

# Data Visualization in R with `ggplot2`

Data Transformation with `dplyr`

**Cédric Scherer**

Physalia Courses | March 2-6 2020

Photo by Richard Strozyński

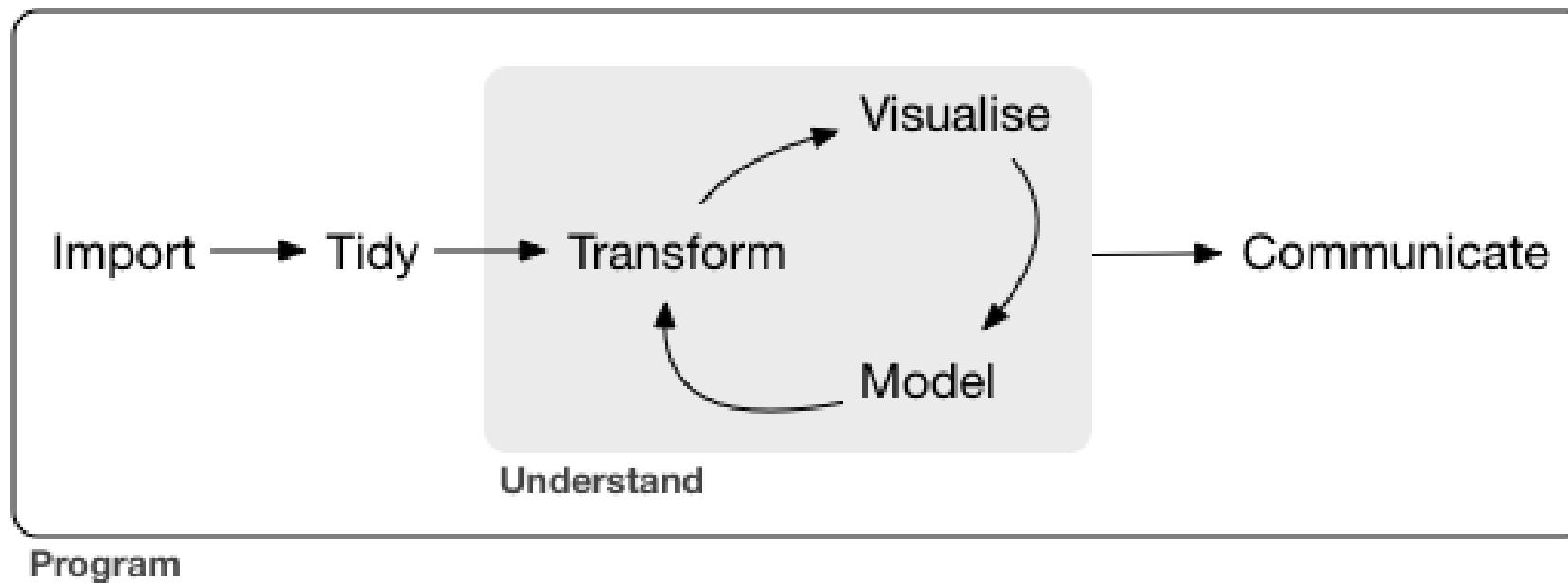
dplyr : go wrangling



Illustration by Allison Horst ([github.com/allisonhorst/stats-illustrations](https://github.com/allisonhorst/stats-illustrations))

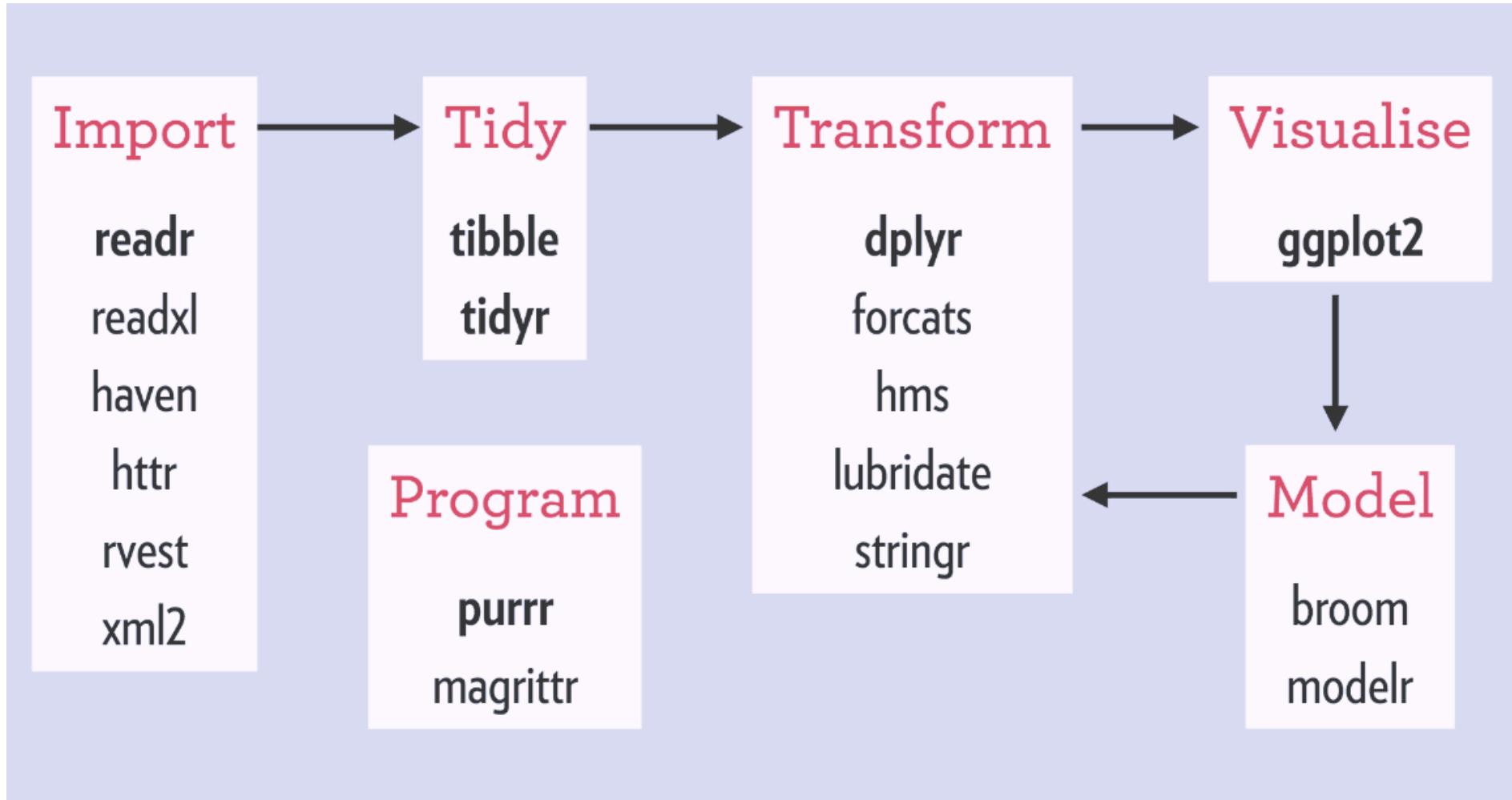
# The **tidyverse**

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



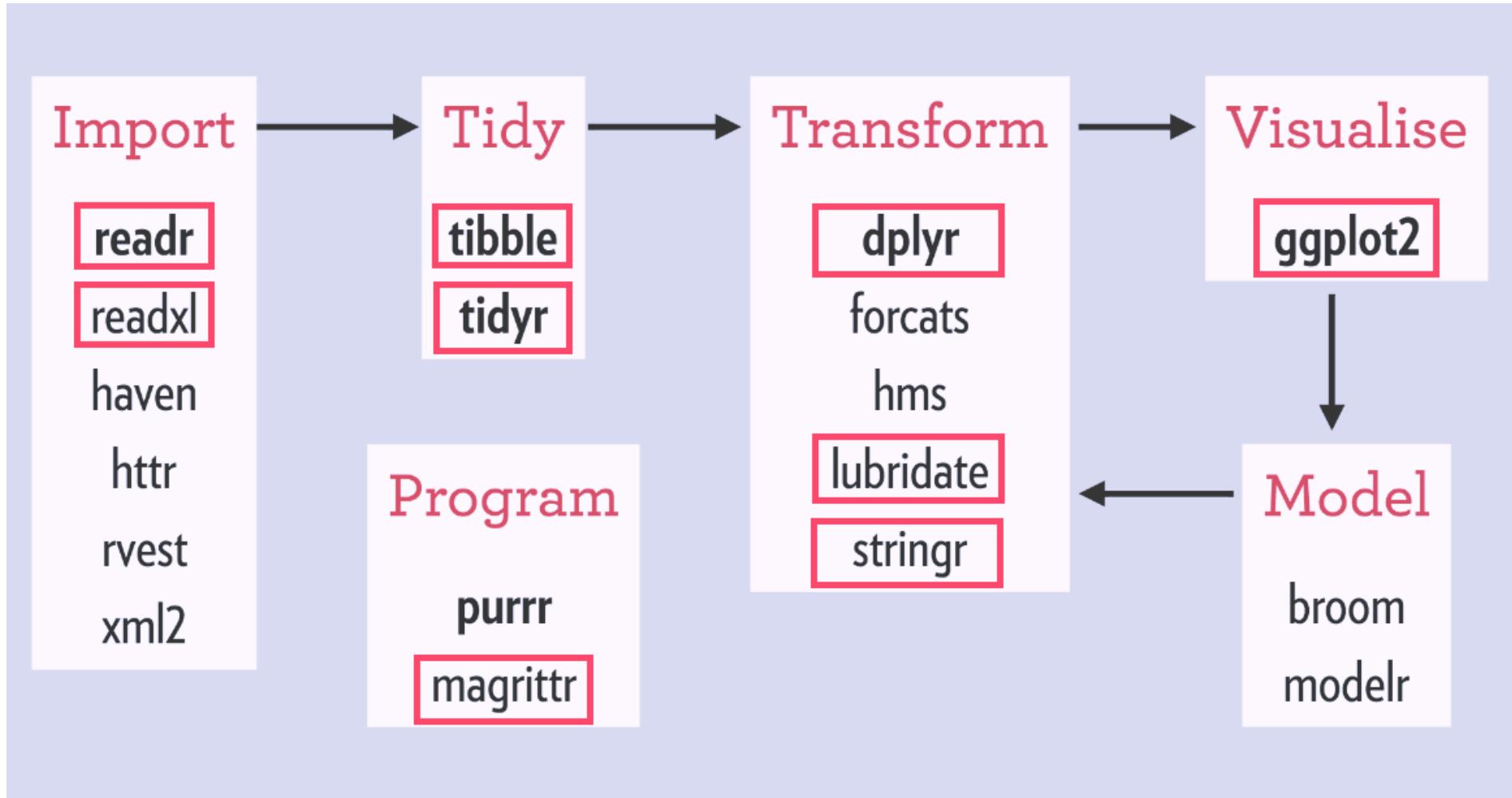
*Source: Hadley Wickham's "R for Data Science" (R4DS)*

# The Typical Data Science Project Flow



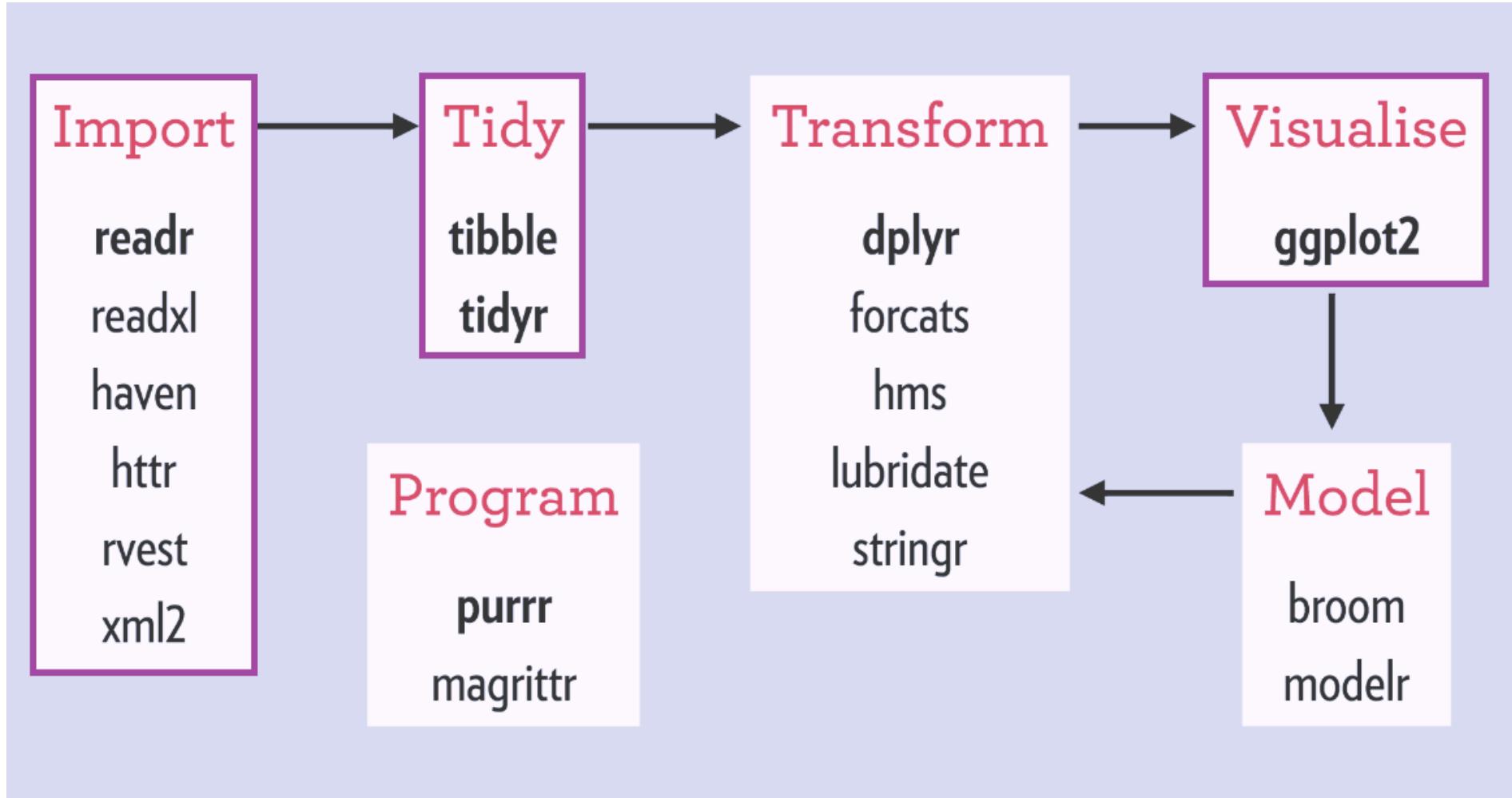
Source: [www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse\\_files/tidyverse1.png](http://www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png)

# The Typical Data Science Project Flow



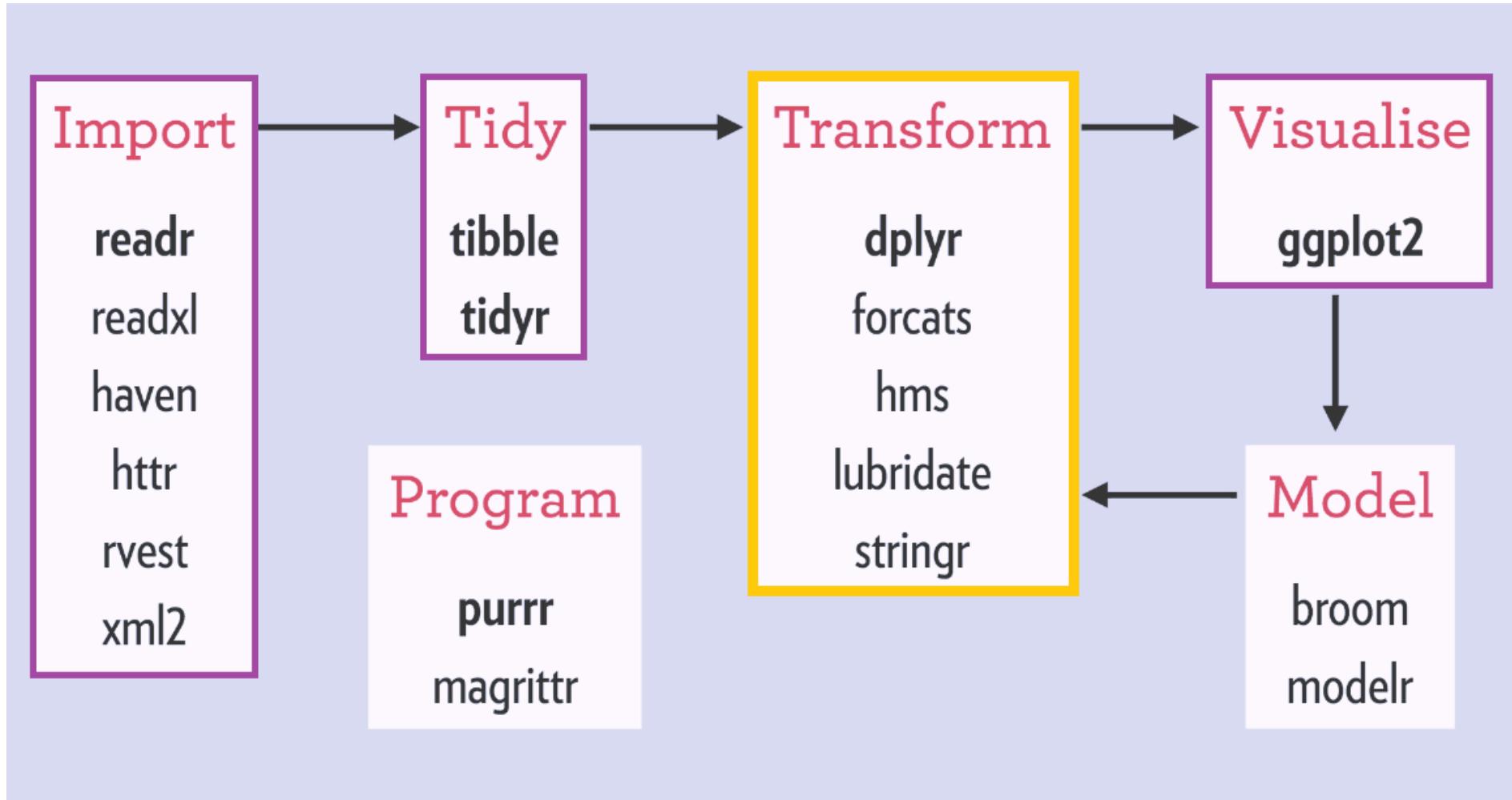
Source: [www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse\\_files/tidyverse1.png](http://www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png)

# The Typical Data Science Project Flow



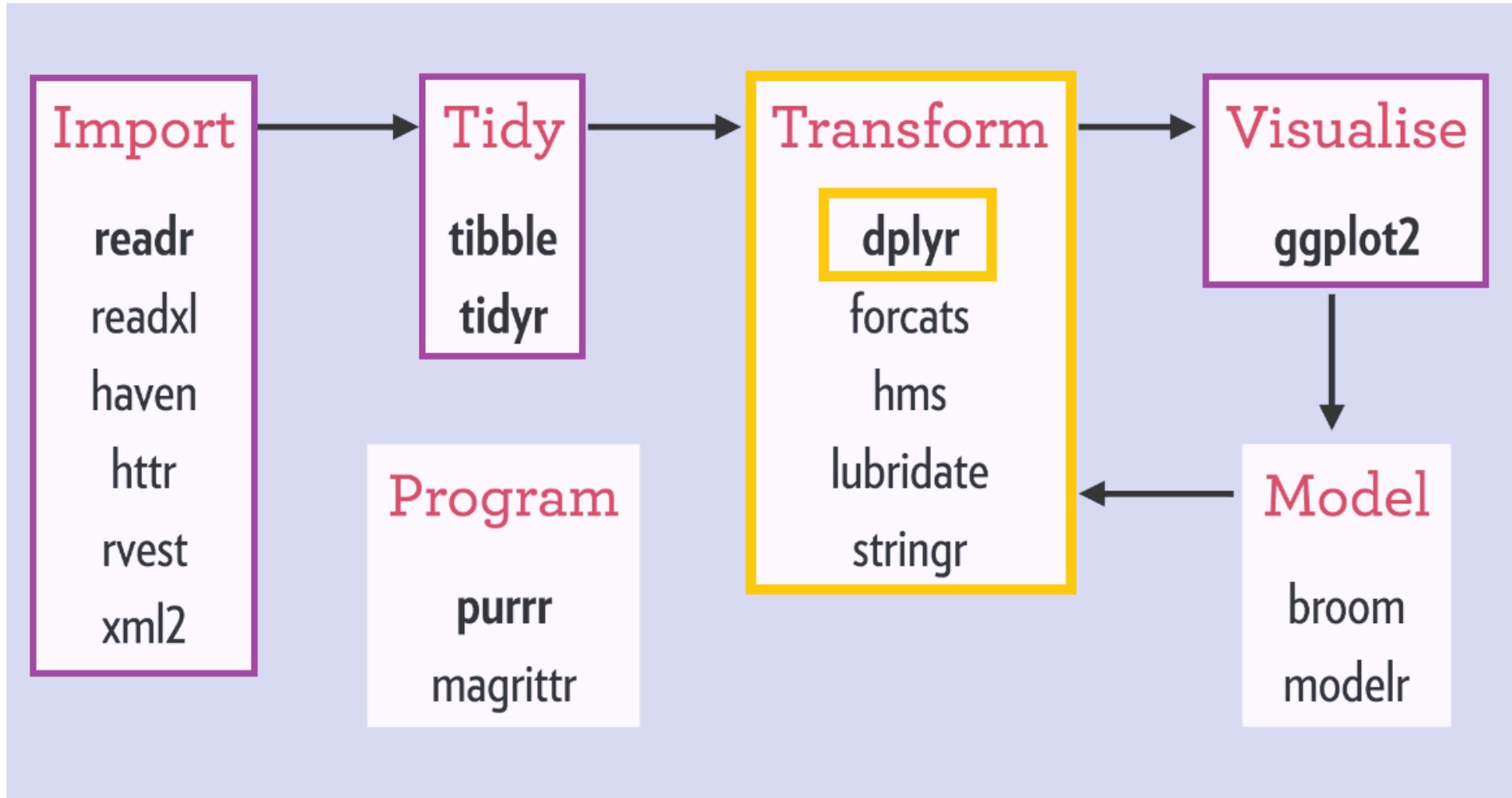
Source: [www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse\\_files/tidyverse1.png](http://www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png)

# The Typical Data Science Project Flow



Source: [www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse\\_files/tidyverse1.png](http://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](http://www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png)

# Our Example Data

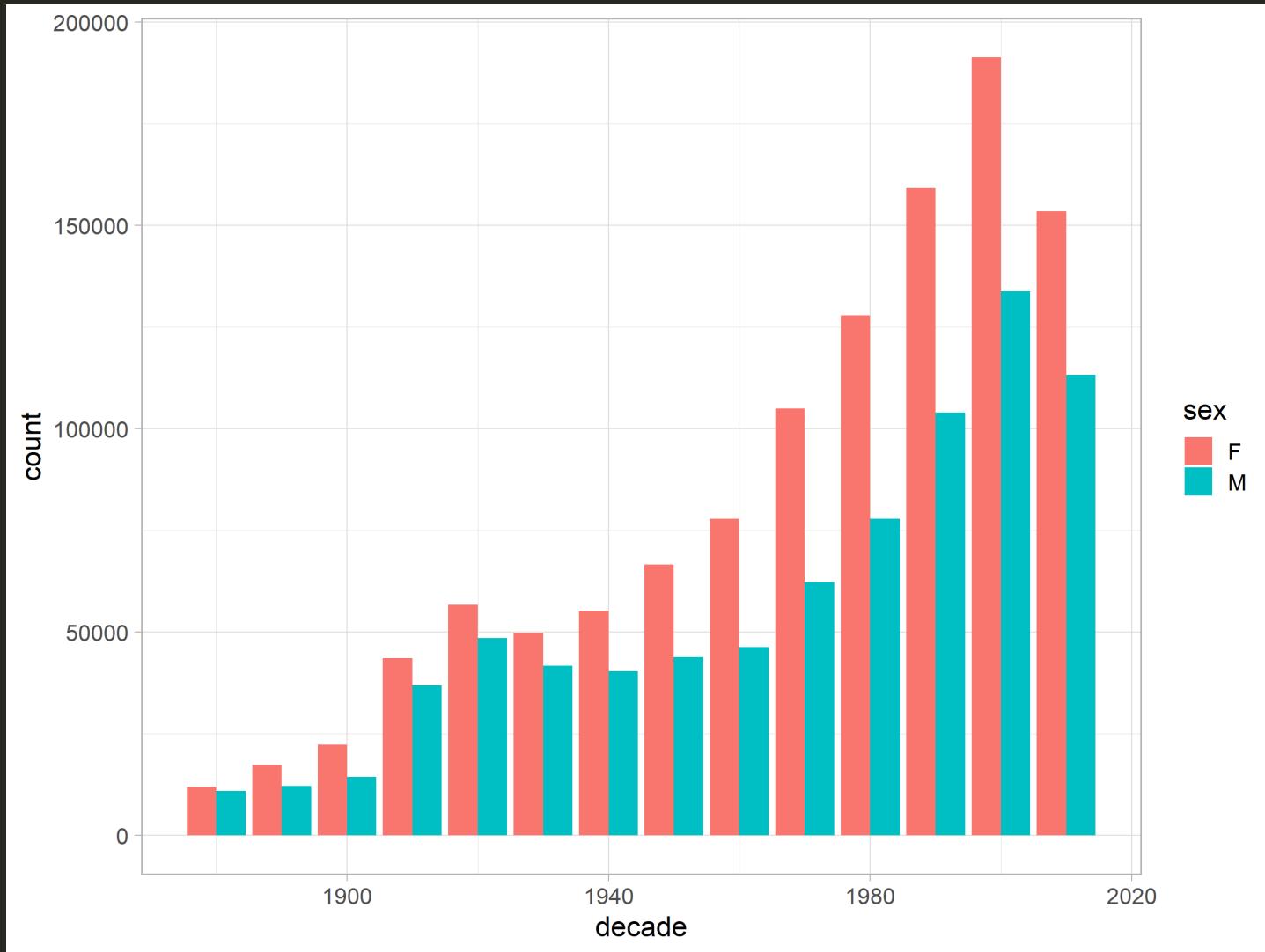
- Count and proportional data of baby names used for at least five children in the US per sex and year from 1880 to 2017
- 1,924,665 rows (observations) and 4 columns (variables)
- Data source: [US Social Security Administration \(SSA\)](#)
- Available as data set `babynames` in the package `babynames`: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

# Our Example Data: babynames

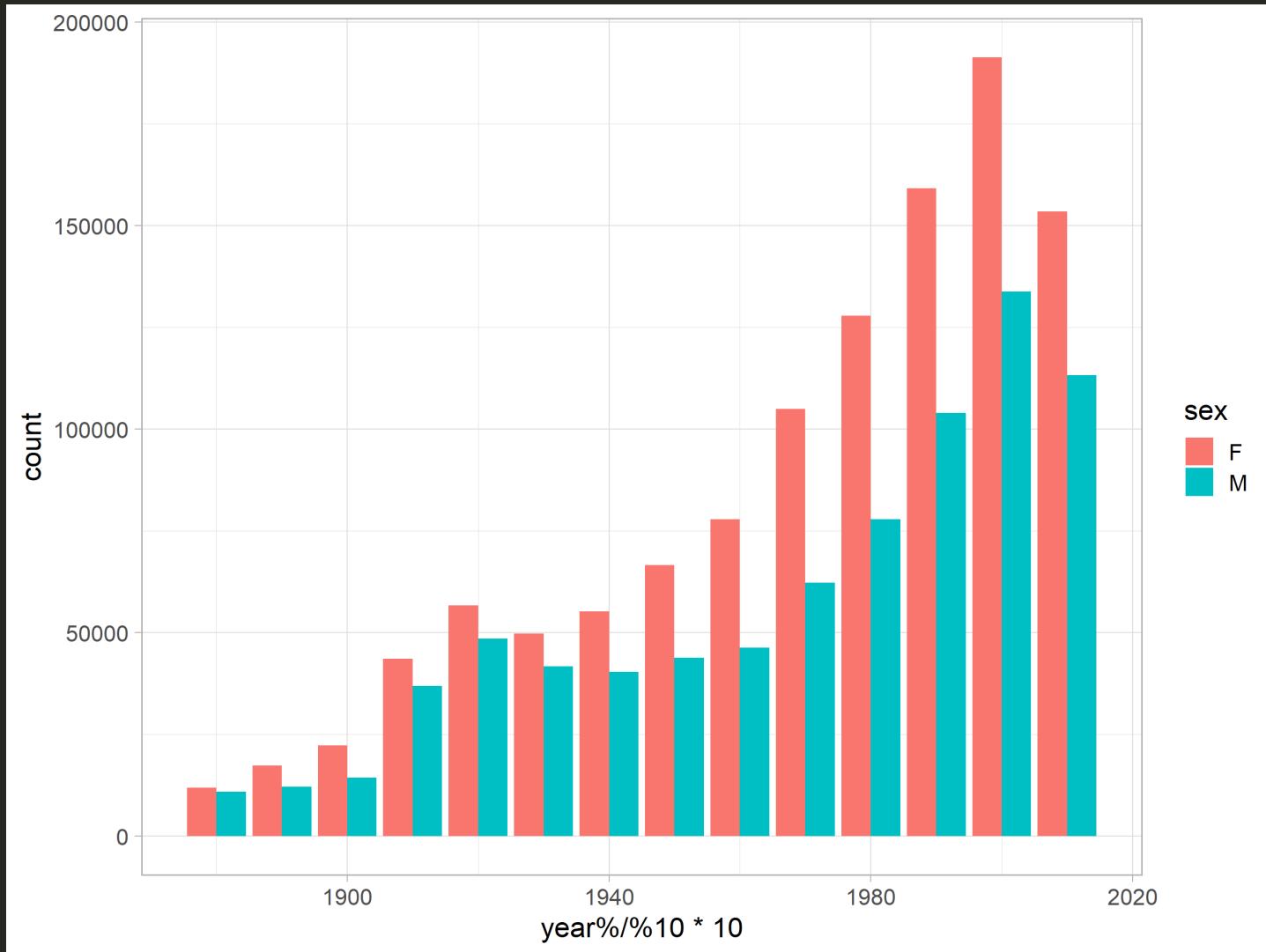
```
library(babynames)

glimpse(babynames)
## Observations: 1,924,665
## Variables: 5
## $ year <dbl> 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 18...
## $ sex <chr> "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!



# Your Turn! (Less Mean)



# Excuse: Integer Division

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

# Excuse: 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

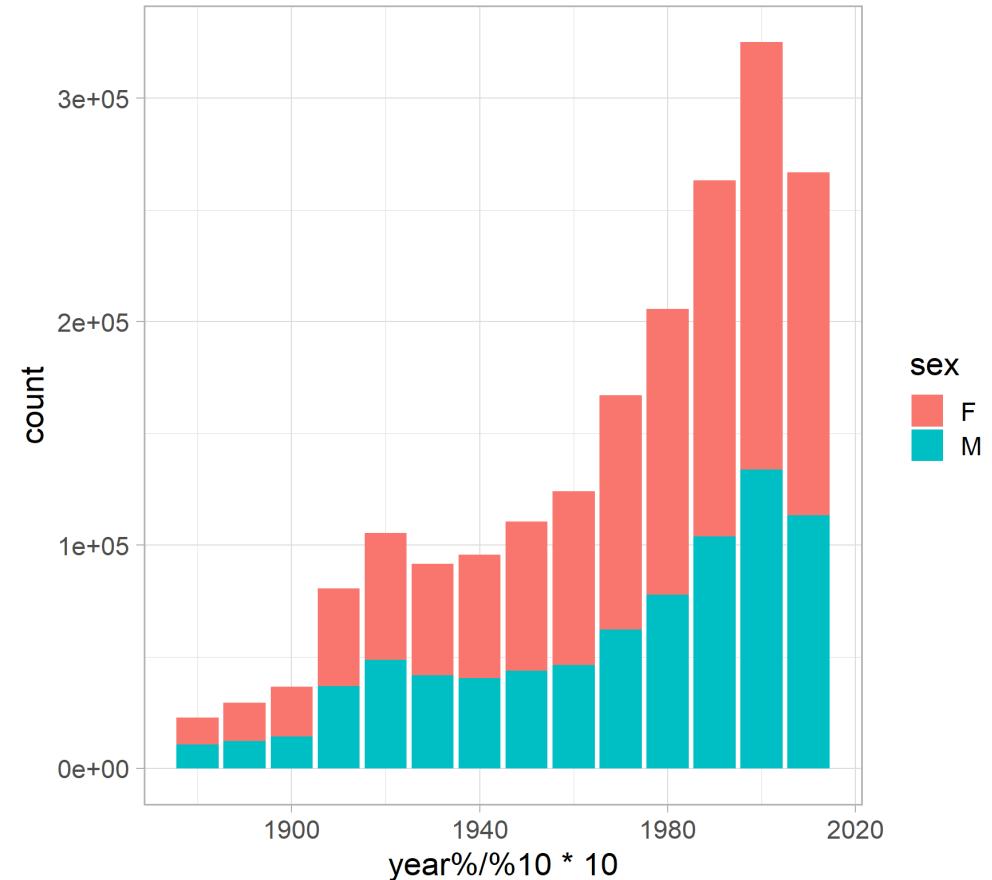
```
1 %% 10
## [1] 0
2 %% 10
## [1] 0
11 %% 10
## [1] 1
12 %% 10
## [1] 1

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

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

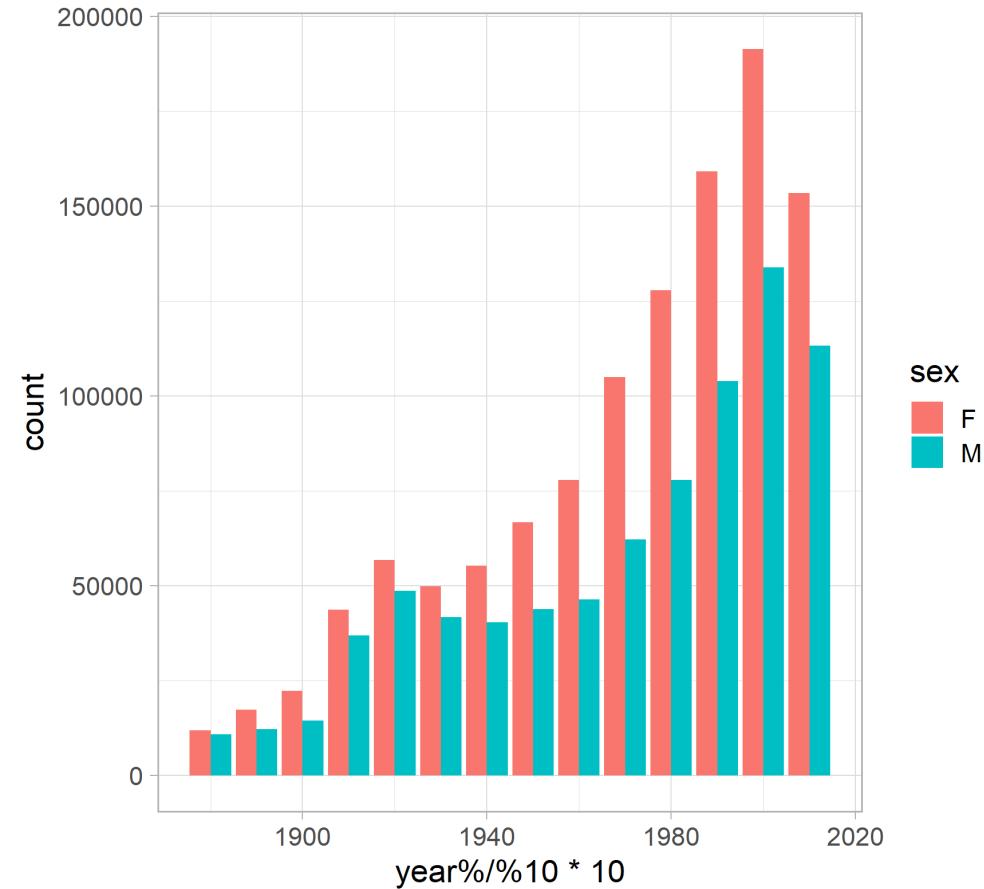
# Your Turn: Dodged Bar Plots

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



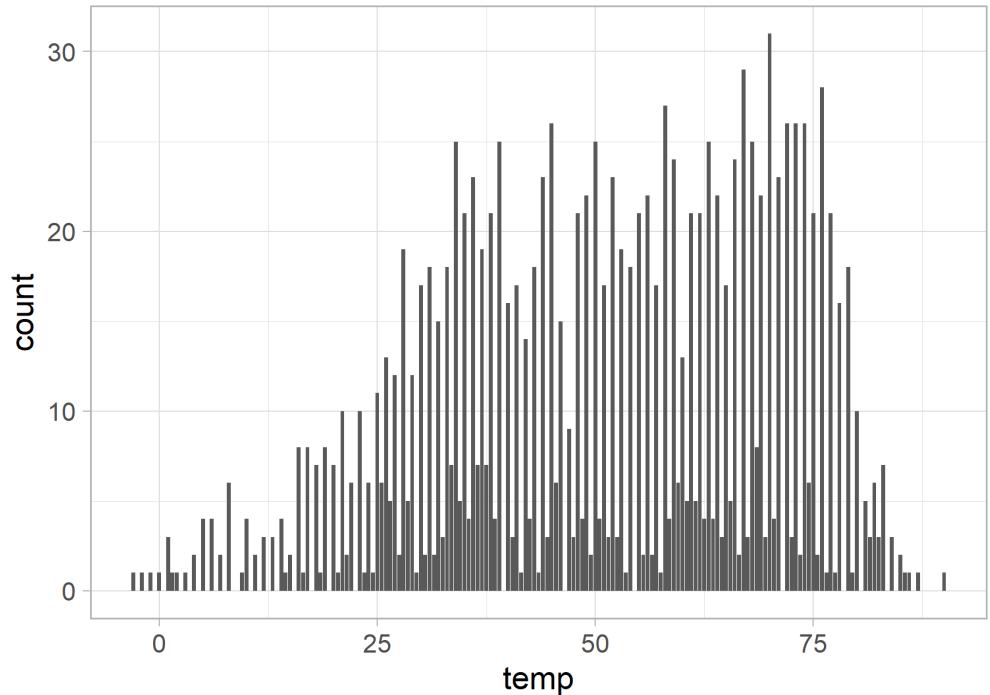
# Your Turn: Dodged Bar Plots

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

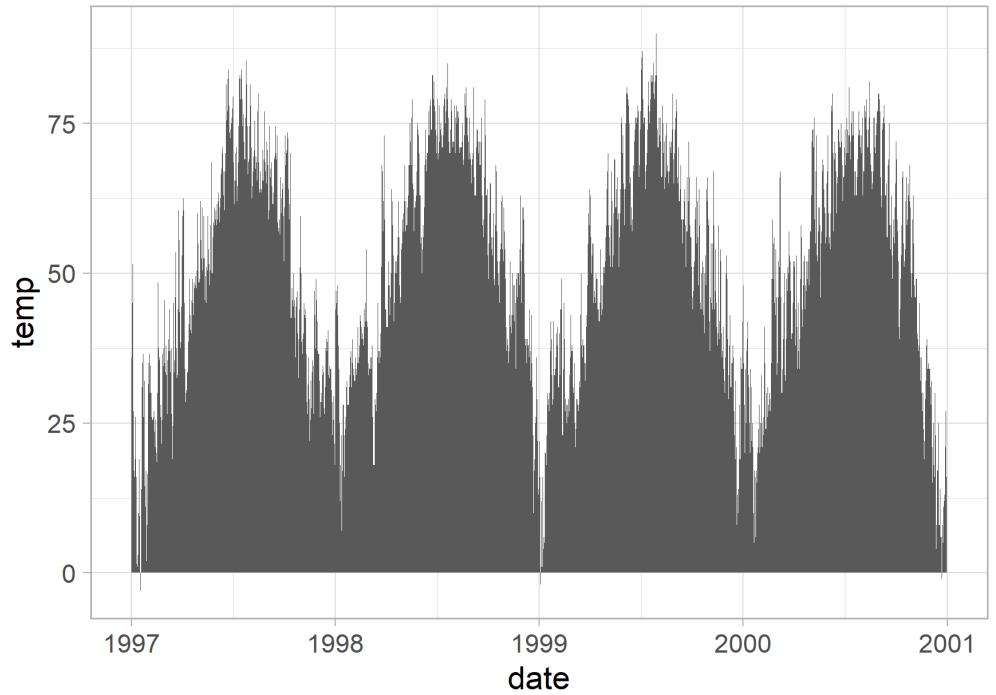


# geom\_bar()

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

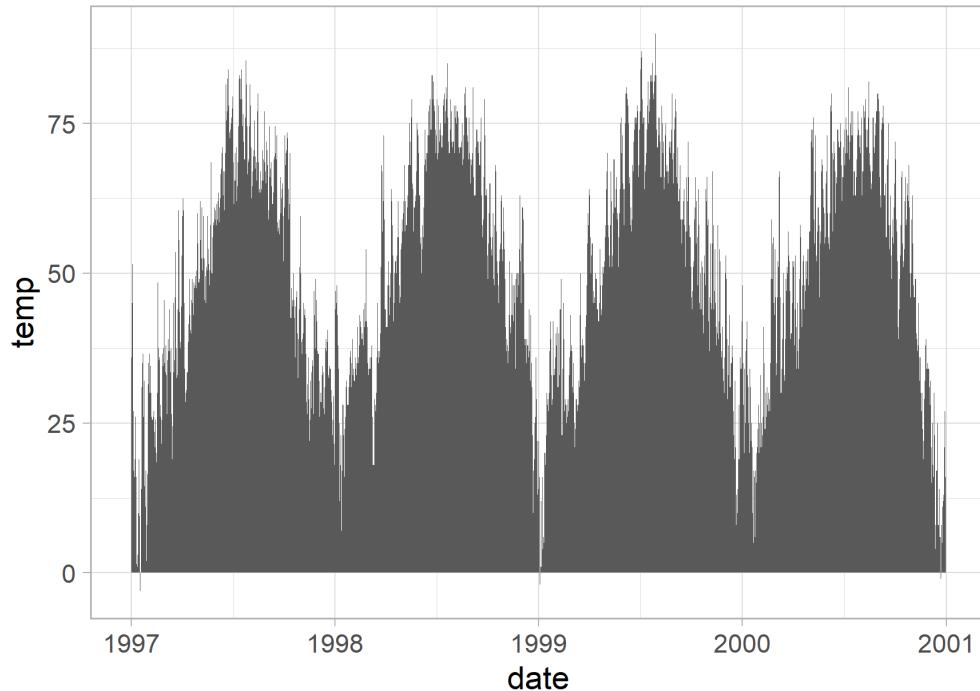


```
ggplot(chic, aes(date, temp)) +  
  geom_bar(stat = "identity")
```

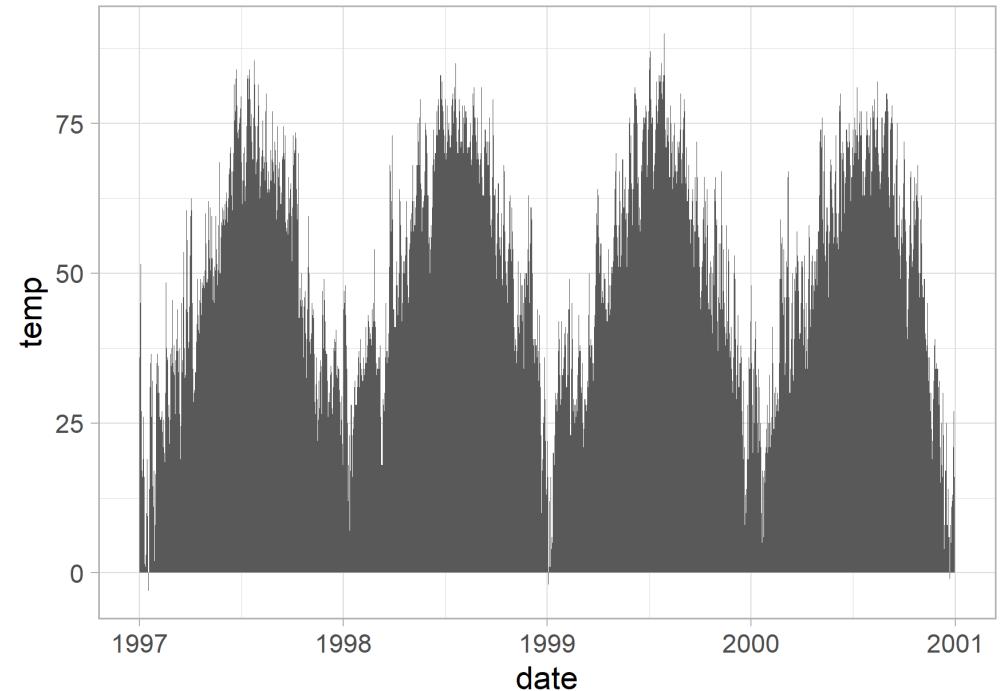


# `geom_bar()` versus `geom_col()`

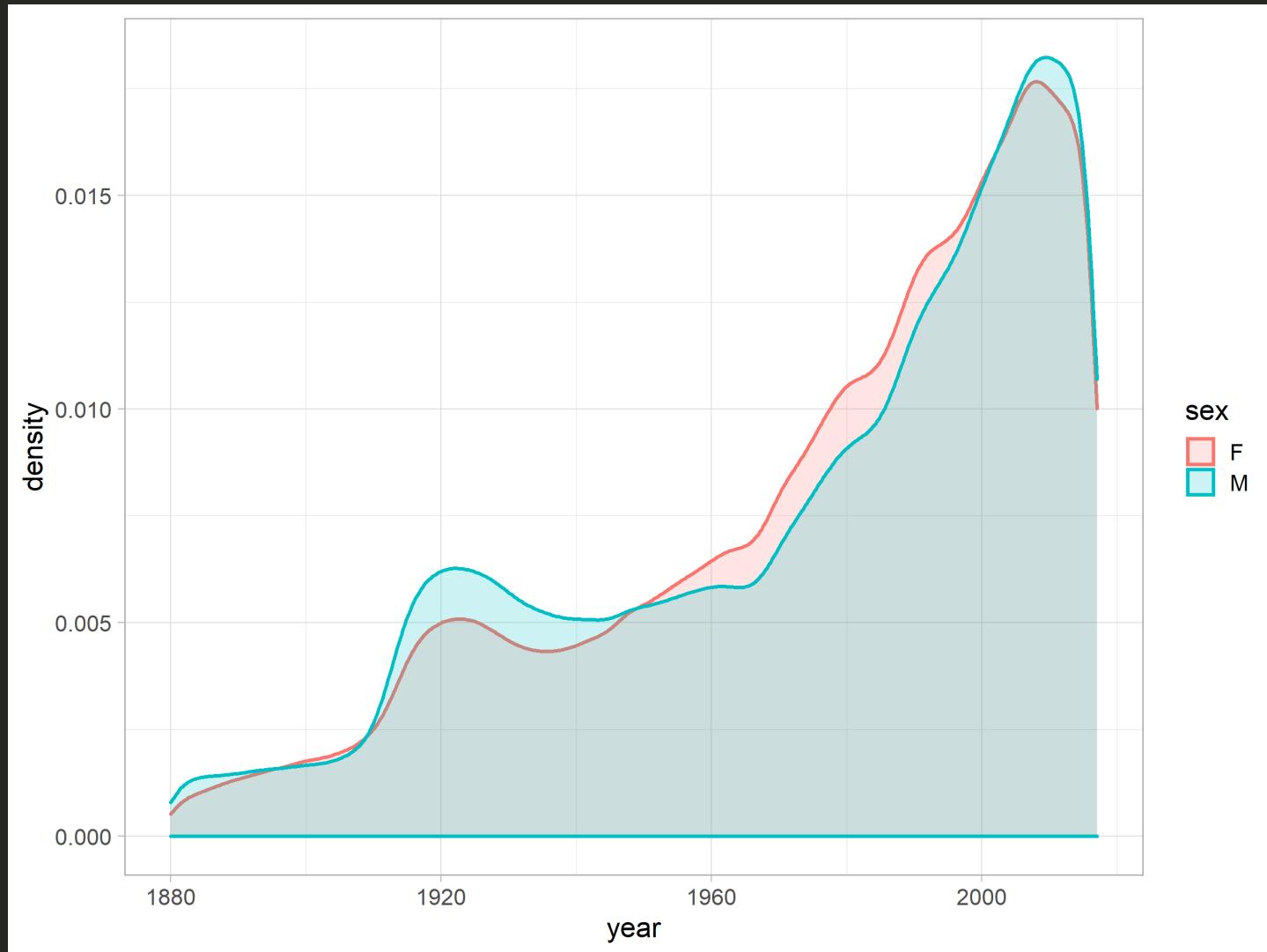
```
ggplot(chic, aes(date, temp)) +  
  geom_col()
```



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

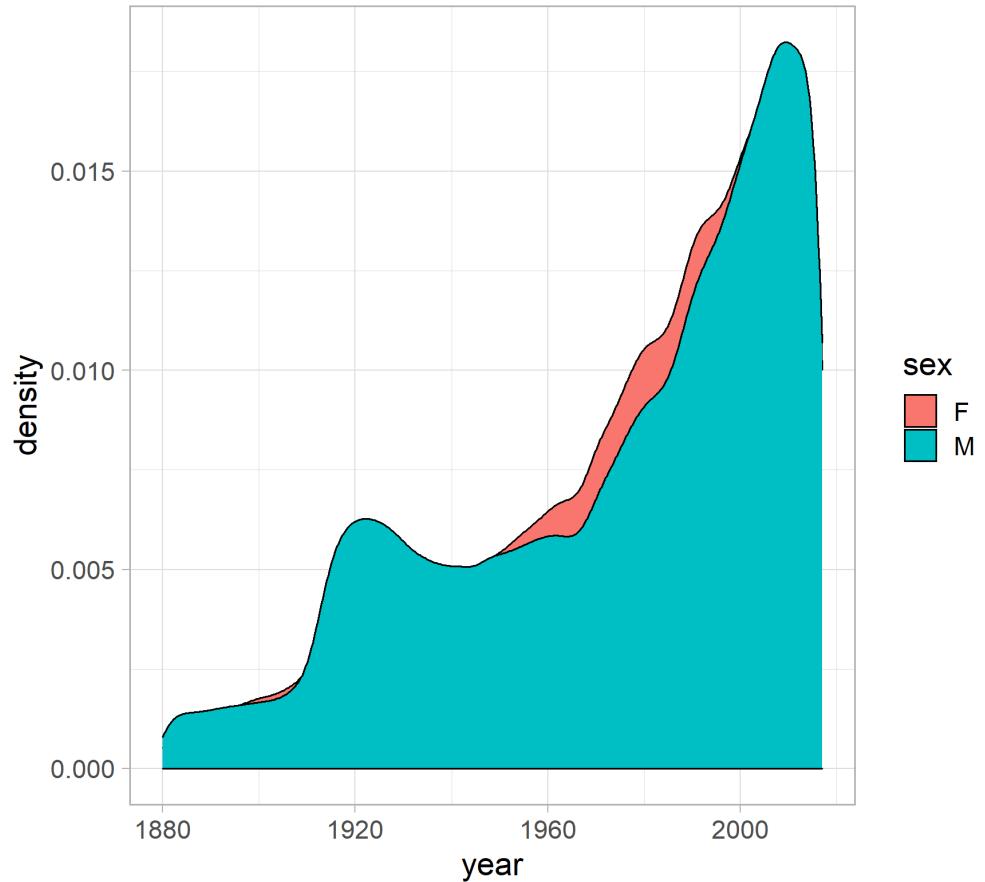


# Your Turn!



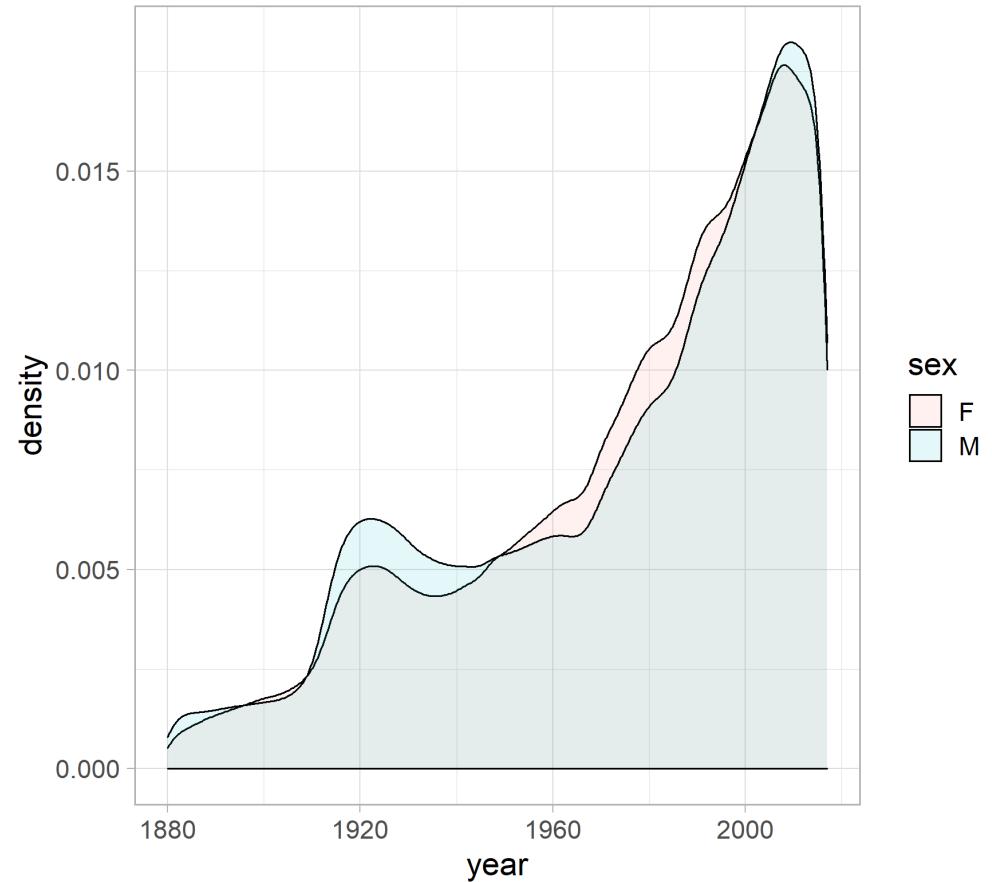
# Your Turn: Density Plots

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



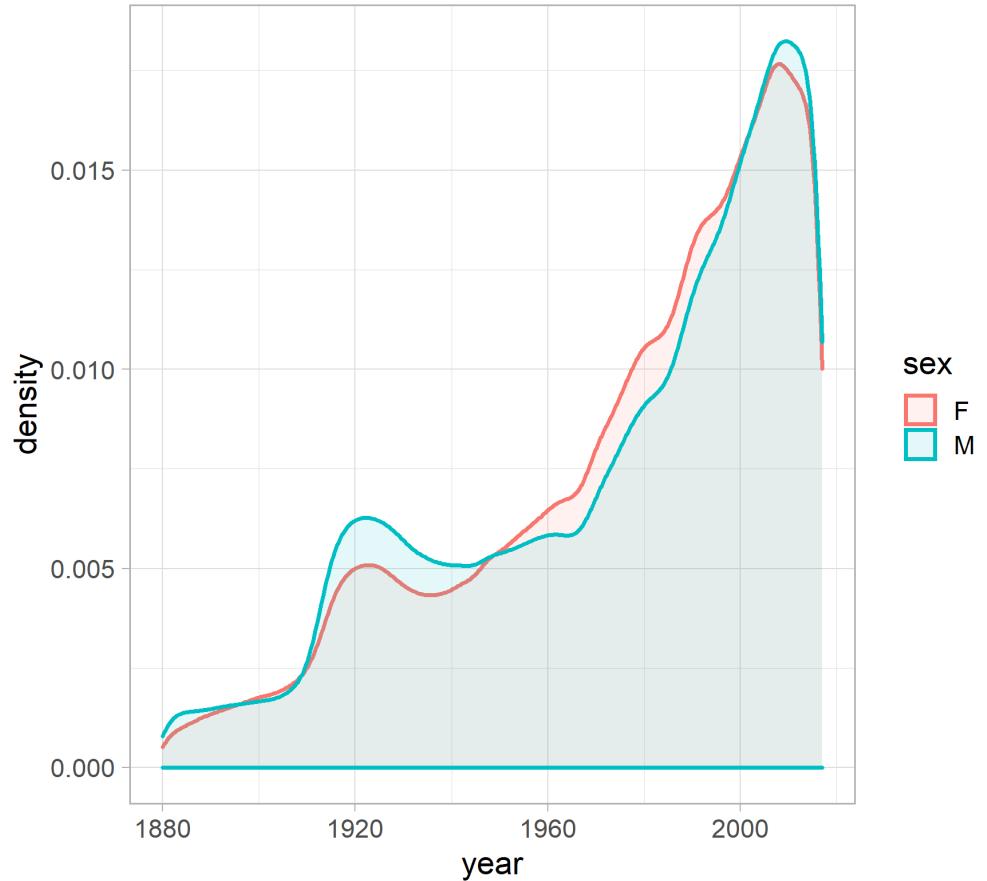
# Your Turn: Density Plots

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

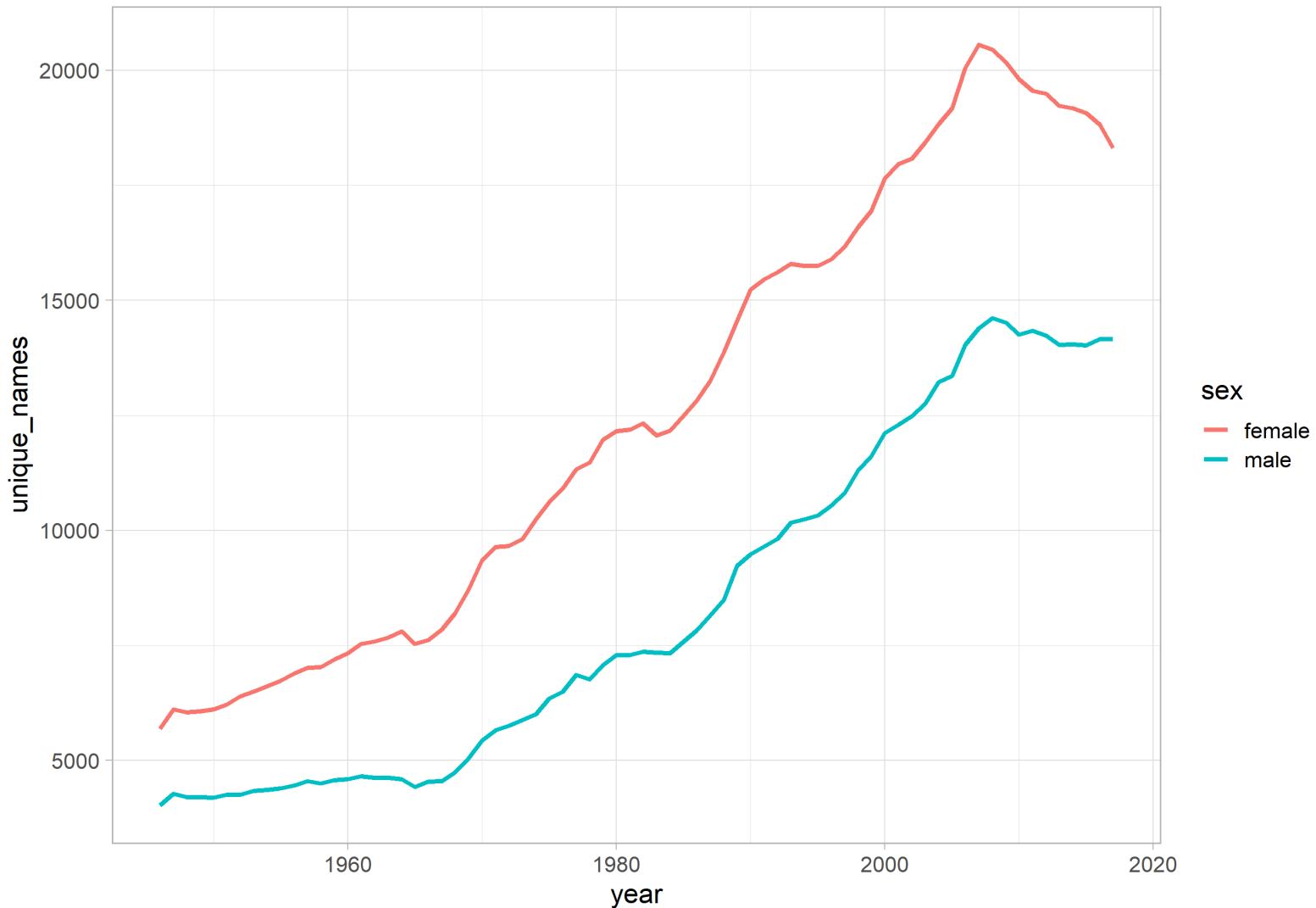


# Your Turn: Density Plots

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

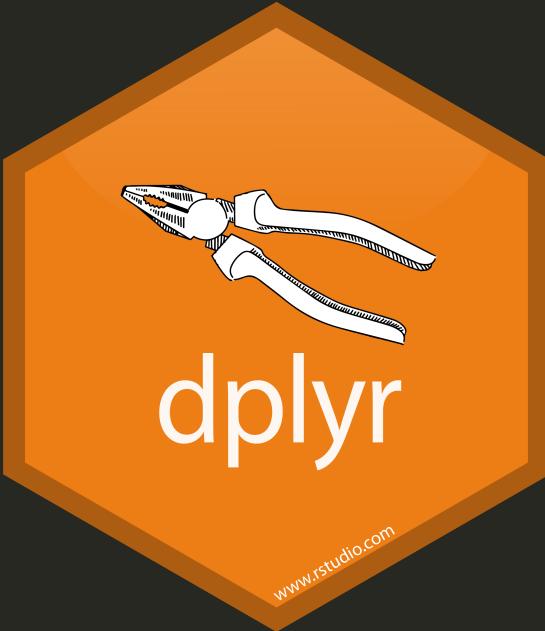


# The Rise of Unique Baby Names in the Post-WWII Era



# 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 until 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 convenient 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
<code>filter()</code>	Pick <b>rows</b> with matching criteria
<code>select()</code>	Pick <b>columns</b> with matching criteria
<code>arrange()</code>	Reorder rows
<code>mutate()</code>	Create new variables
<code>summarize()</code> (or <code>summarise()</code> )	Sum up variables
<code>group_by()</code>	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](https://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:

```
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>  <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)  
##  
## 1           model hwy cty diff      perc  
## 2           a4    29   18   11 0.3793103  
## 3           a4    29   21    8 0.2758621  
## 4           a4    31   20   11 0.3548387  
## 5           a4    30   21    9 0.3000000  
## 6           a4    26   16   10 0.3846154  
## 7           a4    26   18    8 0.3076923  
## 8           a4    27   18    9 0.3333333  
## 9           a4    quattro 26   18    8 0.3076923  
## 10          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
```

# 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
```

A stylized illustration of a superhero character with a dark blue mask and a red cape. The superhero is shown from the chest up, flying towards the viewer against a background of radiating grey and white lines resembling a sunburst or light rays.

# 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!

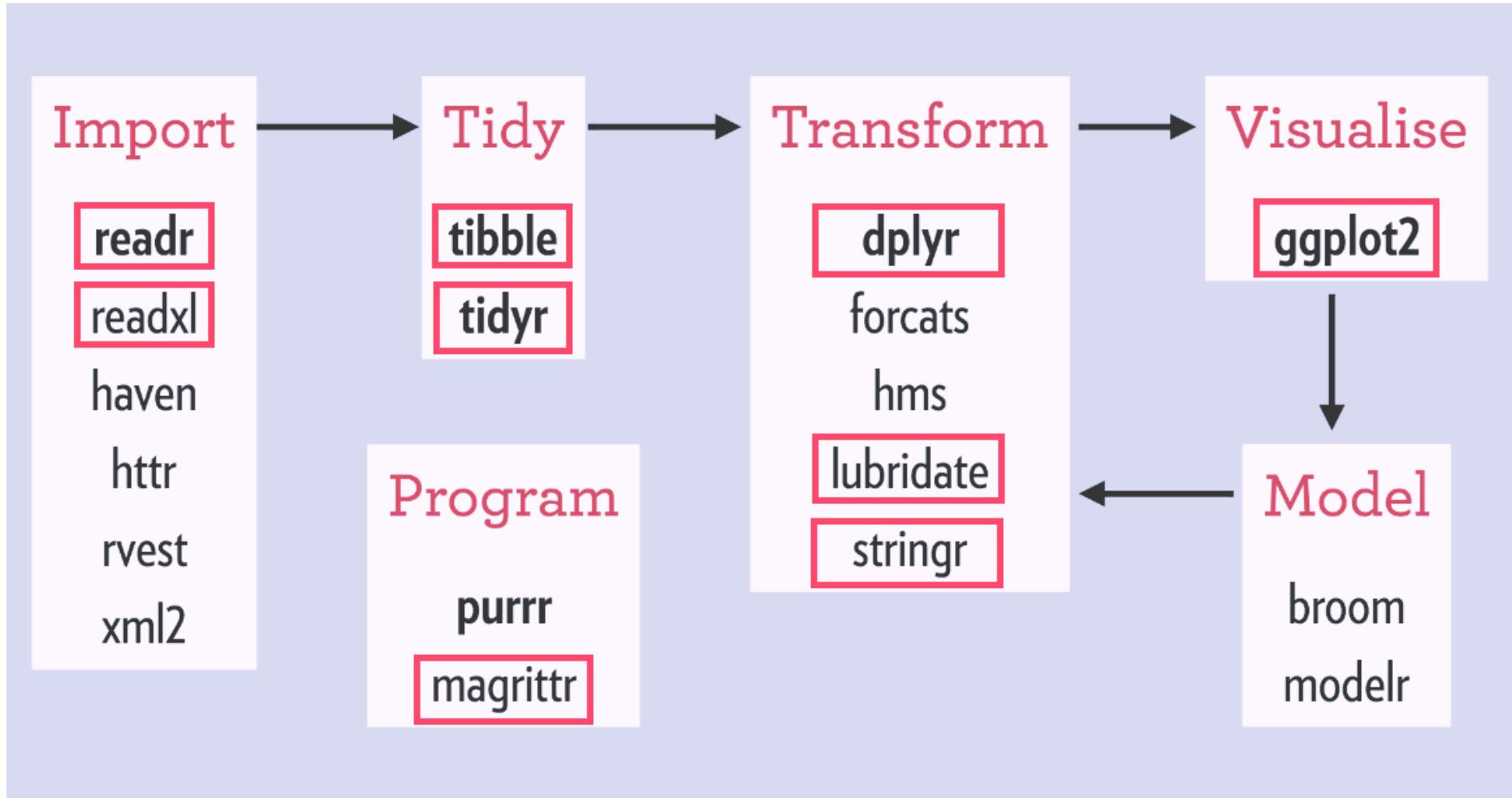
group\_by() in combination with other dplyr verbs allows you to calculate summary statistics for specified groups:

```
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
```



# The **magrittr** Package

# The Typical Data Science Project Flow

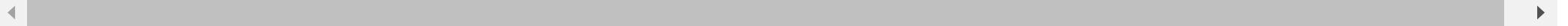


Source: [www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse\\_files/tidyverse6.png](http://www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse6.png)

# 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  
  )
```

# Readability of Code

One could rewrite the code like this:

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

```
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

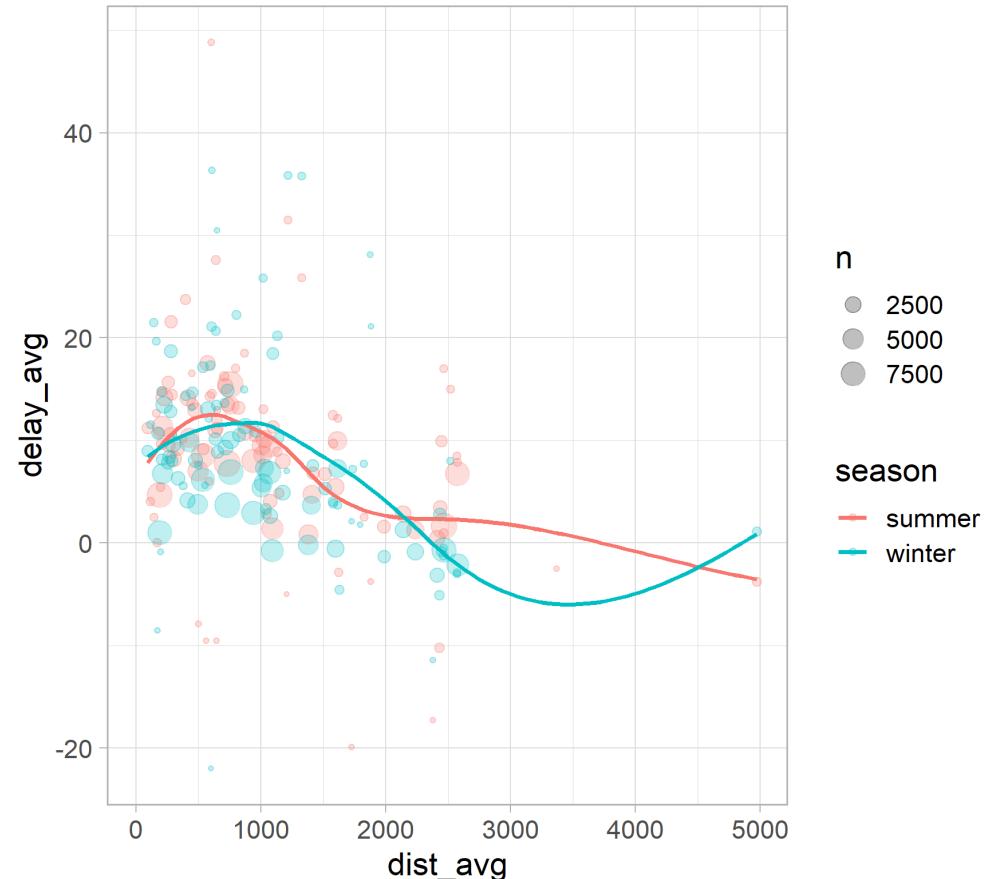
```
flights %>%
  group_by(month, origin) %>%
  summarize(
    dep_delay_avg = round(mean(dep_delay, na.rm = T), 1),
    arr_delay_avg = round(mean(arr_delay, na.rm = T), 1)
  ) %>%
  arrange(-arr_delay_avg)
## # A tibble: 36 x 4
## # Groups:   month [12]
##   month origin dep_delay_avg arr_delay_avg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

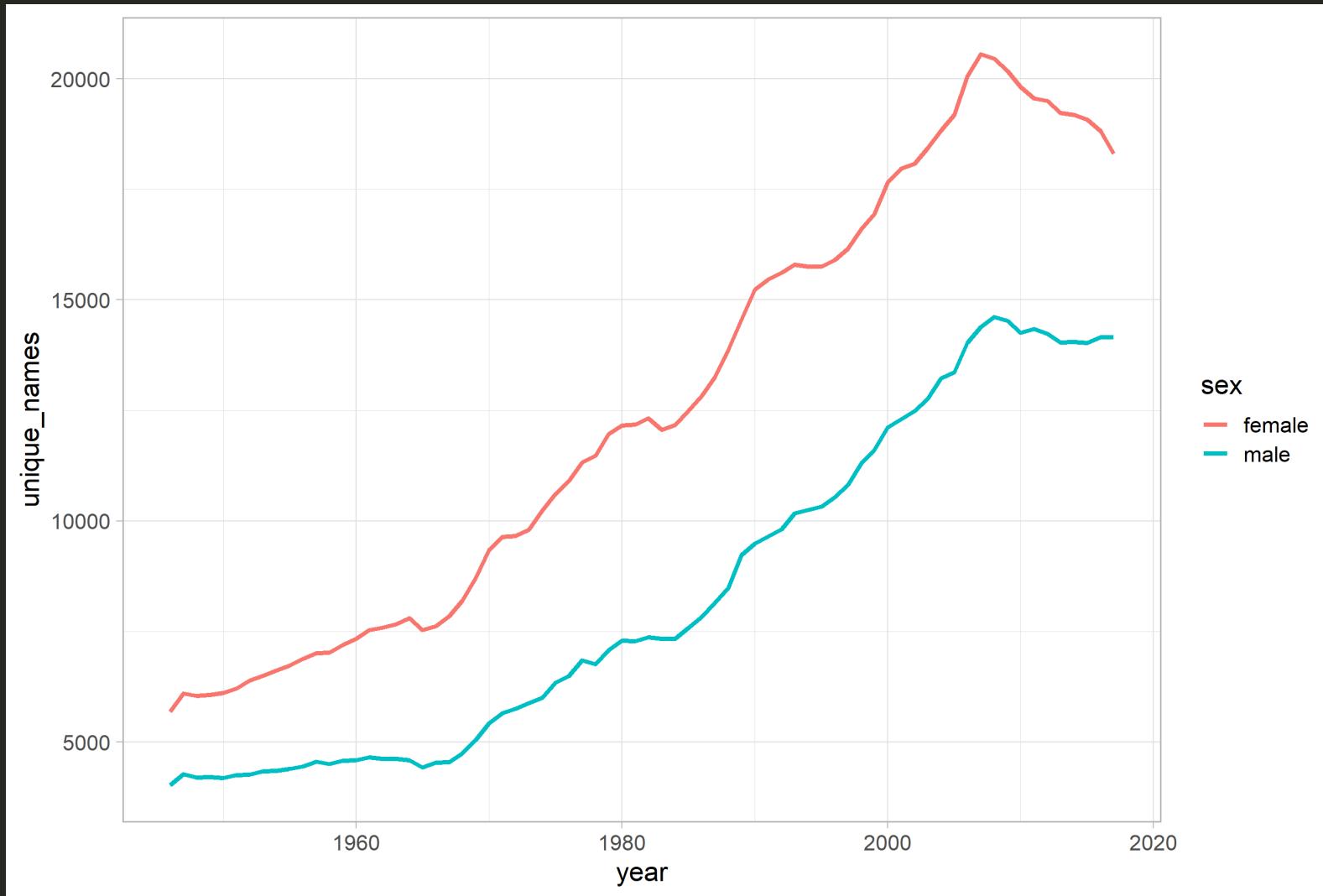
```
flights %>%
  select(month, origin, ends_with("delay")) %>%
  group_by(month, origin) %>%
  summarize_all(funs(avg = round(mean(., na.rm = T), 1))) %>%
  arrange(-arr_delay_avg)
## # A tibble: 36 x 4
## # Groups:   month [12]
##   month origin dep_delay_avg arr_delay_avg
##   <int> <chr>     <dbl>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

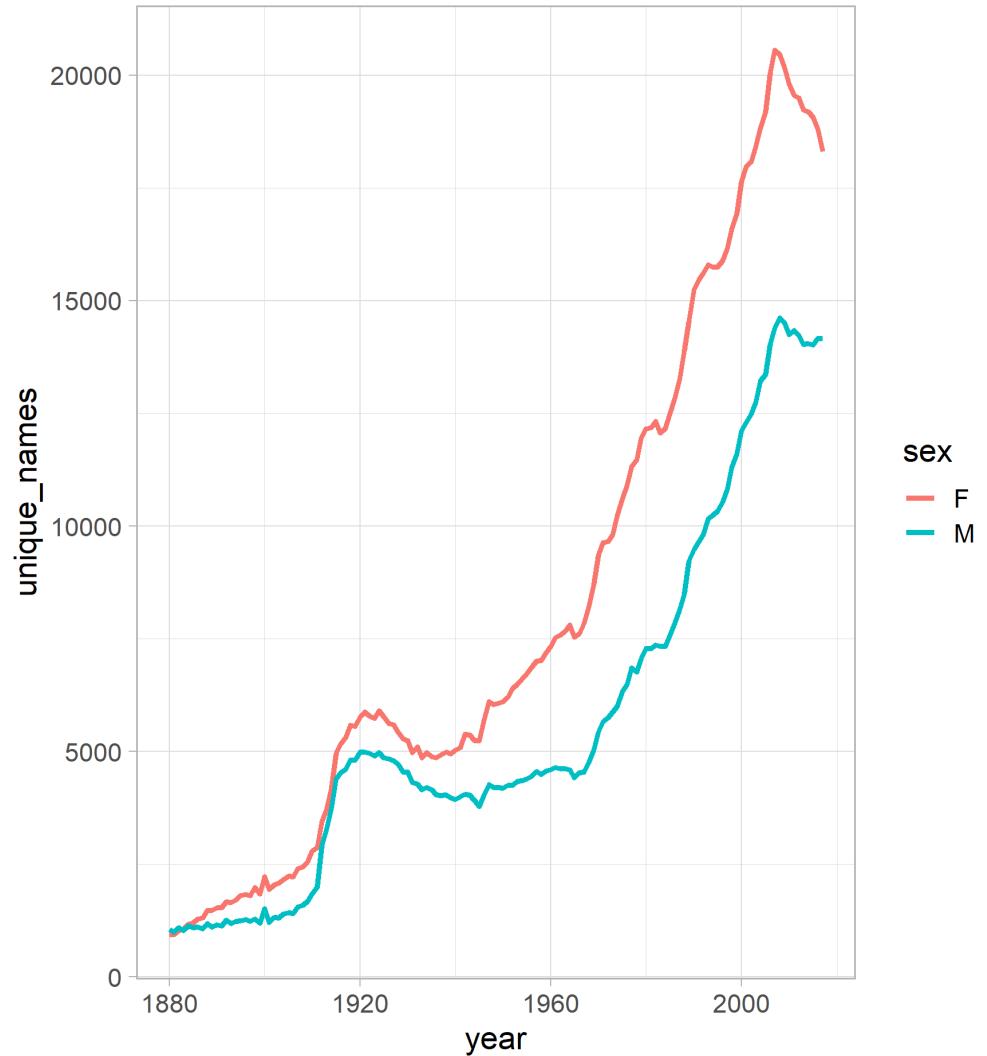


# Your Turn: Post-WWII Line Plot

```
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
```

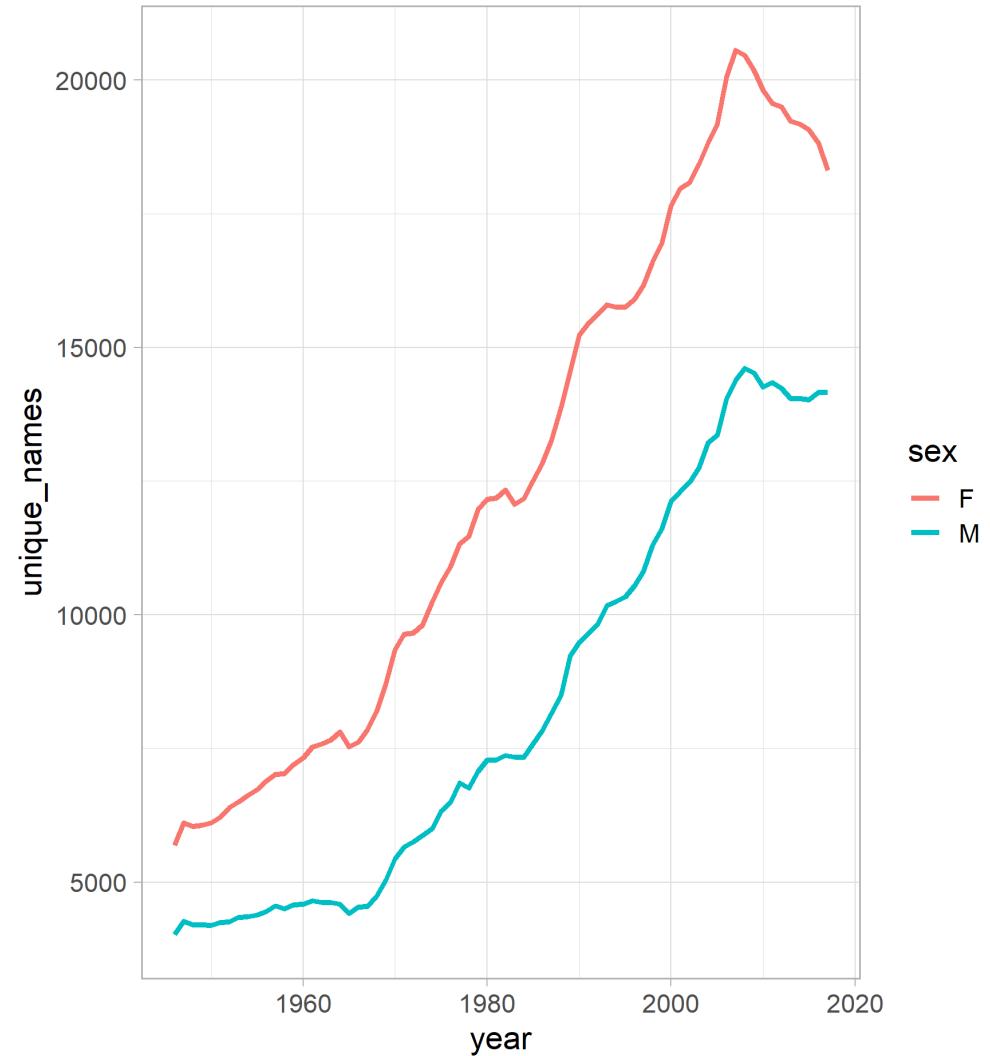
# Your Turn: Post-WWII Line Plot

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



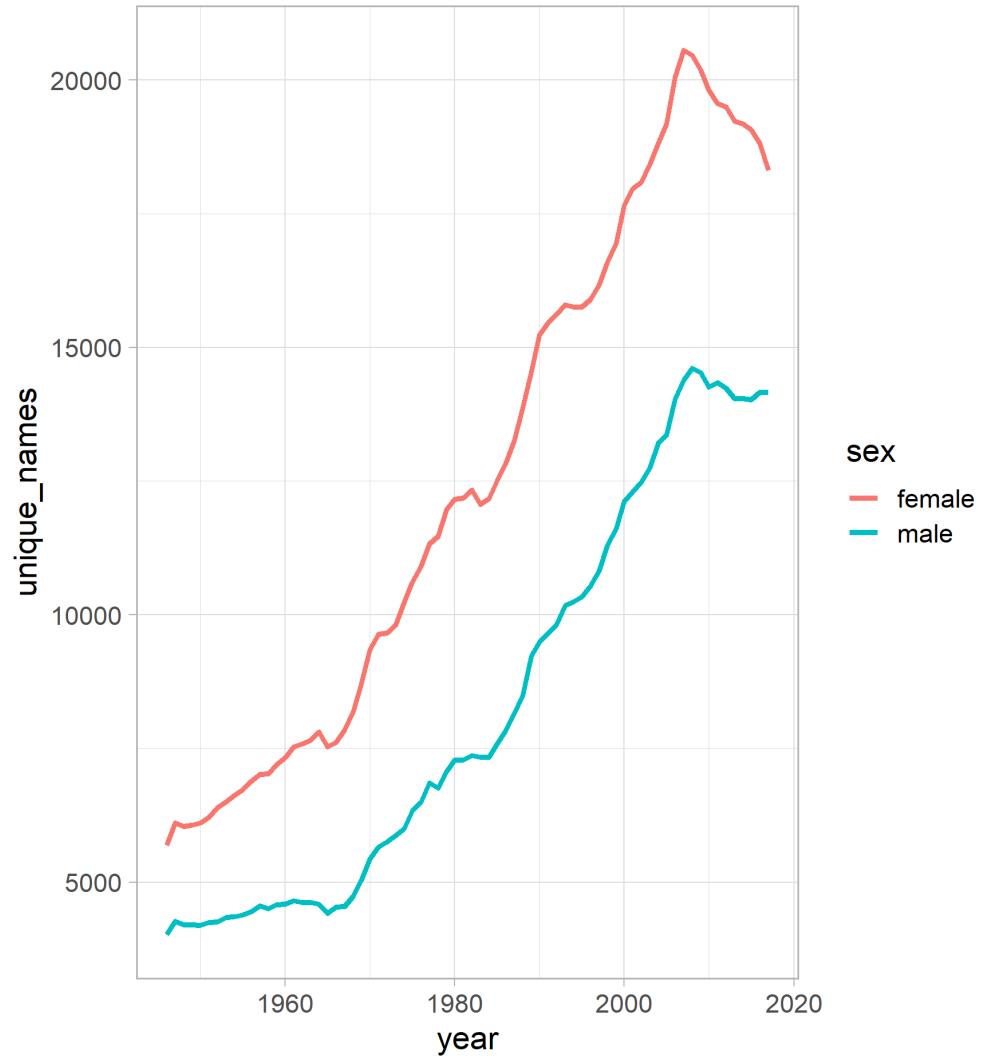
# Your Turn: Post-WWII Line Plot

```
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
<code>slice()</code>	Extract rows by ordinal position
<code>distinct()</code> and <code>n_distinct()</code>	Find and count unique values
<code>sample_n()</code> and <code>sample_frac()</code>	Select rows randomly
<code>top_n()</code> and <code>top_frac()</code>	Pick top or bottom values by variable
<code>count()</code> and <code>top_frac()</code>	Count number of observations
<code>complete()</code>	Create all possible combinations of two variables

# 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 land rover   range r~    4.4  2008     8 auto(s~ 4          12    18 r   SUV
## 2 dodge        caravan~ 3.3   1999     6 auto(l~ f          16    22 r   miniv~
## 3 toyota        corolla 1.8   2008     4 auto(l~ f          26    35 r   compa~
## 4 mercury       mountai~ 4     1999     6 auto(l~ 4          14    17 r   SUV
## 5 honda         civic   1.6   1999     4 manual~ f          25    32 r   subco~
```

# 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    corvette 5.7   1999     8 auto(~ r     15    23 p   2seat~
## 3 dodge        dakota ~ 5.2   1999     8 auto(~ 4    11    15 r   pickup
## 4 ford         expedit~ 5.4   1999     8 auto(~ r    11    17 r   suv
## 5 honda        civic   1.6   1999     4 manua~ f    23    29 p   subco~
## 6 hyundai      sonata  2.4   1999     4 auto(~ f    18    26 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    17 r   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 volkswagen  jetta     1.9   1999     4 manua~ f       33    44 d   comp~
## 2 dodge       durango ~ 5.7   2008     8 auto(~ 4       13    18 r   suv 
## 3 hyundai     sonata    2.5   1999     6 auto(~ f       18    26 r   mids~
## 4 audi        a4 quatt~ 2     2008     4 manua~ 4       20    28 p   comp~
## 5 nissan      altima    3.5   2008     6 auto(~ f       19    26 p   mids~
## 6 nissan      altima    2.5   2008     4 auto(~ f       23    31 r   mids~
## 7 volkswagen  passat    2     2008     4 manua~ f       21    29 p   mids~
## 8 dodge       dakota p~ 3.7   2008     6 auto(~ 4       14    18 r   pick~
## 9 dodge       durango ~ 5.2   1999     8 auto(~ 4       11    16 r   suv 
## 10 jeep       grand ch~ 5.7   2008     8 auto(~ 4       13    18 r   suv 
## 11 toyota     corolla   1.8   1999     4 auto(~ f       24    33 r   comp~
## 12 toyota     camry     2.2   1999     4 manua~ f       21    29 r   mids~
```

# Pick Top or Bottom Values

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

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

# Count Observations

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

```
#count(mpg)
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 statistics - 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 statistics - 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

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

# Count Observations

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

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

# Create all possible combinations of two variables

`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 possible combinations of two variables

`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
```

# Data Transformation with dplyr::

A guide to 37<sup>\*</sup> different behaviours applied to one tibble (tbl)



© R Data Berlin

