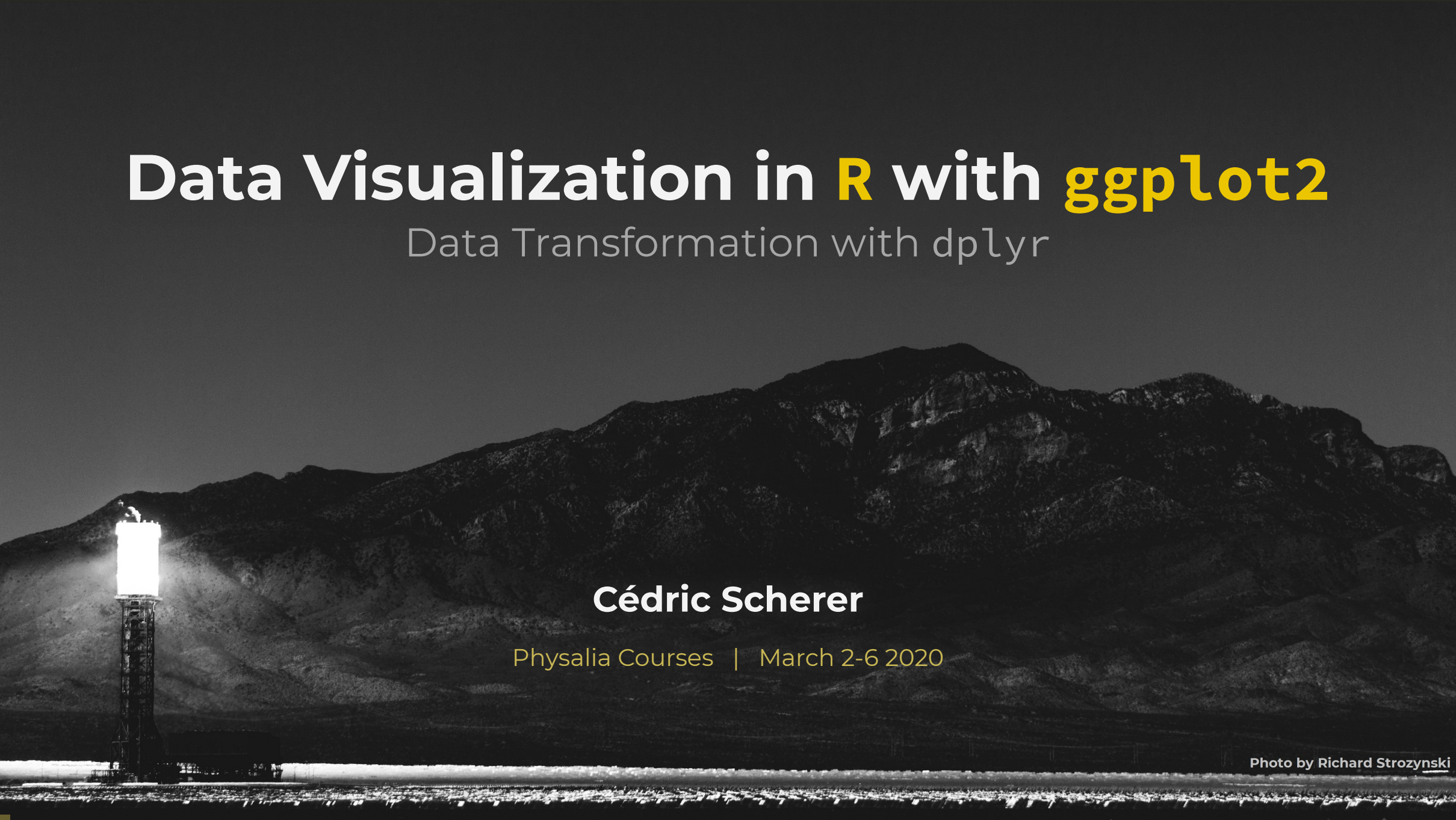
Data Visualization in R with ggplot2

Data Transformation with dplyr



Cédric Scherer

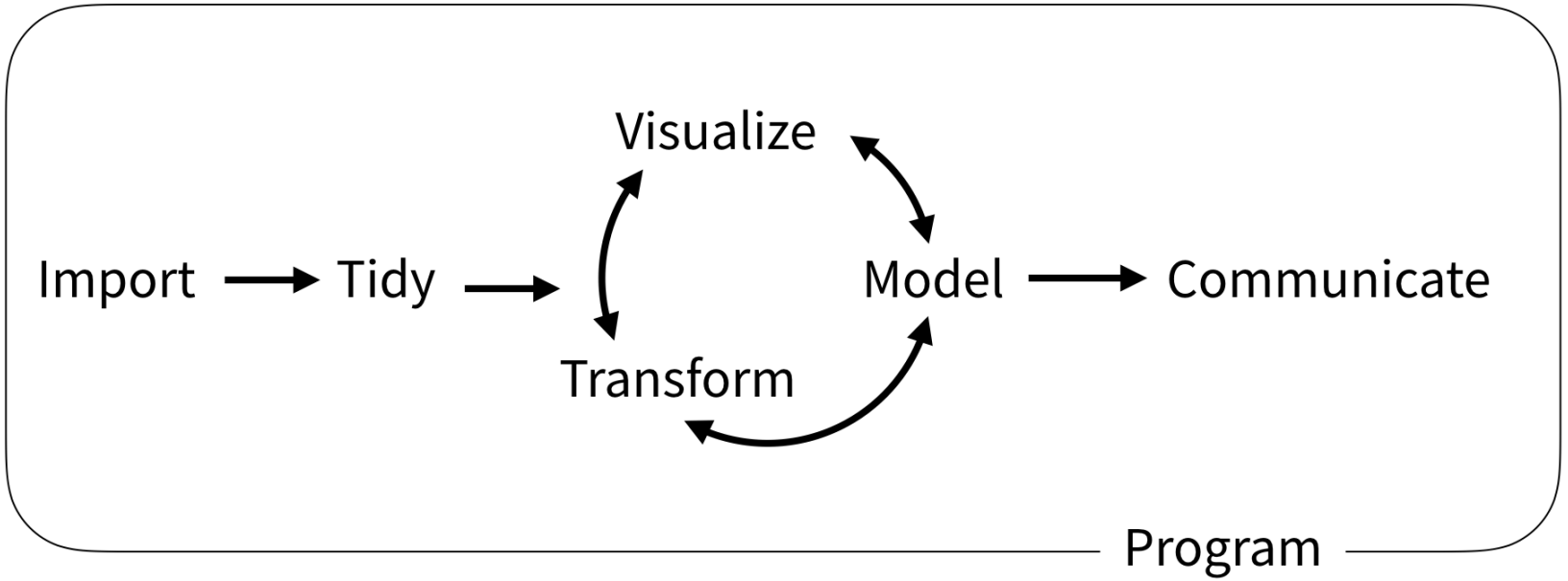
Physalia Courses | March 2-6 2020

Photo by Richard Strozynski

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

The `tidyverse`

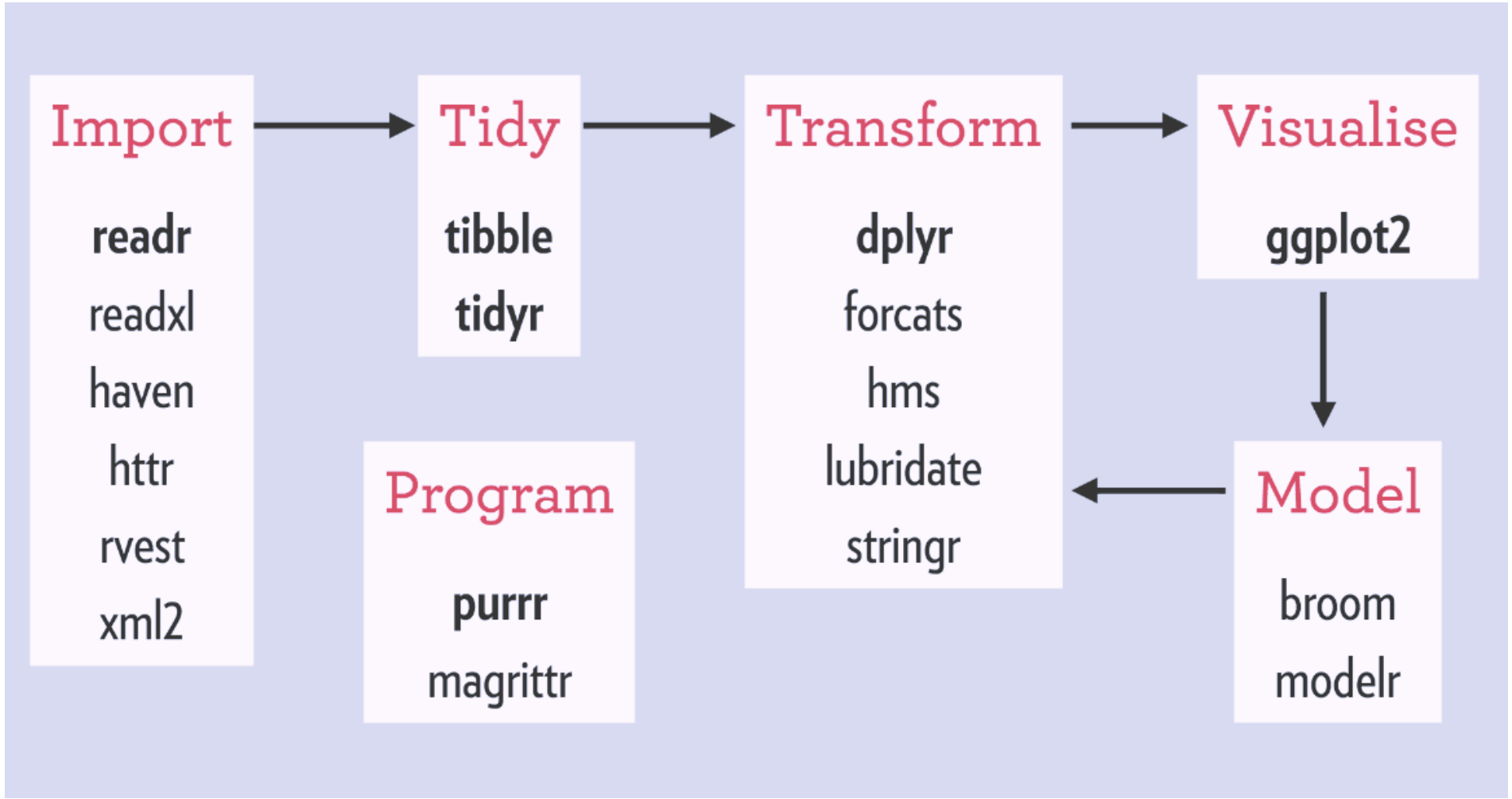
The `tidyverse` provides exemplary support for data wrangling and is the main reason for the recent popularity of `R`, especially in data-driven environments.



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

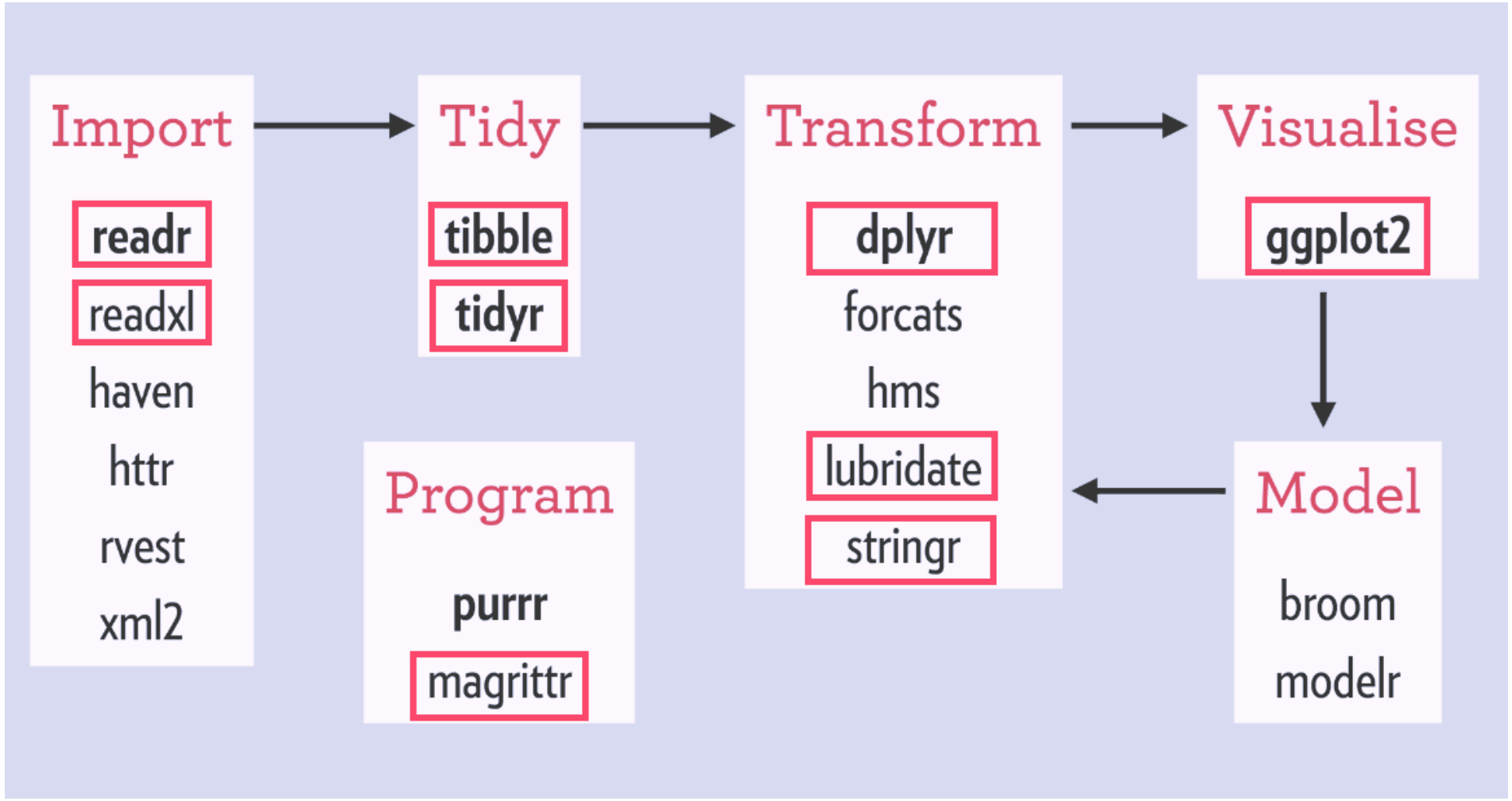
Modified from Hadley Wickham's "R for Data Science" (R4DS)

The Typical Data Science Project Flow



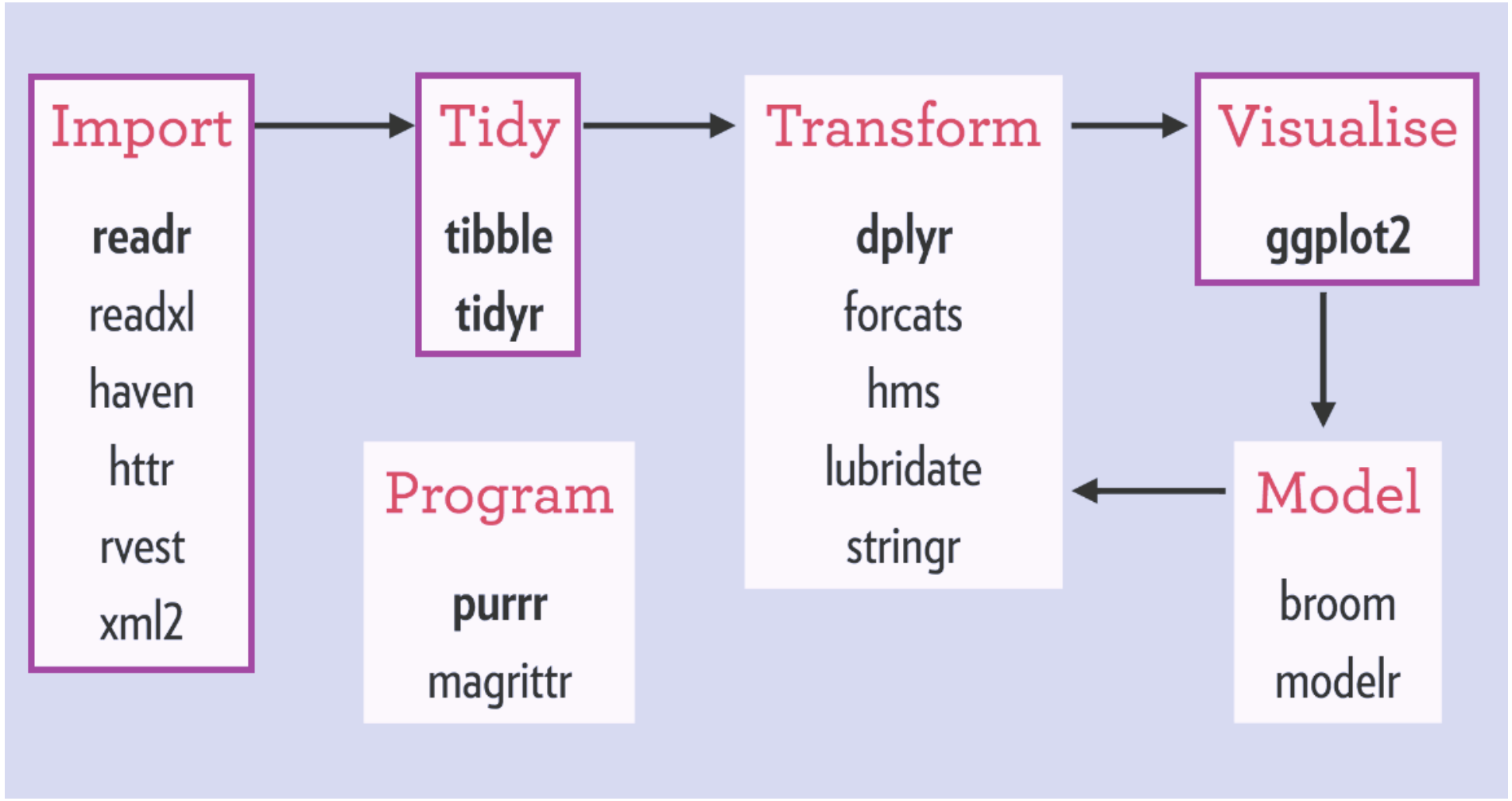
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

The Typical Data Science Project Flow



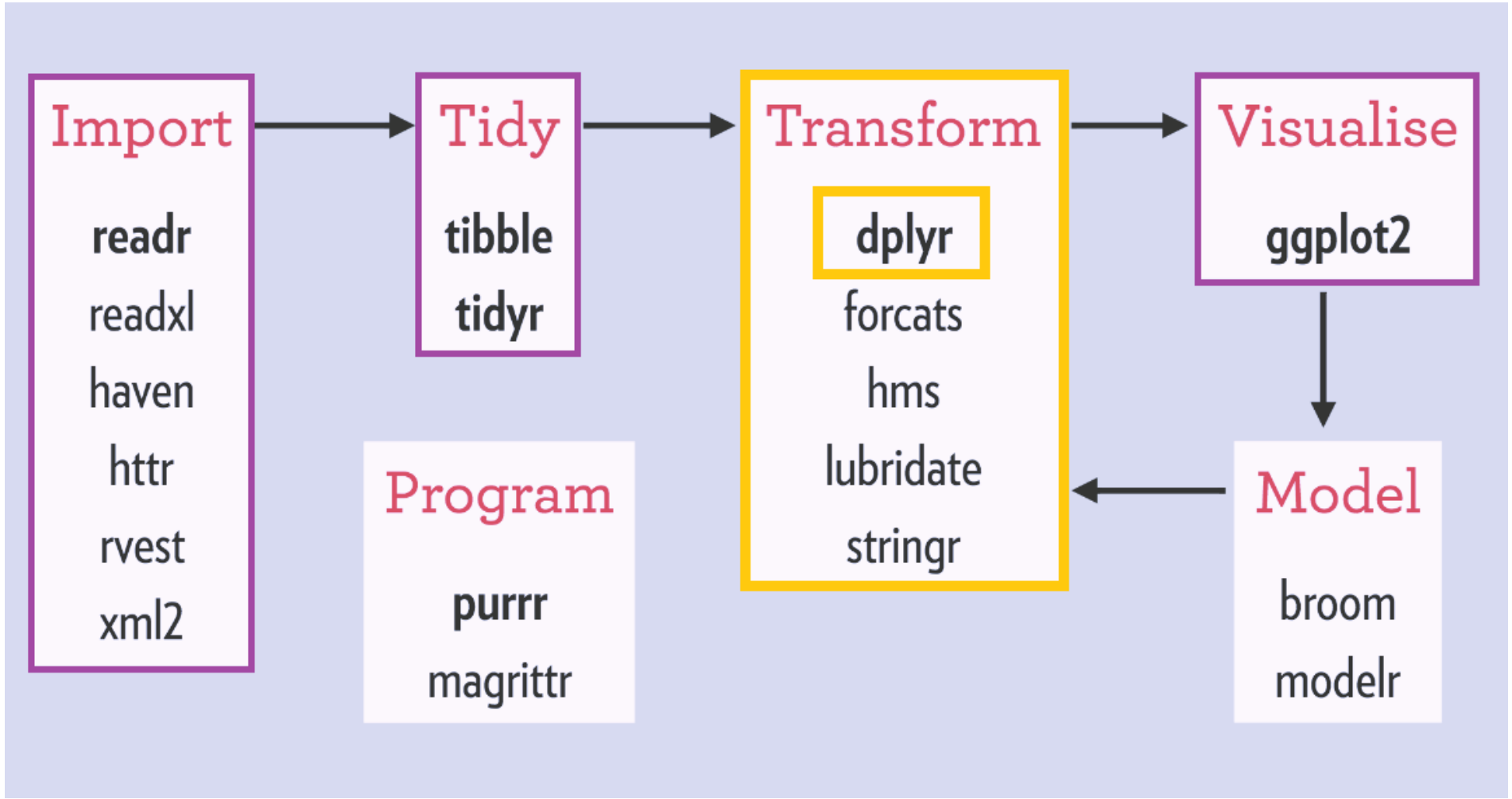
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

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

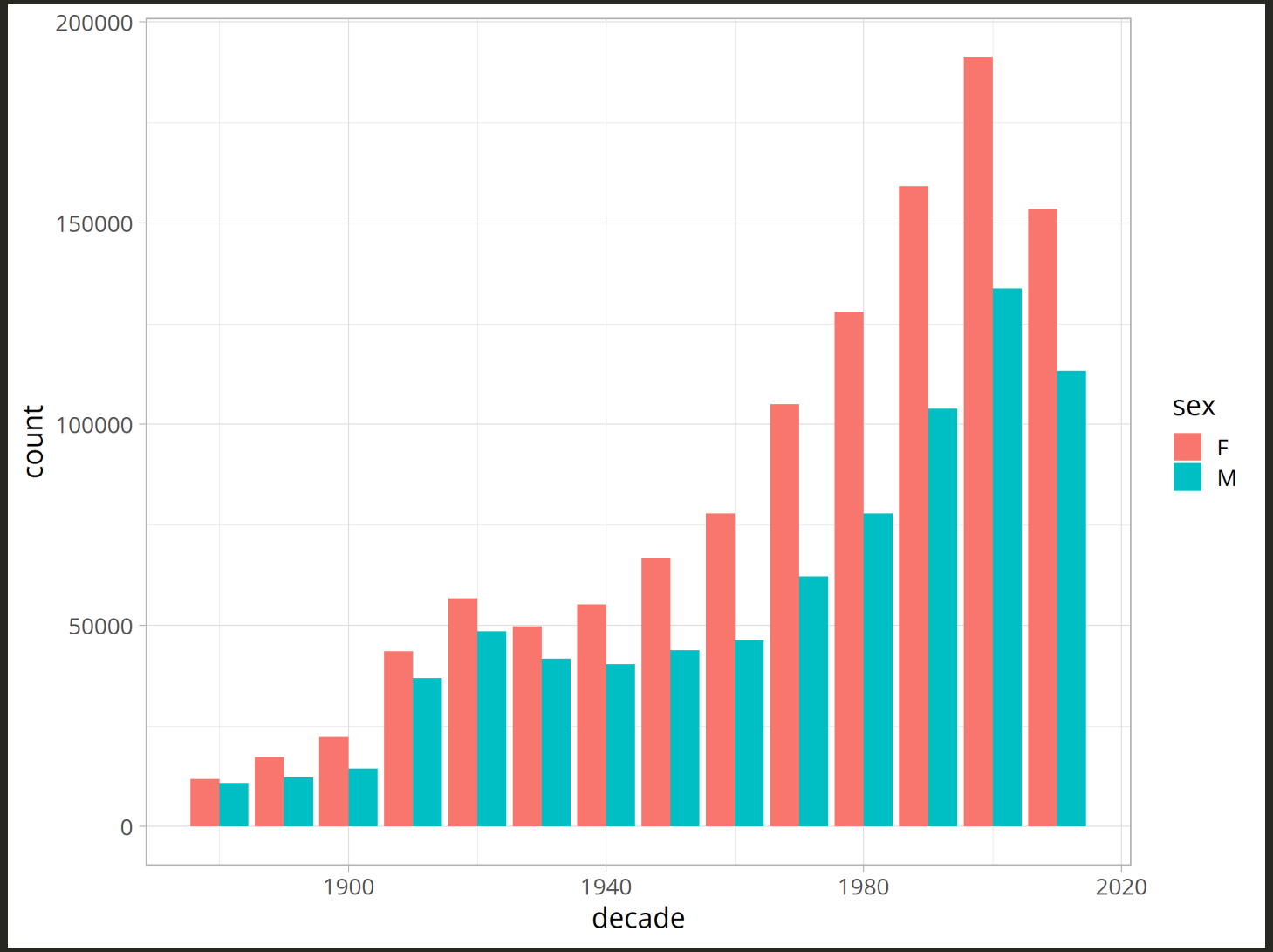
Our Example Data

- 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`

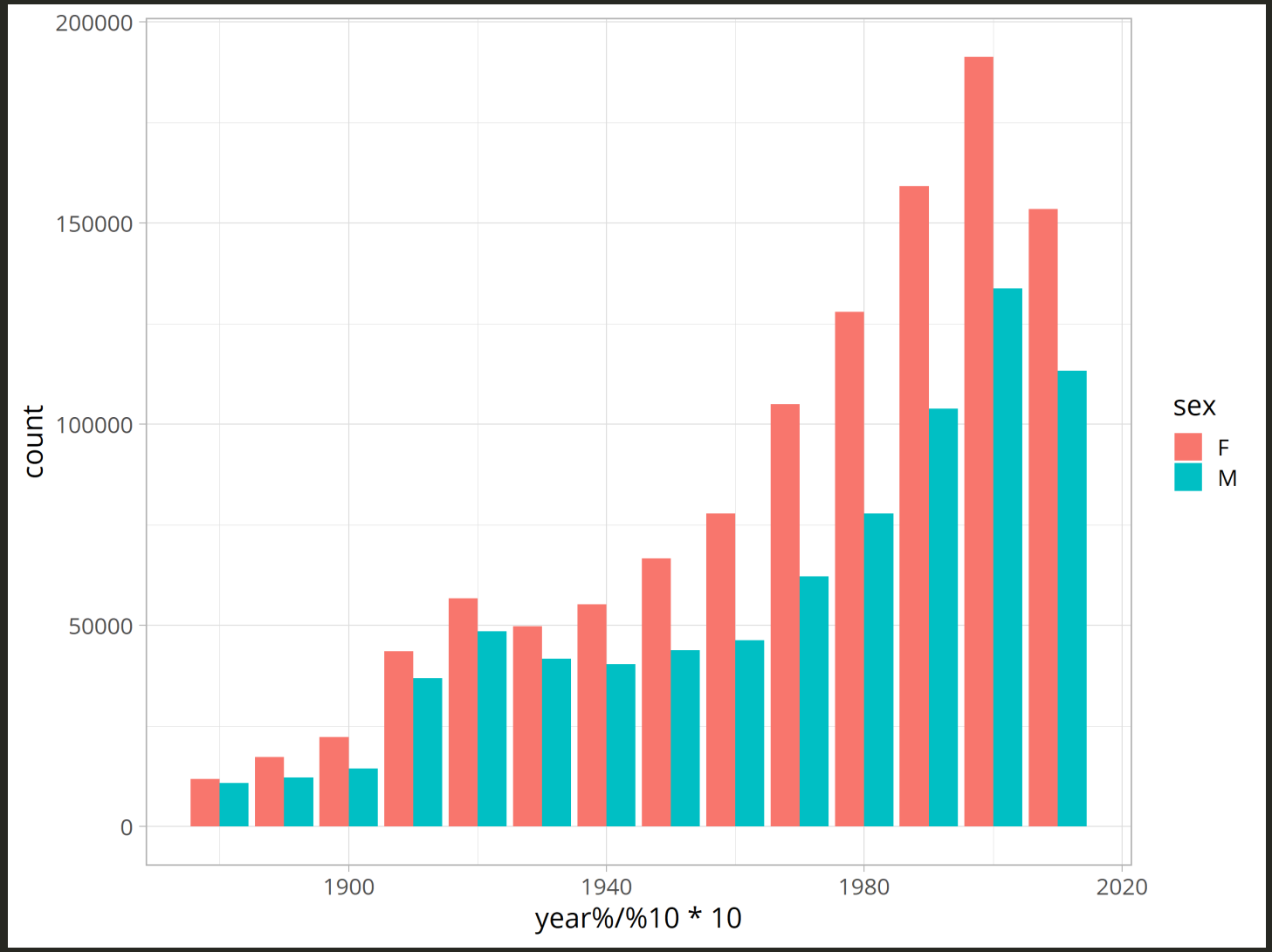
Our Example Data: babynames

```
library(babynames)

glimpse(babynames)
## Observations: 1,924,665
## Variables: 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)

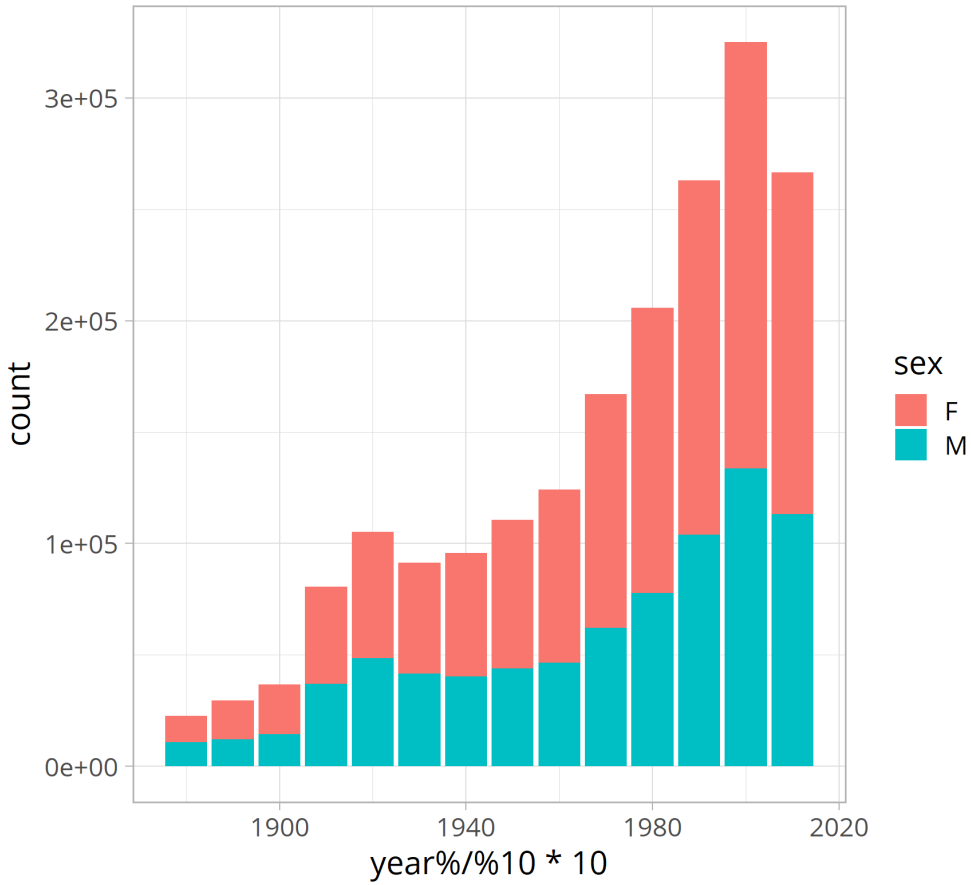


Excourse: Integer Division

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

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

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

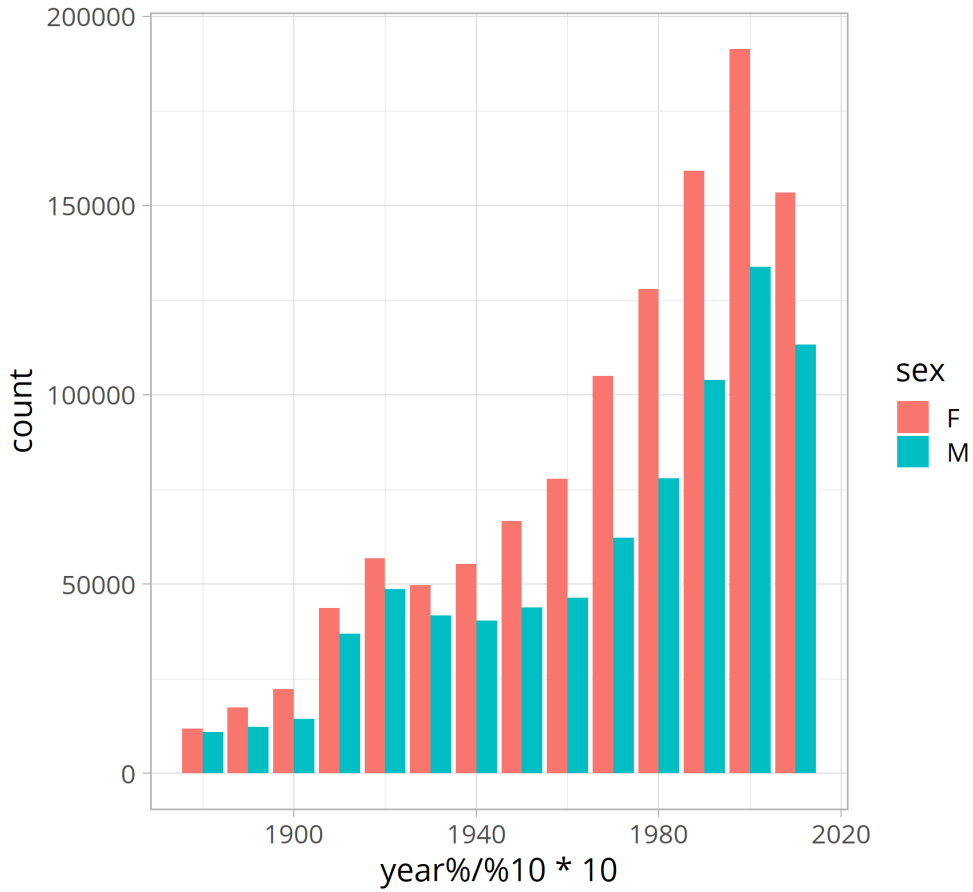
Your Turn: Dodged Bar Plots

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



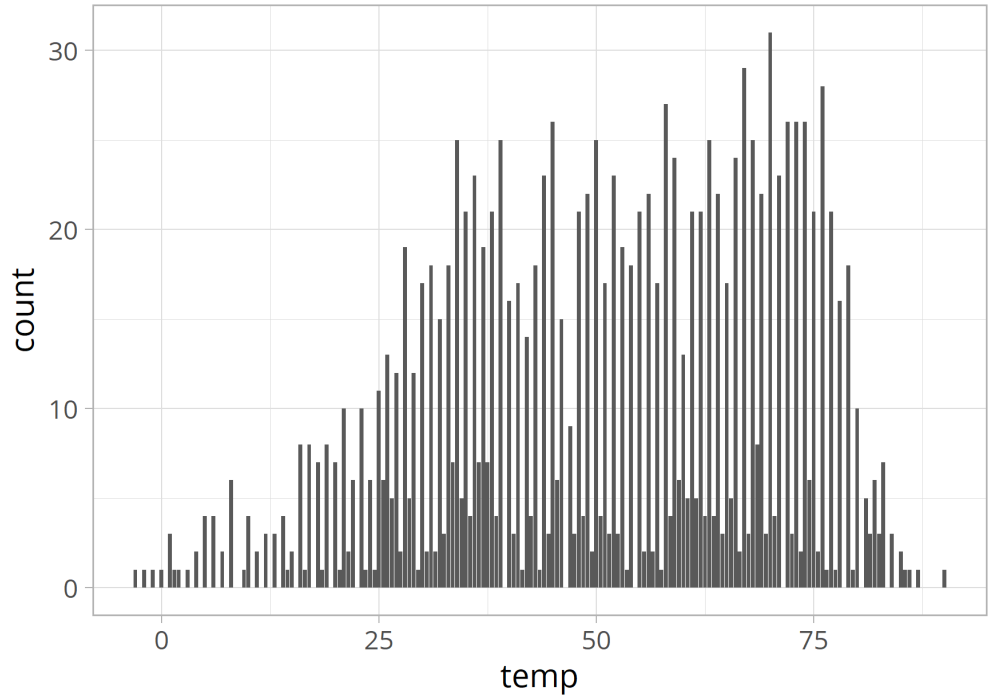
Your Turn: Dodged Bar Plots

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

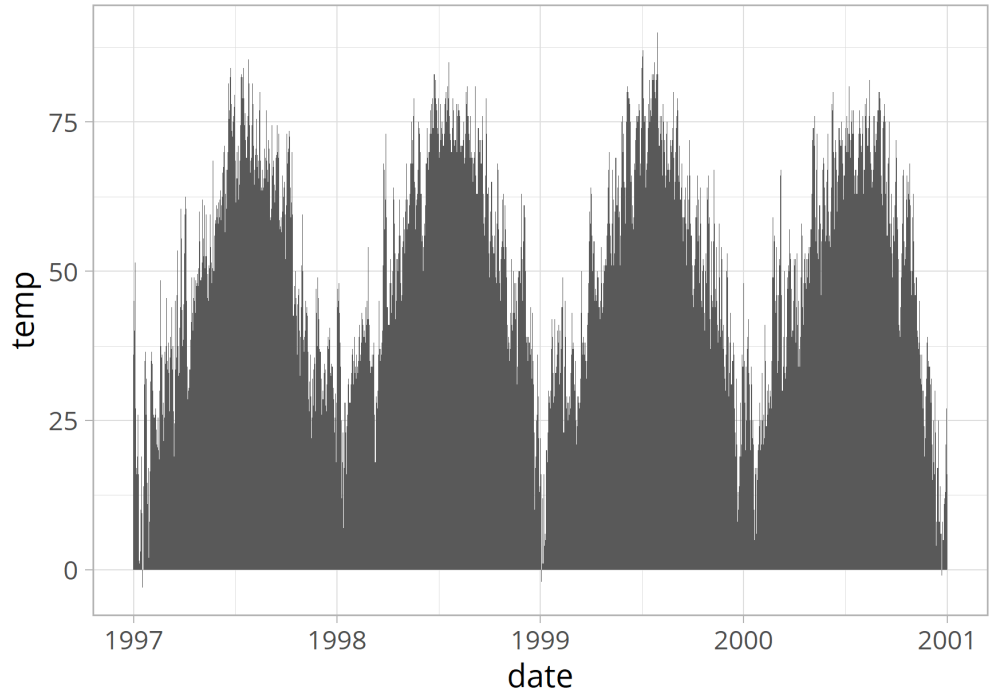


geom_bar()

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

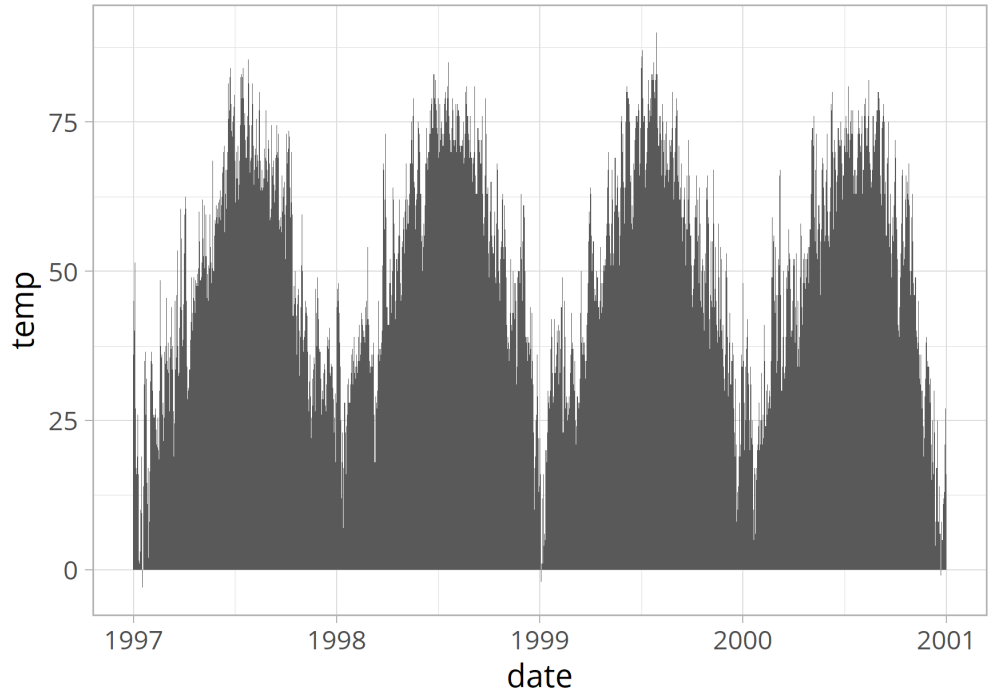



```
ggplot(chic, aes(date, temp)) +
  geom_bar(stat = "identity")
```

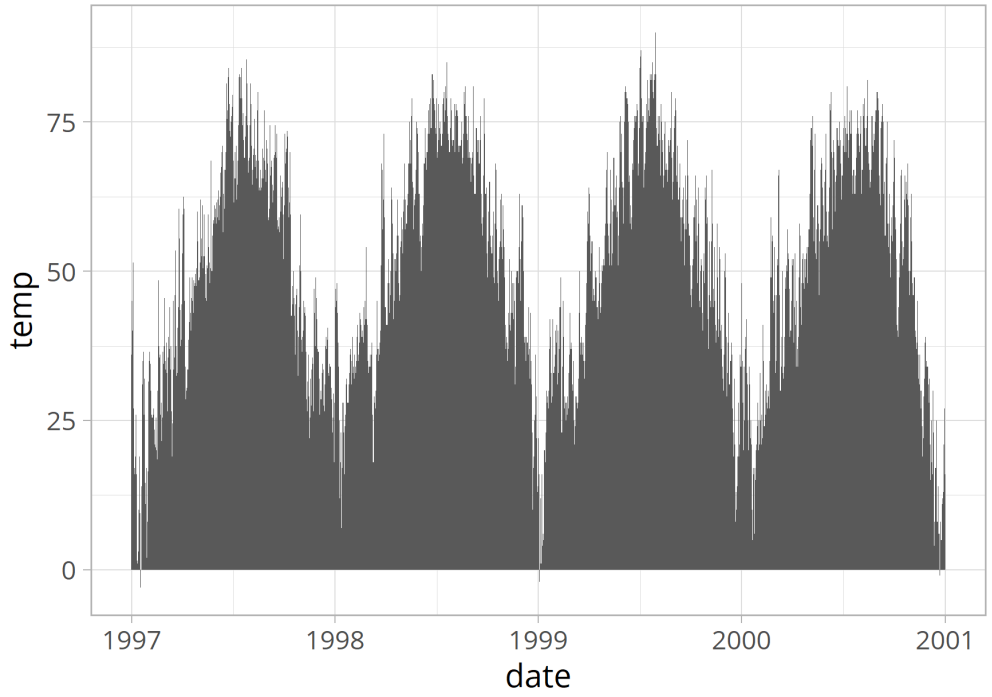


geom_bar() versus geom_col()

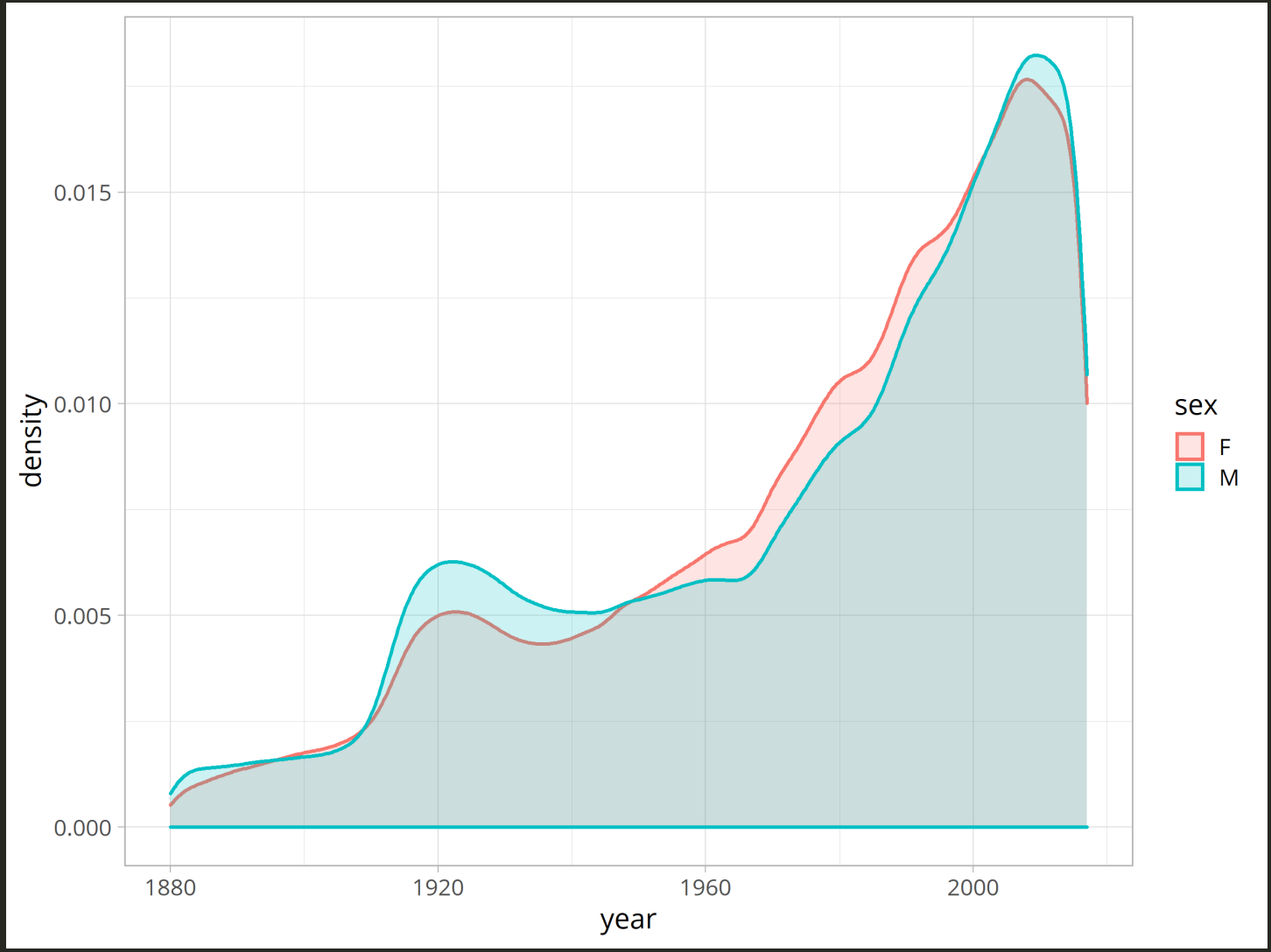
```
ggplot(chic, aes(date, temp)) +
  geom_col()
```




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

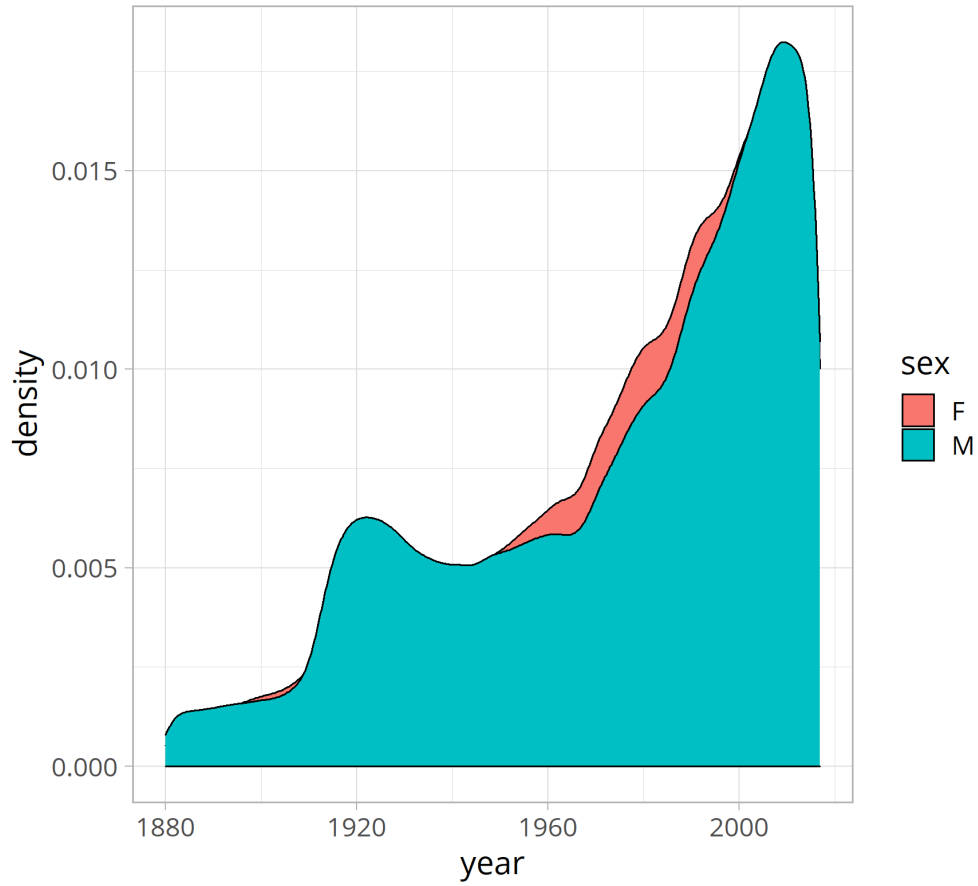


Your Turn!



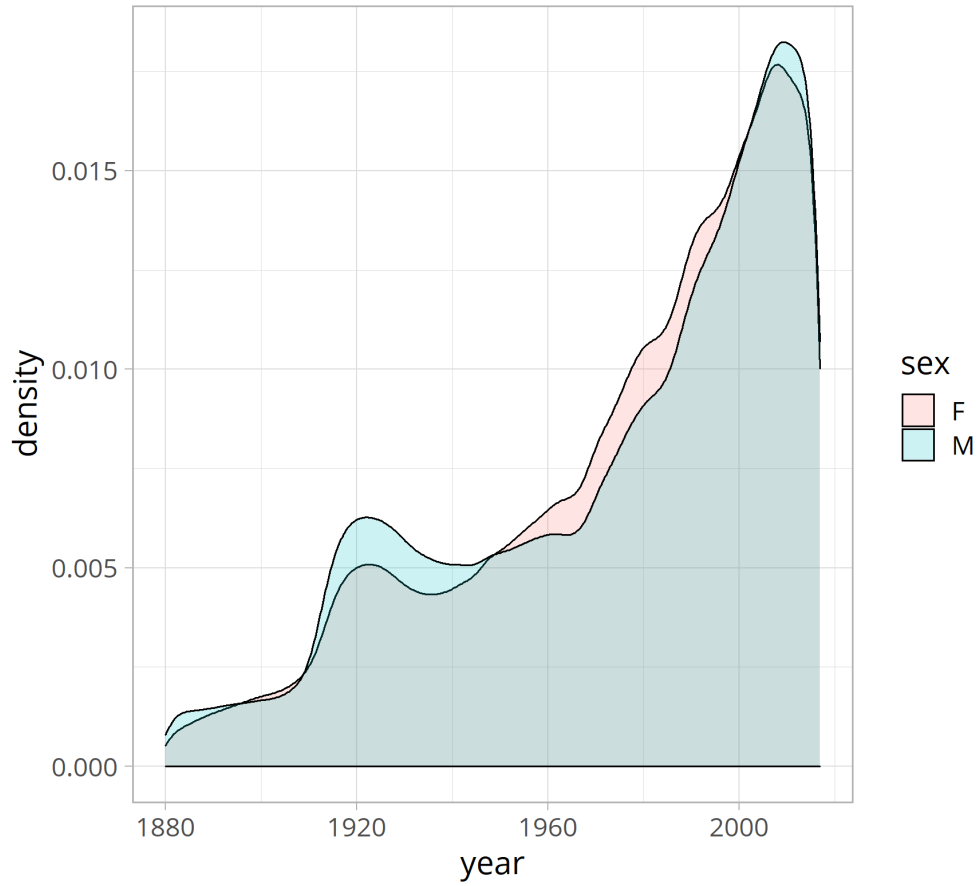
Your Turn: Density Plots

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



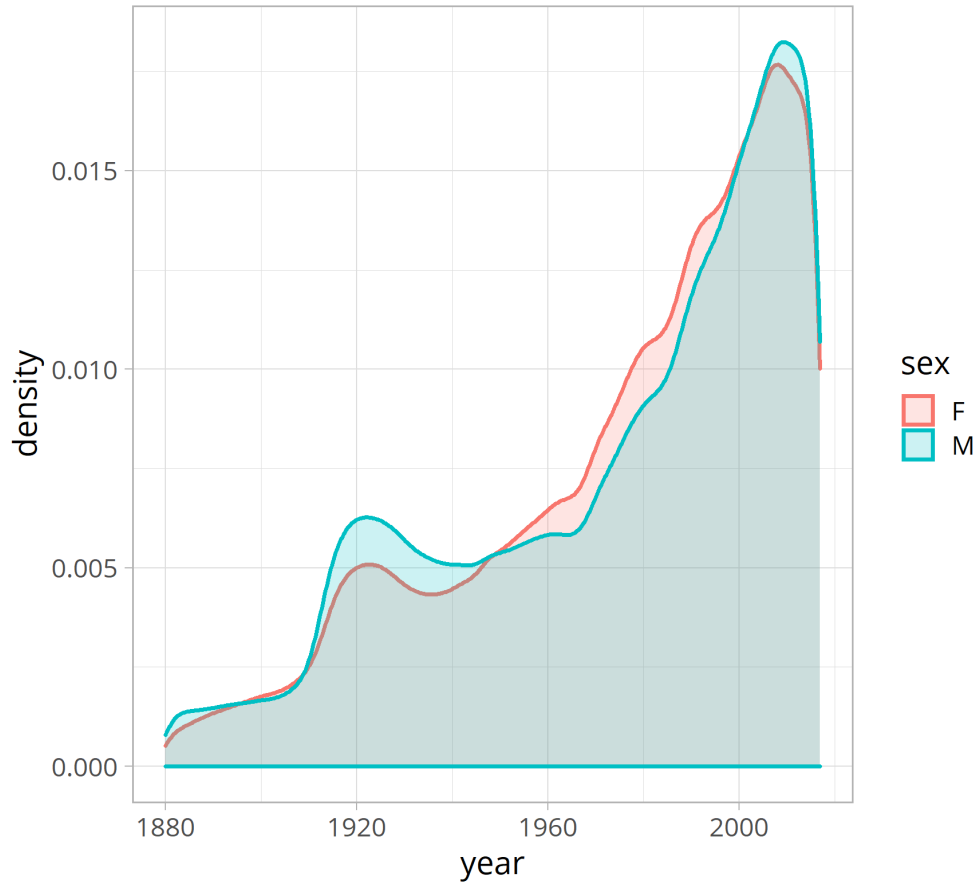
Your Turn: Density Plots

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

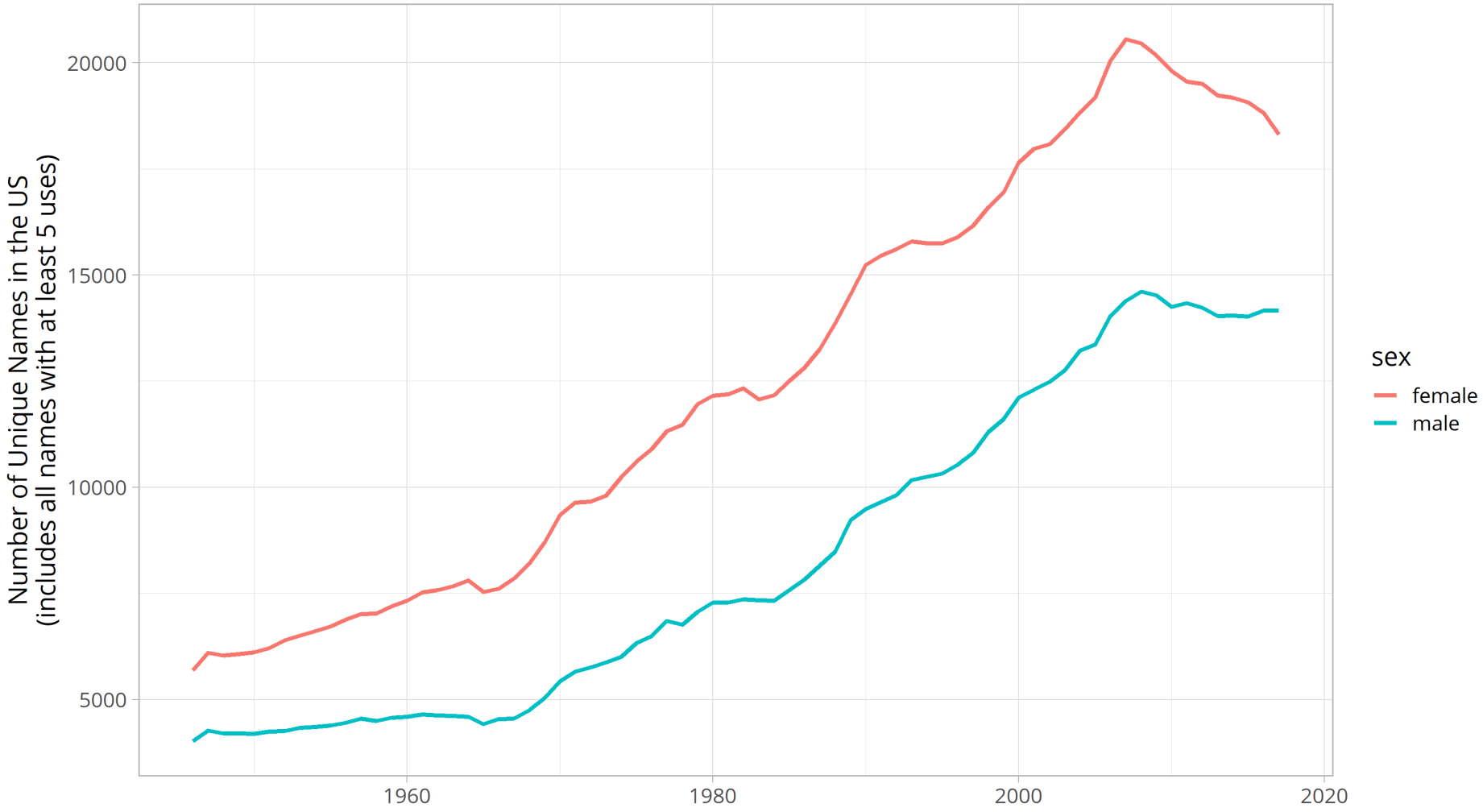


Your Turn: Density Plots

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

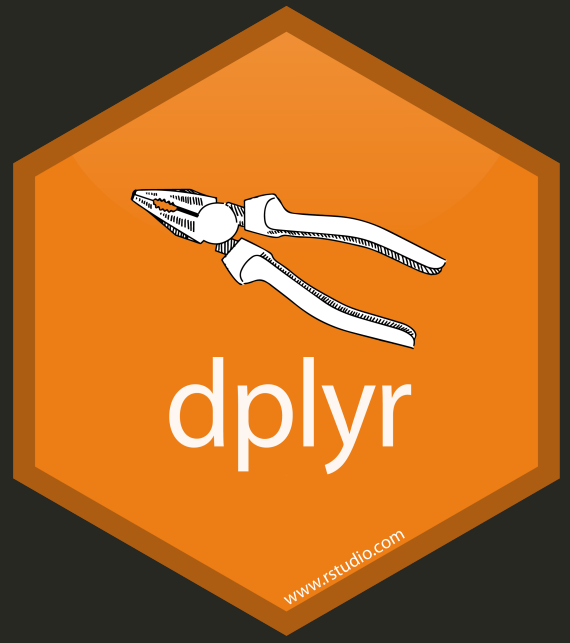


The Rise of Unique Baby Names in the Post-WWII Era



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
`d()`	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`

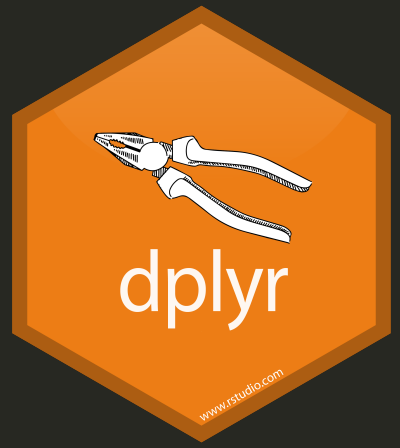
All functions take the same main arguments:

```
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
```

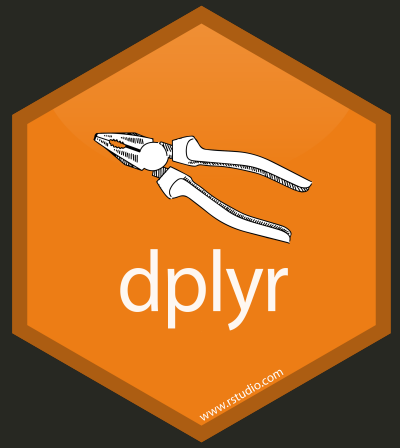
Pick Rows with Matching Criteria

`filter(data, condition)` → 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
```

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

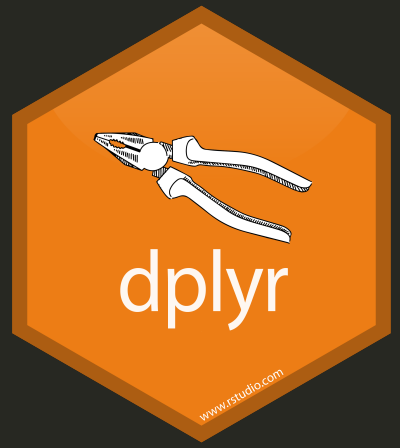
`select(data, condition)` allows you to select a subset of columns:

```
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
```

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

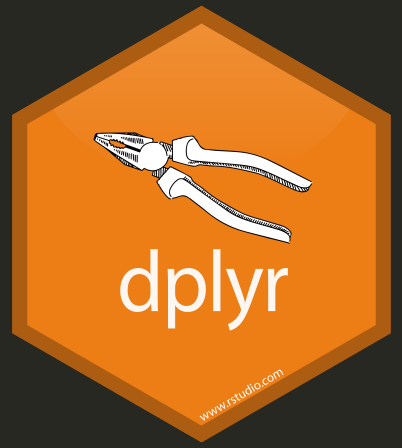


Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

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      0
##  2  1880 F     Anna        2604 0.0267      0
##  3  1880 F     Emma        2003 0.0205      0
##  4  1880 F     Elizabeth   1939 0.0199      0
##  5  1880 F     Minnie      1746 0.0179      0
##  6  1880 F     Margaret    1578 0.0162      0
##  7  1880 F     Ida         1472 0.0151      0
##  8  1880 F     Alice       1414 0.0145      0
##  9  1880 F     Bertha      1320 0.0135      0
## 10  1880 F     Sarah       1288 0.0132      0
## # ... 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`:

```
transform(babynames, decade = year %% 10 * 10)
##       year sex         name    n       prop decade
## 1     1880   F         Mary 7065 0.07238359      0
## 2     1880   F         Anna 2604 0.02667896      0
## 3     1880   F         Emma 2003 0.02052149      0
## 4     1880   F    Elizabeth 1939 0.01986579      0
## 5     1880   F       Minnie 1746 0.01788843      0
## 6     1880   F     Margaret 1578 0.01616720      0
## 7     1880   F          Ida 1472 0.01508119      0
## 8     1880   F        Alice 1414 0.01448696      0
## 9     1880   F       Bertha 1320 0.01352390      0
## 10    1880   F        Sarah 1288 0.01319605      0
## 11    1880   F        Annie 1258 0.01288868      0
## 12    1880   F        Clara 1226 0.01256083      0
## 13    1880   F         Ella 1156 0.01184366      0
```

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_if(data, condition)` allows you to conditionally create several new columns in one step:

```
mutate_if(mpg2, 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
```

Create New Variables

`mutate_if(data, condition)` allows you to conditionally create several new columns in one step:

```
mutate_if(mpg2, is.numeric, list(avg = mean, log_2 = log2))
## # A tibble: 234 x 7
##    model        hwy   cty hwy_avg cty_avg hwy_log_2 cty_log_2
##    <chr>      <int> <int>   <dbl>   <dbl>     <dbl>     <dbl>
##  1 a4            29    18    23.4    16.9      4.86      4.17
##  2 a4            29    21    23.4    16.9      4.86      4.39
##  3 a4            31    20    23.4    16.9      4.95      4.32
##  4 a4            30    21    23.4    16.9      4.91      4.39
##  5 a4            26    16    23.4    16.9      4.70      4
##  6 a4            26    18    23.4    16.9      4.70      4.17
##  7 a4            27    18    23.4    16.9      4.75      4.17
##  8 a4 quattro    26    18    23.4    16.9      4.70      4.17
##  9 a4 quattro    25    16    23.4    16.9      4.64      4
## 10 a4 quattro    28    20    23.4    16.9      4.81      4.32
## # ... with 224 more rows
```

Create New Variables

`mutate_if(data, condition)` allows you to conditionally create several new columns in one step:

```
mutate_if(mpg2, is.numeric, list(~mean(., na.rm = T), ~log2(.)))
## # A tibble: 234 x 7
##    model        hwy   cty hwy_mean cty_mean hwy_log2 cty_log2
##    <chr>      <int> <int>    <dbl>    <dbl>    <dbl>    <dbl>
##  1 a4            29    18     23.4     16.9     4.86     4.17
##  2 a4            29    21     23.4     16.9     4.86     4.39
##  3 a4            31    20     23.4     16.9     4.95     4.32
##  4 a4            30    21     23.4     16.9     4.91     4.39
##  5 a4            26    16     23.4     16.9     4.70     4
##  6 a4            26    18     23.4     16.9     4.70     4.17
##  7 a4            27    18     23.4     16.9     4.75     4.17
##  8 a4 quattro    26    18     23.4     16.9     4.70     4.17
##  9 a4 quattro    25    16     23.4     16.9     4.64     4
## 10 a4 quattro    28    20     23.4     16.9     4.81     4.32
## # ... with 224 more rows
```

Create New Variables

`mutate_at(data, condition)` allows you to conditionally transform several columns in one step:

```
mutate_at(mpg2, c("hwy", "cty"), mean, na.rm = TRUE)
## # A tibble: 234 x 3
##    model        hwy   cty
##    <chr>      <dbl> <dbl>
##  1 a4          23.4  16.9
##  2 a4          23.4  16.9
##  3 a4          23.4  16.9
##  4 a4          23.4  16.9
##  5 a4          23.4  16.9
##  6 a4          23.4  16.9
##  7 a4          23.4  16.9
##  8 a4 quattro  23.4  16.9
##  9 a4 quattro  23.4  16.9
## 10 a4 quattro  23.4  16.9
## # ... with 224 more rows
```

Create New Variables

`mutate_all(data, condition)` allows you to transform all columns in one step:

```
mutate_all(mpg2, mean, na.rm = TRUE)
## # A tibble: 234 x 3
##    model   hwy   cty
##    <dbl> <dbl> <dbl>
##  1    NA  23.4  16.9
##  2    NA  23.4  16.9
##  3    NA  23.4  16.9
##  4    NA  23.4  16.9
##  5    NA  23.4  16.9
##  6    NA  23.4  16.9
##  7    NA  23.4  16.9
##  8    NA  23.4  16.9
##  9    NA  23.4  16.9
## 10    NA  23.4  16.9
## # ... with 224 more rows
```

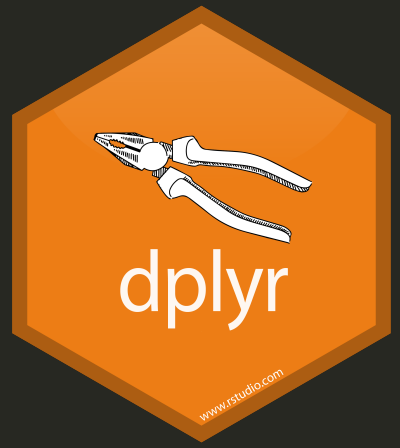
Create New Variables

`mutate_all(data, condition)` allows you to transform all columns in one step:

```
mutate_all(mpg2, list(~mean(., na.rm = T), ~nchar(.)))
## # A tibble: 234 x 9
##    model   hwy   cty model_mean hwy_mean cty_mean model_nchar hwy_nchar
##    <chr> <int> <int>      <dbl>    <dbl>    <dbl>       <int>     <int>
##  1 a4       29    18         NA     23.4     16.9           2         2
##  2 a4       29    21         NA     23.4     16.9           2         2
##  3 a4       31    20         NA     23.4     16.9           2         2
##  4 a4       30    21         NA     23.4     16.9           2         2
##  5 a4       26    16         NA     23.4     16.9           2         2
##  6 a4       26    18         NA     23.4     16.9           2         2
##  7 a4       27    18         NA     23.4     16.9           2         2
##  8 a4 q~    26    18         NA     23.4     16.9          10         2
##  9 a4 q~    25    16         NA     23.4     16.9          10         2
## 10 a4 q~    28    20         NA     23.4     16.9          10         2
## # ... with 224 more rows, and 1 more variable: cty_nchar <int>
```

summarize()

Sum Up Variables



Sum Up Variables

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

```
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 create new variables (columns) based or not based on other variables:

```
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
```

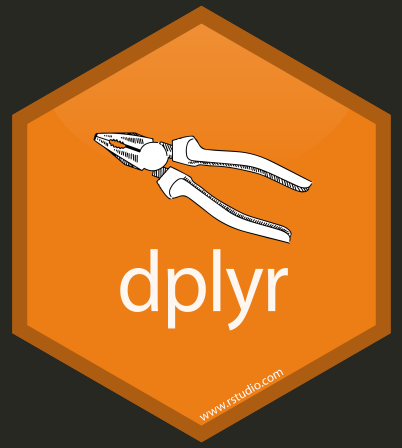
Sum Up Variables

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

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

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
```

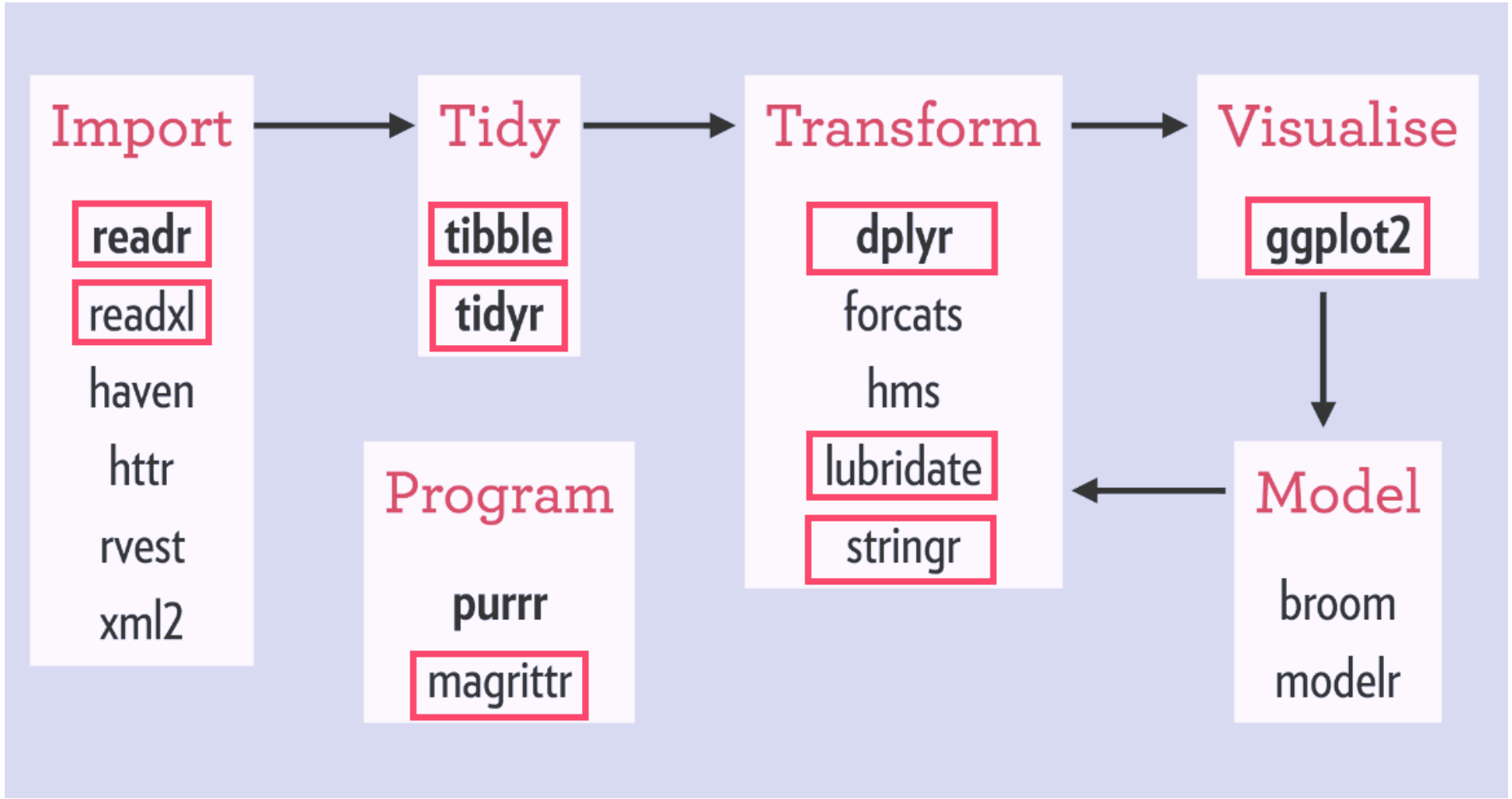
`group_by()` SUPERPOWER!

`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 = "EUR2.80",
    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)

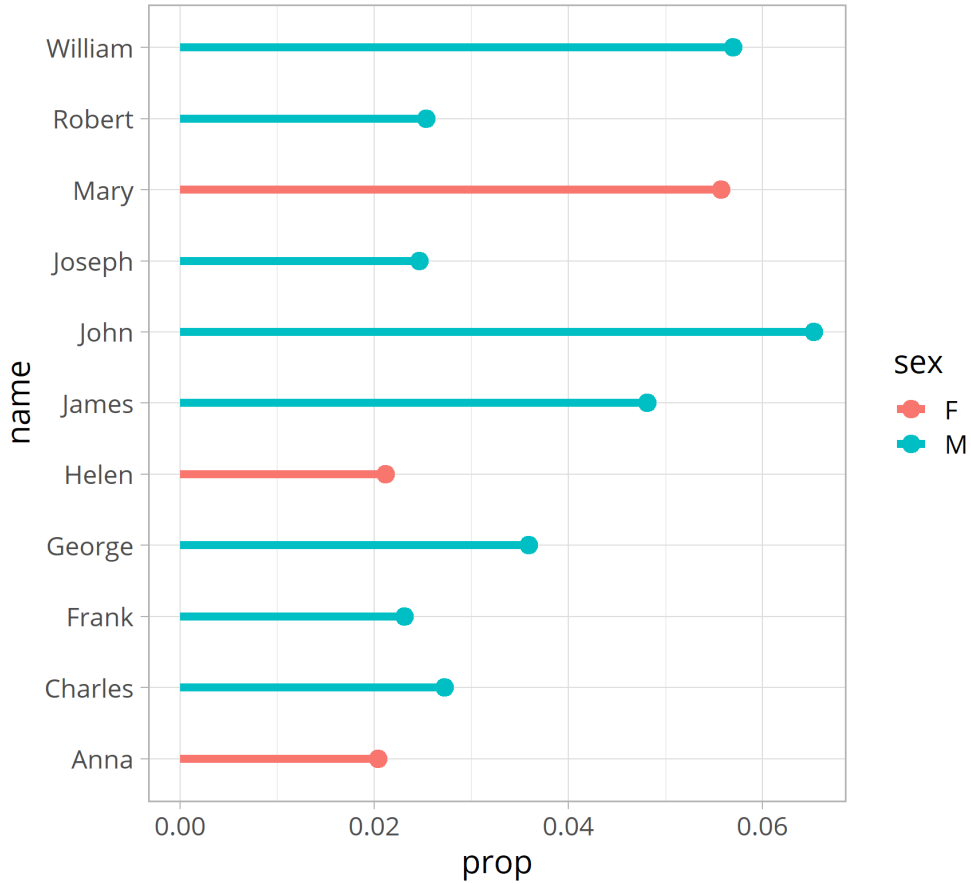
You can easily inactivate verbs or single conditions:

```
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 `%>%`):

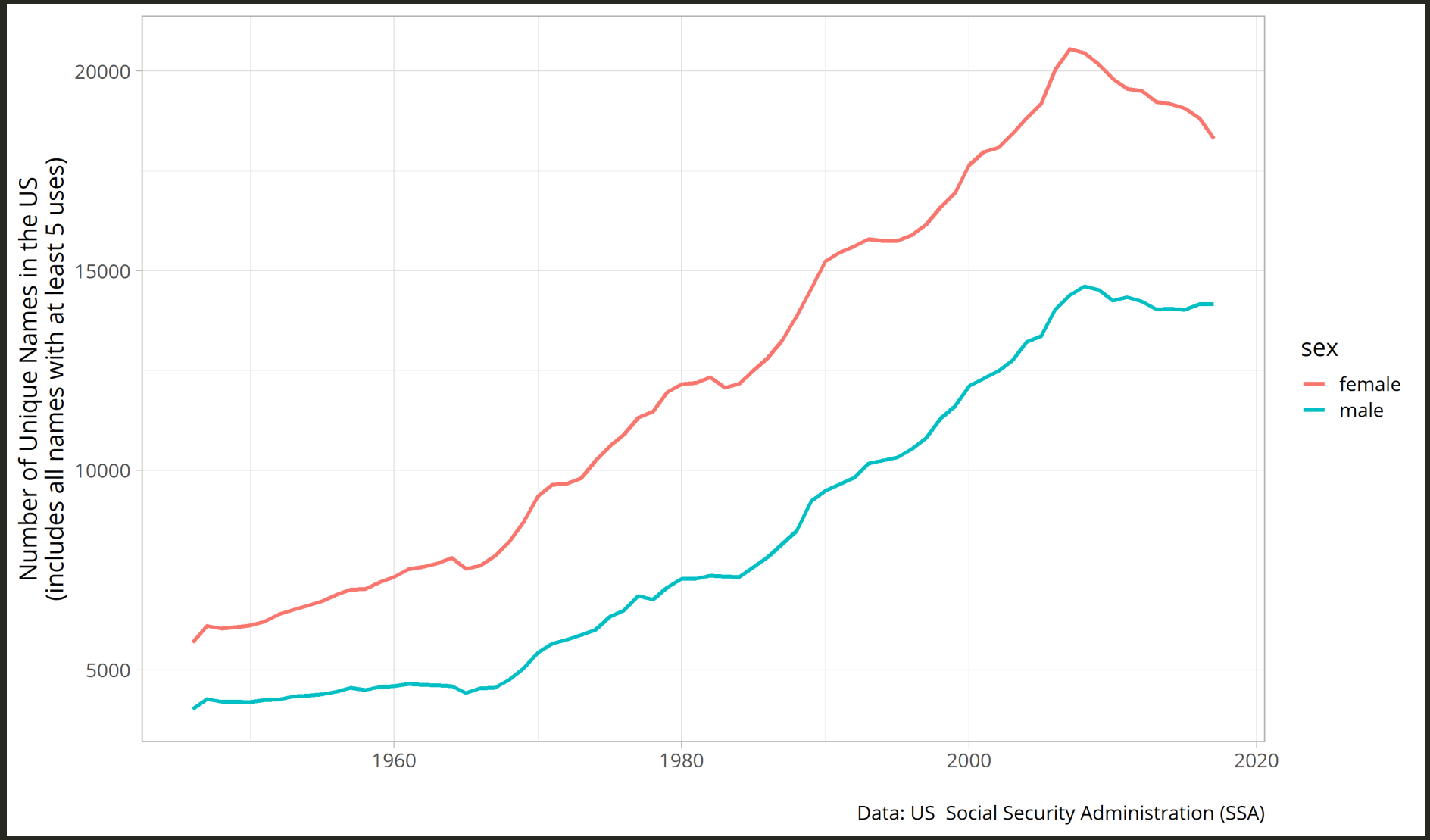
```
babynames %>%
  filter(year == 1900) %>%
  group_by(sex) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, name) %>%
  summarize(
    prop = sum(n) / unique(sum)
  ) %>%
  filter(prop >= .02) %>%
  ggplot(aes(prop, name,
             color = sex)) +
    geom_segment(
      aes(xend = 0, yend = name),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



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 table that contains the average delays 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.

Your Turn: Create this Line Plot!



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)
## [1] 38
```

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 audi         a4 quatt~   2    2008     4 auto(s~ 4        19    27 p     comp~
## 2 ford         f150 pic~   5.4  2008     8 auto(l~ 4        13    17 r     pick~
## 3 ford         f150 pic~   4.2  1999     6 auto(l~ 4        14    17 r     pick~
## 4 nissan       altima      2.4  1999     4 manual~ f        21    29 r     comp~
## 5 ford         f150 pic~   5.4  1999     8 auto(l~ 4        11    15 r     pick~
```

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         1.8  1999     4 manua~ f        21    29 p     compa~
##  2 chevrolet    k1500 t~   5.7  1999     8 auto(~ 4        11    15 r     suv
##  3 dodge        dakota ~   5.2  1999     8 manua~ 4        11    17 r     pickup
##  4 ford         mustang    3.8  1999     6 auto(~ r        18    25 r     subco~
##  5 honda        civic      1.6  1999     4 manua~ f        25    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~   4    1999     6 auto(~ 4        14    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         mustang    5.4  2008     8 manua~ r        14    20 p     subco~
##  2 audi         a4 quat~   1.8  1999     4 auto(~ 4        16    25 p     compa~
##  3 ford         explore~   4    1999     6 auto(~ 4        14    17 r     suv
##  4 dodge        dakota ~   3.9  1999     6 auto(~ 4        13    17 r     pickup
##  5 hyundai      sonata     2.4  1999     4 manua~ f        18    27 r     midsi~
##  6 volkswagen   jetta      2    2008     4 auto(~ f        22    29 p     compa~
##  7 mercury      mountai~   5    1999     8 auto(~ 4        13    17 r     suv
##  8 chevrolet    corvette   5.7  1999     8 manua~ r        16    26 p     2seat~
##  9 toyota       4runner~   4.7  2008     8 auto(~ 4        14    17 r     suv
## 10 hyundai      sonata     2.4  2008     4 manua~ f        21    31 r     midsi~
## 11 volkswagen   jetta      2.8  1999     6 manua~ f        17    24 r     compa~
## 12 nissan       altima     2.4  1999     4 auto(~ f        19    27 r     compa~
```

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

```
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

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

```
mpg %>% group_by(manufacturer) %>% count(sort = TRUE)
## # A tibble: 15 x 2
## # Groups:   manufacturer [15]
##    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()` 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
```

```
summarize(mpg2, n = n())
## # A tibble: 1 x 1
##       n
##   <int>
## 1   234
```

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:

```
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
```

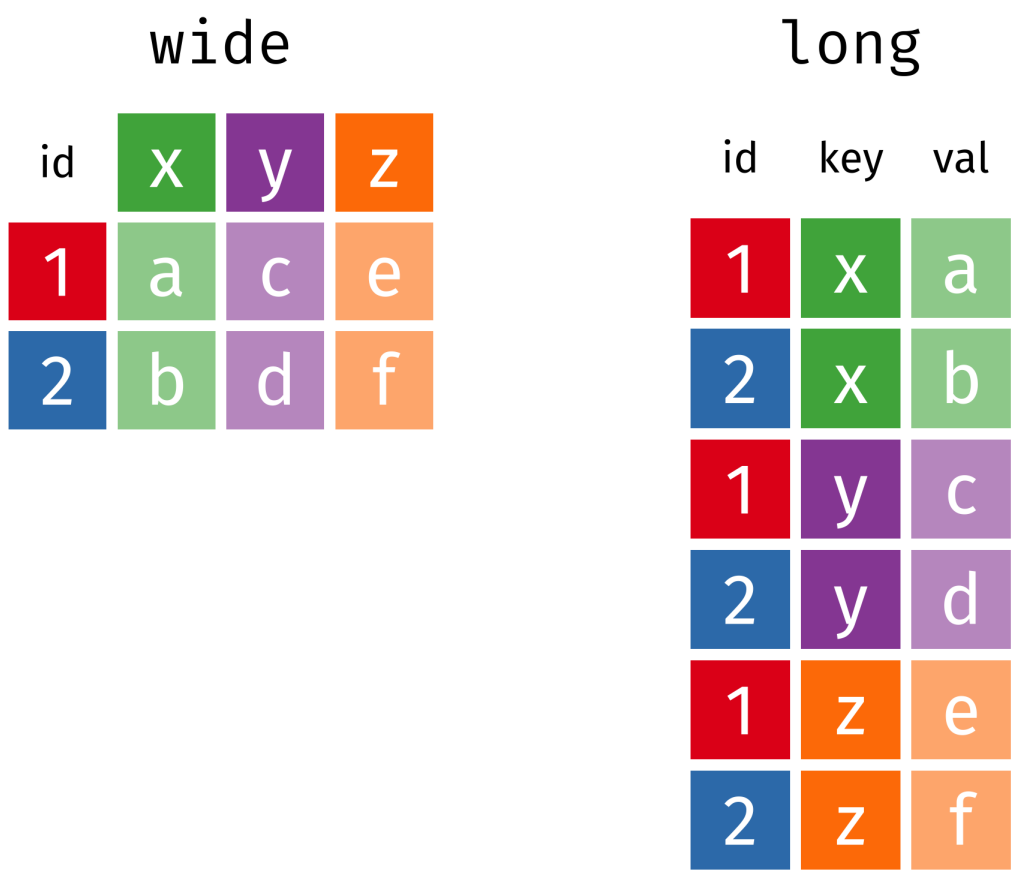
The `join()` Family

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 = T)) %>%
  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:

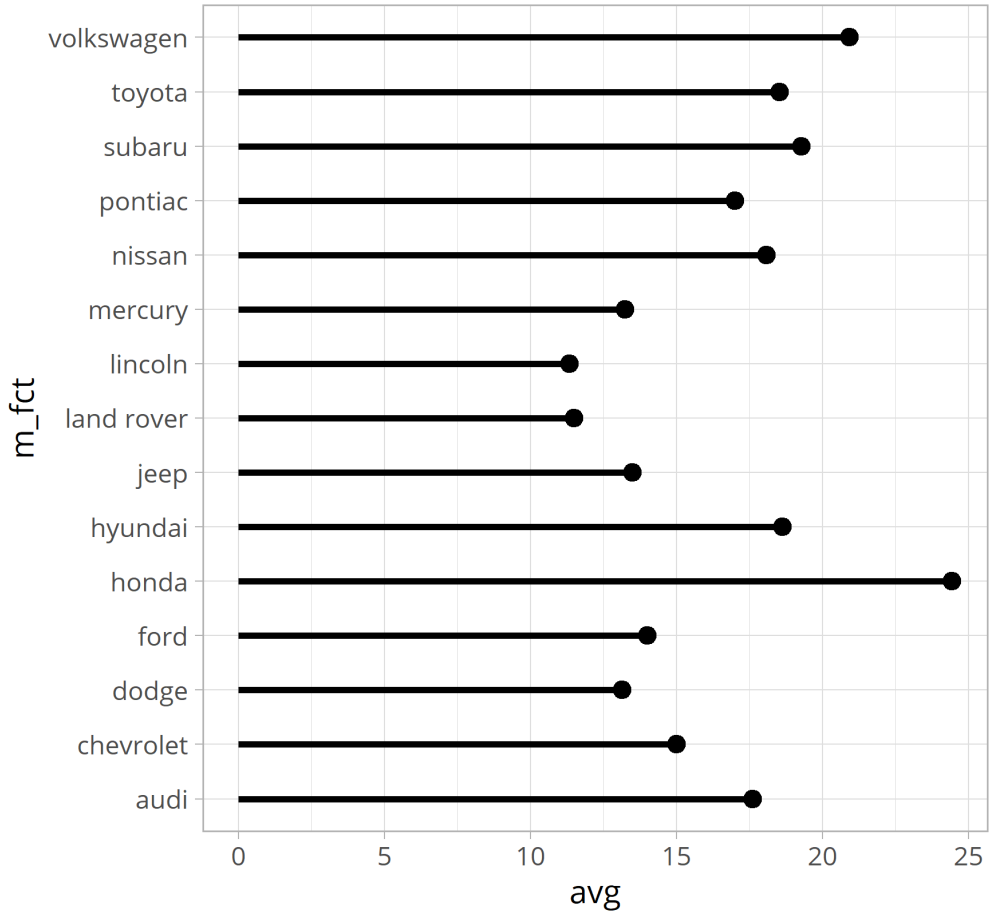
```
mpg_wide %>%
  pivot_longer(
    cols = starts_with("cty_"),
    names_to = "year",
    values_to = "cty",
    names_prefix = "cty_"
  )
## # A tibble: 314 x 6
## # Groups:   manufacturer, model, trans, cyl [157]
##    manufacturer model trans        cyl year    cty
##    <chr>        <chr> <chr>      <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:

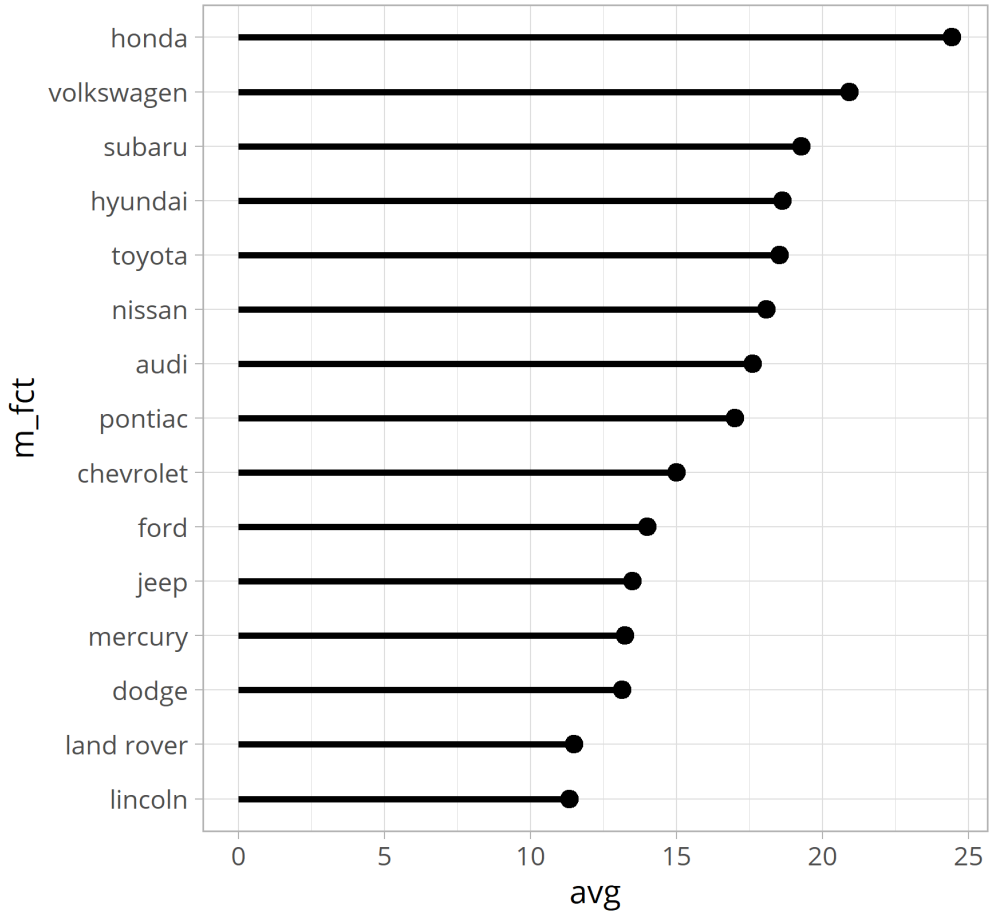
```
mpg_avg <-
  mpg %>%
  mutate(
    m_fct = factor(manufacturer)
  ) %>%
  filter(!is.na(cty)) %>%
  group_by(m_fct) %>%
  summarize(avg = mean(cty)) %>%
  ungroup()

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:

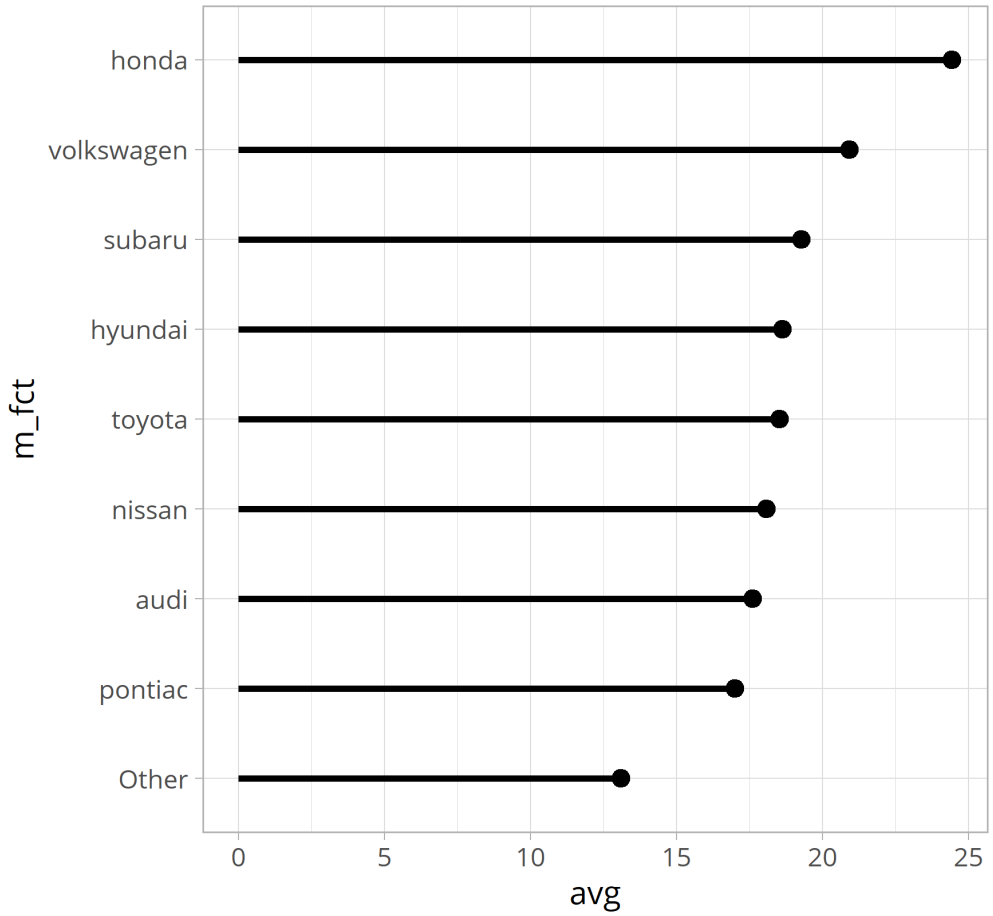
```
mpg_avg %>%
  mutate(
    m_fct = fct_reorder(m_fct, avg)
  ) %>%
  ggplot(aes(avg, m_fct)) +
    geom_segment(
      aes(xend = 0, yend = m_fct),
      size = 1.5
    ) +
    geom_point(size = 4)
```



Functions from the `forcats` Package

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

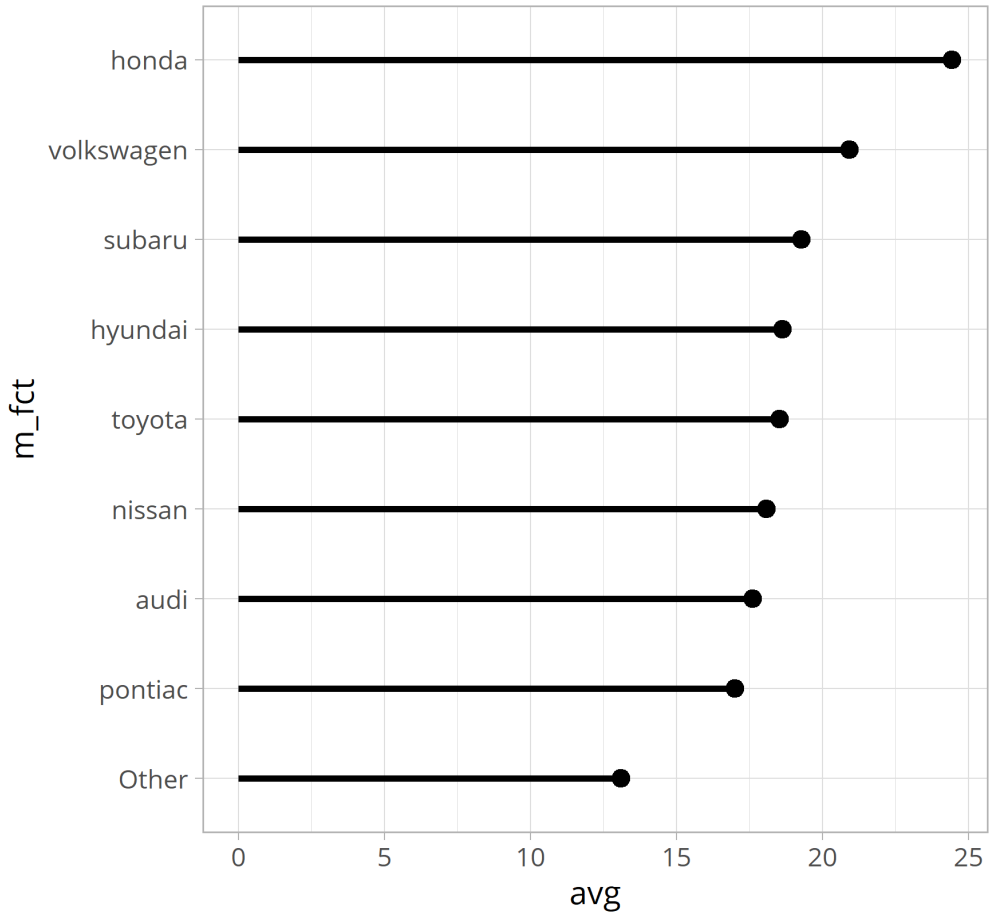
```
mpg_avg %>%
  mutate(m_fct = fct_lump(
    m_fct,
    n = 8,
    w = avg
  )) %>%
  group_by(m_fct) %>%
  summarize(avg = mean(avg)) %>%
  ungroup() %>%
  mutate(m_fct = fct_reorder(m_fct, avg)) 
  ggplot(aes(avg, m_fct)) +
    geom_segment(
      aes(xend = 0, yend = m_fct),
      size = 1.5
  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



Functions from the `forcats` Package

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

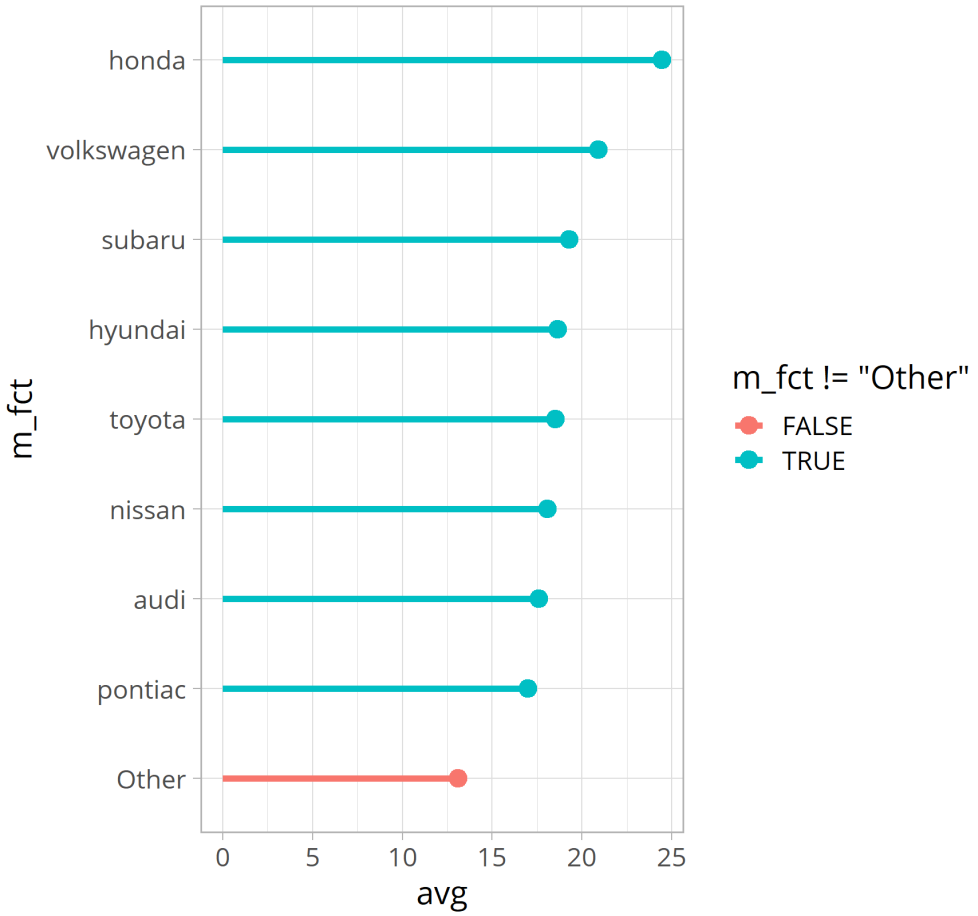
```
mpg_avg %>%
  mutate(m_fct = fct_lump(
    m_fct,
    n = 8,
    w = avg
  )) %>%
  group_by(m_fct) %>%
  summarize(avg = mean(avg)) %>%
  ungroup() %>%
  mutate(m_fct = fct_reorder(m_fct, avg))
  ggplot(aes(avg, m_fct)) +
    geom_segment(
      aes(xend = 0, yend = m_fct),
      size = 1.5
   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

```
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
```

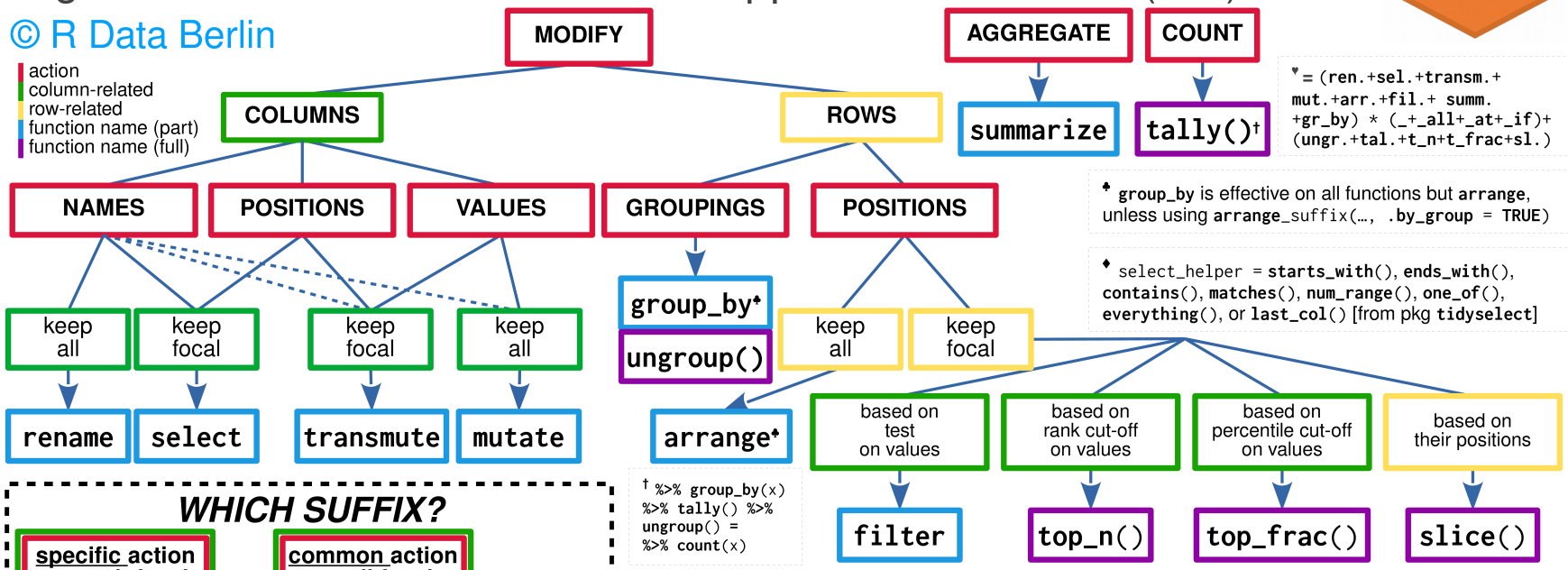
glue in combination with stringr

`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::`

A guide to 37♥ different behaviours applied to one tibble (`tbl`)



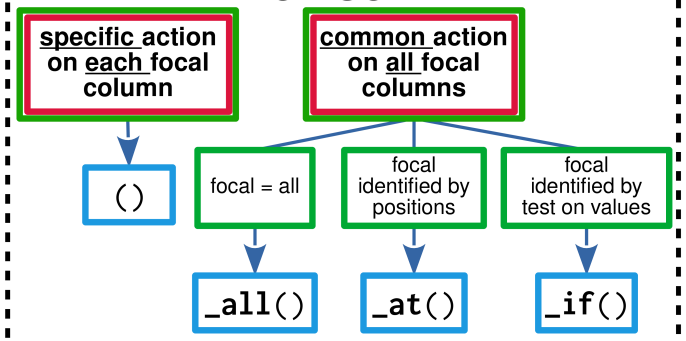
♥ = (ren.+sel.+transm.+ mut.+arr.+fil.+ summ. +gr_by) * (_+_all+_at+_if)+ (ungr.+tal.+t_n+t_frac+sl.)

♠ `group_by` is effective on all functions but `arrange`, unless using `arrange_suffix(…, .by_group = TRUE)`

♦ `select_helper` = `starts_with()`, `ends_with()`, `contains()`, `matches()`, `num_range()`, `one_of()`, `everything()`, or `last_col()` [from pkg `tidyselect`]

† `%>% group_by(x) %>% tally() %>% ungroup() = %>% count(x)`

WHICH SUFFIX?



♠ if the function `fn` does not return a scalar or a `nrow`-long vector, use `list(fn())` to nest the output ♠♠ for multiple columns dispatch use `list(suffix_for_column_names = fn)` or in case of ♠+♠♠ `list(suffix_for_column_names = ~ list(fn(.x, fn_args))`

SYNTAX: `tbl %>% verb_suffix(., …, .args)`

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

```
tbl %>% rename(new_name_columnX = old_name_columnX)
tbl %>% select(name_columnX, new_name_columnY = old_name_columnY, select_helper♦("text"))
tbl %>% transmute(name_columnZ = fn_scalar_or_vectorized♠(name_columnX))
tbl %>% mutate(name_columnZ = fn_scalar_or_vectorized♠(name_columnX))
tlb %>% tally()†; tlb %>% tally(name_numeric_column, wt = weights)†
tbl %>% arrange(name_columnX, desc(name_columnY))
tbl %>% filter(fn_test_vectorized(name_columnX), fn_test_vectorized(name_columnY))
tbl %>% top_n(n = number, name_columnX); tbl %>% top_frac(n = fraction, name_columnX)
tbl %>% slice(row_indices)
tbl %>% summarize(name_columnZ = fn_scalar♠(name_columnX))
tbl %>% group_by(name_columnX, name_columnY) %>% verb_suffix() & tbl %>% ungroup()
tbl %>% verb_all(.funs = fn♠♠, fn_arg_name1 = fn_arg1,fn_arg_name2 = fn_arg2) # included below
tbl %>% verb_at(.vars = indices_for_focal_columns OR .vars = vars(select_helper♦("text")), )
tbl %>% verb_if(.predicate = logicals_for_each_column, )
```