Data Visualization in R with ggplot2

Data Transformation with dplyr

Cédric Scherer

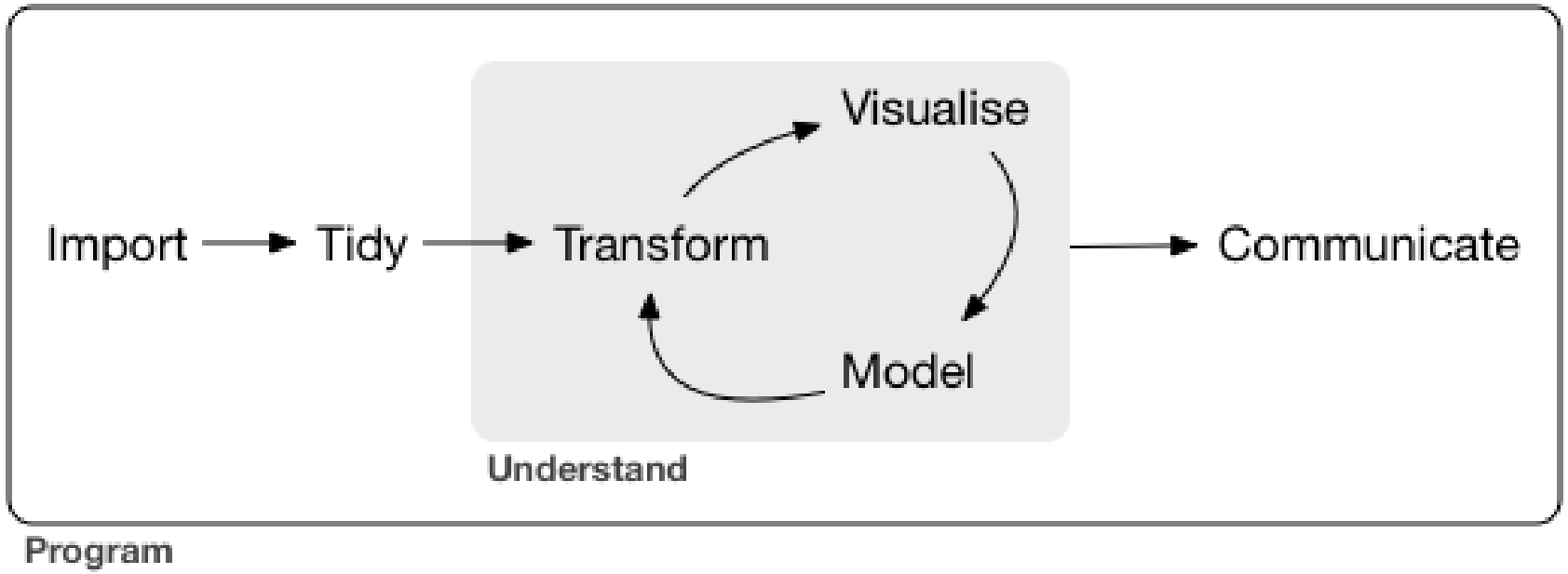
Physalia Courses | March 2-6 2020

Photo by Richard Strozynski

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

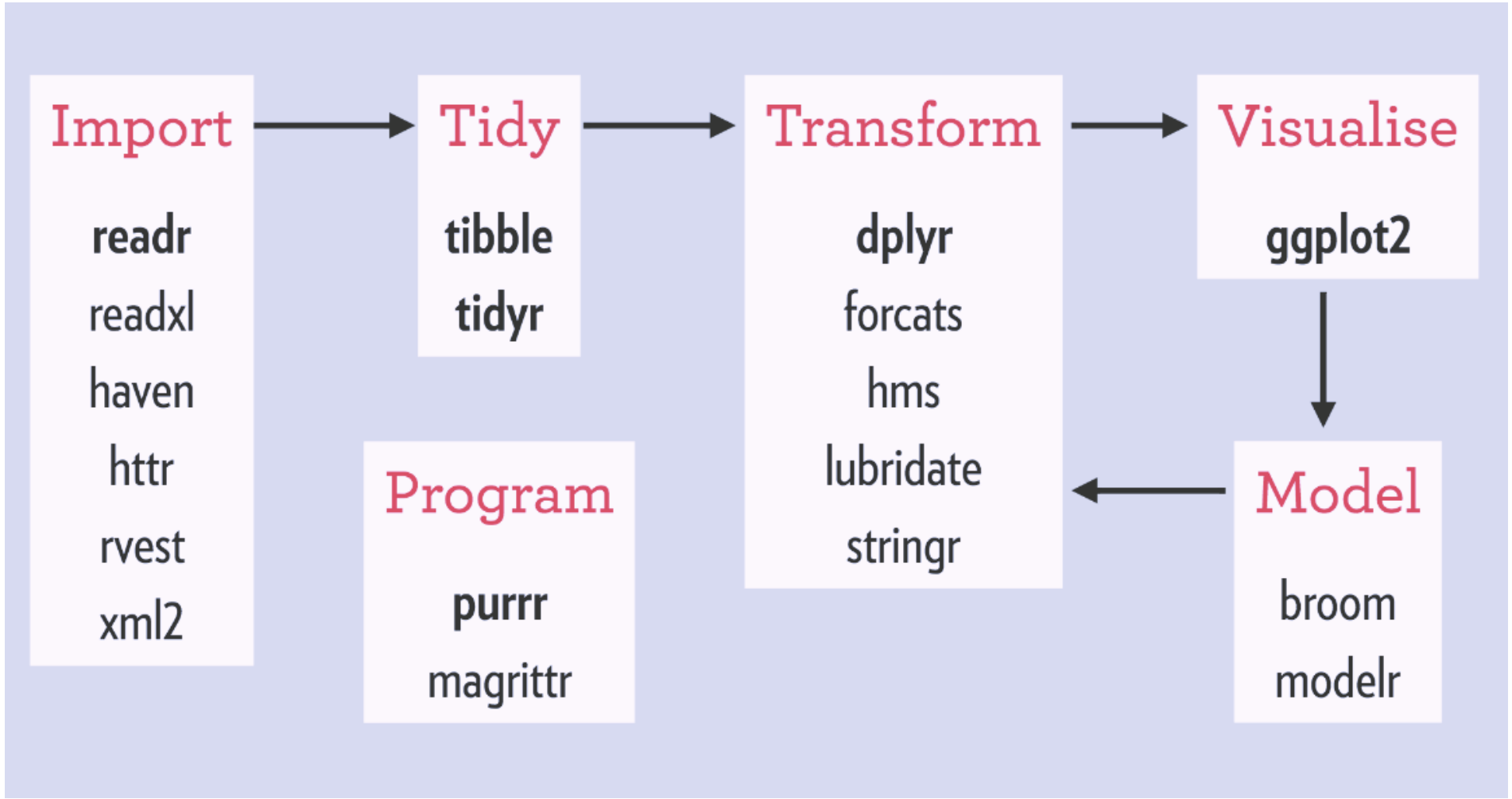
The `tidyverse`

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



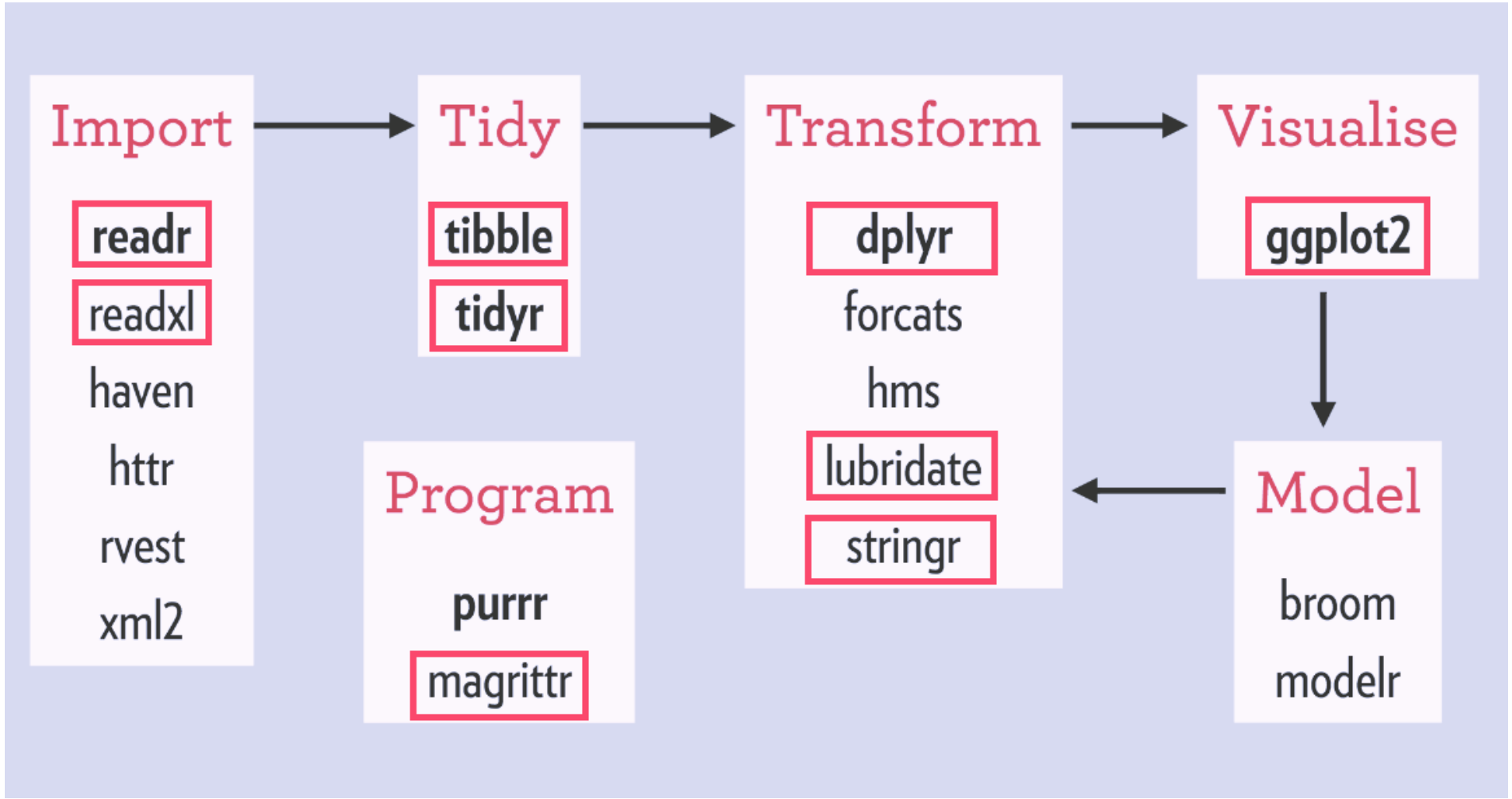
Source: Hadley Wickham's "R for Data Science" (R4DS)

The Typical Data Science Project Flow



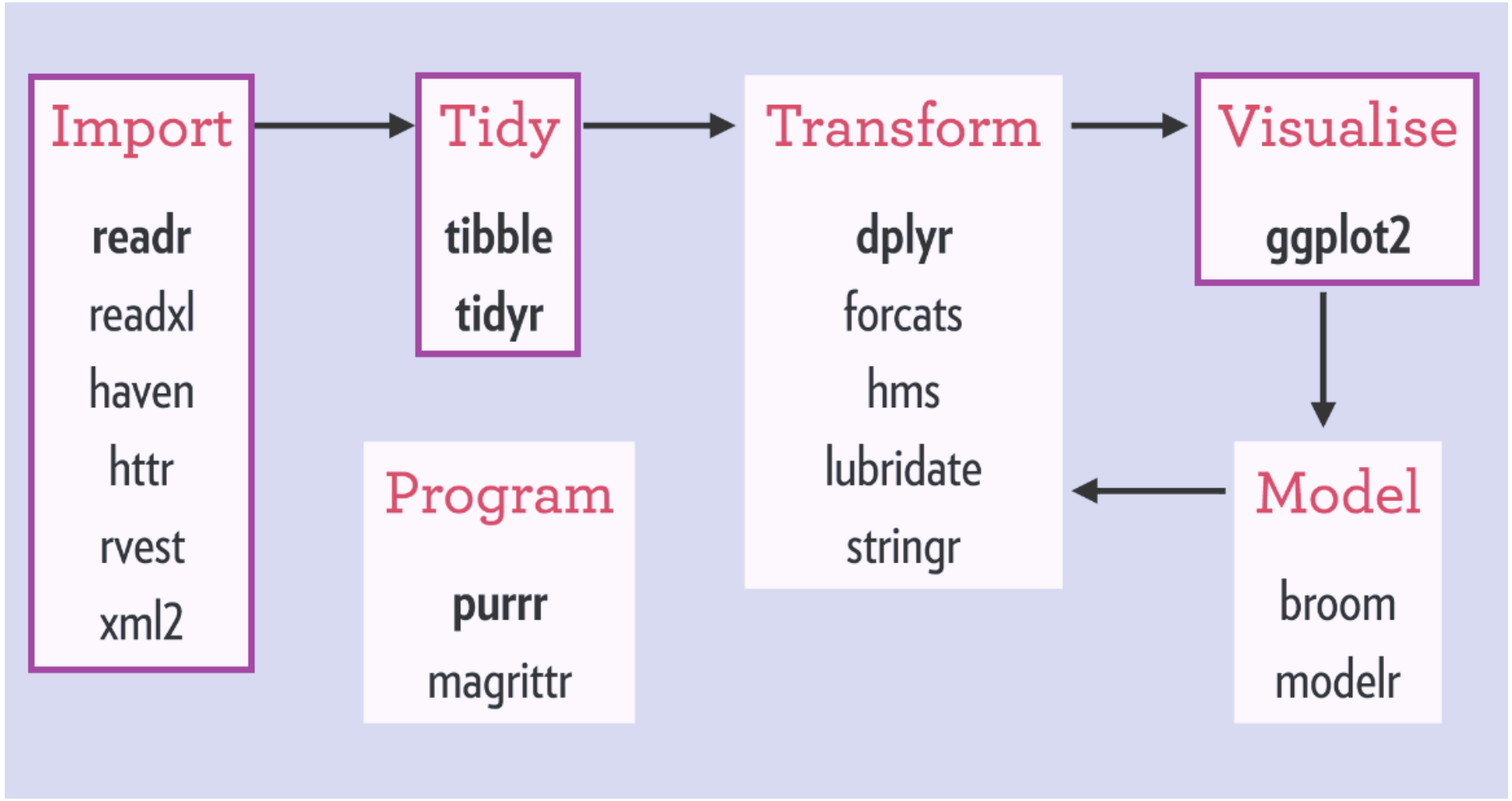
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

The Typical Data Science Project Flow



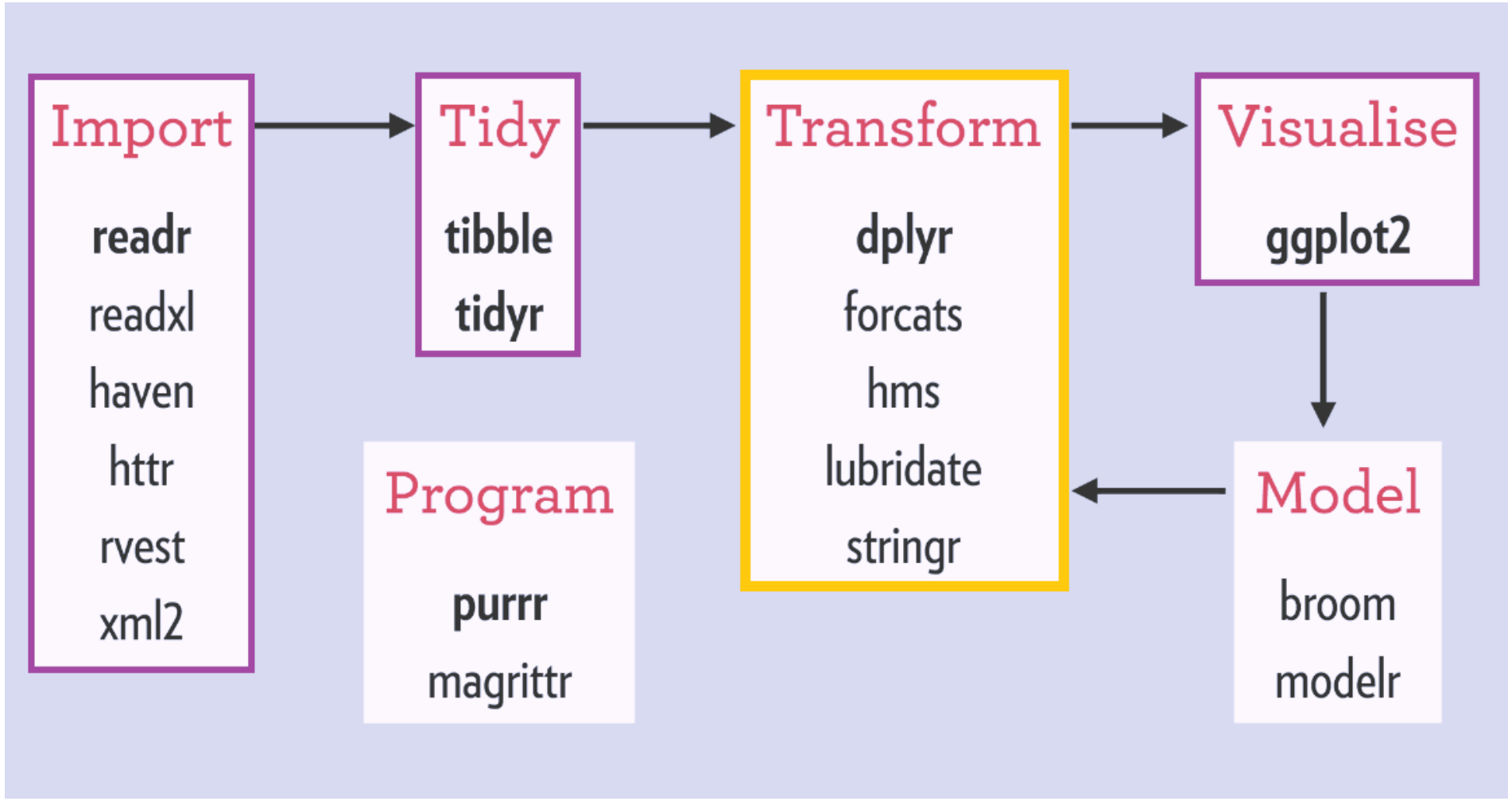
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

The Typical Data Science Project Flow



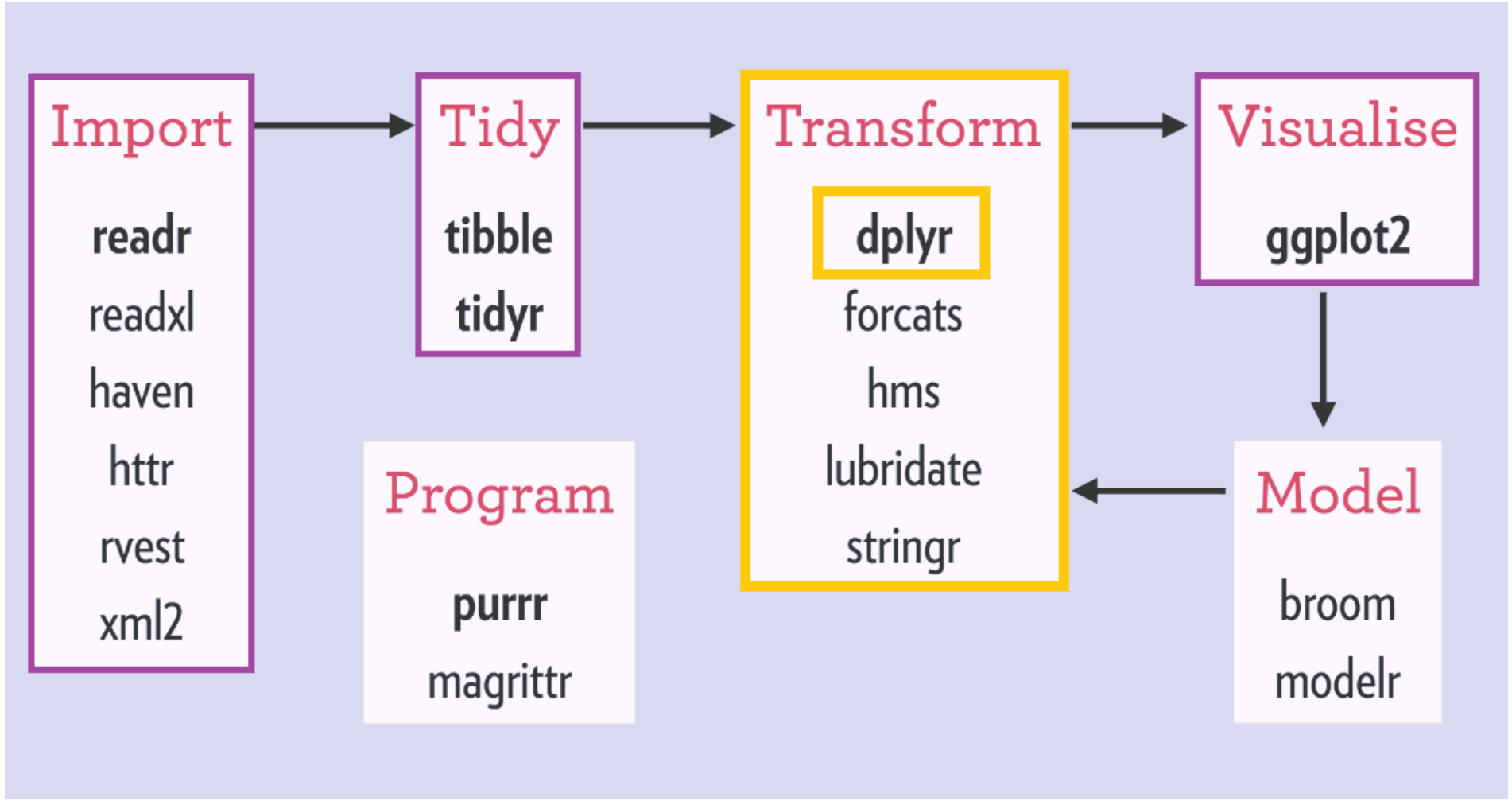
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

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

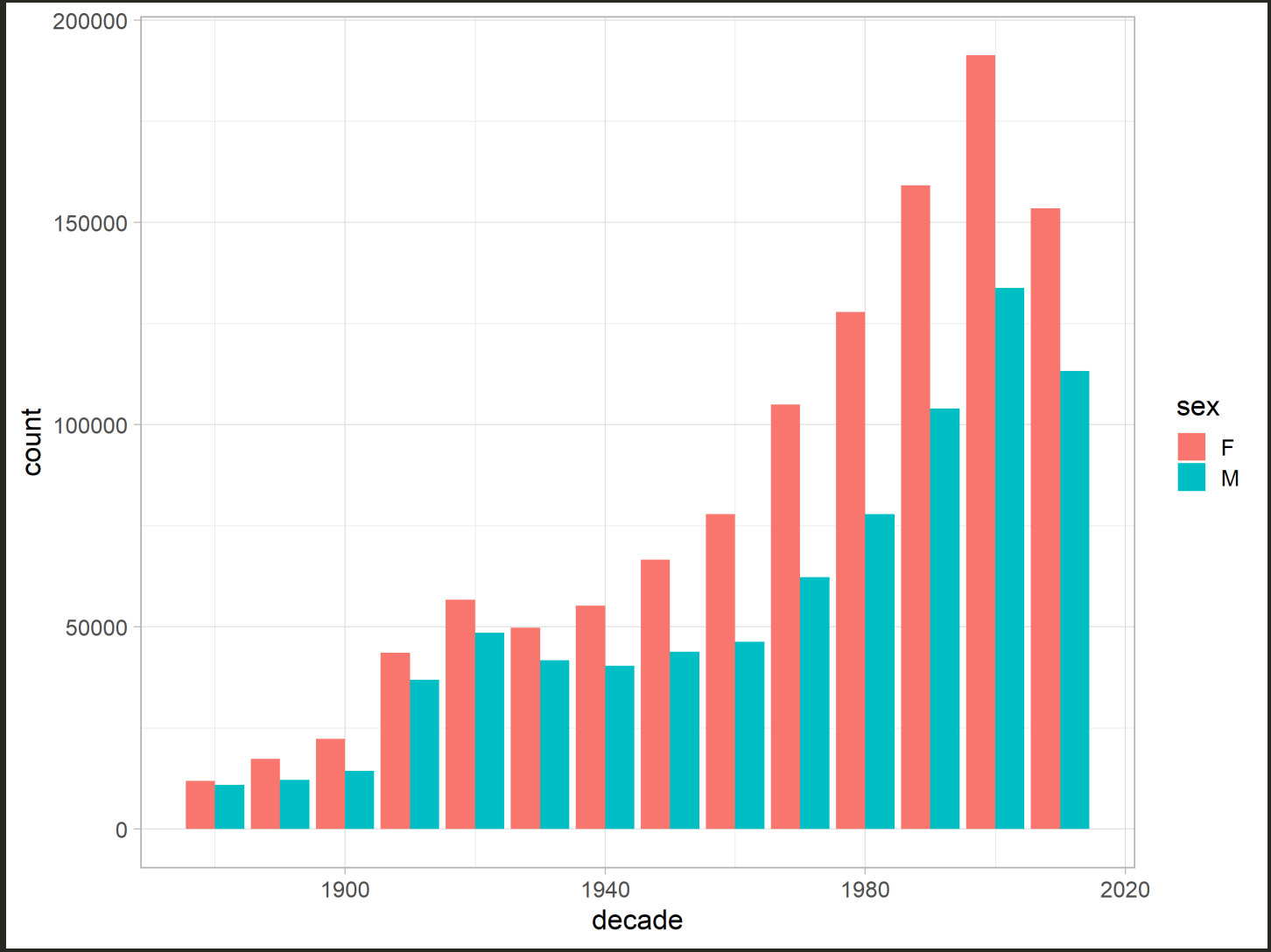
- Count and proportional data of baby names used for at least five children in the US per sex and year from 1880 to 2017
- 1,924,665 rows (observations) and 4 columns (variables)
- Data source: **US Social Security Administration (SSA)**
- Available as data set `babynames` in the package `babynames`: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

Our Example Data: babynames

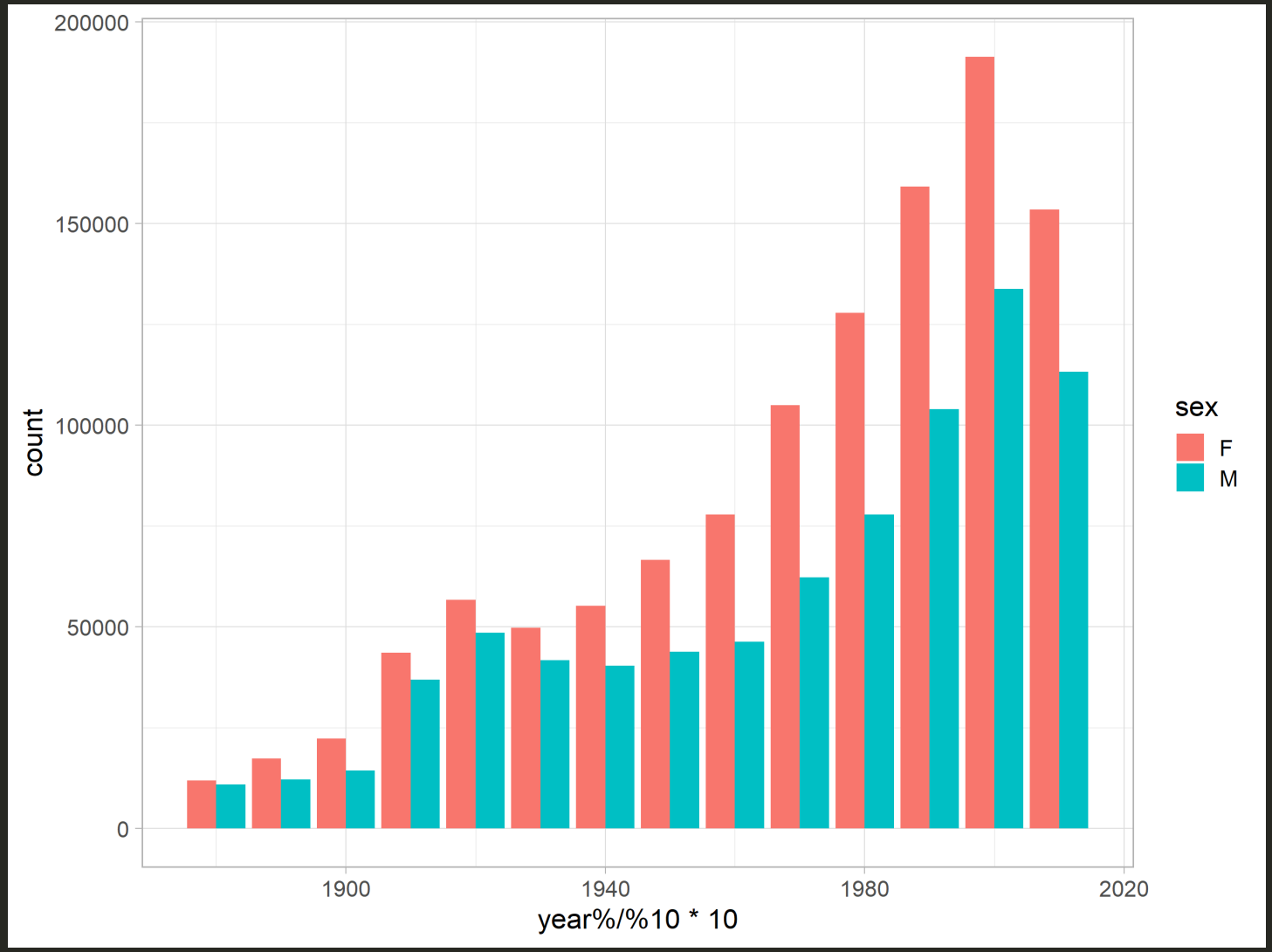
```
library(babynames)

glimpse(babynames)
## 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
```

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

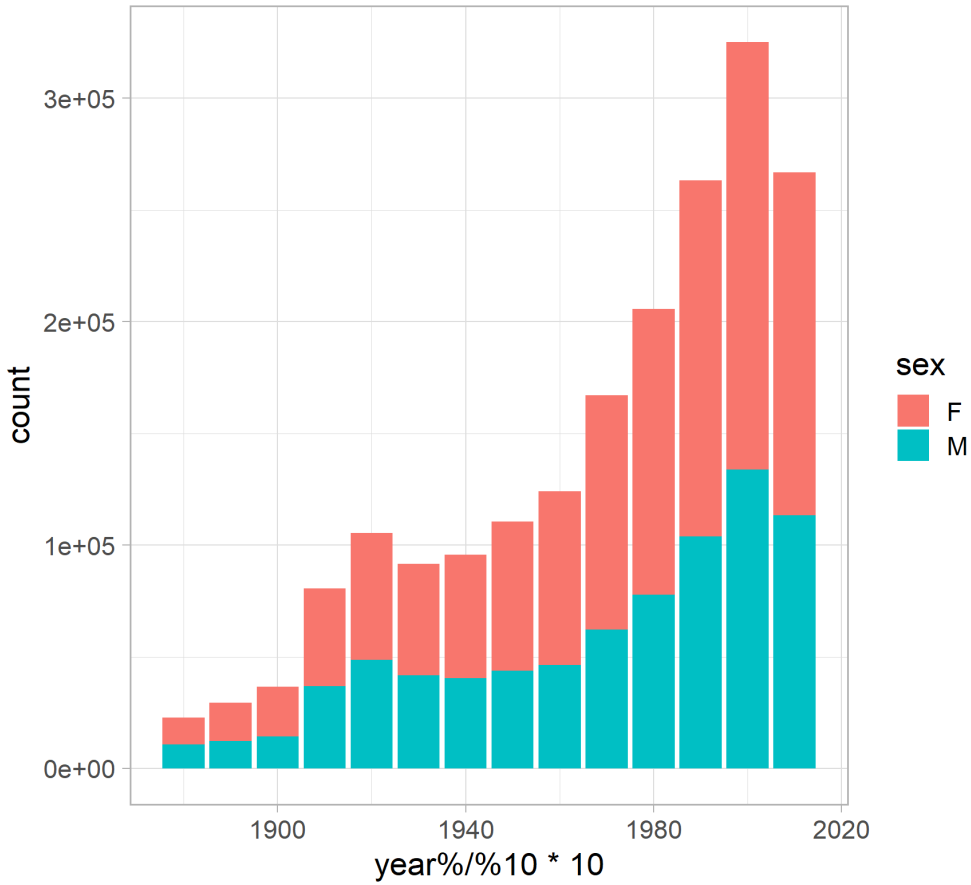
```
1 %/% 10
## [1] 0
2 %/% 10
## [1] 0
11 %/% 10
## [1] 1
12 %/% 10
## [1] 1

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

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

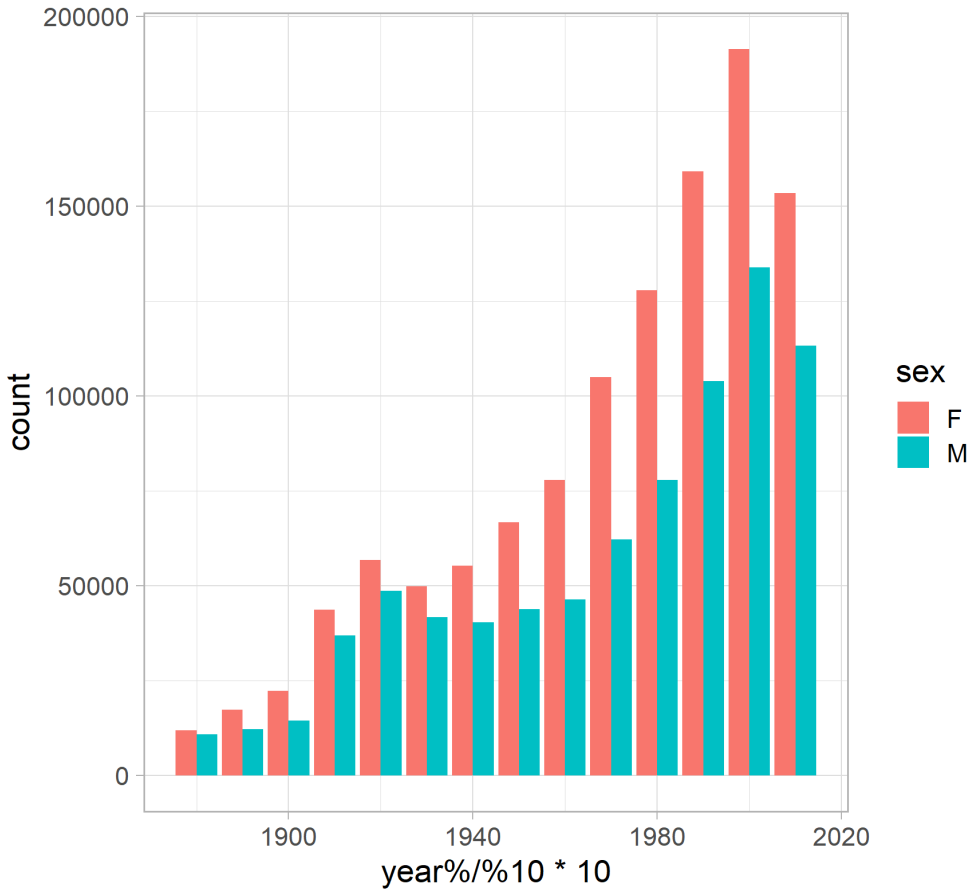
Your Turn: Dodged Bar Plots

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



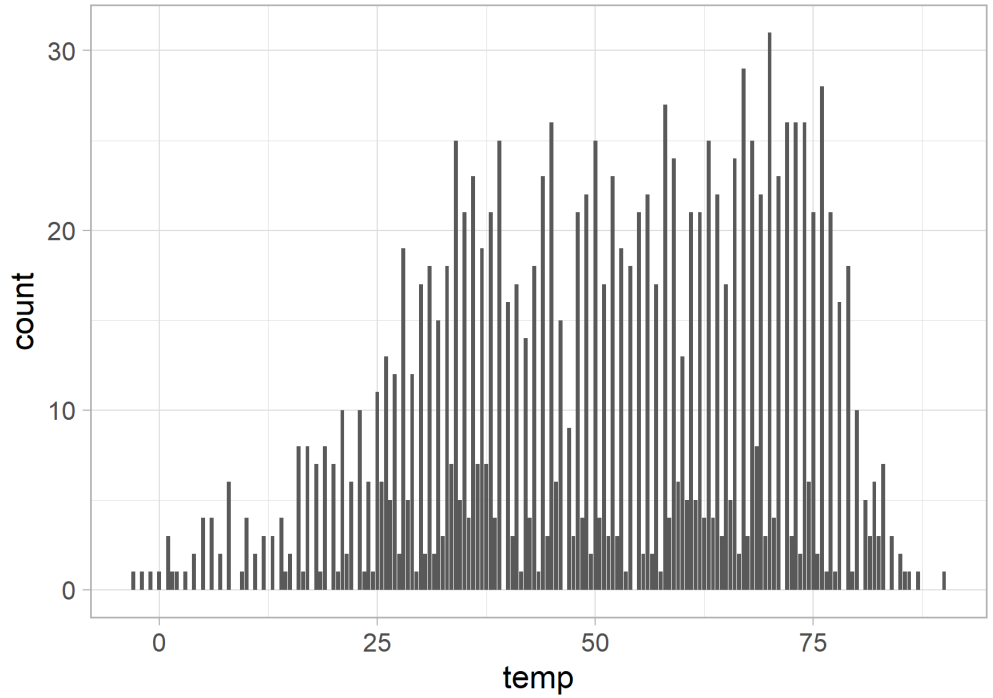
Your Turn: Dodged Bar Plots

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

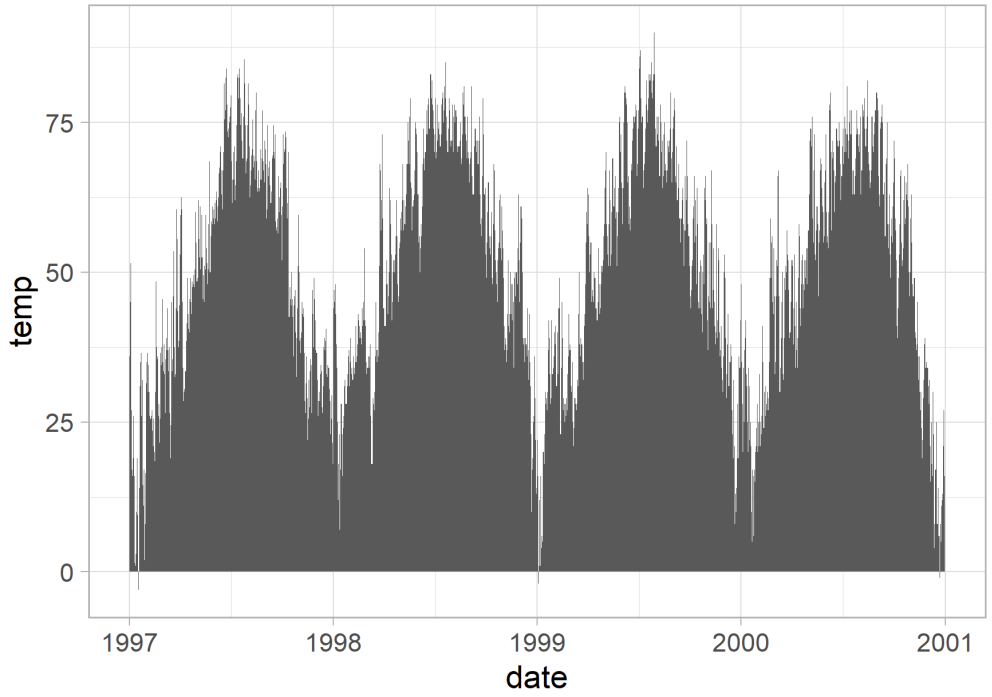


geom_bar()

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

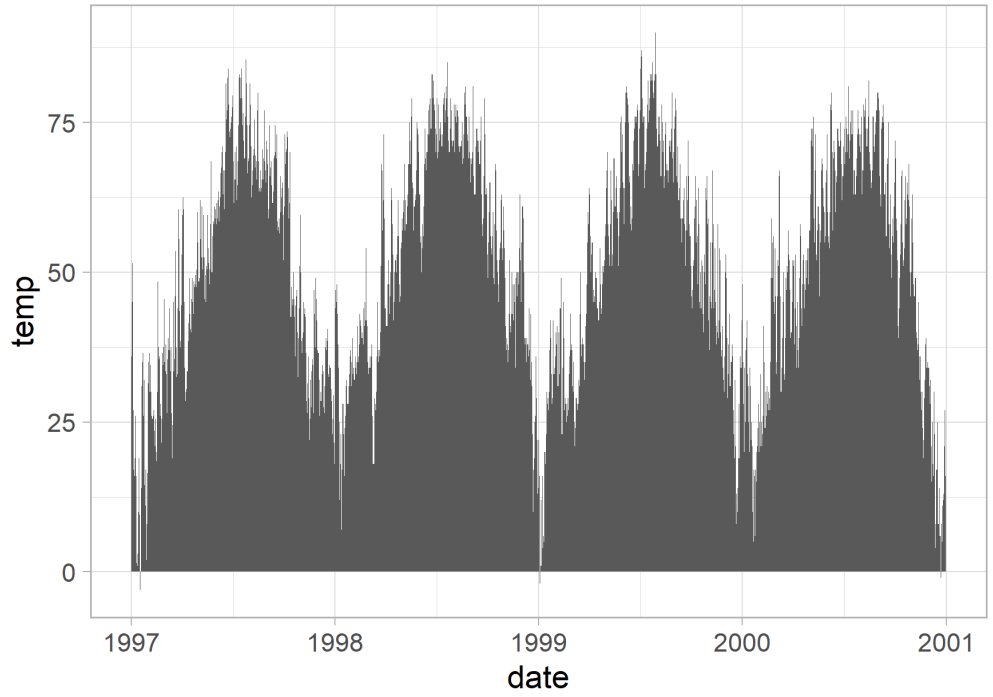



```
ggplot(chic, aes(date, temp)) +
  geom_bar(stat = "identity")
```

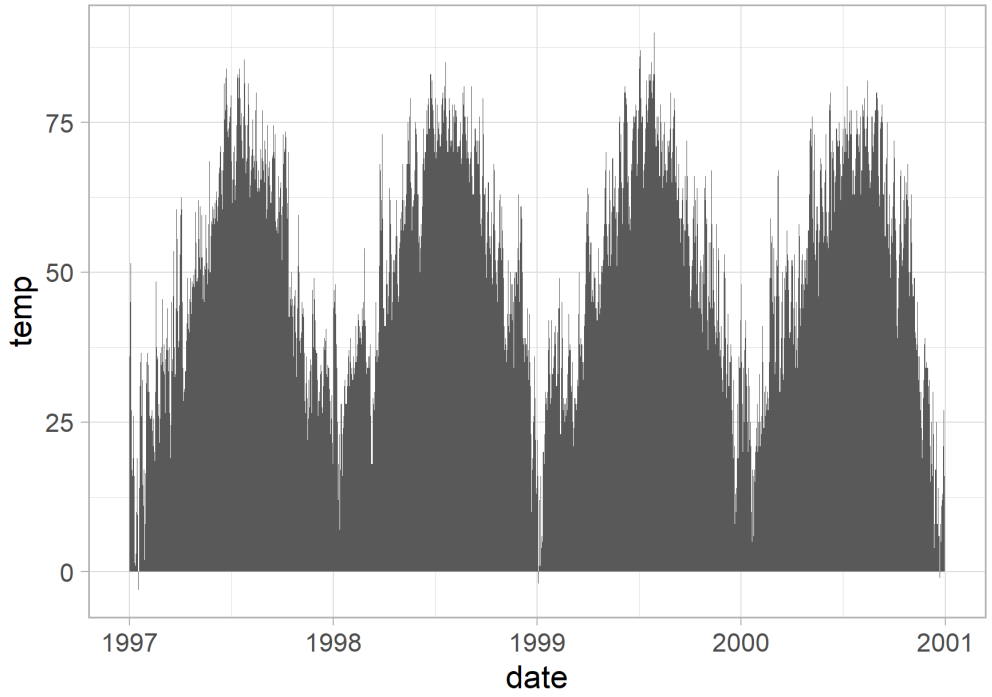


`geom_bar()` versus `geom_col()`

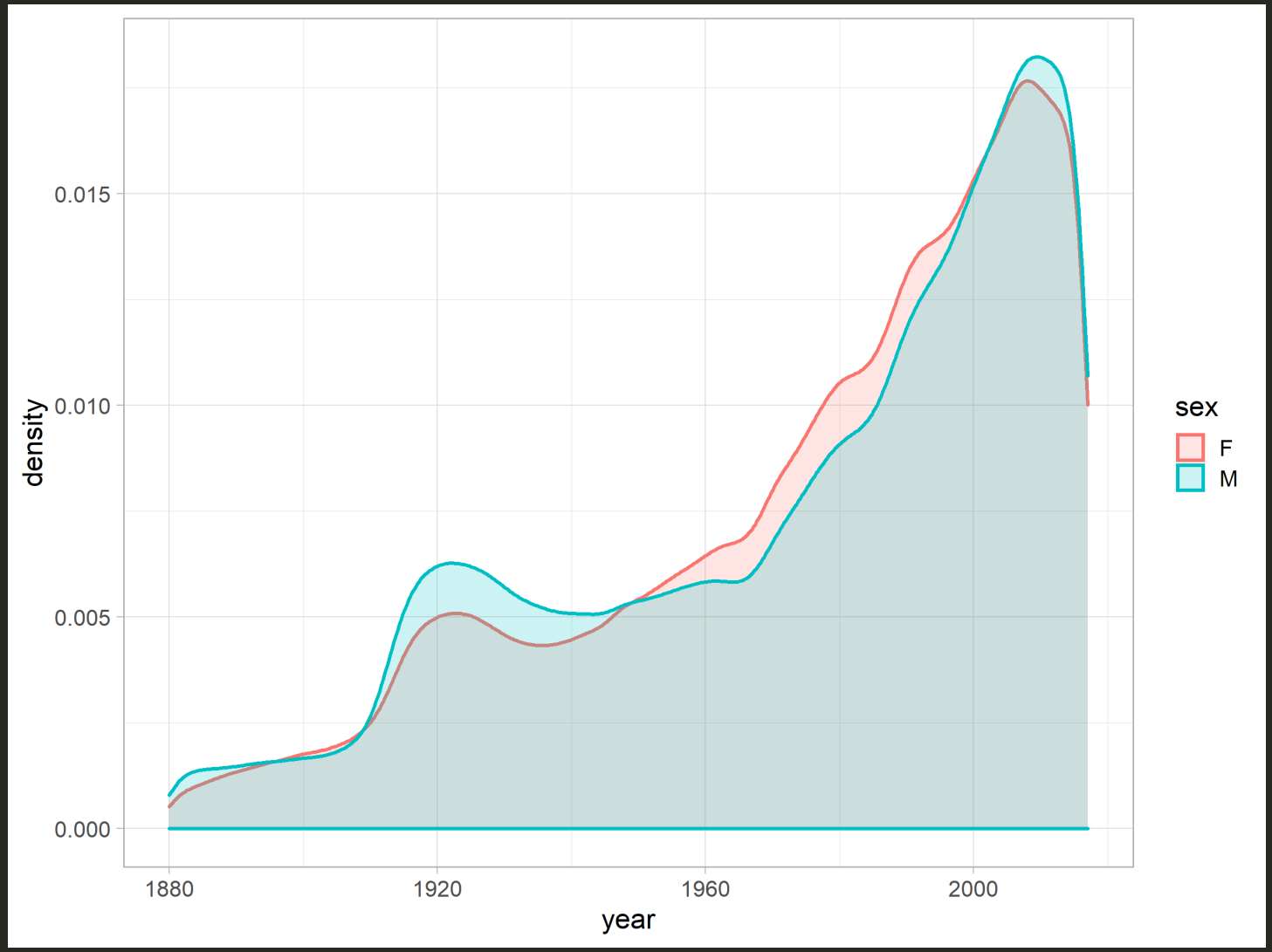
```
ggplot(chic, aes(date, temp)) +
  geom_col()
```




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

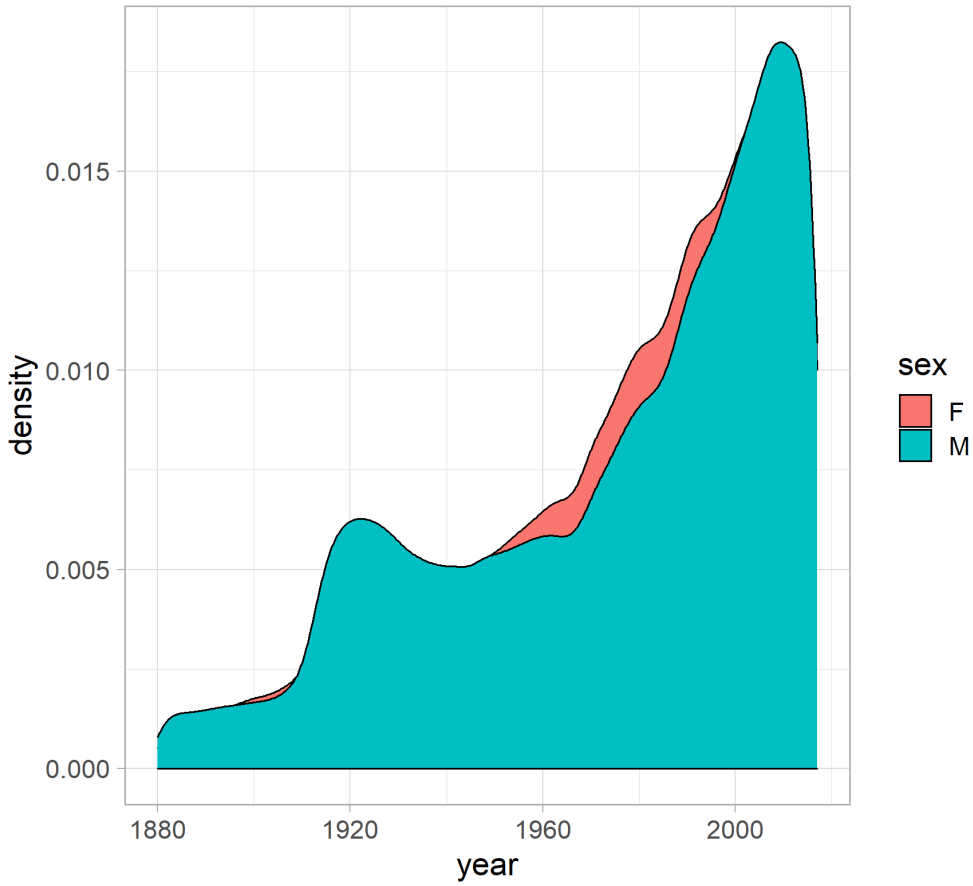


Your Turn!



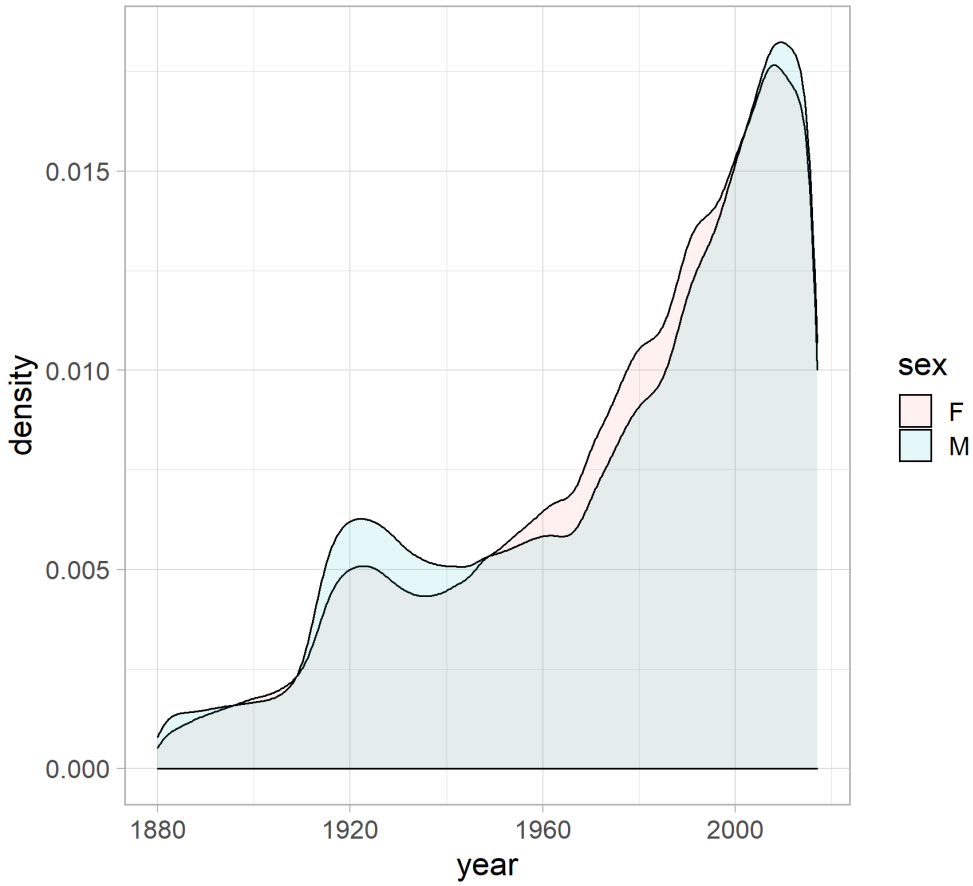
Your Turn: Density Plots

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



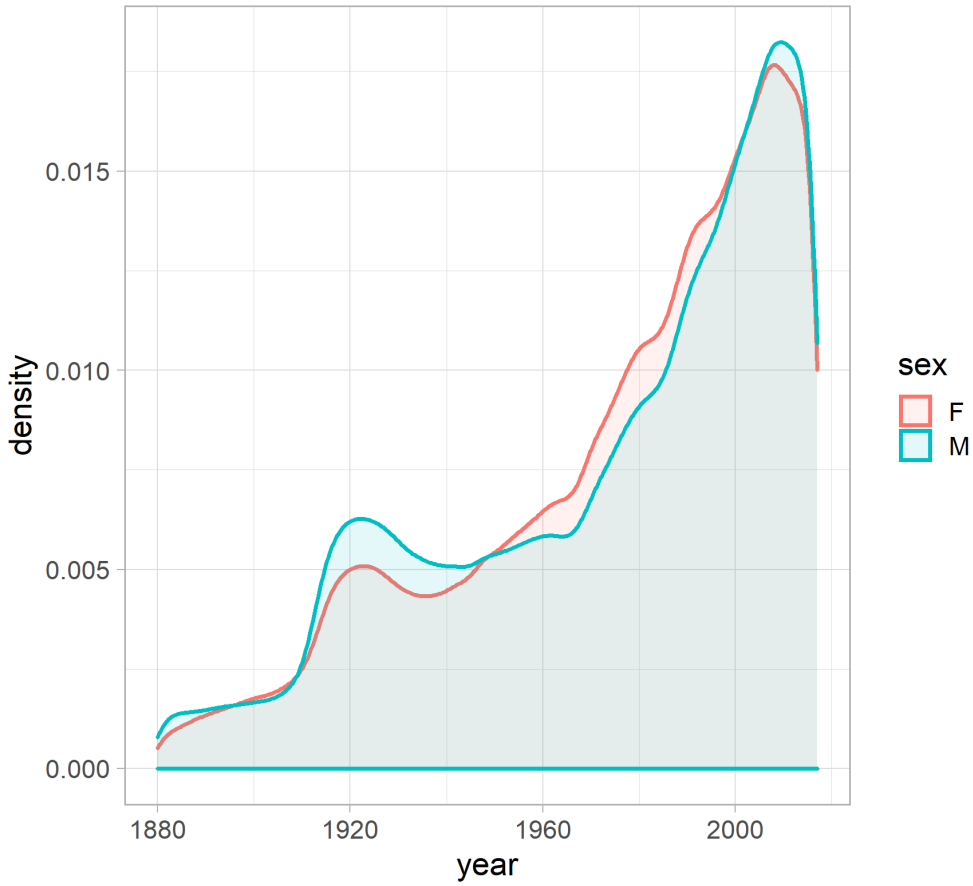
Your Turn: Density Plots

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

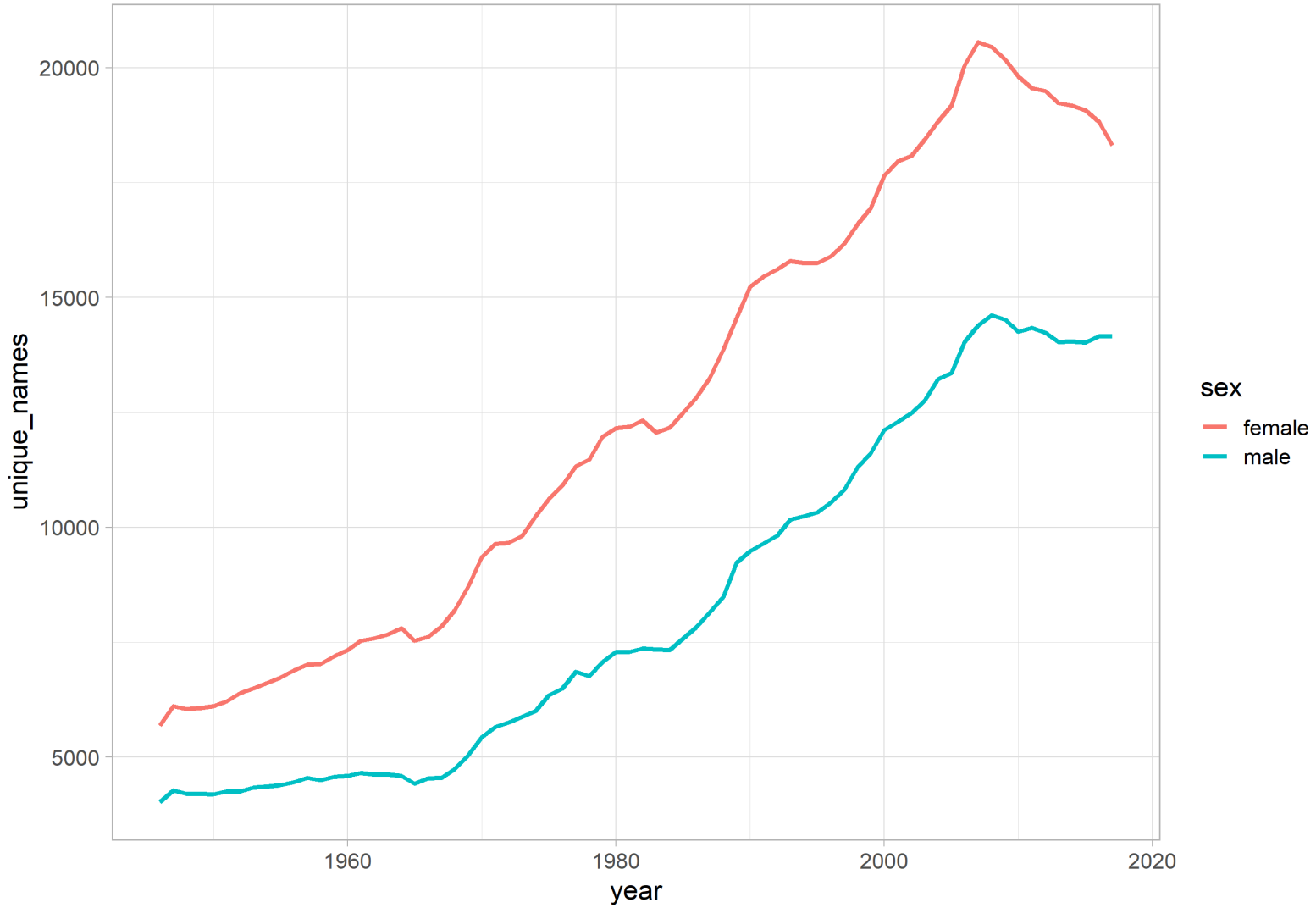


Your Turn: Density Plots

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

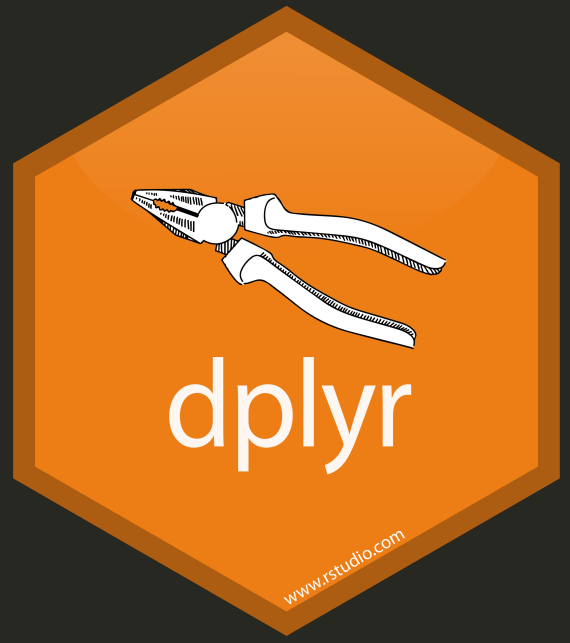


The Rise of Unique Baby Names in the Post-WWII Era



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`

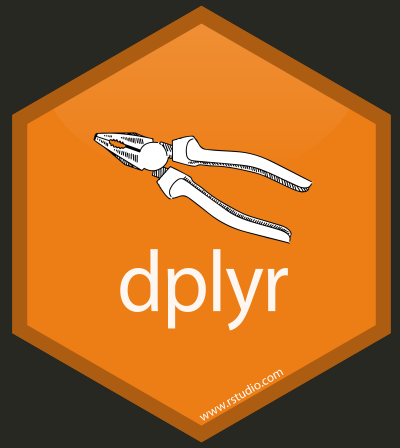
All functions take the same main arguments:

```
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)` → 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
```

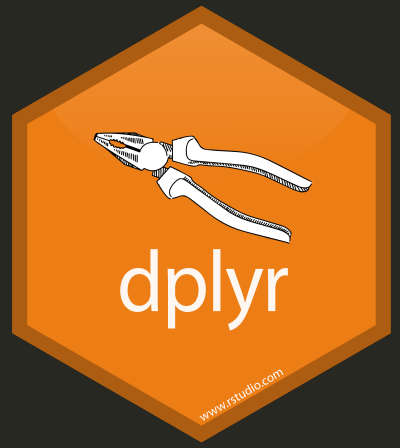
Pick Rows with Matching Criteria

`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 **rows**:

```
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

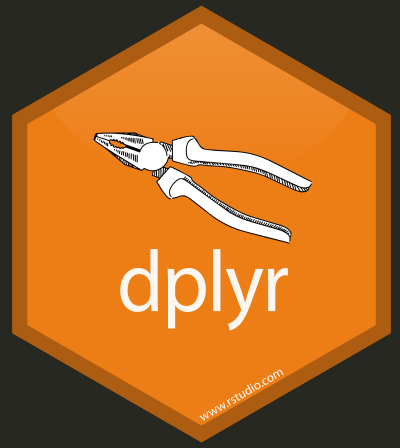
`select(data, condition)` allows you to select a subset of **columns**:

```
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

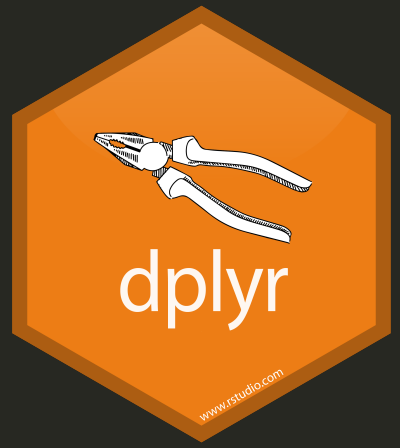


Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

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, 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:

```
mpg <- select(mpg, manufacturer:year, hwy, cty) ## only for visualization
mutate(mpg, diff = hwy - cty, perc = diff / hwy)
## # A tibble: 234 x 8
##    manufacturer model      displ  year   hwy   cty  diff  perc
##    <chr>        <chr>      <dbl> <int> <int> <int> <int> <dbl>
##  1 audi         a4           1.8  1999    29    18    11 0.379
##  2 audi         a4           1.8  1999    29    21     8 0.276
##  3 audi         a4           2    2008    31    20    11 0.355
##  4 audi         a4           2    2008    30    21     9 0.3
##  5 audi         a4           2.8  1999    26    16    10 0.385
##  6 audi         a4           2.8  1999    26    18     8 0.308
##  7 audi         a4           3.1  2008    27    18     9 0.333
##  8 audi         a4 quattro   1.8  1999    26    18     8 0.308
##  9 audi         a4 quattro   1.8  1999    25    16     9 0.36
## 10 audi         a4 quattro   2    2008    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:

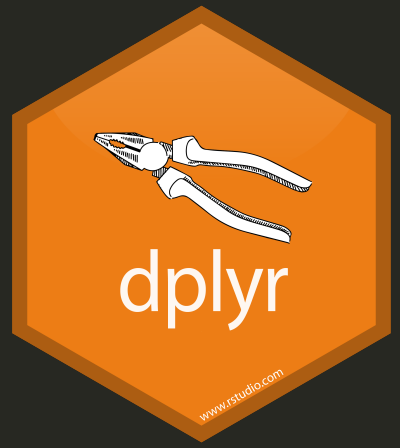
```
mpg <- select(mpg, manufacturer:year, hwy, cty) ## only for visualization
mutate(mpg, diff = hwy - cty, perc = diff / hwy)
```

Equivalent code in base R:

```
mpg_diff <- transform(mpg, diff = hwy - cty)
transform(mpg_diff, perc = diff / hwy)
##     manufacturer                 model displ year hwy cty diff      perc
## 1           audi                    a4   1.8 1999  29  18   11 0.3793103
## 2           audi                    a4   1.8 1999  29  21    8 0.2758621
## 3           audi                    a4   2.0 2008  31  20   11 0.3548387
## 4           audi                    a4   2.0 2008  30  21    9 0.3000000
## 5           audi                    a4   2.8 1999  26  16   10 0.3846154
## 6           audi                    a4   2.8 1999  26  18    8 0.3076923
## 7           audi                    a4   3.1 2008  27  18    9 0.3333333
## 8           audi             a4 quattro   1.8 1999  26  18    8 0.3076923
## 9           audi             a4 quattro   1.8 1999  25  16    9 0.3600000
## 10          audi             a4 quattro   2.0 2008  28  20    8 0.2857143
## 11          audi             a4 quattro   2.0 2008  27  19    8 0.2962963
```

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
```

Create New 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
```

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
```


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_1900 <- filter(babynames, year == 1900)
names_1900 <- group_by(names_1900, sex, year)
names_1900
## # A tibble: 3,730 x 5
## # Groups:   sex, year [2]
##     year sex   name           n   prop
##    <dbl> <chr> <chr>      <int>  <dbl>
##  1  1900 F     Mary       16706 0.0526
##  2  1900 F     Helen       6343 0.0200
##  3  1900 F     Anna        6114 0.0192
##  4  1900 F     Margaret    5304 0.0167
##  5  1900 F     Ruth        4765 0.0150
##  6  1900 F     Elizabeth   4096 0.0129
##  7  1900 F     Florence    3920 0.0123
##  8  1900 F     Ethel       3896 0.0123
##  9  1900 F     Marie       3856 0.0121
## 10  1900 F     Lillian     3414 0.0107
## # ... with 3,720 more rows
```

`group_by()` SUPERPOWER!

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

```
names_1900 <- filter(babynames, year == 1900)
names_1900 <- group_by(names_1900, sex, year)
names_1900 <- mutate(names_1900, sum = sum(n))
names_1900 <- group_by(names_1900, sex, year, name)
names_1900
## # A tibble: 3,730 x 6
## # Groups:   sex, year, name [3,730]
##     year sex   name           n   prop    sum
##    <dbl> <chr> <chr>      <int>  <dbl>  <int>
##  1  1900 F     Mary       16706 0.0526 299800
##  2  1900 F     Helen       6343 0.0200 299800
##  3  1900 F     Anna        6114 0.0192 299800
##  4  1900 F     Margaret    5304 0.0167 299800
##  5  1900 F     Ruth        4765 0.0150 299800
##  6  1900 F     Elizabeth   4096 0.0129 299800
##  7  1900 F     Florence    3920 0.0123 299800
##  8  1900 F     Ethel       3896 0.0123 299800
##  9  1900 F     Marie       3856 0.0121 299800
## 10  1900 F     Lillian     3414 0.0107 299800
## # ... with 3,720 more rows
```

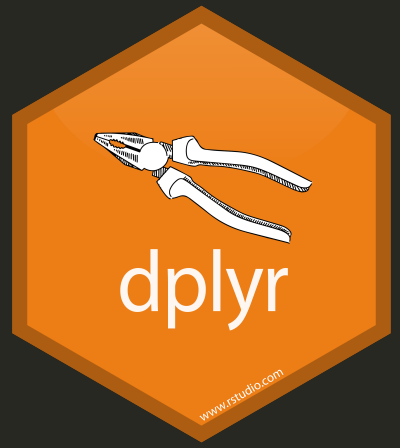
`group_by()` SUPERPOWER!

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

```
names_1900 <- filter(babynames, year == 1900)
names_1900 <- group_by(names_1900, sex, year)
names_1900 <- mutate(names_1900, sum = sum(n))
names_1900 <- group_by(names_1900, sex, year, name)
names_1900_sum <- summarize(names_1900, prop = prop, prop_check = sum(n) / unique(sum)
arrange(names_1900_sum, desc(sex), -prop_check)
## # A tibble: 3,730 x 5
## # Groups:   sex, year [2]
##    sex    year name      prop prop_check
##    <chr> <dbl> <chr>    <dbl>      <dbl>
##  1 M      1900 John    0.0606     0.0653
##  2 M      1900 William 0.0529     0.0570
##  3 M      1900 James   0.0447     0.0481
##  4 M      1900 George  0.0333     0.0359
##  5 M      1900 Charles 0.0253     0.0272
##  6 M      1900 Robert  0.0236     0.0254
##  7 M      1900 Joseph  0.0229     0.0247
##  8 M      1900 Frank   0.0214     0.0231
##  9 M      1900 Edward  0.0168     0.0181
## 10 M      1900 Henry   0.0161     0.0173
```

group_by()

Create Subsets



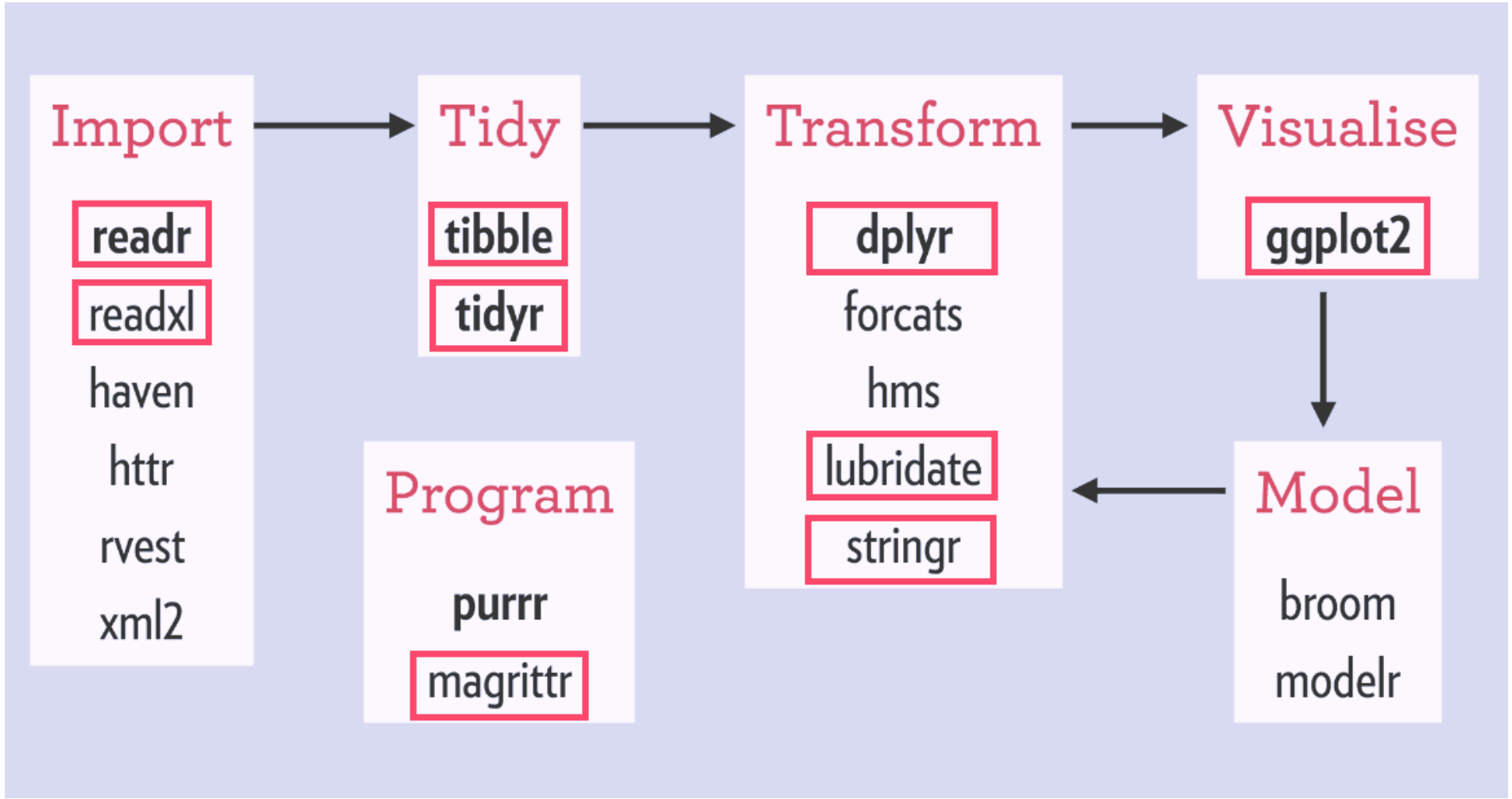
Ungroup Data Afterwards

... and `ungroup()` lets you ungroup your data:

```
names_1990_sum <- ungroup(names_1900_sum)
arrange(names_1990_sum, desc(sex), -prop_check)
## # A tibble: 3,730 x 5
##    sex    year name      prop prop_check
##    <chr> <dbl> <chr>    <dbl>      <dbl>
##  1 M      1900 John    0.0606     0.0653
##  2 M      1900 William 0.0529     0.0570
##  3 M      1900 James   0.0447     0.0481
##  4 M      1900 George  0.0333     0.0359
##  5 M      1900 Charles 0.0253     0.0272
##  6 M      1900 Robert  0.0236     0.0254
##  7 M      1900 Joseph  0.0229     0.0247
##  8 M      1900 Frank   0.0214     0.0231
##  9 M      1900 Edward  0.0168     0.0181
## 10 M      1900 Henry   0.0161     0.0173
## # ... with 3,720 more rows
```

Excourse: 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_1900 <- filter(babynames, year == 1900)
names_1900 <- group_by(names_1900, sex, year)
names_1900 <- mutate(names_1900, sum = sum(n))
names_1900 <- group_by(names_1900, sex, year, name)
names_1900_sum <- summarize(names_1900, prop = prop, prop_check = sum(n) / unique(sum))
names_1990_sum <- ungroup(names_1900_sum)
names_1900_sum <- arrange(names_1900_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_1900 <-
  group_by(mutate(group_by(filter(babynames, year == 1900), sex, year), sum = sum(n)),

names_1900_sum <-
  ungroup(arrange(summarize(names_1900, prop = prop, prop_check = sum(n) / unique(sum)

names_1900_sum
## # A tibble: 3,730 x 5
##    sex    year name      prop prop_check
##    <chr> <dbl> <chr>    <dbl>      <dbl>
##  1 M      1900 John    0.0606     0.0653
##  2 M      1900 William 0.0529     0.0570
##  3 M      1900 James   0.0447     0.0481
##  4 M      1900 George  0.0333     0.0359
##  5 M      1900 Charles 0.0253     0.0272
##  6 M      1900 Robert  0.0236     0.0254
##  7 M      1900 Joseph  0.0229     0.0247
##  8 M      1900 Frank   0.0214     0.0231
##  9 M      1900 Edward  0.0168     0.0181
## 10 M      1900 Henry   0.0161     0.0173
## # ... with 3,720 more rows
```

Readability of Code

One could rewrite the code like this:

```
names_1900 <-
  group_by(
    mutate(
      group_by(
        filter(
          babynames,
          year == 1900
        ),
        sex, year
        ),
      sum = sum(n)
    ),
    sex, year, name
  )

names_1900_sum <-
  ungroup(
    arrange(
      summarize(
        names_1900,
        prop = prop,
```

```
physalia(
  bvg(
    breakfast(
      shower(
        wake_up(
          Cedric,
          7.0
        ),
        temp == 38
      ),
      c("cornflakes", "tea"),
    ),
    price = "EUR2.80",
    delay = 10
  ),
  course = "Data Visualization in R with ggplot2"
)
```

```
Cedric %>%
  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"
  )
```

```
Cedric %>%
  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_1900 <-
  babynames %>%
  filter(year == 1900) %>%
  group_by(sex, year) %>%
  mutate(sum = sum(n)) %>%
  group_by(sex, year, name)

names_1900_sum <-
  names_1900 %>%
  summarize(
    prop = prop,
    prop_check = sum(n) / unique(sum)
  ) %>%
  arrange(desc(sex), -prop_check) %>%
  ungroup()
```

Readability of Code: %>% (The Pipe)

With the pipe one can rewrite the code like this:

```
names_1900_sum <-
  babynames %>%
  filter(year == 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)

With the pipe one can rewrite the code like this:

```
names_1900_sum
## # A tibble: 3,730 x 5
##    sex    year name      prop prop_check
##    <chr> <dbl> <chr>    <dbl>      <dbl>
##  1 M      1900 John    0.0606     0.0653
##  2 M      1900 William 0.0529     0.0570
##  3 M      1900 James   0.0447     0.0481
##  4 M      1900 George  0.0333     0.0359
##  5 M      1900 Charles 0.0253     0.0272
##  6 M      1900 Robert  0.0236     0.0254
##  7 M      1900 Joseph  0.0229     0.0247
##  8 M      1900 Frank   0.0214     0.0231
##  9 M      1900 Edward  0.0168     0.0181
## 10 M      1900 Henry   0.0161     0.0173
## # ... with 3,720 more rows
```

Readability of Code: %>% (The Pipe)

You can easily inactivate verbs or single conditions:

```
names_1900_sum <-
  babynames %>%
  filter(year == 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_1900_sum <-
  babynames %>%
  filter(year == 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_1900_sum <-
  babynames %>%
  filter(year == 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_1900_sum <-
  babynames %>%
  filter(year == 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_1900_sum <-
  babynames %>%
  filter(year == 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() %>%
  {} #
```

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.

Flights in June and July 2013

```
filter(flights, month == 11 | month == 12)
## # A tibble: 55,403 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    11     1        5           2359         6      352            345
##  2  2013    11     1       35           2250       105      123           2356
##  3  2013    11     1      455            500        -5      641            651
##  4  2013    11     1      539            545        -6      856            827
##  5  2013    11     1      542            545        -3      831            855
##  6  2013    11     1      549            600       -11      912            923
##  7  2013    11     1      550            600       -10      705            659
##  8  2013    11     1      554            600        -6      659            701
##  9  2013    11     1      554            600        -6      826            827
## 10  2013    11     1      554            600        -6      749            751
## # ... with 55,393 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(11, 12))
## # A tibble: 55,403 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    11     1        5           2359         6      352            345
##  2  2013    11     1       35           2250       105      123           2356
##  3  2013    11     1      455            500        -5      641            651
##  4  2013    11     1      539            545        -6      856            827
##  5  2013    11     1      542            545        -3      831            855
##  6  2013    11     1      549            600       -11      912            923
##  7  2013    11     1      550            600       -10      705            659
##  8  2013    11     1      554            600        -6      659            701
##  9  2013    11     1      554            600        -6      826            827
## 10  2013    11     1      554            600        -6      749            751
## # ... with 55,393 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>
```

Average Delays per Origin and Month

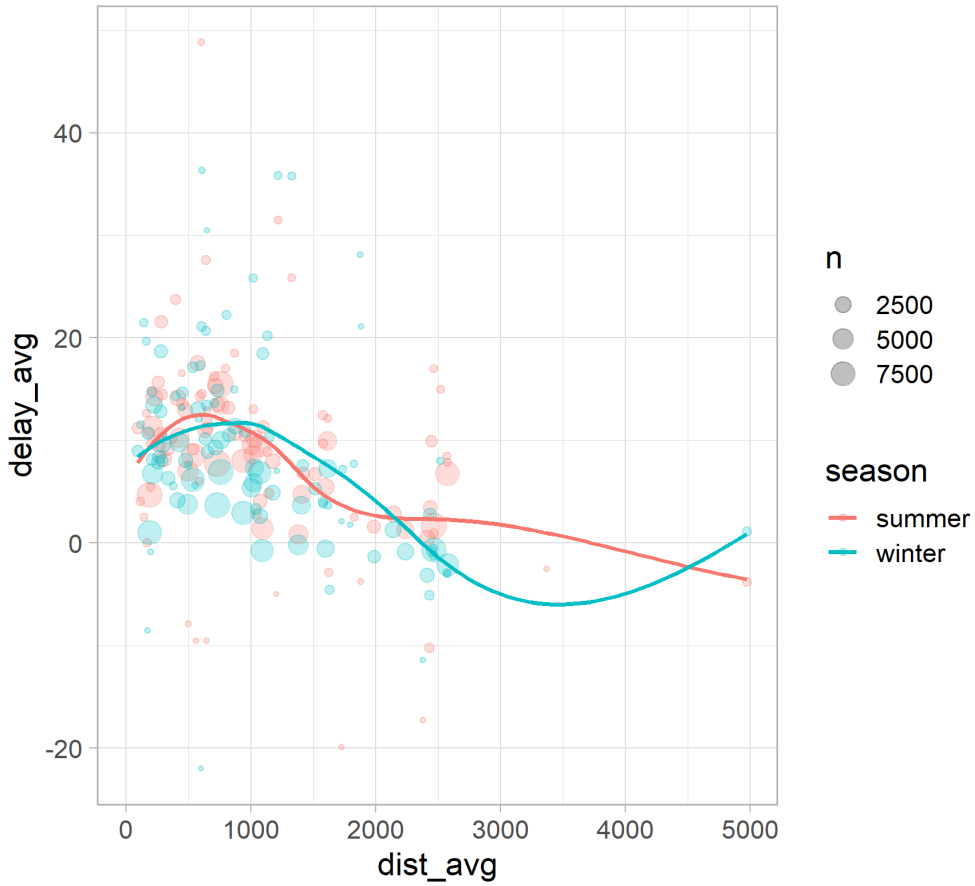
```
flights %>%
  group_by(month, origin) %>%
  summarize(
    dep_delay_avg = round(mean(dep_delay, na.rm = T), 1),
    arr_delay_avg = round(mean(arr_delay, na.rm = T), 1)
  ) %>%
  arrange(-arr_delay_avg)
## # A tibble: 36 x 4
## # Groups:   month [12]
##    month origin dep_delay_avg arr_delay_avg
##    <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

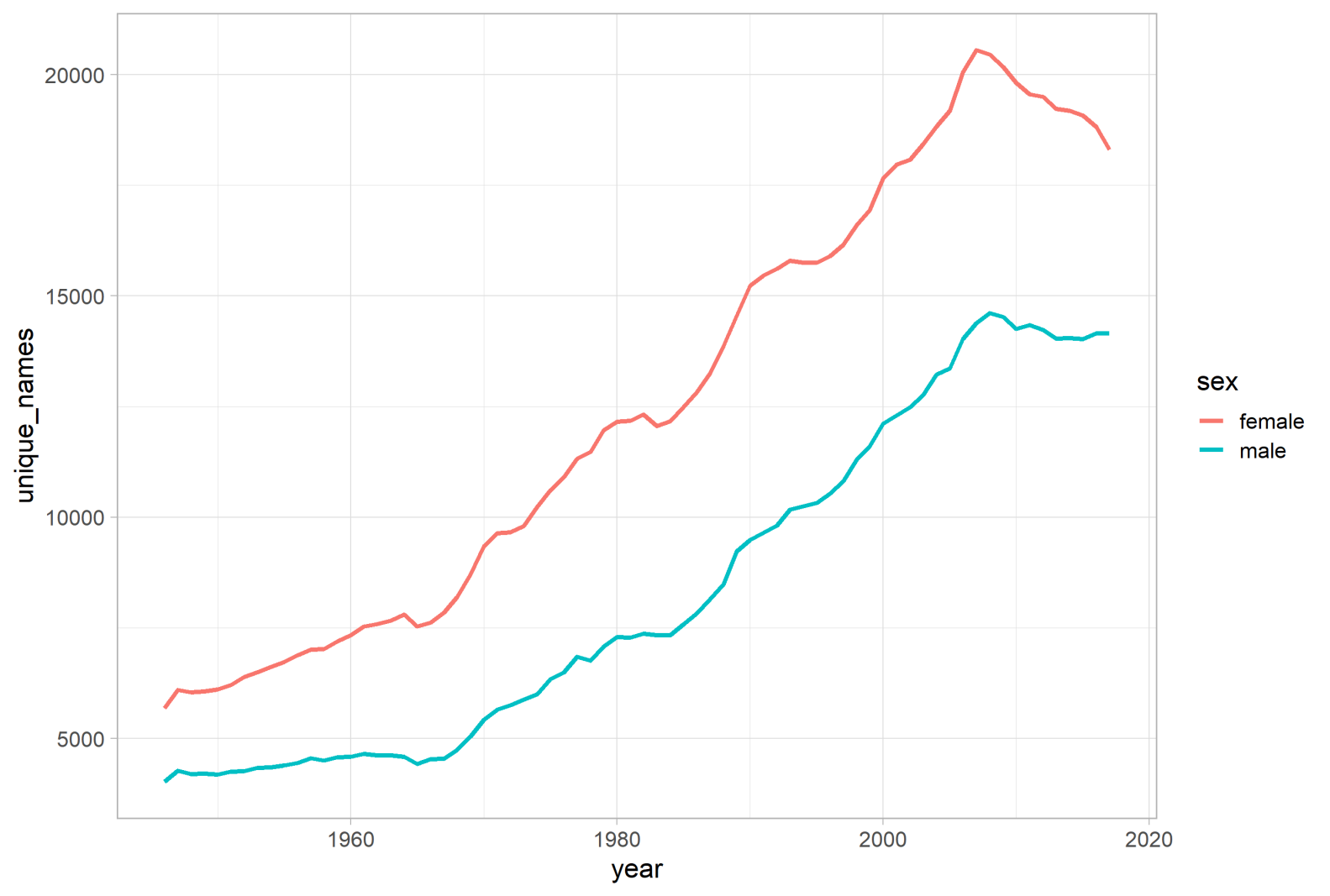
```
flights %>%
  select(month, origin, ends_with("delay")) %>%
  group_by(month, origin) %>%
  summarize_all(funs(avg = round(mean(., na.rm = T), 1))) %>%
  arrange(-arr_delay_avg)
## # A tibble: 36 x 4
## # Groups:   month [12]
##    month origin dep_delay_avg arr_delay_avg
##    <int> <chr>          <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
```

Distance versus Delay

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



Your Turn: Create this Line Plot!

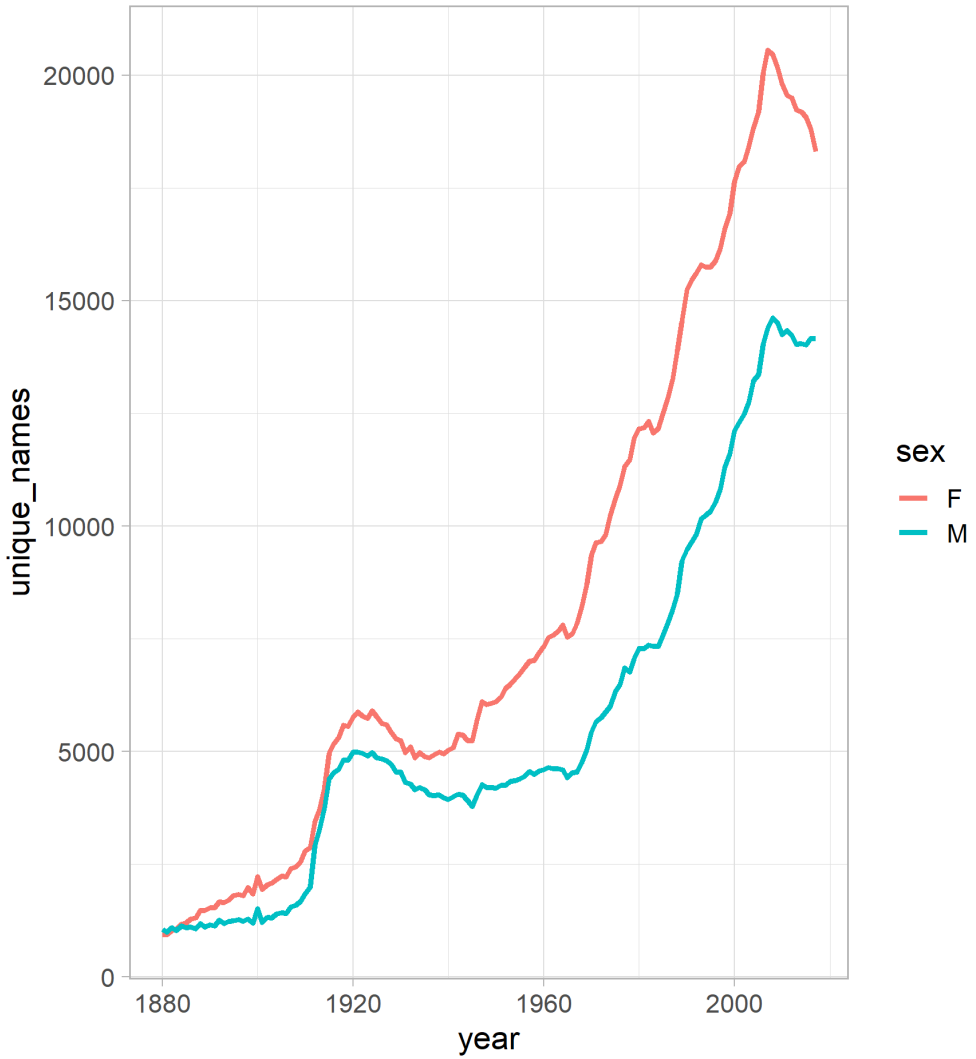


Your Turn: Post-WWII Line Plot

```
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
```

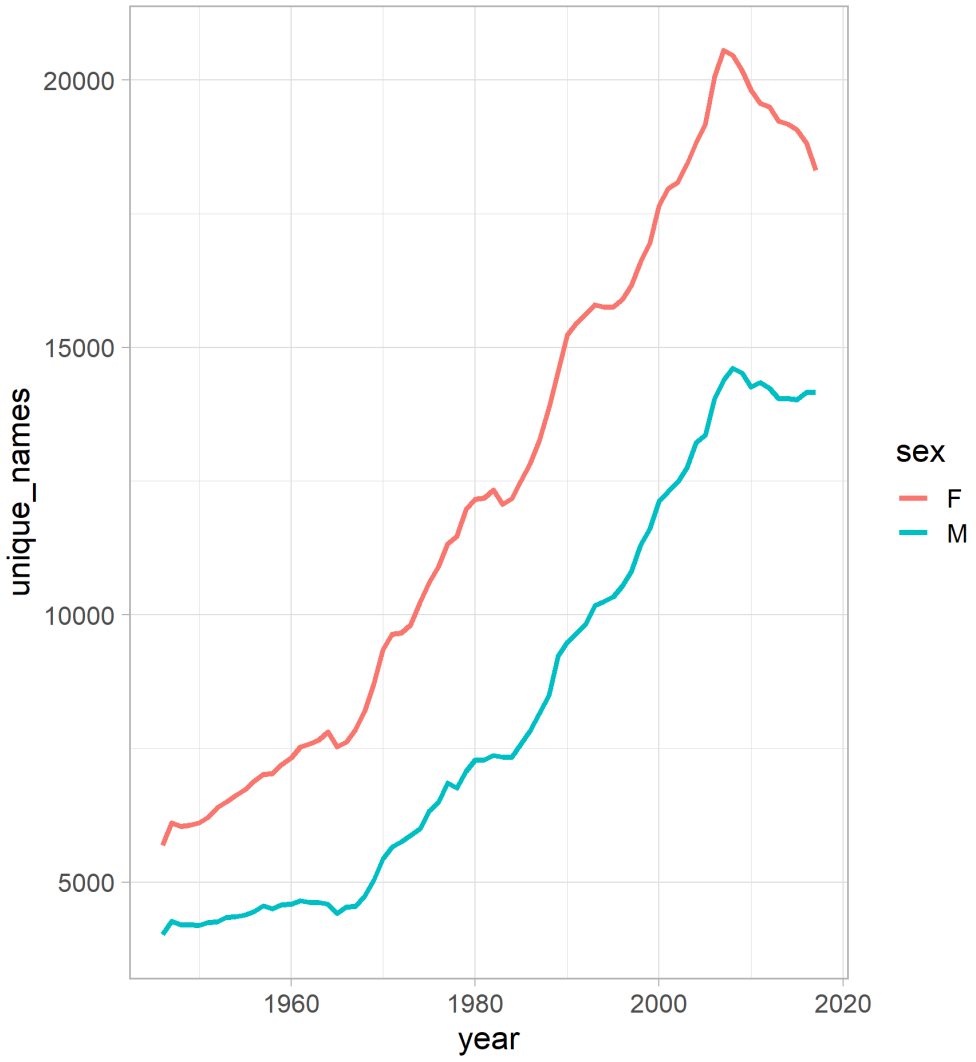
Your Turn: Post-WWII Line Plot

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



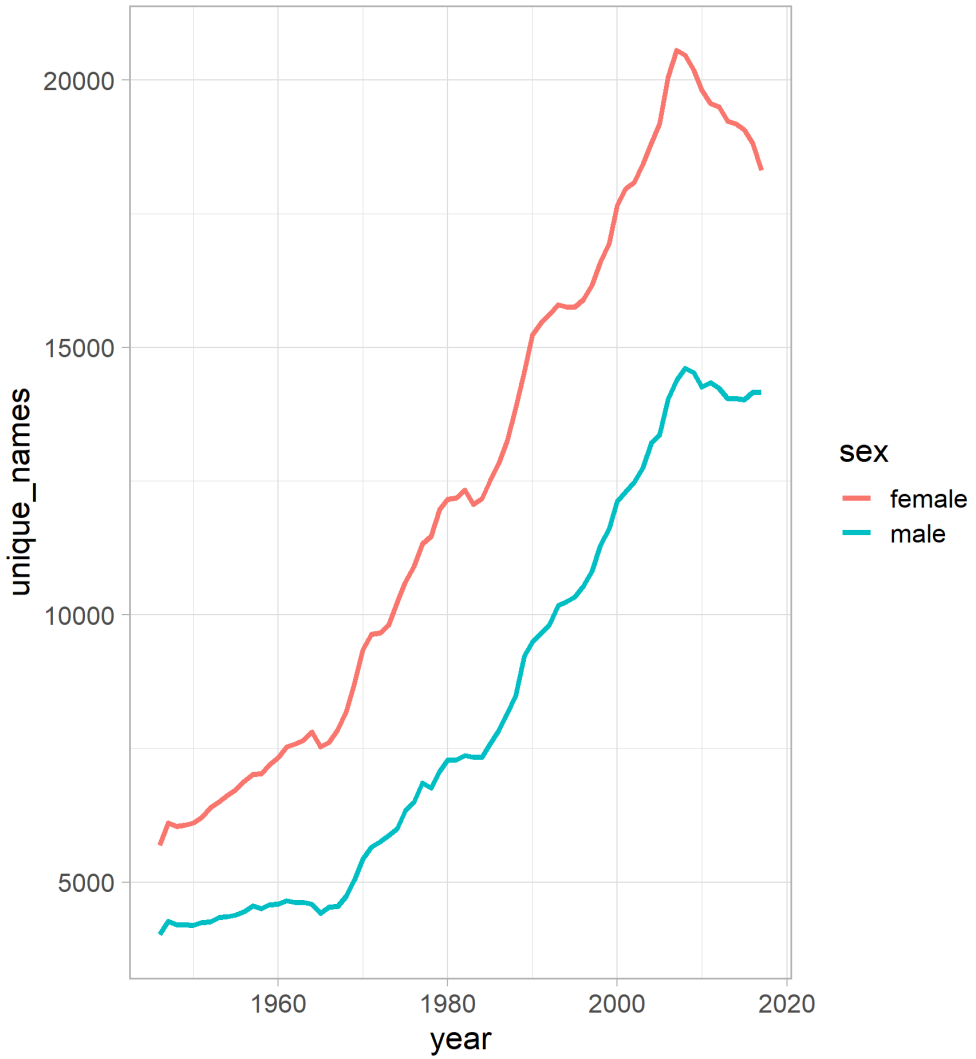
Your Turn: Post-WWII Line Plot

```
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)
```



Other Helpful Functions of `dplyr`

Function	Explanation
`slice()`	Filter rows by position
`distinct()` and `n_distinct()`	Find unique values
`sample_n()` and `sample_frac()`	Create samples of the data
`top_n()` and `top_frac()`	Pick top or bottom values by variable
`complete()`	XCreate all possible combinations of two variables

Data Transformation with `dplyr::`

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

action
column-related
row-related
function name (part)
function name (full)

MODIFY

- **COLUMNS**
  - **NAMES**
    - keep all → **rename**
    - keep focal → **select**
  - **POSITIONS**
    - keep focal → **transmute**
    - keep all → **mutate**
  - **VALUES**
- **ROWS**
  - **GROUPINGS**
    - **group_by**♠
    - **ungroup()**
  - **POSITIONS**
    - keep all → **arrange**♠
    - keep focal
      - based on test on values → **filter**
      - based on rank cut-off on values → **top_n()**
      - based on percentile cut-off on values → **top_frac()**
      - based on their positions → **slice()**

AGGREGATE
- **summarize**

COUNT
- **tally()**†

♥ = (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?

- **specific action on each focal column**
  - `()`
- **common action on all focal columns**
  - focal = all → **_all()**
  - focal identified by positions → **_at()**
  - focal identified by test on values → **_if()**

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)
```