402
##  8 M      1890 Charles 0.0339     0.0366
##  9 M      1900 George  0.0333     0.0359
## 10 M      1890 Frank   0.0257     0.0277
```

The magrittr Package

The Typical Data Science Project Flow



Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse6.png

Readability of Code

Have a look again at this chunk of code:

```
names_cent <- filter(babynames, year %in% c(1890, 1900))
names_cent <- group_by(names_cent, sex, year)
names_cent <- mutate(names_cent, sum = sum(n))
names_cent <- group_by(names_cent, sex, year, name)
names_sum <- summarize(names_cent, prop = prop, prop_check = sum(n) / unique(sum))
names_1990_sum <- ungroup(names_sum)
names_sum <- arrange(names_sum, desc(sex), -prop_check)
```

There is a lot redundancy and also no need to save each step as an object.

Readability of Code

One could rewrite the code like this:

```
names_cent <-
  group_by(mutate(group_by(filter(babynames, year %in% c(1890, 1900)), sex, year), sum

names_sum <-
  ungroup(arrange(summarize(names_cent, prop = prop, prop_check = sum(n) / unique(sum)

names_sum
## # A tibble: 6,425 x 5
##    sex    year name      prop prop_check
##    <chr> <dbl> <chr>    <dbl>      <dbl>
##  1 M      1890 John    0.0710     0.0766
##  2 M      1890 William 0.0626     0.0675
##  3 M      1900 John    0.0606     0.0653
##  4 M      1900 William 0.0529     0.0570
##  5 M      1900 James   0.0447     0.0481
##  6 M      1890 James   0.0426     0.0459
##  7 M      1890 George  0.0372     0.0402
##  8 M      1890 Charles 0.0339     0.0366
##  9 M      1900 George  0.0333     0.0359
## 10 M      1890 Frank   0.0257     0.0277
## # ... with 6,415 more rows
```

Readability of Code

One could rewrite the code like this (Part 1)

```
names_cent <-
  group_by(
    mutate(
      group_by(
        filter(
          babynames,
          year %in% c(1890, 1900)
        ),
        sex, year
        ),
      sum = sum(n)
    ),
    sex, year, name
  )
```

Readability of Code

One could rewrite the code like this (Part 2)

```
names_sum <-
  ungroup(
    arrange(
      summarize(
        names_cent,
        prop = prop,
        prop_check = sum(n) / unique(sum)
      ),
      desc(sex), -prop_check
    )
  )
```

```
physalia(
  bvg(
    breakfast(
      shower(
        wake_up(
          Cédric,
          7.0
        ),
        temp == 38
      ),
      c("cornflakes", "tea"),
    ),
    price = "EUR2.90",
    delay = 10
  ),
  course = "Data Visualization in R with ggplot2"
)
```

```
Cédric %>%
  wake_up(7.0) %>%
  shower(temp == 38) %>%
  breakfast(c("cereals", "tea")) %>%
  bvg(
    price = "EUR2.90",
    delay = 10
  ) %>%
  physalia(
    course = "Data Visualization in R with ggplot2"
  )
```

```
Cédric %>%
  wake_up(., 7.0) %>%
  shower(., temp == 38) %>%
  breakfast(., c("cereals", "tea")) %>%
  bvg(
    .,
    price = "EUR2.90",
    delay = 10
  ) %>%
  physalia(
    .,
    course = "Data Visualization in R with ggplot2"
  )
```

Readability of Code: %>% (The Pipe)

With the pipe one can rewrite the code like this:

```
names_cent <-
  babynames %>%
  filter(year %in% c(1890, 1900)) %>%
  group_by(sex, year) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, year, name)

names_sum <-
  names_cent %>%
  summarize(
    prop = prop,
    prop_check = sum(n) / unique(sum)
  ) %>%
  arrange(desc(sex), -prop_check) %>%
  ungroup()
```

Readability of Code: %>% (The Pipe)

... or like this by omitting the intermediate assignment:

```
names_sum <-
  babynames %>%
  filter(year %in% c(1890, 1900)) %>%
  group_by(sex, year) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, year, name) %>%
  summarize(
    prop = prop,
    prop_check = sum(n) / unique(sum)
  ) %>%
  arrange(desc(sex), -prop_check) %>%
  ungroup()
```

Readability of Code: %>% (The Pipe)

(Just to show it worked.)

```
names_sum
## # A tibble: 6,425 x 5
##    sex    year name      prop prop_check
##    <chr> <dbl> <chr>    <dbl>      <dbl>
##  1 M      1890 John    0.0710     0.0766
##  2 M      1890 William 0.0626     0.0675
##  3 M      1900 John    0.0606     0.0653
##  4 M      1900 William 0.0529     0.0570
##  5 M      1900 James   0.0447     0.0481
##  6 M      1890 James   0.0426     0.0459
##  7 M      1890 George  0.0372     0.0402
##  8 M      1890 Charles 0.0339     0.0366
##  9 M      1900 George  0.0333     0.0359
## 10 M      1890 Frank   0.0257     0.0277
## # ... with 6,415 more rows
```

Readability of Code: %>% (The Pipe)

This way, you can also easily inactivate verbs or single conditions:

```
names_sum <-
  babynames %>%
  filter(year %in% c(1890, 1900)) %>%
  group_by(sex, year) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, year, name) %>%
  summarize(
    #prop = prop,
    prop_check = sum(n) / unique(sum)
  ) %>%
  arrange(desc(sex), -prop_check) %>%
  ungroup()
```

Readability of Code: %>% (The Pipe)

You can easily inactivate verbs or single conditions:

```
names_sum <-
  babynames %>%
  filter(year %in% c(1890, 1900)) %>%
  group_by(sex, year) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, year, name) %>%
  summarize(
    #prop = prop,
    prop_check = sum(n) / unique(sum)
  ) %>%
  #arrange(desc(sex), -prop_check) %>%
  ungroup()
```

Readability of Code: %>% (The Pipe)

You can easily inactivate verbs or single conditions:

```
names_sum <-
  babynames %>%
  filter(year %in% c(1890, 1900)) %>%
  group_by(sex, year) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, year, name) %>%
  summarize(
    #prop = prop,
    prop_check = sum(n) / unique(sum)
  ) %>%  ## expecting another verb!
  #arrange(desc(sex), -prop_check) %>%
  #ungroup()
```

Readability of Code: %>% (The Pipe)

You can easily inactivate verbs or single conditions:

```
names_sum <-
  babynames %>%
  filter(year %in% c(1890, 1900)) %>%
  group_by(sex, year) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, year, name) %>%
  summarize(
    #prop = prop,
    prop_check = sum(n) / unique(sum)
  ) #%>% #<< ## comment as well
  #arrange(desc(sex), -prop_check) %>%
  #ungroup()
```

Readability of Code: %>% (The Pipe)

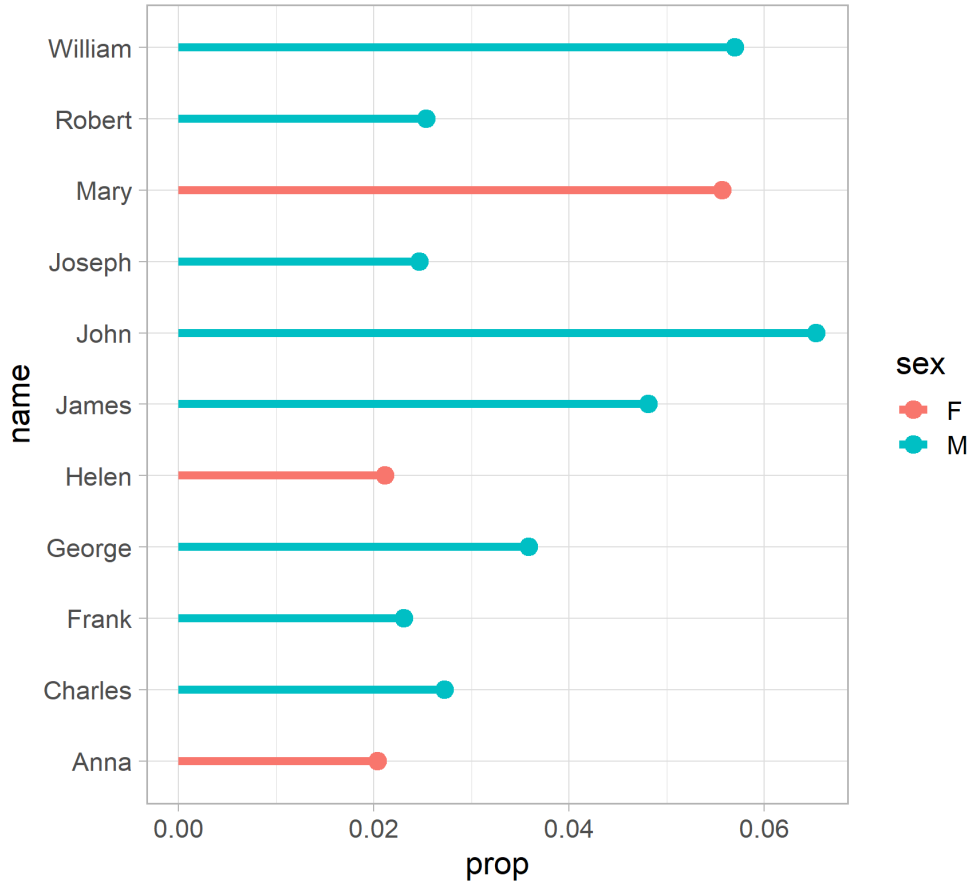
You can easily inactivate verbs or single conditions:

```
names_sum <-
  babynames %>%
  filter(year %in% c(1890, 1900)) %>%
  group_by(sex, year) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, year, name) %>%
  summarize(
    #prop = prop,
    prop_check = sum(n) / unique(sum)
  ) %>%
  #arrange(desc(sex), -prop_check) %>%
  #ungroup() %>%
  {} # ... or add this  #
```

Readability of Code: %>% (The Pipe)

You can also pipe into a `ggplot()` call (but don't forget to use `+` afterwards, not `%>%`):

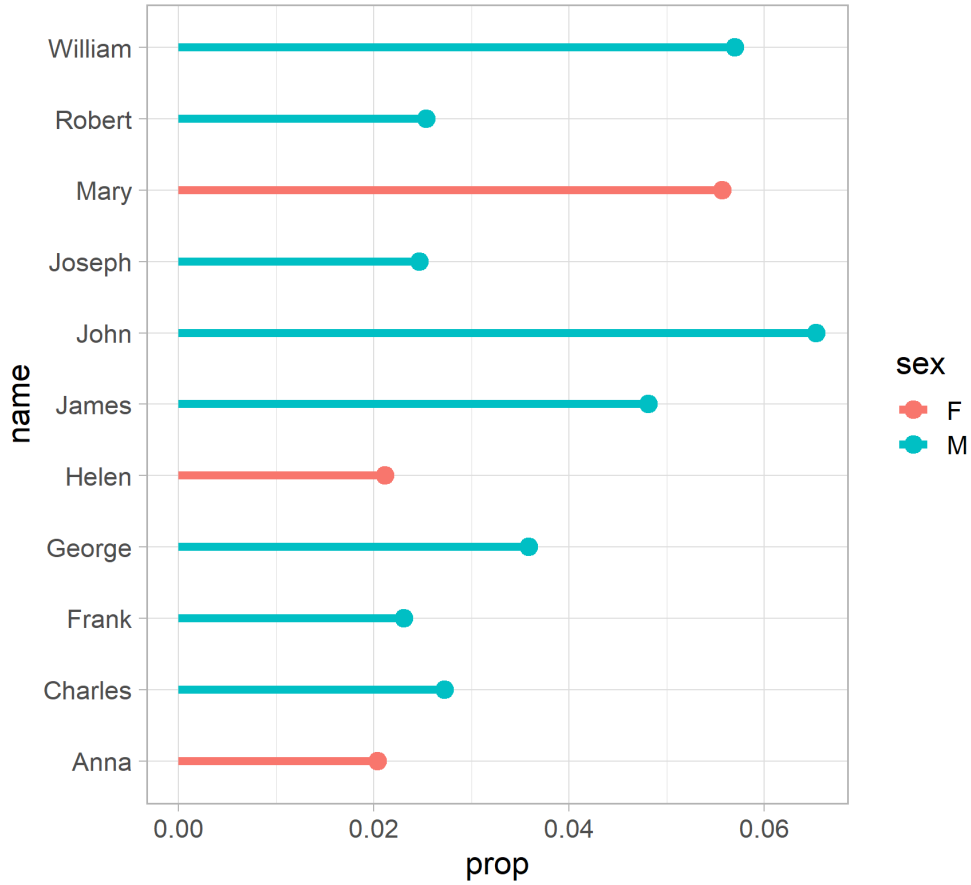
```
babynames %>%
  filter(year == 1900) %>%
  group_by(sex) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, name) %>%
  summarize(
    prop = sum(n) / unique(sum)
  ) %>%
  filter(prop >= .02) %>%
  ggplot(aes(prop, name,
             color = sex)) +
    geom_segment(
      aes(xend = 0, yend = name),
      size = 2
    ) +
    geom_point(size = 4)
```



Readability of Code: %>% (The Pipe)

But of course you can also assign it first and call it with `ggplot()` afterwards:

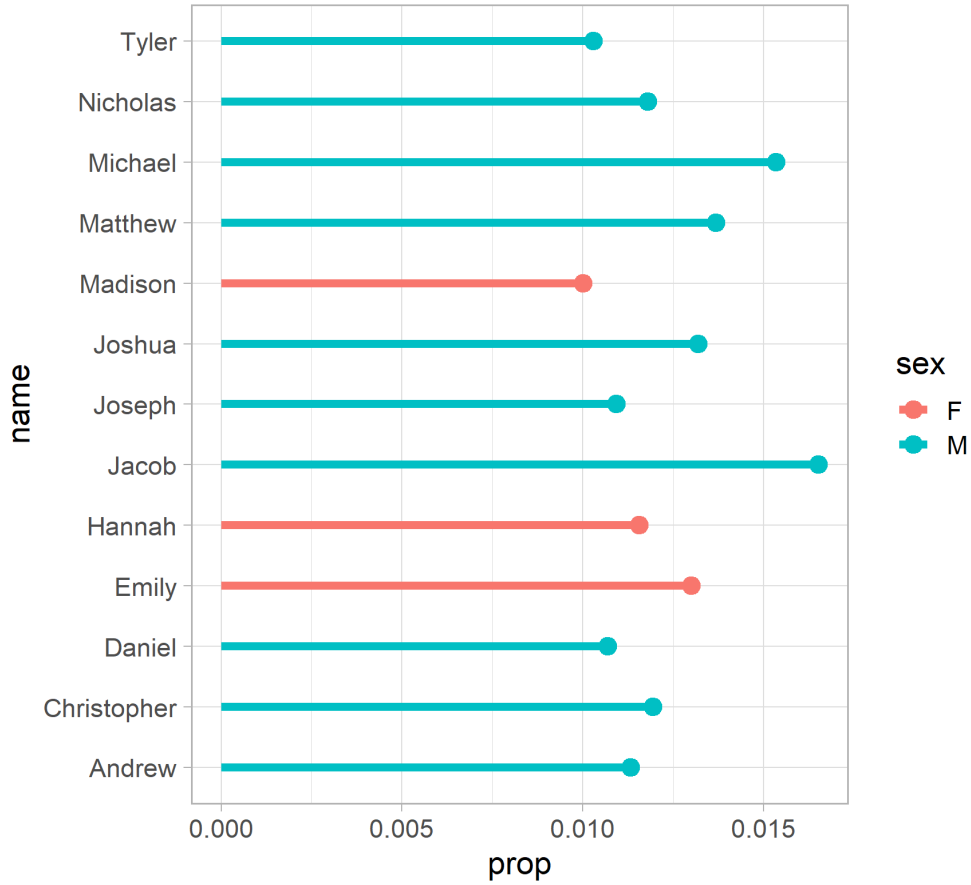
```
babynames_prep <- babynames %>%
  filter(year == 1900) %>%
  group_by(sex) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, name) %>%
  summarize(
    prop = sum(n) / unique(sum)
  ) %>%
  filter(prop >= .02)

ggplot(
    babynames_prep,
    aes(prop, name,
        color = sex)) +
    geom_segment(
      aes(xend = 0, yend = name),
      size = 2
    ) +
    geom_point(size = 4)
```



Readability of Code: %>% (The Pipe)

One can also use such a chain *inside* the `ggplot()` call:

```
ggplot(
    babynames %>%
        filter(year == 2000,
               prop > .01),
    aes(prop, name,
        color = sex)) +
    geom_segment(
      aes(xend = 0, yend = name),
      size = 2
    ) +
    geom_point(
      size = 4
    )
```



Your Turn!

- Inspect the `flights` data set from the `nycflights13` package.
- Count the number of flights in June and July 2013.
- Find out how many flights did catch up their departure delay.
- Create a data frame that contains the average delays (departure & arrival) per origin and month, ordered by maximum arrival delay. (Bonus: Delays with 1 digit only.)
- Explore the relationship between the average distance and average delay per season and destination.

Flights in June and July 2013

```
library(nycflights13)
filter(flights, month == 6 | month == 7)
## # A tibble: 57,668 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     6     1        2           2359         3      341            350
##  2  2013     6     1      451            500        -9      624            640
##  3  2013     6     1      506            515        -9      715            800
##  4  2013     6     1      534            545       -11      800            829
##  5  2013     6     1      538            545        -7      925            922
##  6  2013     6     1      539            540        -1      832            840
##  7  2013     6     1      546            600       -14      850            910
##  8  2013     6     1      551            600        -9      828            850
##  9  2013     6     1      552            600        -8      647            655
## 10  2013     6     1      553            600        -7      700            711
## # ... with 57,658 more rows, and 11 more variables: 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>
```

Flights in June and July 2013

```
filter(flights, month %in% c(6, 7))
## # A tibble: 57,668 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     6     1        2           2359         3      341            350
##  2  2013     6     1      451            500        -9      624            640
##  3  2013     6     1      506            515        -9      715            800
##  4  2013     6     1      534            545       -11      800            829
##  5  2013     6     1      538            545        -7      925            922
##  6  2013     6     1      539            540        -1      832            840
##  7  2013     6     1      546            600       -14      850            910
##  8  2013     6     1      551            600        -9      828            850
##  9  2013     6     1      552            600        -8      647            655
## 10  2013     6     1      553            600        -7      700            711
## # ... with 57,658 more rows, and 11 more variables: 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>
```

Catch-Up Delay

```
filter(flights, dep_delay > 0, arr_delay <= 0)
## # A tibble: 35,442 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      601            600         1      844            850
##  2  2013     1     1      644            636         8      931            940
##  3  2013     1     1      646            645         1      910            916
##  4  2013     1     1      646            645         1     1023           1030
##  5  2013     1     1      701            700         1     1123           1154
##  6  2013     1     1      752            750         2     1025           1029
##  7  2013     1     1      803            800         3     1132           1144
##  8  2013     1     1      826            817         9     1145           1158
##  9  2013     1     1      846            845         1     1138           1205
## 10  2013     1     1      856            855         1     1140           1203
## # ... with 35,432 more rows, and 11 more variables: 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>
```

Catch-Up Delay (Version 2)

```
filter(flights, dep_delay > 0, arr_delay < dep_delay)
## # A tibble: 83,728 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      601            600         1      844            850
##  2  2013     1     1      623            610        13      920            915
##  3  2013     1     1      632            608        24      740            728
##  4  2013     1     1      644            636         8      931            940
##  5  2013     1     1      646            645         1      910            916
##  6  2013     1     1      646            645         1     1023           1030
##  7  2013     1     1      701            700         1     1123           1154
##  8  2013     1     1      732            729         3     1041           1039
##  9  2013     1     1      732            645        47     1011            941
## 10  2013     1     1      743            730        13     1107           1100
## # ... with 83,718 more rows, and 11 more variables: 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>
```

Average Delays per Origin and Month

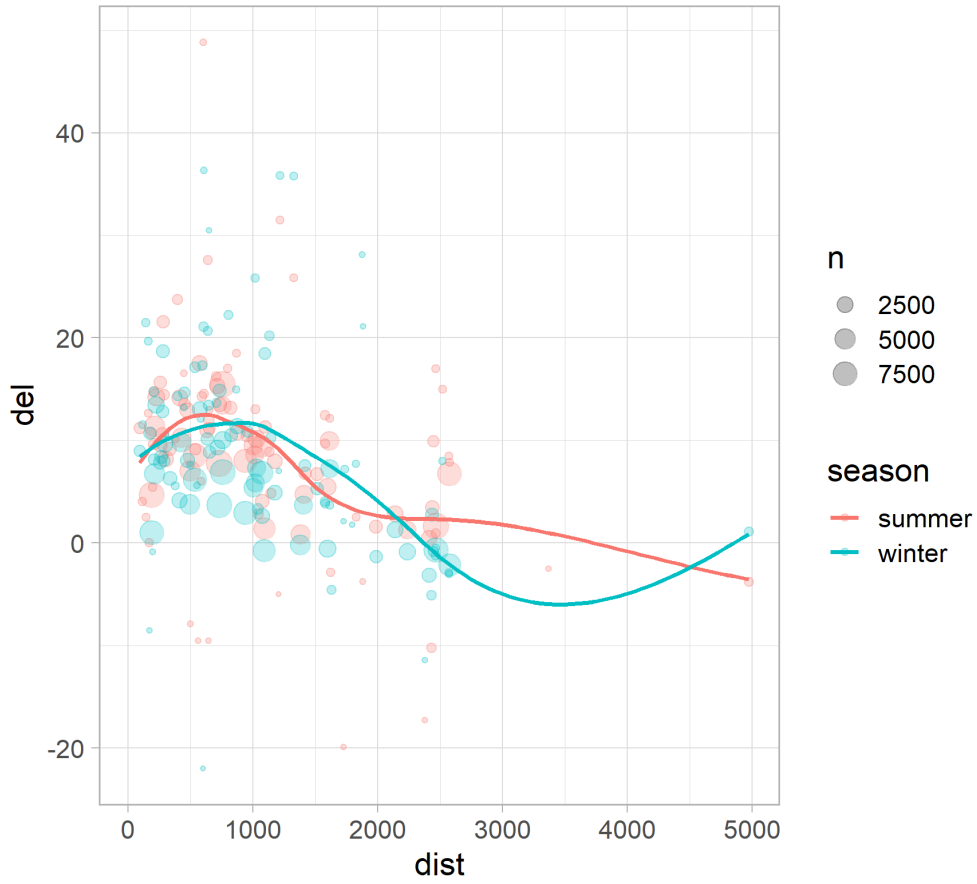
```
flights %>%
  group_by(month, origin) %>%
  summarize(
    dep_delay_avg = round(mean(dep_delay, na.rm = TRUE), 1),
    arr_delay_avg = round(mean(arr_delay, na.rm = TRUE), 1)
  ) %>%
  arrange(-arr_delay_avg)
## # A tibble: 36 x 4
## # Groups:   month [12]
##    month origin dep_delay_avg arr_delay_avg
##    <int> <chr>           <dbl>         <dbl>
##  1     7 JFK              23.8          20.2
##  2    12 EWR              21            19.6
##  3     6 JFK              20.5          17.6
##  4     6 EWR              22.5          16.9
##  5     7 EWR              22            15.5
##  6     6 LGA              19.3          14.8
##  7     7 LGA              19            14.2
##  8     4 EWR              17.4          14.1
##  9     1 EWR              14.9          12.8
## 10    12 JFK              14.8          12.7
## # ... with 26 more rows
```

Average Delays per Origin and Month

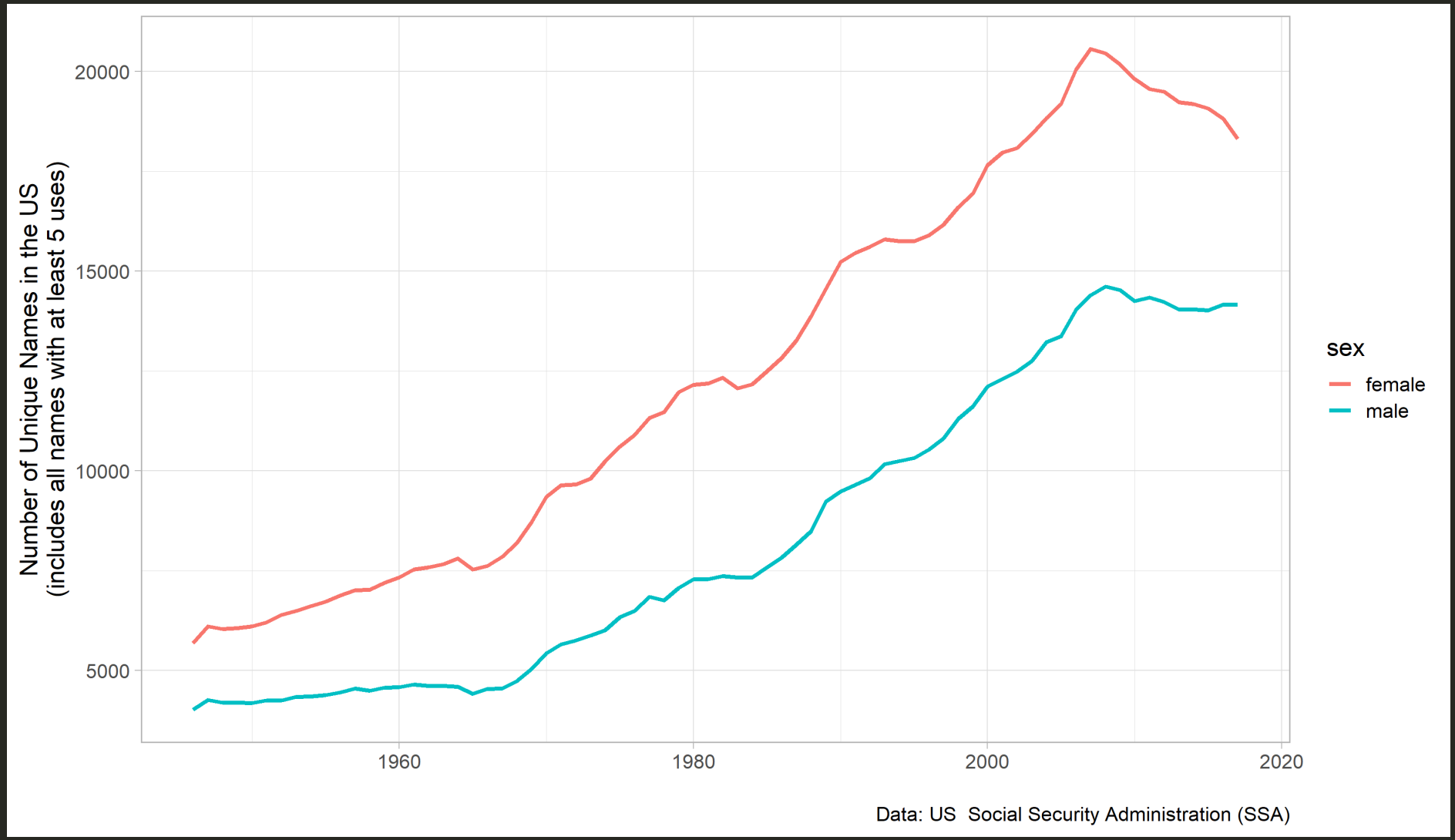
```
flights %>%
  select(month, origin, ends_with("delay")) %>%
  group_by(month, origin) %>%
  summarize_all(list(~round(mean(., na.rm = TRUE), 1))) %>%
  arrange(-arr_delay)
## # A tibble: 36 x 4
## # Groups:   month [12]
##    month origin dep_delay arr_delay
##    <int> <chr>      <dbl>     <dbl>
##  1     7 JFK         23.8      20.2
##  2    12 EWR         21        19.6
##  3     6 JFK         20.5      17.6
##  4     6 EWR         22.5      16.9
##  5     7 EWR         22        15.5
##  6     6 LGA         19.3      14.8
##  7     7 LGA         19        14.2
##  8     4 EWR         17.4      14.1
##  9     1 EWR         14.9      12.8
## 10    12 JFK         14.8      12.7
## # ... with 26 more rows
```

Distance versus Delay

```
flights %>%
  mutate(
    season = if_else(month %in% 4:9,
                     "summer",
                     "winter")
  ) %>%
  group_by(season, dest) %>%
  summarize(
    del = mean(arr_delay, na.rm = TRUE),
    dist = mean(distance, na.rm = TRUE),
    n = n()
  ) %>%
  ggplot(aes(dist, del,
             color = season)) +
    geom_point(
      aes(size = n),
      alpha = .25
    ) +
    geom_smooth(se = FALSE)
```



Your Turn: Create this Line Plot!

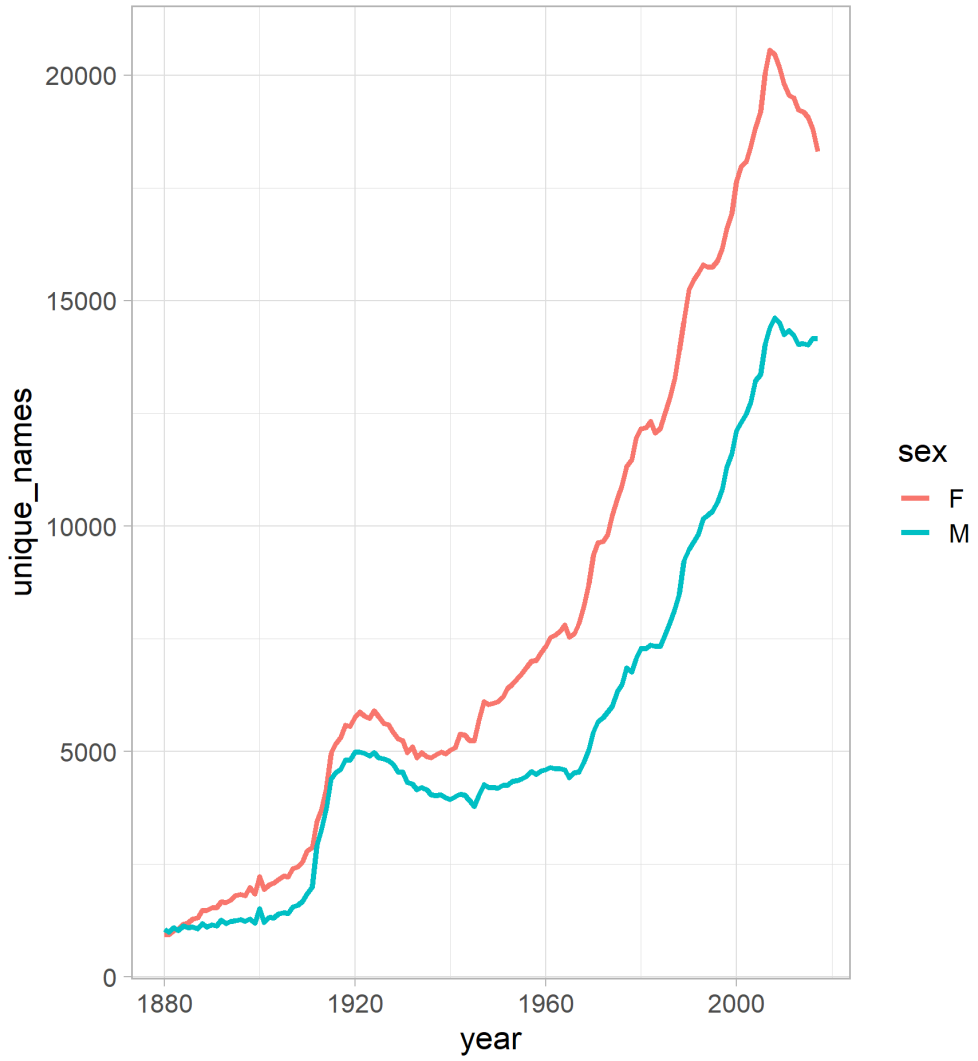


Your Turn: Post-WWII Line Plot

```
babynames %>%
  group_by(year, sex) %>%
  summarize(unique_names = n_distinct(name))
## # A tibble: 276 x 3
## # Groups:   year [138]
##     year sex   unique_names
##    <dbl> <chr>        <int>
##  1  1880 F              942
##  2  1880 M             1058
##  3  1881 F              938
##  4  1881 M              997
##  5  1882 F             1028
##  6  1882 M             1099
##  7  1883 F             1054
##  8  1883 M             1030
##  9  1884 F             1172
## 10  1884 M             1125
## # ... with 266 more rows
```

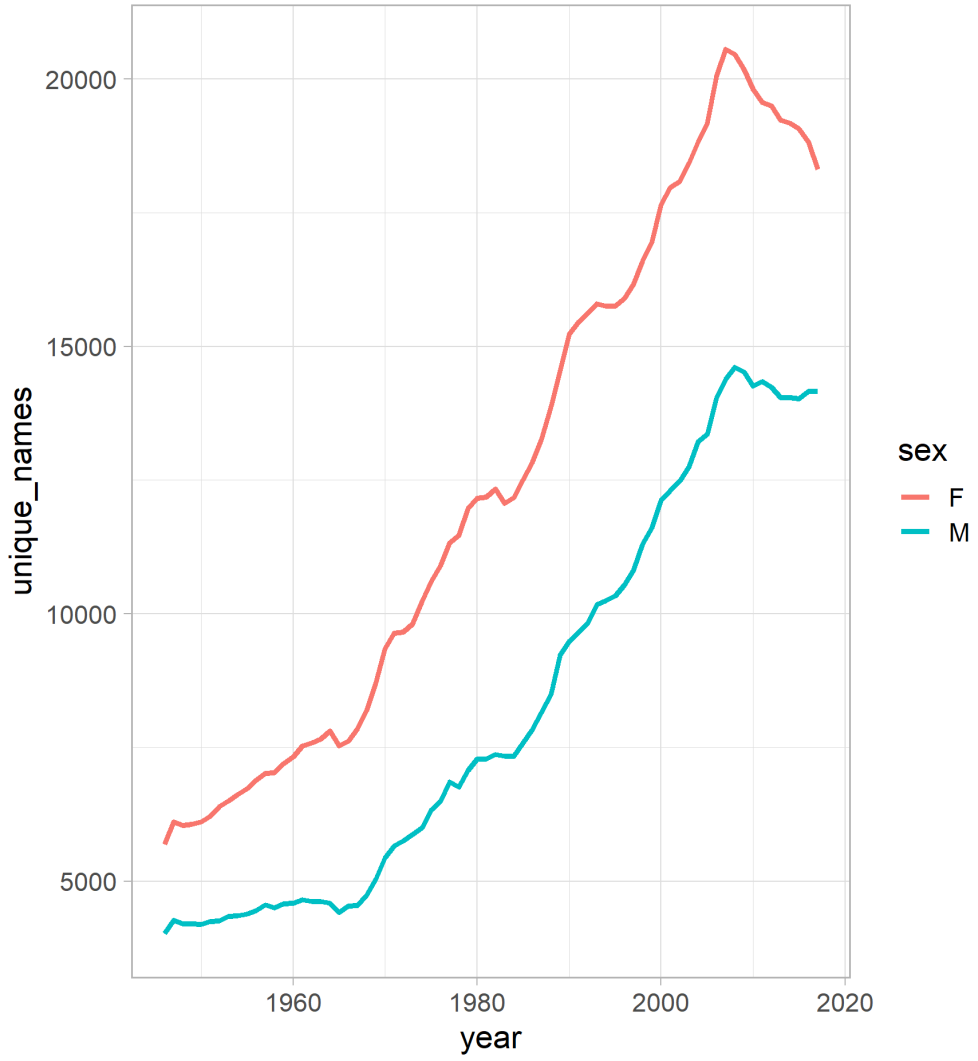
Your Turn: Post-WWII Line Plot

```
babynames %>%
  group_by(year, sex) %>%
  summarize(
    unique_names = n_distinct(name)
  ) %>%
  ggplot(aes(year, unique_names,
             color = sex)) +
    geom_line(size = 1.3)
```



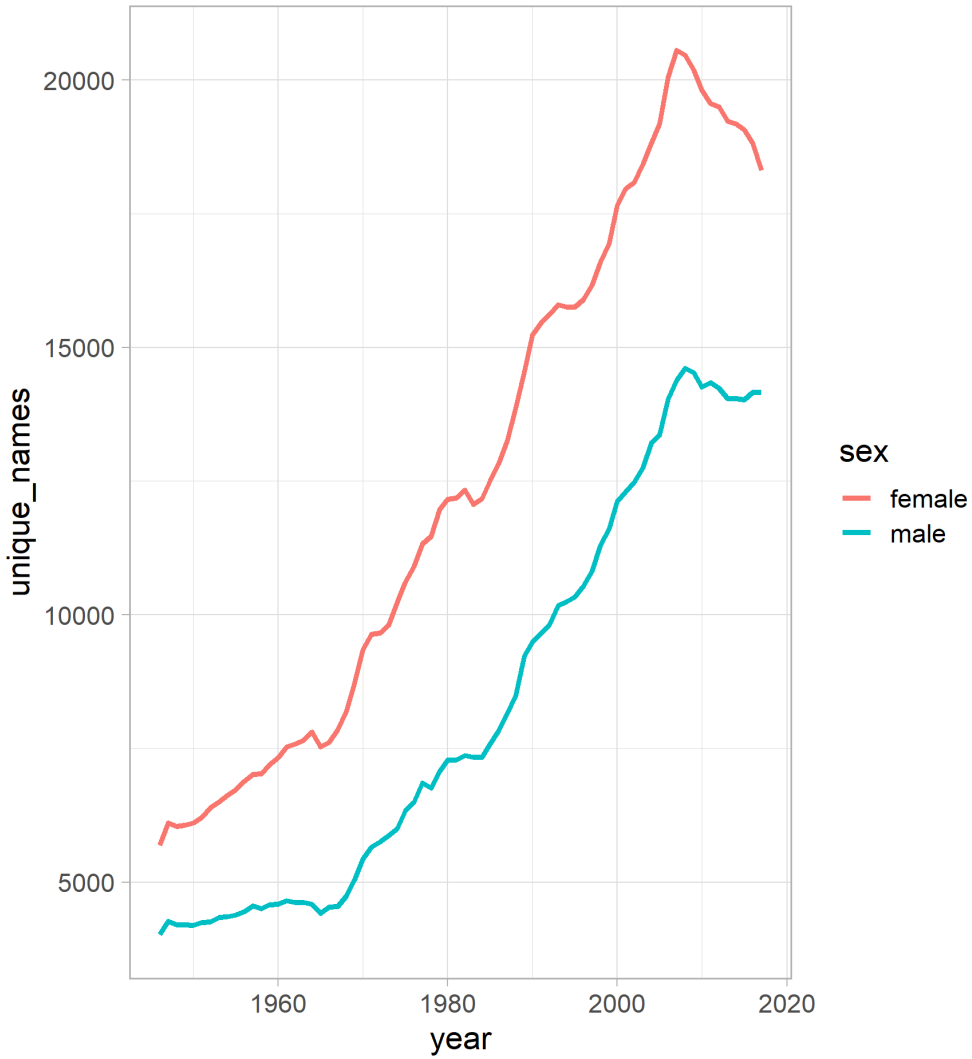
Your Turn: Post-WWII Line Plot

```
babynames %>%
  filter(year > 1945) %>%
  group_by(year, sex) %>%
  summarize(
    unique_names = n_distinct(name)
  ) %>%
  ggplot(aes(year, unique_names,
             color = sex)) +
    geom_line(size = 1.3)
```



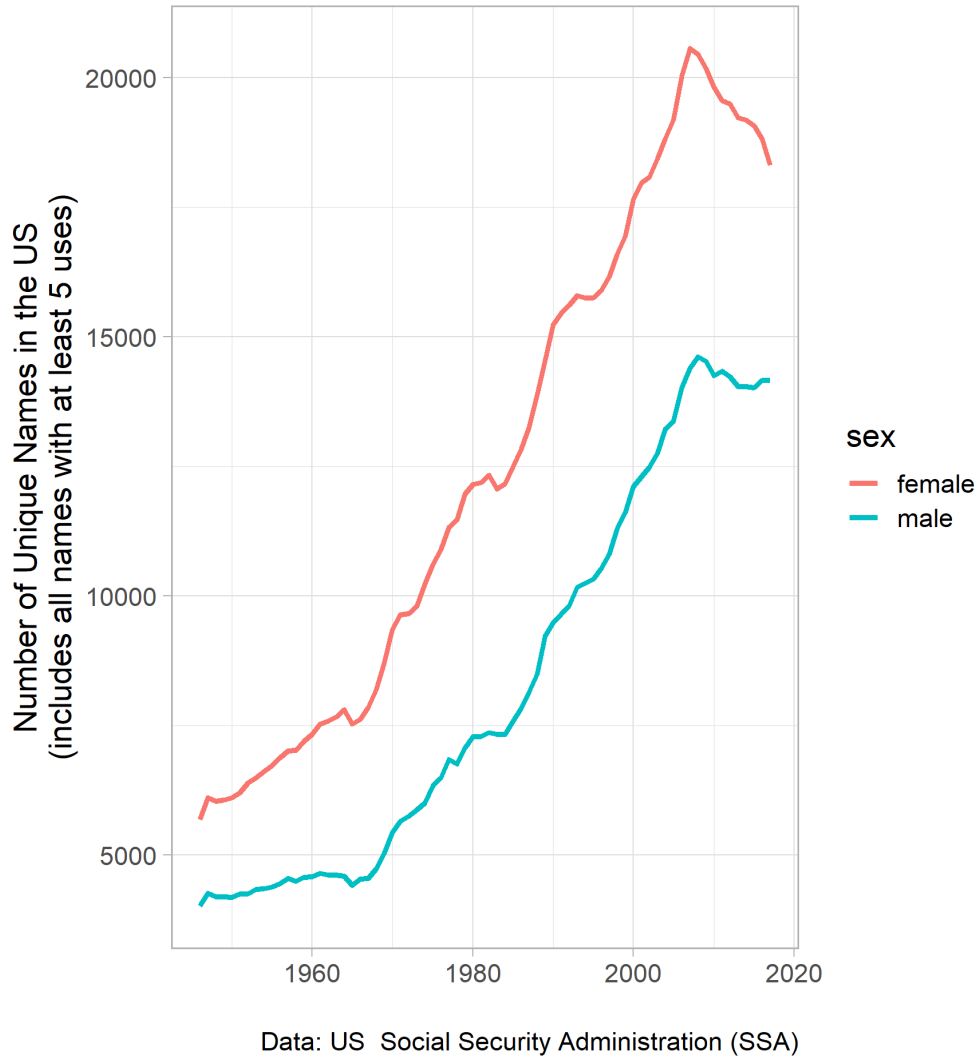
Your Turn: Post-WWII Line Plot

```
babynames %>%
  filter(year > 1945) %>%
  mutate(
    sex = if_else(sex == "M",
                  "male",
                  "female")
  ) %>%
  group_by(year, sex) %>%
  summarize(
    unique_names = n_distinct(name)
  ) %>%
  ggplot(aes(year, unique_names,
             color = sex)) +
    geom_line(size = 1.3)
```



Your Turn: Post-WWII Line Plot

```
babynames %>%
  filter(year > 1945) %>%
  mutate(
    sex = if_else(sex == "M",
                  "male",
                  "female")
  ) %>%
  group_by(year, sex) %>%
  summarize(
    unique_names = n_distinct(name)
  ) %>%
  ggplot(aes(year, unique_names,
             color = sex)) +
    geom_line(size = 1.3) +
    labs(
      x = NULL,
      y = "Number of Unique Names in the
      caption = "\nData: US  Social Secur
    )
```



Other Helpful `dplyr` and `tidyverse` Functions

Function	Explanation
`slice()`	Extract rows by ordinal position
`distinct()` and `n_distinct()`	Find and count unique values
`sample_n()` and `sample_frac()`	Select rows randomly
`top_n()` and `top_frac()`	Pick top or bottom values by variable
`count()` and `top_frac()`	Count number of observations
`complete()`	Create all possible combinations of two variables
`*_join()`	Merge two tables
`pivot_longer()` and `pivot_wider()`	Turn wide data into long and vice versa
`fct_*()`	Change factor levels dynamically
`str_*()`	Manipulate character strings
`glue()`	Combine strings

Extract Rows by Ordinal Position

`slice()` allows you to filter rows based on their row value:

```
slice(mpg, 100:120)
## # A tibble: 21 x 11
##    manufacturer model  displ  year   cyl trans   drv     cty   hwy fl    class
##    <chr>        <chr>  <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
##  1 honda        civic    1.6  1999     4 manual~ f        28    33 r     subcom~
##  2 honda        civic    1.6  1999     4 auto(l~ f        24    32 r     subcom~
##  3 honda        civic    1.6  1999     4 manual~ f        25    32 r     subcom~
##  4 honda        civic    1.6  1999     4 manual~ f        23    29 p     subcom~
##  5 honda        civic    1.6  1999     4 auto(l~ f        24    32 r     subcom~
##  6 honda        civic    1.8  2008     4 manual~ f        26    34 r     subcom~
##  7 honda        civic    1.8  2008     4 auto(l~ f        25    36 r     subcom~
##  8 honda        civic    1.8  2008     4 auto(l~ f        24    36 c     subcom~
##  9 honda        civic    2    2008     4 manual~ f        21    29 p     subcom~
## 10 hyundai      sonata   2.4  1999     4 auto(l~ f        18    26 r     midsize
## # ... with 11 more rows
```

Extract Rows by Ordinal Position

`slice()` allows you to filter rows based on their row value - works also per group:

```
mpg %>% group_by(manufacturer) %>% slice(1)
## # A tibble: 15 x 11
## # Groups:   manufacturer [15]
##    manufacturer model    displ  year   cyl trans  drv     cty   hwy fl    class
##    <chr>        <chr>    <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
##  1 audi         a4         1.8  1999     4 auto(~ f        18    29 p     compa~
##  2 chevrolet    c1500 s~   5.3  2008     8 auto(~ r        14    20 r     suv
##  3 dodge        caravan~   2.4  1999     4 auto(~ f        18    24 r     miniv~
##  4 ford         expedit~   4.6  1999     8 auto(~ r        11    17 r     suv
##  5 honda        civic      1.6  1999     4 manua~ f        28    33 r     subco~
##  6 hyundai      sonata     2.4  1999     4 auto(~ f        18    26 r     midsi~
##  7 jeep         grand c~   3    2008     6 auto(~ 4        17    22 d     suv
##  8 land rover   range r~   4    1999     8 auto(~ 4        11    15 p     suv
##  9 lincoln      navigat~   5.4  1999     8 auto(~ r        11    17 r     suv
## 10 mercury      mountai~   4    1999     6 auto(~ 4        14    17 r     suv
## 11 nissan       altima     2.4  1999     4 manua~ f        21    29 r     compa~
## 12 pontiac      grand p~   3.1  1999     6 auto(~ f        18    26 r     midsi~
## 13 subaru       foreste~   2.5  1999     4 manua~ 4        18    25 r     suv
## 14 toyota       4runner~   2.7  1999     4 manua~ 4        15    20 r     suv
## 15 volkswagen   gti        2    1999     4 manua~ f        21    29 r     compa~
```

Find Unique Values

`distinct()` allows you to retain only unique values (as `unique()` does):

```
distinct(mpg, manufacturer)
## # A tibble: 15 x 1
##    manufacturer
##    <chr>
##  1 audi
##  2 chevrolet
##  3 dodge
##  4 ford
##  5 honda
##  6 hyundai
##  7 jeep
##  8 land rover
##  9 lincoln
## 10 mercury
## 11 nissan
## 12 pontiac
## 13 subaru
## 14 toyota
## 15 volkswagen
```

Find Unique Values

`distinct()` allows you to retain only unique values (as `unique()` does):

```
distinct(mpg, hwy)
## # A tibble: 27 x 1
##      hwy
##    <int>
##  1    29
##  2    31
##  3    30
##  4    26
##  5    27
##  6    25
##  7    28
##  8    24
##  9    23
## 10    20
## # ... with 17 more rows
```

Count Unique Values

`n_distinct()` allows you to calculate the number of unique values (as `length(unique())` does):

```
n_distinct(mpg$model) ## needs a vector :(
## [1] 38
```

Find Unique Values

`n_distinct()` is useful in combination with `group_by()` and `mutate|summarize()`:

```
mpg %>% group_by(manufacturer) %>% summarize(n_model = n_distinct(model))
## # A tibble: 15 x 2
##    manufacturer n_model
##    <chr>          <int>
##  1 audi               3
##  2 chevrolet          4
##  3 dodge              4
##  4 ford               4
##  5 honda              1
##  6 hyundai            2
##  7 jeep               1
##  8 land rover         1
##  9 lincoln            1
## 10 mercury            1
## 11 nissan             3
## 12 pontiac            1
## 13 subaru             2
## 14 toyota             6
## 15 volkswagen         4
```

Select Rows Randomly

`sample_n()` allows you to create random subsets based on a number of rows:

```
sample_n(mpg, 5)
## # A tibble: 5 x 11
##   manufacturer model     displ  year   cyl trans   drv     cty   hwy fl    class
##   <chr>        <chr>     <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
## 1 chevrolet    k1500 ta~   6.5  1999     8 auto(l~ 4        14    17 d     suv
## 2 nissan       pathfind~   3.3  1999     6 manual~ 4        15    17 r     suv
## 3 ford         f150 pic~   4.6  1999     8 auto(l~ 4        13    16 r     pick~
## 4 toyota       camry       3    1999     6 auto(l~ f        18    26 r     mids~
## 5 subaru       forester~   2.5  2008     4 manual~ 4        20    27 r     suv
```

Select Rows Randomly

`sample_n()` allows you to create random subsets based on a number of rows - also works with groups:

```
mpg %>% group_by(year, manufacturer) %>% sample_n(1)
## # A tibble: 30 x 11
## # Groups:   year, manufacturer [30]
##    manufacturer model    displ  year   cyl trans  drv     cty   hwy fl    class
##    <chr>        <chr>    <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
##  1 audi         a4 quat~   2.8  1999     6 auto(~ 4        15    25 p     compa~
##  2 chevrolet    k1500 t~   6.5  1999     8 auto(~ 4        14    17 d     suv
##  3 dodge        dakota ~   5.2  1999     8 auto(~ 4        11    15 r     pickup
##  4 ford         f150 pi~   5.4  1999     8 auto(~ 4        11    15 r     pickup
##  5 honda        civic      1.6  1999     4 auto(~ f        24    32 r     subco~
##  6 hyundai      sonata     2.4  1999     4 manua~ f        18    27 r     midsi~
##  7 jeep         grand c~   4.7  1999     8 auto(~ 4        14    17 r     suv
##  8 land rover   range r~   4    1999     8 auto(~ 4        11    15 p     suv
##  9 lincoln      navigat~   5.4  1999     8 auto(~ r        11    16 p     suv
## 10 mercury      mountai~   5    1999     8 auto(~ 4        13    17 r     suv
## # ... with 20 more rows
```

Select Rows Randomly

`sample_frac()` allows you to create random subsets based on a fraction:

```
sample_frac(mpg, .05)
## # A tibble: 12 x 11
##    manufacturer model    displ  year   cyl trans  drv     cty   hwy fl    class
##    <chr>        <chr>    <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
##  1 ford         expedit~   5.4  1999     8 auto(~ r        11    17 r     suv
##  2 jeep         grand c~   6.1  2008     8 auto(~ 4        11    14 p     suv
##  3 audi         a6 quat~   4.2  2008     8 auto(~ 4        16    23 p     midsi~
##  4 audi         a6 quat~   3.1  2008     6 auto(~ 4        17    25 p     midsi~
##  5 toyota       toyota ~   4    2008     6 auto(~ 4        16    20 r     pickup
##  6 chevrolet    c1500 s~   6    2008     8 auto(~ r        12    17 r     suv
##  7 toyota       camry      2.4  2008     4 auto(~ f        21    31 r     midsi~
##  8 dodge        durango~   3.9  1999     6 auto(~ 4        13    17 r     suv
##  9 toyota       4runner~   2.7  1999     4 auto(~ 4        16    20 r     suv
## 10 ford         mustang    4    2008     6 manua~ r        17    26 r     subco~
## 11 toyota       toyota ~   4    2008     6 manua~ 4        15    18 r     pickup
## 12 nissan       pathfin~   5.6  2008     8 auto(~ 4        12    18 p     suv
```

Pick Top or Bottom Values

`top_n()` allows you to filter the **top** or bottom values, ranked by a variable:

```
top_n(mpg, 5, hwy)
## # A tibble: 6 x 11
##   manufacturer model   displ  year   cyl trans   drv     cty   hwy fl    class
##   <chr>        <chr>   <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
## 1 honda        civic     1.8  2008     4 auto(l~ f        25    36 r     subcom~
## 2 honda        civic     1.8  2008     4 auto(l~ f        24    36 c     subcom~
## 3 toyota       corolla   1.8  2008     4 manual~ f        28    37 r     compact
## 4 volkswagen   jetta     1.9  1999     4 manual~ f        33    44 d     compact
## 5 volkswagen   new be~   1.9  1999     4 manual~ f        35    44 d     subcom~
## 6 volkswagen   new be~   1.9  1999     4 auto(l~ f        29    41 d     subcom~
```

Pick Top or Bottom Values

`top_n()` allows you to filter the top or **bottom** values, ranked by a variable:

```
top_n(mpg, 5, -hwy)
## # A tibble: 5 x 11
##   manufacturer model      displ  year   cyl trans  drv     cty   hwy fl    class
##   <chr>        <chr>      <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
## 1 dodge        dakota pi~   4.7  2008     8 auto(~ 4         9    12 e     pick~
## 2 dodge        durango 4~   4.7  2008     8 auto(~ 4         9    12 e     suv
## 3 dodge        ram 1500 ~   4.7  2008     8 auto(~ 4         9    12 e     pick~
## 4 dodge        ram 1500 ~   4.7  2008     8 manua~ 4         9    12 e     pick~
## 5 jeep         grand che~   4.7  2008     8 auto(~ 4         9    12 e     suv
```

Pick Top or Bottom Values

`top_n()` allows you to filter the top or bottom values, ranked by a variable:

```
mpg %>% group_by(manufacturer) %>% top_n(1, hwy)
## # A tibble: 25 x 11
## # Groups:   manufacturer [15]
##    manufacturer model   displ  year   cyl trans   drv     cty   hwy fl    class
##    <chr>        <chr>   <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
##  1 audi         a4        2    2008     4 manual~ f        20    31 p     compa~
##  2 chevrolet    malibu    2.4  2008     4 auto(l~ f        22    30 r     midsi~
##  3 dodge        carava~   2.4  1999     4 auto(l~ f        18    24 r     miniv~
##  4 dodge        carava~   3    1999     6 auto(l~ f        17    24 r     miniv~
##  5 dodge        carava~   3.3  2008     6 auto(l~ f        17    24 r     miniv~
##  6 dodge        carava~   3.3  2008     6 auto(l~ f        17    24 r     miniv~
##  7 ford         mustang   3.8  1999     6 manual~ r        18    26 r     subco~
##  8 ford         mustang   4    2008     6 manual~ r        17    26 r     subco~
##  9 honda        civic     1.8  2008     4 auto(l~ f        25    36 r     subco~
## 10 honda        civic     1.8  2008     4 auto(l~ f        24    36 c     subco~
## # ... with 15 more rows
```

Pick Top or Bottom Values

`top_frac()` allows you to filter the top or bottom values, ranked by a variable:

```
top_frac(mpg, .01, displ)
## # A tibble: 2 x 11
##   manufacturer model     displ  year   cyl trans   drv     cty   hwy fl    class
##   <chr>        <chr>     <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
## 1 chevrolet    corvette    7    2008     8 manual~ r        15    24 p     2sea~
## 2 chevrolet    k1500 ta~   6.5  1999     8 auto(l~ 4        14    17 d     suv
```

Count Observations

`count()` calculates the sample size of your data as summary statistic:

```
count(mpg)
## # A tibble: 1 x 1
##       n
##   <int>
## 1   234
```

Count Observations

`count()` is a short-cut for `summarize(n = n())`:

```
count(mpg)
## # A tibble: 1 x 1
##       n
##   <int>
## 1   234
```

```
summarize(mpg, n = n())
## # A tibble: 1 x 1
##       n
##   <int>
## 1   234
```

Count Observations

`count()` calculates the sample size of your data as summary statistic - also per group:

```
mpg %>% group_by(manufacturer) %>% count()
## # A tibble: 15 x 2
## # Groups:   manufacturer [15]
##    manufacturer     n
##    <chr>        <int>
##  1 audi            18
##  2 chevrolet       19
##  3 dodge           37
##  4 ford            25
##  5 honda            9
##  6 hyundai         14
##  7 jeep             8
##  8 land rover       4
##  9 lincoln          3
## 10 mercury          4
## 11 nissan          13
## 12 pontiac          5
## 13 subaru          14
## 14 toyota          34
## 15 volkswagen      27
```

Count Observations

And you can actually also run this without `group_by()`:

```
count(mpg, manufacturer)
## # A tibble: 15 x 2
##    manufacturer     n
##    <chr>        <int>
##  1 audi            18
##  2 chevrolet       19
##  3 dodge           37
##  4 ford            25
##  5 honda            9
##  6 hyundai         14
##  7 jeep             8
##  8 land rover       4
##  9 lincoln          3
## 10 mercury          4
## 11 nissan          13
## 12 pontiac          5
## 13 subaru          14
## 14 toyota          34
## 15 volkswagen      27
```

Count Observations

And you can actually also run this without `group_by()`:

```
mpg %>% count(manufacturer)
## # A tibble: 15 x 2
##    manufacturer     n
##    <chr>        <int>
##  1 audi            18
##  2 chevrolet       19
##  3 dodge           37
##  4 ford            25
##  5 honda            9
##  6 hyundai         14
##  7 jeep             8
##  8 land rover       4
##  9 lincoln          3
## 10 mercury          4
## 11 nissan          13
## 12 pontiac          5
## 13 subaru          14
## 14 toyota          34
## 15 volkswagen      27
```

Count Observations

`count()` calculates the sample size of your data as summary statistic - also per group:

```
mpg %>% count(manufacturer, sort = TRUE)
## # A tibble: 15 x 2
##    manufacturer     n
##    <chr>        <int>
##  1 dodge           37
##  2 toyota          34
##  3 volkswagen      27
##  4 ford            25
##  5 chevrolet       19
##  6 audi            18
##  7 hyundai         14
##  8 subaru          14
##  9 nissan          13
## 10 honda            9
## 11 jeep             8
## 12 pontiac          5
## 13 land rover       4
## 14 mercury          4
## 15 lincoln          3
```

Count Observations

`tally()` does basically the same as `count()`:

```
tally(mpg)
## # A tibble: 1 x 1
##       n
##   <int>
## 1   234
```

Count Observations

`add_count()` adds the sample size of your data as additional column:

```
add_count(mpg2)
## # A tibble: 234 x 4
##    model        hwy   cty     n
##    <chr>      <int> <int> <int>
##  1 a4            29    18   234
##  2 a4            29    21   234
##  3 a4            31    20   234
##  4 a4            30    21   234
##  5 a4            26    16   234
##  6 a4            26    18   234
##  7 a4            27    18   234
##  8 a4 quattro    26    18   234
##  9 a4 quattro    25    16   234
## 10 a4 quattro    28    20   234
## # ... with 224 more rows
```

Count Observations

`add_count()` is a short-cut for `mutate(n = n())`:

```
add_count(mpg2)
## # A tibble: 234 x 4
##    model        hwy   cty     n
##    <chr>      <int> <int> <int>
##  1 a4            29    18   234
##  2 a4            29    21   234
##  3 a4            31    20   234
##  4 a4            30    21   234
##  5 a4            26    16   234
##  6 a4            26    18   234
##  7 a4            27    18   234
##  8 a4 quattro    26    18   234
##  9 a4 quattro    25    16   234
## 10 a4 quattro    28    20   234
## # ... with 224 more rows
```

```
mutate(mpg2, n = n())
## # A tibble: 234 x 4
##    model        hwy   cty     n
##    <chr>      <int> <int> <int>
##  1 a4            29    18   234
##  2 a4            29    21   234
##  3 a4            31    20   234
##  4 a4            30    21   234
##  5 a4            26    16   234
##  6 a4            26    18   234
##  7 a4            27    18   234
##  8 a4 quattro    26    18   234
##  9 a4 quattro    25    16   234
## 10 a4 quattro    28    20   234
## # ... with 224 more rows
```

Count Observations

Similarly, there is also `add_tally()`:

```
add_tally(mpg2)
## # A tibble: 234 x 4
##    model        hwy   cty     n
##    <chr>      <int> <int> <int>
##  1 a4            29    18   234
##  2 a4            29    21   234
##  3 a4            31    20   234
##  4 a4            30    21   234
##  5 a4            26    16   234
##  6 a4            26    18   234
##  7 a4            27    18   234
##  8 a4 quattro    26    18   234
##  9 a4 quattro    25    16   234
## 10 a4 quattro    28    20   234
## # ... with 224 more rows
```

Create All Combinations of Two Columns

`complete()` turns missing values into explicit missing values:

```
mpg3 <- mpg %>% group_by(manufacturer, class) %>% count() %>% ungroup()
complete(mpg3, manufacturer, nesting(class))
## # A tibble: 105 x 3
##    manufacturer class          n
##    <chr>        <chr>      <int>
##  1 audi         2seater       NA
##  2 audi         compact       15
##  3 audi         midsize        3
##  4 audi         minivan       NA
##  5 audi         pickup        NA
##  6 audi         subcompact    NA
##  7 audi         suv           NA
##  8 chevrolet    2seater        5
##  9 chevrolet    compact       NA
## 10 chevrolet    midsize        5
## # ... with 95 more rows
```

Create All Combinations of Two Columns

`complete()` turns missing values into explicit missing values:

```
complete(mpg3, manufacturer, nesting(class), fill = list(n = 0))
## # A tibble: 105 x 3
##    manufacturer class          n
##    <chr>        <chr>      <dbl>
##  1 audi         2seater        0
##  2 audi         compact       15
##  3 audi         midsize        3
##  4 audi         minivan        0
##  5 audi         pickup         0
##  6 audi         subcompact     0
##  7 audi         suv            0
##  8 chevrolet    2seater        5
##  9 chevrolet    compact        0
## 10 chevrolet    midsize        5
## # ... with 95 more rows
```

Create All Combinations of Two Columns

`complete()` turns missing values into explicit missing values:

```
mpg3 <- mpg %>% group_by(manufacturer, class) %>% count() %>% ungroup()
complete(mpg3, manufacturer, nesting(class), fill = list(n = 0))
## # A tibble: 105 x 3
##    manufacturer class          n
##    <chr>        <chr>      <dbl>
##  1 audi         2seater        0
##  2 audi         compact       15
##  3 audi         midsize        3
##  4 audi         minivan        0
##  5 audi         pickup         0
##  6 audi         subcompact     0
##  7 audi         suv            0
##  8 chevrolet    2seater        5
##  9 chevrolet    compact        0
## 10 chevrolet    midsize        5
## # ... with 95 more rows
```

The `join()` Family

Several `*_join()` functions let you merge two data tables:

```
band_members
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
band_instruments
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

The `join()` Family

Several `*_join()` functions let you merge two data tables:

```
left_join(band_members, band_instruments)
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

The `join()` Family

Several `*_join()` functions let you merge two data tables:

```
right_join(band_members, band_instruments)
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
## 3 Keith <NA>    guitar
```

The `join()` Family

Several `*_join()` functions let you merge two data tables:

```
inner_join(band_members, band_instruments)
## # A tibble: 2 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
```

The `join()` Family

Several `*_join()` functions let you merge two data tables:

```
full_join(band_members, band_instruments)
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

The `join()` Family

Several `*_join()` functions let you merge two data tables:

```
full_join(band_members, band_instruments, by = "name")
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

The `join()` Family

Several `*_join()` functions let you merge two data tables:

```
band_members
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
band_instruments2
## # A tibble: 3 x 2
##   artist plays
##   <chr>  <chr>
## 1 John   guitar
## 2 Paul   bass
## 3 Keith  guitar
```

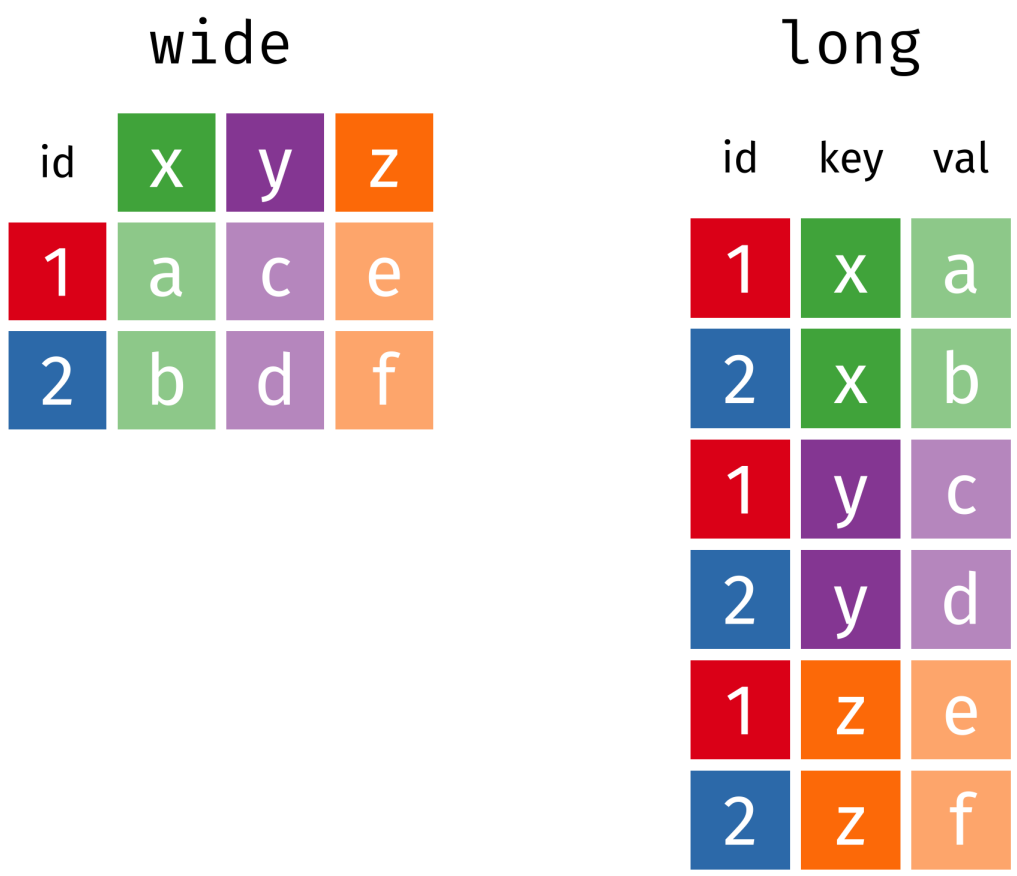
The `join()` Family

Several `*_join()` functions let you merge two data tables:

```
full_join(band_members, band_instruments2, by = c("name" = "artist"))
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

The tidyr Package

pivot_wider() and pivot_longer() (tidyr)



Created by Garrick Aden-Buie (github.com/gadenbuie/tidyexplain)

`pivot_wider()` and `pivot_longer()` (`tidyr`)

`pivot_wider(names_from = , values_from = )` let's you turn a long data frame into a wide one:

```
(mpg_wide <-
  mpg %>%
  group_by(manufacturer, model, trans, cyl, year) %>%
  summarize(cty = mean(cty, na.rm = TRUE)) %>%
  pivot_wider(
    id_cols = c(manufacturer, model, trans, cyl),
    names_from = year,
    values_from = cty,
    names_prefix = "cty_"
  ))
## # A tibble: 157 x 6
## # Groups:   manufacturer, model, trans, cyl [157]
##    manufacturer model      trans        cyl cty_2008 cty_1999
##    <chr>        <chr>      <chr>      <int>    <dbl>    <dbl>
##  1 audi         a4         auto(av)       4       21       NA
##  2 audi         a4         auto(av)       6       18       NA
##  3 audi         a4         auto(l5)       4       NA       18
##  4 audi         a4         auto(l5)       6       NA       16
##  5 audi         a4         manual(m5)     4       NA       21
##  6 audi         a4         manual(m5)     6       NA       18
```

`pivot_wider()` and `pivot_longer()` (`tidyr`)

`pivot_longer(names_to = , values_to = )` let's you turn a wide data frame into a long one:

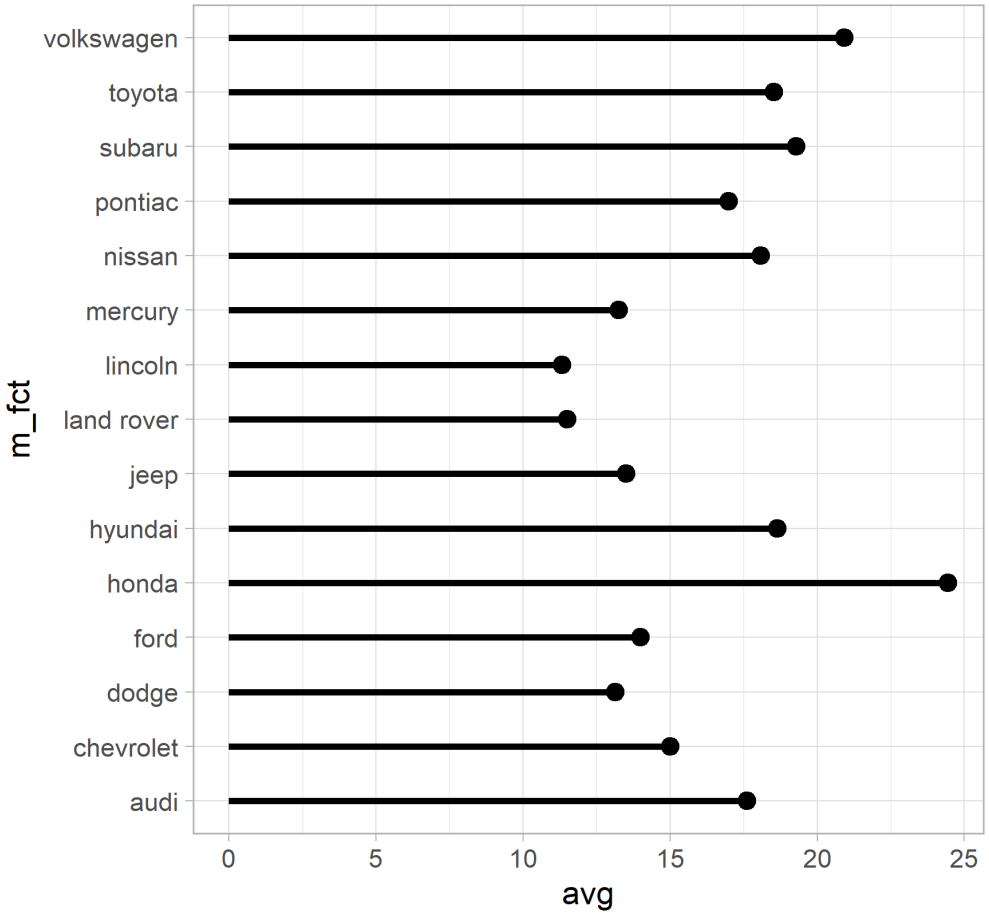
```
mpg_wide %>%
  pivot_longer(
    cols = starts_with("cty_"),
    names_to = "year",
    values_to = "cty",
    names_prefix = "cty_"
  )
## # A tibble: 314 x 6
## # Groups:   manufacturer, model, trans, cyl [157]
##    manufacturer model trans        cyl year    cty
##    <chr>        <chr> <chr>      <int> <chr> <dbl>
##  1 audi         a4    auto(av)       4 2008     21
##  2 audi         a4    auto(av)       4 1999     NA
##  3 audi         a4    auto(av)       6 2008     18
##  4 audi         a4    auto(av)       6 1999     NA
##  5 audi         a4    auto(l5)       4 2008     NA
##  6 audi         a4    auto(l5)       4 1999     18
##  7 audi         a4    auto(l5)       6 2008     NA
##  8 audi         a4    auto(l5)       6 1999     16
##  9 audi         a4    manual(m5)     4 2008     NA
```

The forcats Package

Functions from the `forcats` Package

The `forcats` package provides helpers for reordering factor levels:

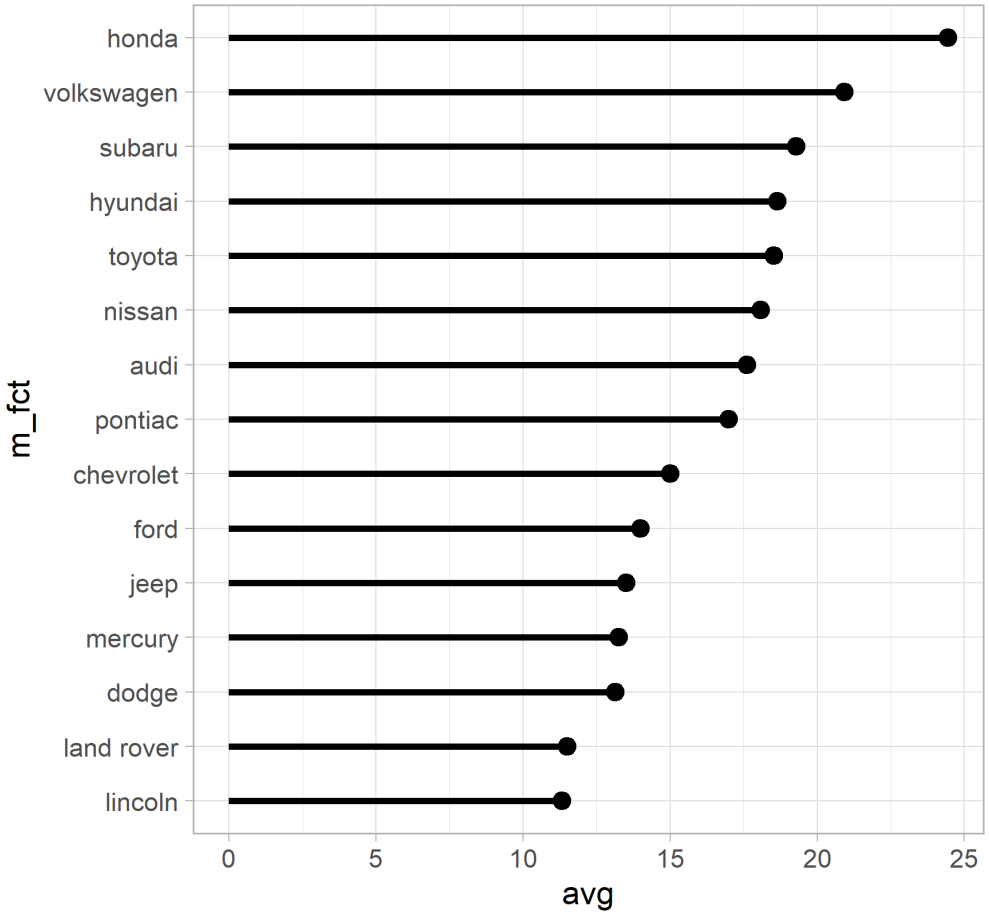
```
mpg_avg <-
  mpg %>%
  mutate(
    m_fct = factor(manufacturer)
  ) %>%
  filter(!is.na(cty)) %>%
  group_by(m_fct) %>%
  summarize(avg = mean(cty)) %>%
  ungroup()

mpg_avg %>%
  ggplot(aes(avg, m_fct)) +
    geom_segment(
      aes(xend = 0, yend = m_fct),
      size = 1.5
    ) +
    geom_point(size = 4)
```



Functions from the `forcats` Package

The `forcats` package provides helpers for reordering factor levels:

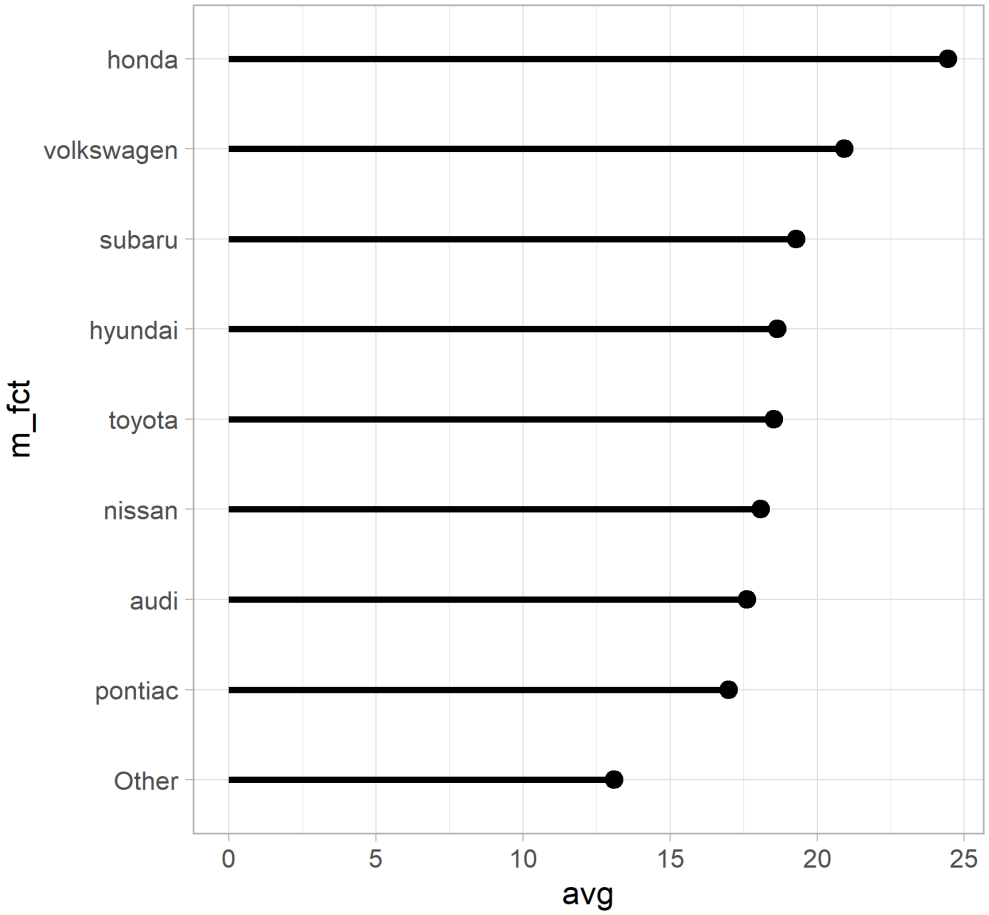
```
mpg_avg %>%
  mutate(
    m_fct = fct_reorder(m_fct, avg)
  ) %>%
  ggplot(aes(avg, m_fct)) +
    geom_segment(
      aes(xend = 0, yend = m_fct),
      size = 1.5
    ) +
    geom_point(size = 4)
```



Functions from the `forcats` Package

The `forcats` package provides helpers for reordering factor levels:

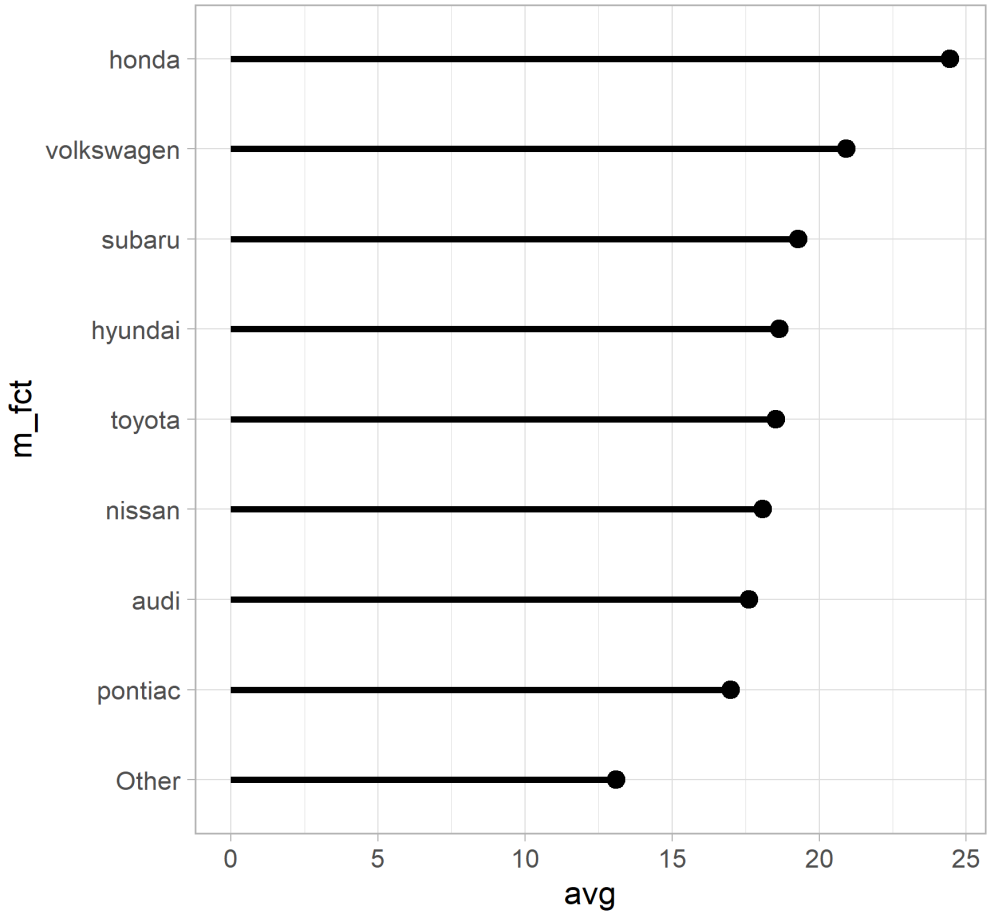
```
mpg_avg %>%
  mutate(m_fct = fct_lump(
    m_fct,
    n = 8,
    w = avg
  )) %>%
  group_by(m_fct) %>%
  summarize(avg = mean(avg)) %>%
  ungroup() %>%
  mutate(m_fct = fct_reorder(m_fct, avg)) 
  ggplot(aes(avg, m_fct)) +
    geom_segment(
      aes(xend = 0, yend = m_fct),
      size = 1.5
    ) +
    geom_point(size = 4)
```



Functions from the `forcats` Package

The `forcats` package provides helpers for reordering factor levels:

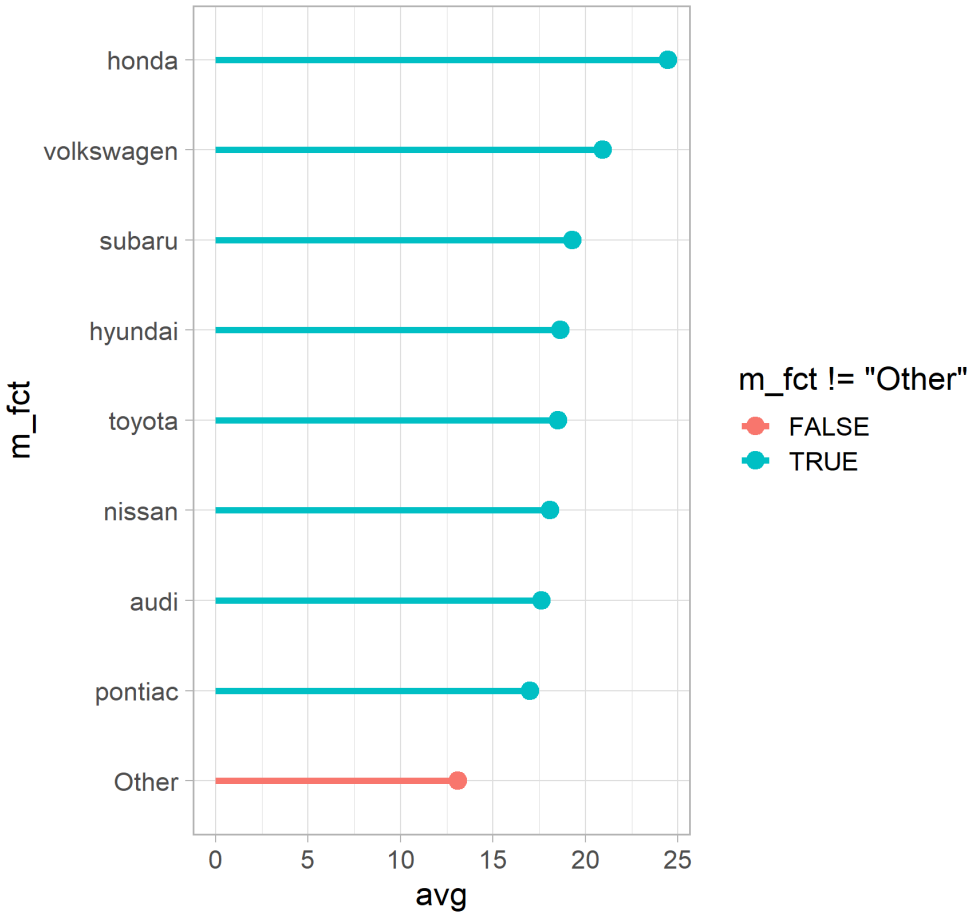
```
mpg_avg %>%
  mutate(m_fct = fct_lump(
    m_fct,
    n = 8,
    w = avg
  )) %>%
  group_by(m_fct) %>%
  summarize(avg = mean(avg)) %>%
  ungroup() %>%
  mutate(m_fct = fct_reorder(m_fct, avg))
  ggplot(aes(avg, m_fct)) +
    geom_segment(
      aes(xend = 0, yend = m_fct),
      size = 1.5
    ) +
    geom_point(size = 4)
```



Functions from the `forcats` Package

The `forcats` package provides helpers for reordering factor levels:

```
mpg_avg %>%
  mutate(m_fct = fct_lump(
    m_fct,
    n = 8,
    w = avg
  )) %>%
  group_by(m_fct) %>%
  summarize(avg = mean(avg)) %>%
  ungroup() %>%
  mutate(m_fct = fct_reorder(m_fct, avg)) %>%
  ggplot(aes(avg, m_fct,
             color = m_fct != "Other")) +
    geom_segment(
      aes(xend = 0, yend = m_fct),
      size = 1.5
    ) +
    geom_point(size = 4)
```



The stringr Package

Functions from the `stringr` Package

Functions from the `stringr` package operate on character strings:

```
(manufacturers <- mpg %>% distinct(manufacturer) %>% pull(manufacturer))
##  [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"
##  [6] "hyundai"    "jeep"       "land rover" "lincoln"    "mercury"
## [11] "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen"
```

Functions from the `stringr` Package

Functions from the `stringr` package operate on character strings:

```
manufacturers <- mpg %>% distinct(manufacturer) %>% pull(manufacturer)

str_sub(manufacturers, 1, 3)
##  [1] "aud" "che" "dod" "for" "hon" "hyu" "jee" "lan" "lin" "mer" "nis" "pon"
## [13] "sub" "toy" "vol"
```

Functions from the `stringr` Package

Functions from the `stringr` package operate on character strings, often using **REG**ular **EX**pressions:

```
manufacturers <- mpg %>% distinct(manufacturer) %>% pull(manufacturer)

str_sub(manufacturers, 1, 3)
##  [1] "aud" "che" "dod" "for" "hon" "hyu" "jee" "lan" "lin" "mer" "nis" "pon"
## [13] "sub" "toy" "vol"

str_subset(manufacturers, "y")
## [1] "hyundai" "mercury" "toyota"
str_subset(manufacturers, "[ab]")
## [1] "audi"       "honda"      "hyundai"    "land rover" "nissan"
## [6] "pontiac"    "subaru"     "toyota"     "volkswagen"
```

Functions from the `stringr` Package

Functions from the `stringr` package operate on character strings, often using **REG**ular **EX**pressions:

```
manufacturers <- mpg %>% distinct(manufacturer) %>% pull(manufacturer)

str_sub(manufacturers, 1, 3)
##  [1] "aud" "che" "dod" "for" "hon" "hyu" "jee" "lan" "lin" "mer" "nis" "pon"
## [13] "sub" "toy" "vol"

str_subset(manufacturers, "y")
## [1] "hyundai" "mercury" "toyota"
str_subset(manufacturers, "[ab]")
## [1] "audi"       "honda"      "hyundai"    "land rover" "nissan"
## [6] "pontiac"    "subaru"     "toyota"     "volkswagen"

str_detect(manufacturers, "[ab]")
##  [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE
```

Functions from the `stringr` Package

Functions from the `stringr` package operate on character strings, often using **REG**ular **EX**pressions:

```
manufacturers <- mpg %>% distinct(manufacturer) %>% pull(manufacturer)

str_sub(manufacturers, 1, 3)
##  [1] "aud" "che" "dod" "for" "hon" "hyu" "jee" "lan" "lin" "mer" "nis" "pon"
## [13] "sub" "toy" "vol"

str_subset(manufacturers, "y")
## [1] "hyundai" "mercury" "toyota"
str_subset(manufacturers, "[ab]")
## [1] "audi"       "honda"      "hyundai"    "land rover" "nissan"
## [6] "pontiac"    "subaru"     "toyota"     "volkswagen"

str_detect(manufacturers, "[ab]")
##  [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE

str_count(manufacturers, "[aeiou]")
##  [1] 3 3 2 1 2 3 2 3 2 2 2 3 3 3 3
```

Functions from the `stringr` Package

Functions from the `stringr` package operate on character strings, often using **REG**ular **EX**pressions:

```
manufacturers <- mpg %>% distinct(manufacturer) %>% pull(manufacturer)

str_sub(manufacturers, 1, 3)
##  [1] "aud" "che" "dod" "for" "hon" "hyu" "jee" "lan" "lin" "mer" "nis" "pon"
## [13] "sub" "toy" "vol"

str_subset(manufacturers, "y")
## [1] "hyundai" "mercury" "toyota"
str_subset(manufacturers, "[aeiou]$")
## [1] "audi"    "dodge"   "honda"   "hyundai" "subaru"  "toyota"

str_detect(manufacturers, "[ab]")
##  [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE

str_count(manufacturers, "[aeiou]")
##  [1] 3 3 2 1 2 3 2 3 2 2 2 3 3 3 3

str_replace(manufacturers, "[aeiou]", "X")
##  [1] "Xudi"       "chXvrolet"  "dXdge"      "fXrd"       "hXnda"
##  [6] "hyXndai"    "jXep"       "lXnd rover" "lXncoln"    "mXrcury"
```

The glue Package

glue

`glue()` allows you to *glue strings together*.

```
mpg %>%
  select(manufacturer, model, year, trans) %>%
  mutate(
    text = glue::glue("The model {manufacturer} {model} with {trans} as transission wa
  )
## # A tibble: 234 x 5
##    manufacturer model      year trans    text
##    <chr>        <chr>     <int> <chr>    <glue>
##  1 audi         a4         1999 auto(l5) The model audi a4 with auto(l5) as tran~
##  2 audi         a4         1999 manual(~ The model audi a4 with manual(m5) as tr~
##  3 audi         a4         2008 manual(~ The model audi a4 with manual(m6) as tr~
##  4 audi         a4         2008 auto(av) The model audi a4 with auto(av) as tran~
##  5 audi         a4         1999 auto(l5) The model audi a4 with auto(l5) as tran~
##  6 audi         a4         1999 manual(~ The model audi a4 with manual(m5) as tr~
##  7 audi         a4         2008 auto(av) The model audi a4 with auto(av) as tran~
##  8 audi         a4 quat~   1999 manual(~ The model audi a4 quattro with manual(m~
##  9 audi         a4 quat~   1999 auto(l5) The model audi a4 quattro with auto(l5)~
## 10 audi         a4 quat~   2008 manual(~ The model audi a4 quattro with manual(m~
## # ... with 224 more rows
```

glue in combination with stringr

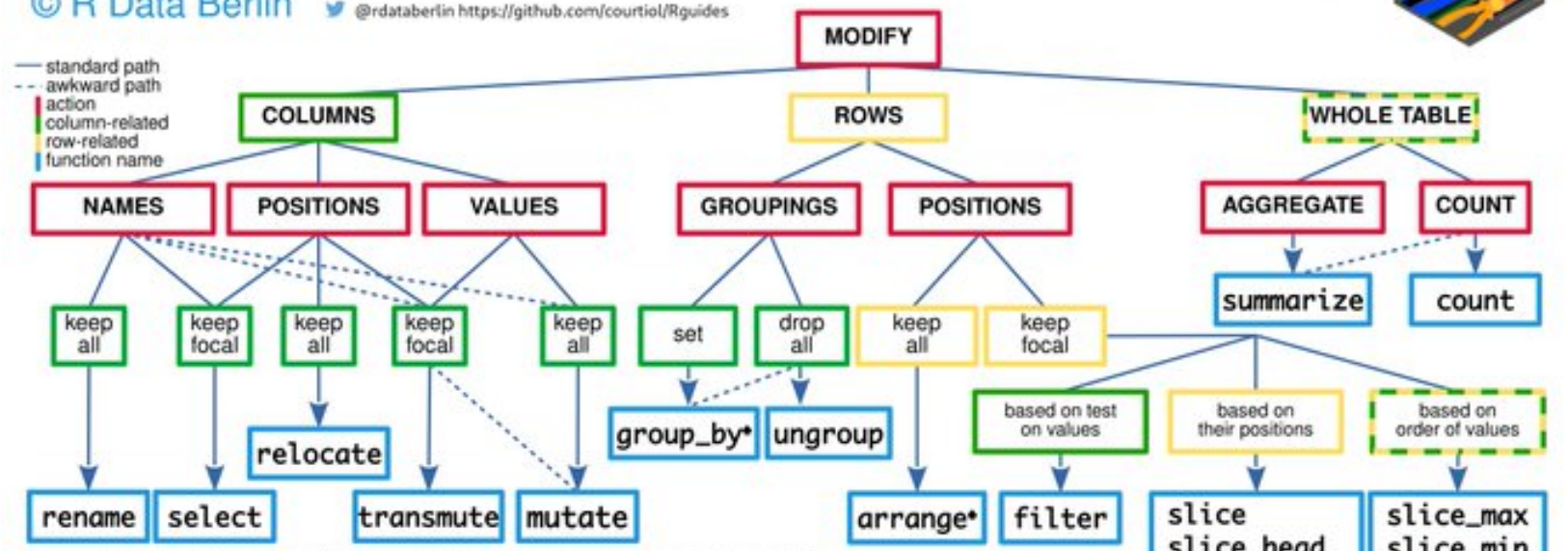
`glue()` allows you to *glue strings together*.

```
mpg %>%
  select(manufacturer, model, year, trans) %>%
  mutate(text =
    glue::glue("The {str_to_title(manufacturer)} {str_to_title(model)} with {trans} as
  )
## # A tibble: 234 x 5
##    manufacturer model      year trans    text
##    <chr>        <chr>     <int> <chr>    <glue>
##  1 audi         a4         1999 auto(l5) The Audi A4 with auto(l5) as transissio~
##  2 audi         a4         1999 manual(~ The Audi A4 with manual(m5) as transiss~
##  3 audi         a4         2008 manual(~ The Audi A4 with manual(m6) as transiss~
##  4 audi         a4         2008 auto(av) The Audi A4 with auto(av) as transissio~
##  5 audi         a4         1999 auto(l5) The Audi A4 with auto(l5) as transissio~
##  6 audi         a4         1999 manual(~ The Audi A4 with manual(m5) as transiss~
##  7 audi         a4         2008 auto(av) The Audi A4 with auto(av) as transissio~
##  8 audi         a4 quat~   1999 manual(~ The Audi A4 Quattro with manual(m5) as ~
##  9 audi         a4 quat~   1999 auto(l5) The Audi A4 Quattro with auto(l5) as tr~
## 10 audi         a4 quat~   2008 manual(~ The Audi A4 Quattro with manual(m6) as ~
## # ... with 224 more rows
```

Data Transformation with **dplyr 1.0** (part 1)

A guide to 17 modifications applied to one tibble (`tbl`) or data.frame

 @rdataberlin https://github.com/courtiol/Rguides



* group_by defines groupings in all functions but arrange, unless using arrange(..., .by_group = TRUE)

SYNTAX: `tbl %>% verb()`

[font style varies to tease apart placeholders from true R_commands]

```
tbl %>% rename(new_name_col_X = old_name_col_X)
tbl %>% select(name_col_X, name_col_Y [+ std_op.*], selection_helper*)
tbl %>% relocate(same_as_for_select, .before[or .after] = name_col_Z/selection_helper*)
tbl %>% transmute(name_col_Z = fn▼(name_col_X))
tbl %>% mutate(name_col_Z = fn▼(name_col_X))
tbl %>% group_by(name_col_X, name_col_Y) %>% verb() %>% ungroup()
tbl %>% arrange(name_col_X, desc(name_col_Y))
tbl %>% filter(fn_test_vectorized(name_col_X), fn_test_vectorized(name_col_Y))
tbl %>% slice(row_indices); tbl %>% slice_head/tail/sample(number_rows_to_keep);
tbl %>% slice_min/max(name_col_X, n = nb_rows_to_keep [or prop = proportion_rows_to_keep])
tbl %>% summarize(name_col_Z = fn▼(name_col_X))
tbl %>% count(); tbl %>% count(name_col_X)
```

* standard operators may be used to combine (`c()`, `&`, `|`) or negate elements (`!`)

* selection helpers from pkg **tidyselect** may be used to select columns based on:
  - column values → `where(fn)`, e.g. `fn = is.numeric`
  - column names → `starts_with("text")`, `ends_with("text")`, `contains("text")`, `matches("regex")`, `num_range("text", min:max)`, `all_of(vector_of_text)`, `any_of(vector_of_text)`
  - column positions → `everything()`, `last_col()` -> *See guide part 2 for more details*

▼ if the function `fn` does not return a scalar or a nrow-long output, use `list(fn())` to create a list column (i.e. for nesting the content); for creating multiple columns at once `fn` should return a `data.frame` or `tibble` and no name should be defined when calling the **dplyr** verb (i.e. ~~`name_col_Z =`~~; if a name is defined, the output will be nested); to unnest to content of a list column, use one of the functions provided by the pkg **tidyr** (e.g. **`unnest_wider`**)

Data Transformation with **dplyr 1.0** (part 2)

A guide to using (c_)across() to apply the same functions repeatedly

 @rdataberlin https://github.com/courtiol/Rguides

Use across() to apply the same function(s) on multiple columns

```
tbl %>%
  [group_by(name_grouping_col_X, name_grouping_col_Y...) %>% ] ## grouping is optional
  verb*(across(selected_columns, fn_with_args, [.names = prototype_for_new_col_names*])) ## prototype for naming is optional
```

- mutate ☺
- transmute ♪
- group_by ●
- arrange
- filter ☼
- summarize ♫
- count 🖷

selection from column-names, values, positions:

- fn+g: name_col_X:name_col_Y 🖷
- c(name_col_X, name_col_Y) ☺
- all_of/any_of(c("name_col_X", "name_col_Y"))
- starts_with/ends_with/contains("partial_name_col") ♪
- matches("regular_expression") ●
- num_range("name_col_without_number", vector_numbers)
- where(fn_returning_TRUE/FALSE_for_each_col) ☼
- c(position_col_X, position_col_Y)
- everything() ♫
- last_col()

* when fn runs, .x will be internally replaced by the content of the selected columns

1 function:

- fn [, args*] ☺
- ~ fn(.x* [, args*]) ☼

*args = possible arguments for fn

≥ 1 functions:

- list(suffix1_new_col_names* = fn1, suffix2_new_col_names* = fn2), [, args*] ●
- list(suffix1_new_col_names* = ~ fn1(.x* [, args*]), ♪ suffix2_new_col_names* = ~ fn2(.x* [, args*]))

A quoted expression with any text and the placeholders {.fn} referring to function position and/or {.col} ☺ referring to former column names

Only useful when no suffix defined in fn_with_args

* if no name is defined, some will automatically be generated by dplyr

* across() and c_across() should not to be used within dplyr verbs shown in part 1 and not here, as it would not make sense. Note however that there is a special function to rename multiple columns: rename_with(fn_with_args, selected_columns)

EXAMPLES for across()

```
## turning specific columns into z-scores:
☺ iris %>%
    mutate(across(c("Sepal.Length", "Sepal.Width"), scale, .names = "{.col}_z"))

## keeping only rows where numeric values are > 2 in all numeric columns:
☼ iris %>% filter(across(where(is.numeric), ~ .x > 2))

## defining 3 groups for all columns with name made of 2 words around a point:
● iris %>% group_by(across(matches("\\w\\.\\w"), list(discrete = cut), breaks = 3))
                                  suffix for new col names   fn   args for fn

## counting NA in each column:
♫ iris %>% summarise(across(everything(), list("NA" = ~ sum(is.na(.x)))))
```

Use c_across() to apply the same function across multiple columns within each row

```
tbl %>%
  rowwise() %>%
  verb*(fn*(c_across(selected_columns), [args*]))
```

Never forget! If you do, c_across() will concatenate values across rows, which is wrong

as for across(), see above

- mutate ☺
- transmute ♪
- filter ☼
- count 🖷

* unlike fn_with_args in across() calls, the function fn is not here provided as a definition but used directly, so you can only use one function that has already been defined (you cannot define it on the fly using ~ & .x) ♪🖷

EXAMPLES for c_across()

```
## computing the area of petal (approximated as rectangles) and only keep that:
♪ iris %>%
    rowwise() %>%
    transmute(Petal_Area = prod(c_across(contains("Petal"))))

## counting rows where at least one numeric values is > 6 for a range of columns:
🖷 iris %>%
    rowwise() %>%
    count(any(c_across(Sepal.Length:Petal.Width) > 6))
```