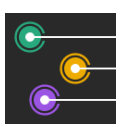
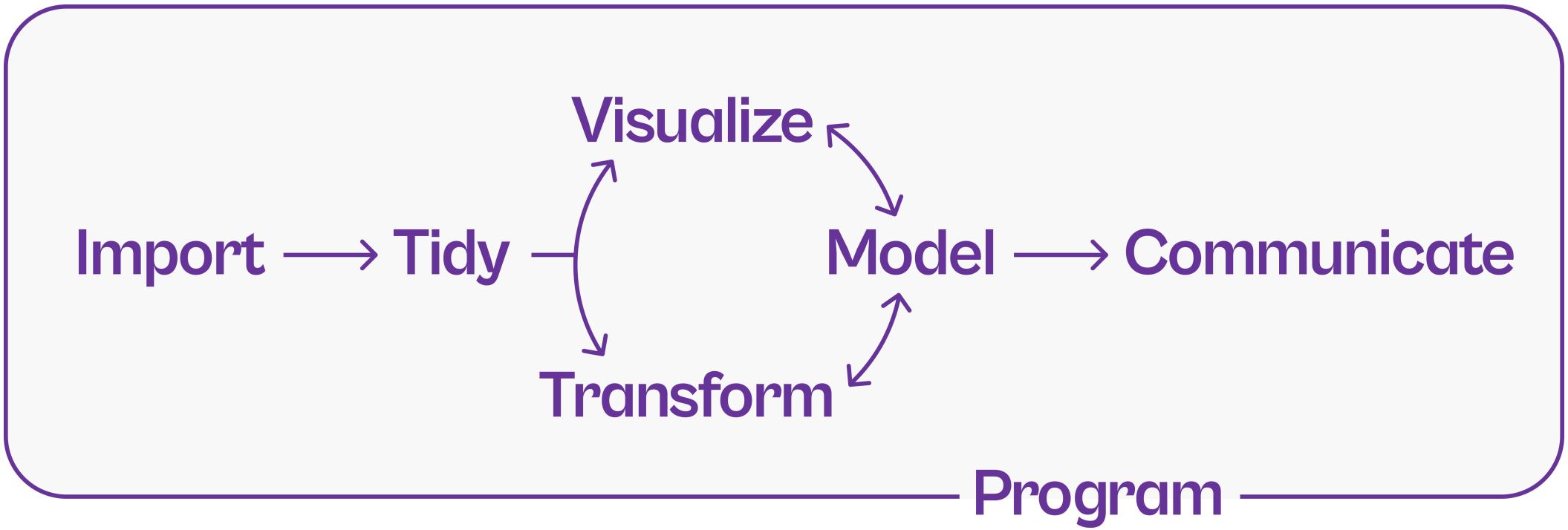
Reproducible Data Analysis with R

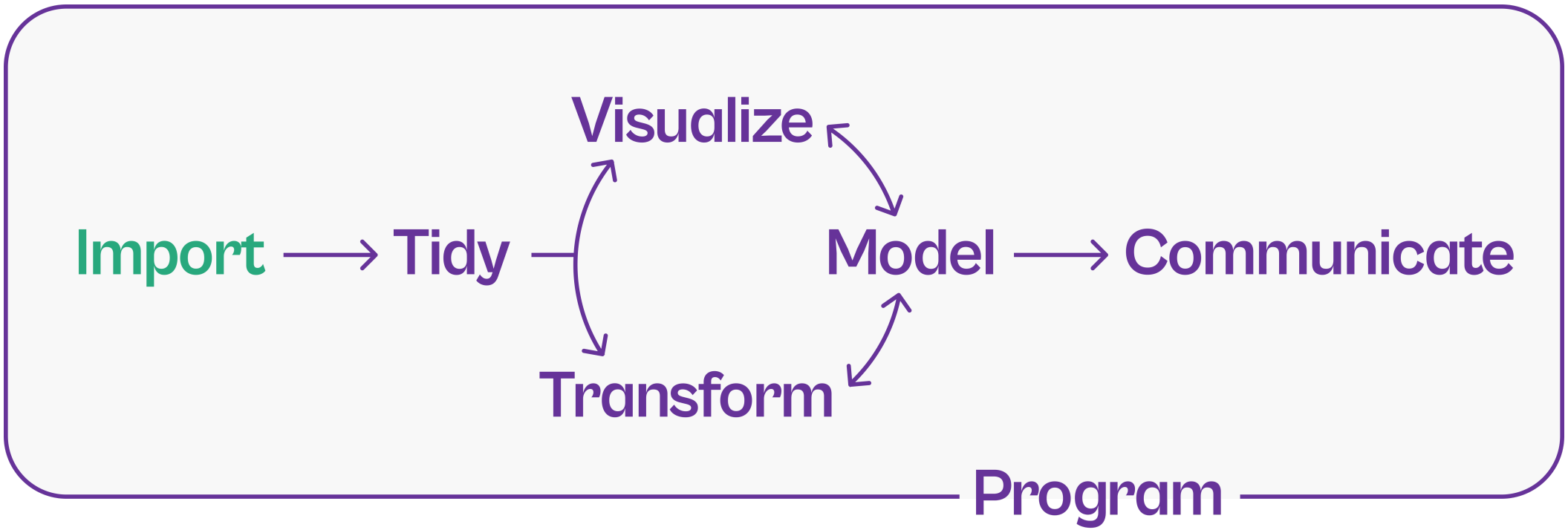
— Data Wrangling with the {tidyverse} —

Cédric Scherer // R Course TU Dresden // Feb 27–Mar 3, 2023

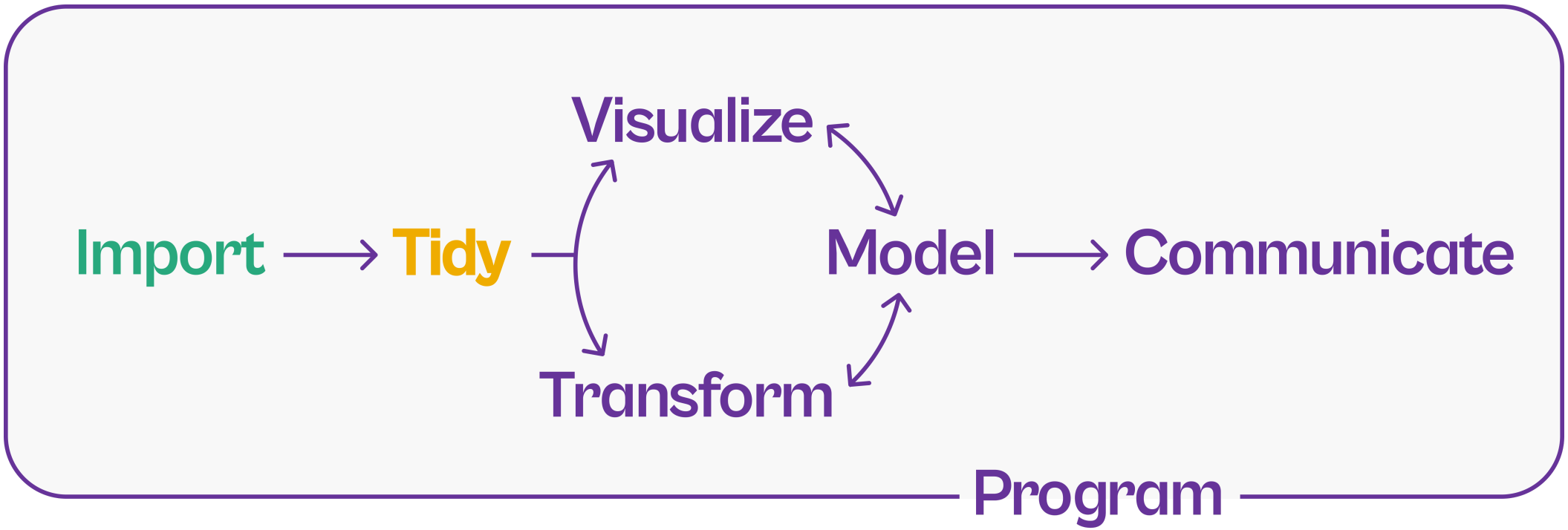




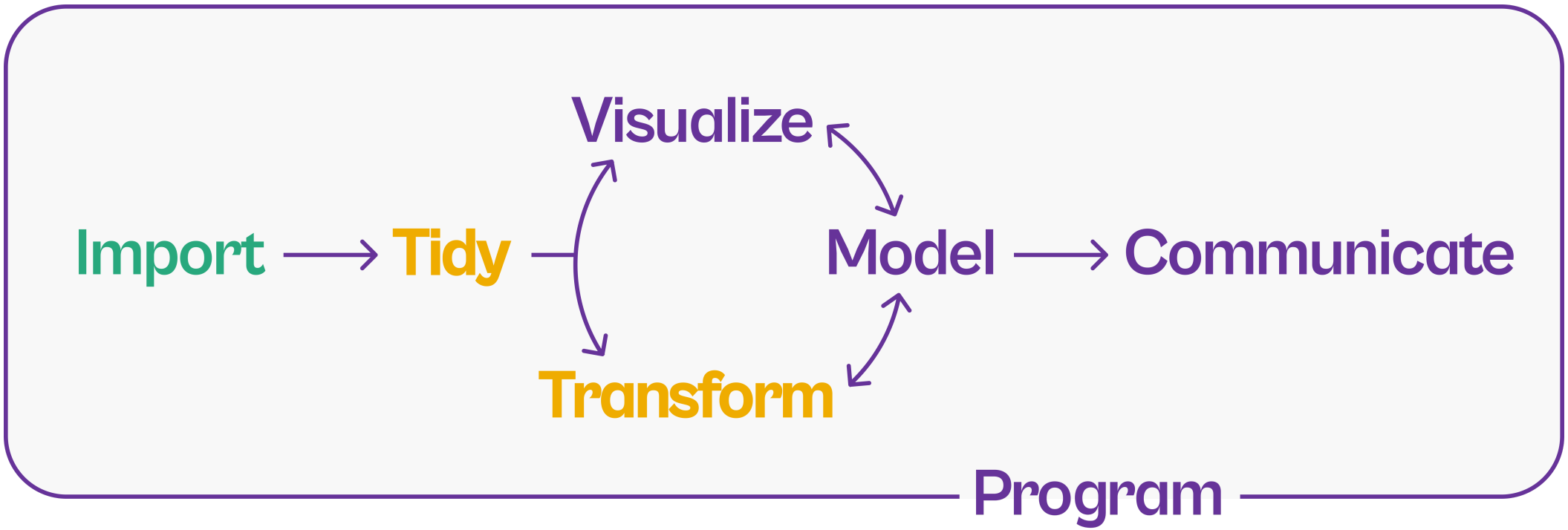
The data science workflow, modified from "R for Data Science"



The data science workflow, modified from "R for Data Science"

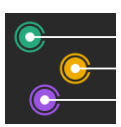


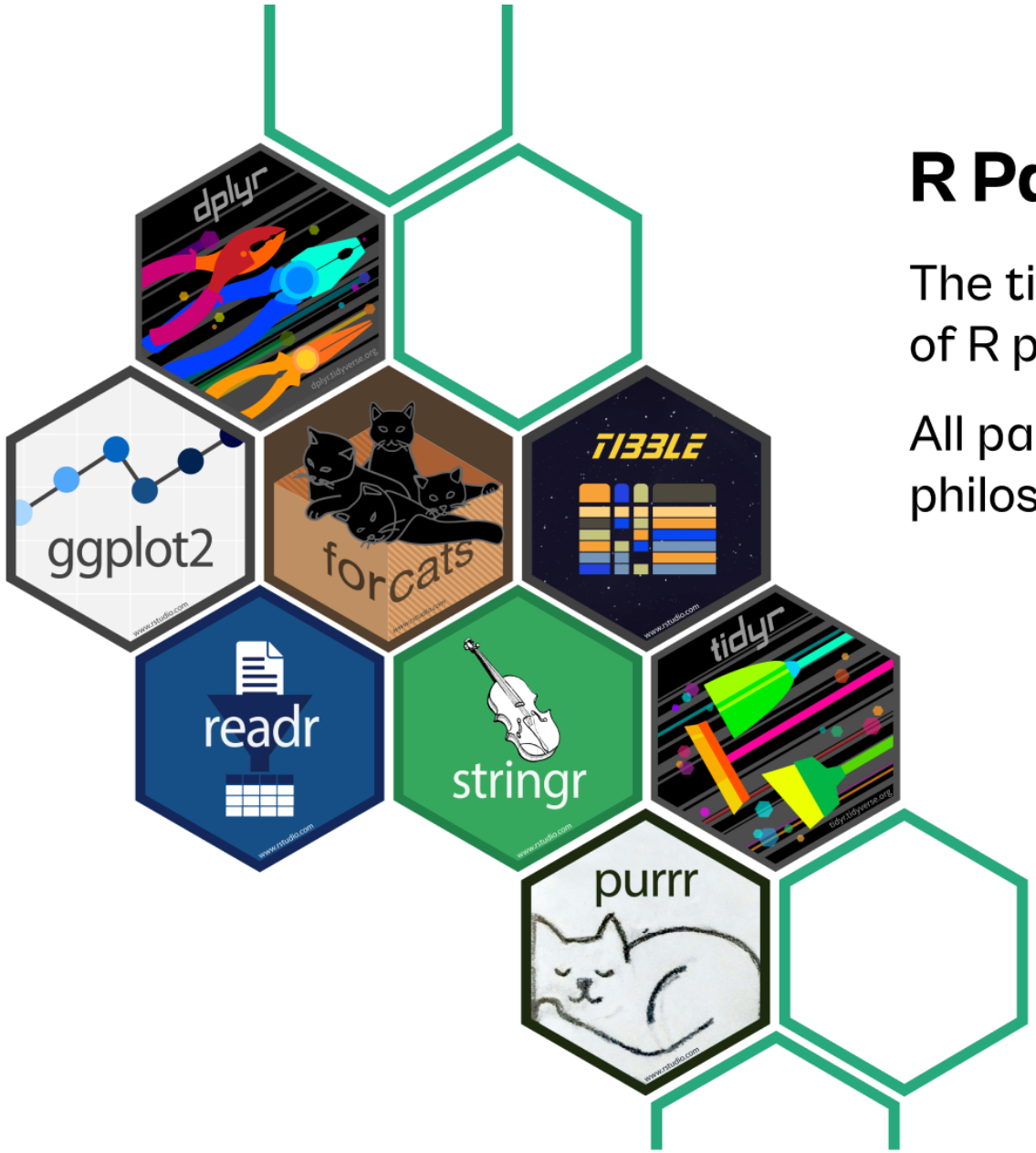
The data science workflow, modified from "R for Data Science"



The data science workflow, modified from "R for Data Science"

Data Wrangling with the {tidyverse}



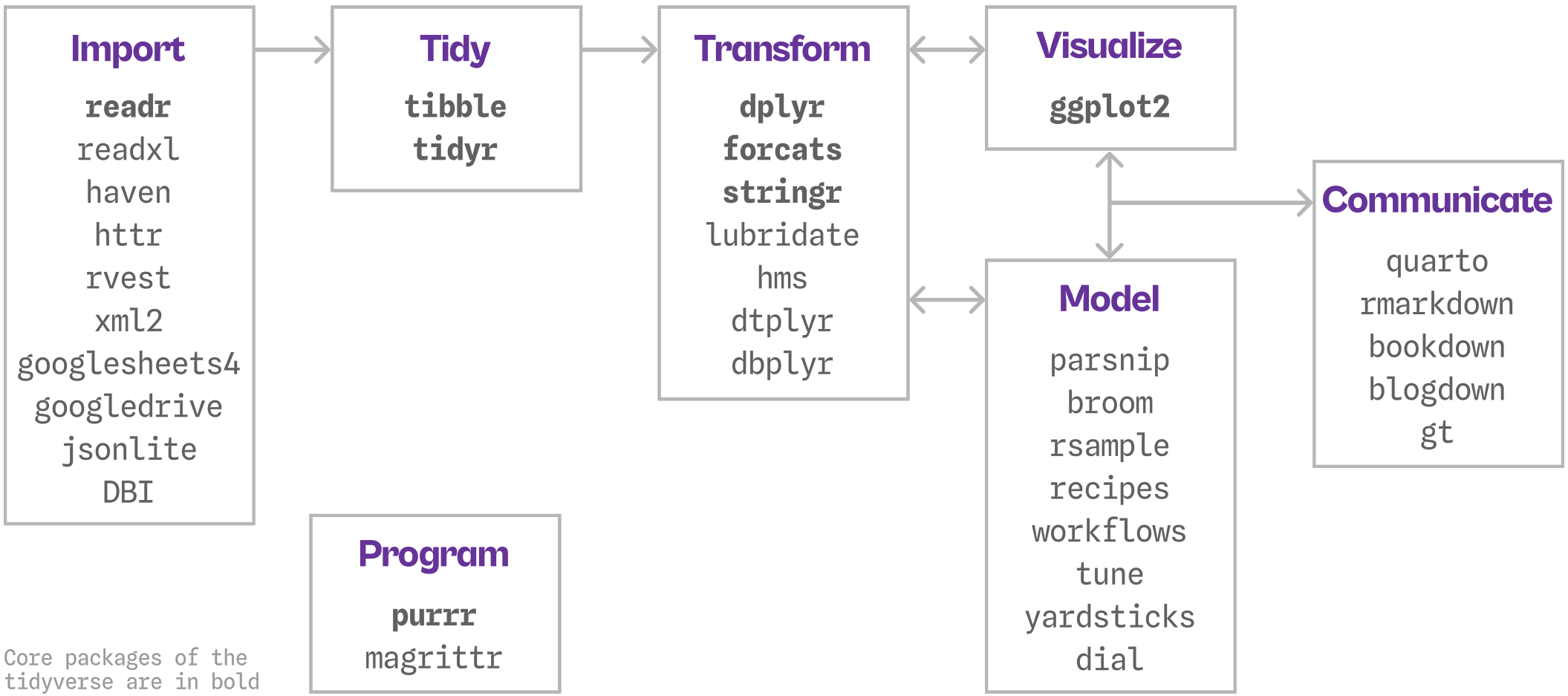


R Packages for Data Science

The tidyverse is an opinionated collection of R packages designed for data science.

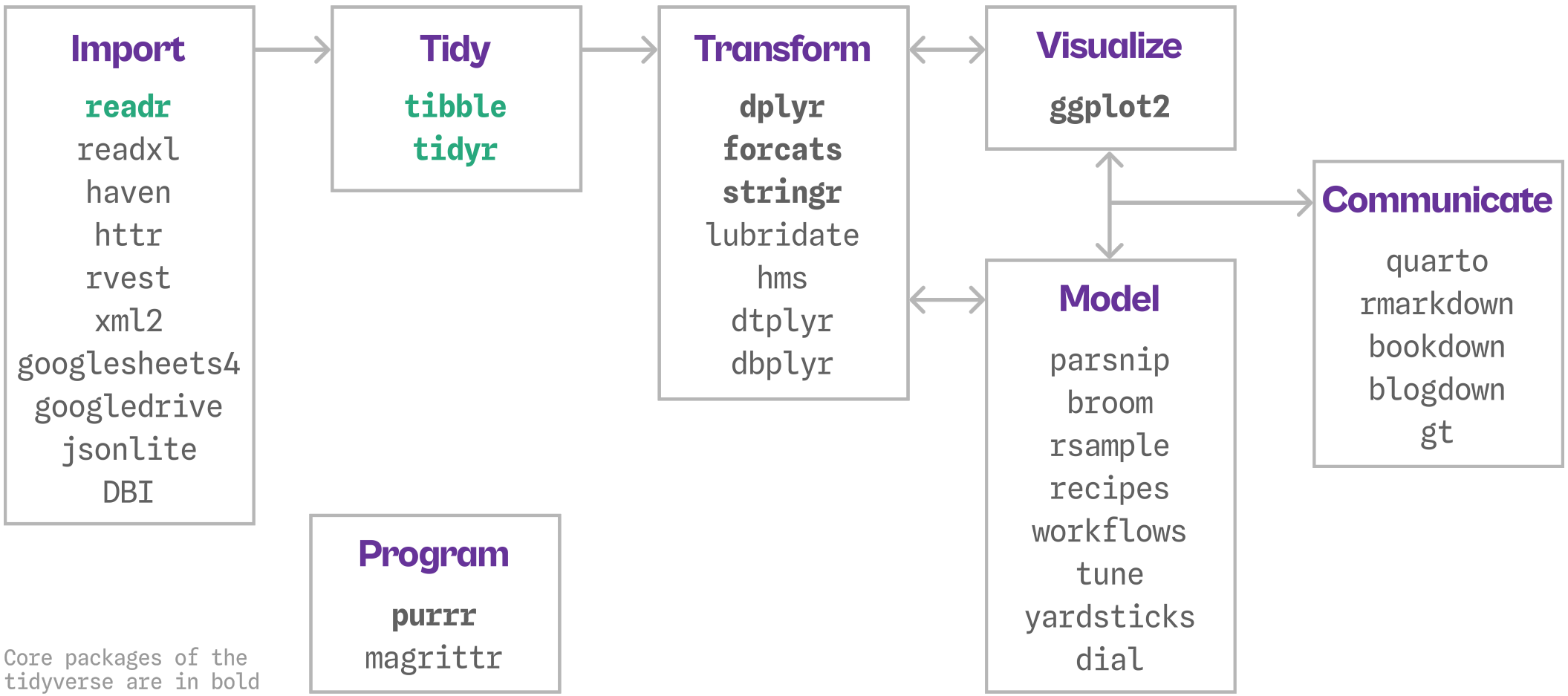
All packages share an underlying design philosophy, grammar, and data structures.

The {tidyverse}

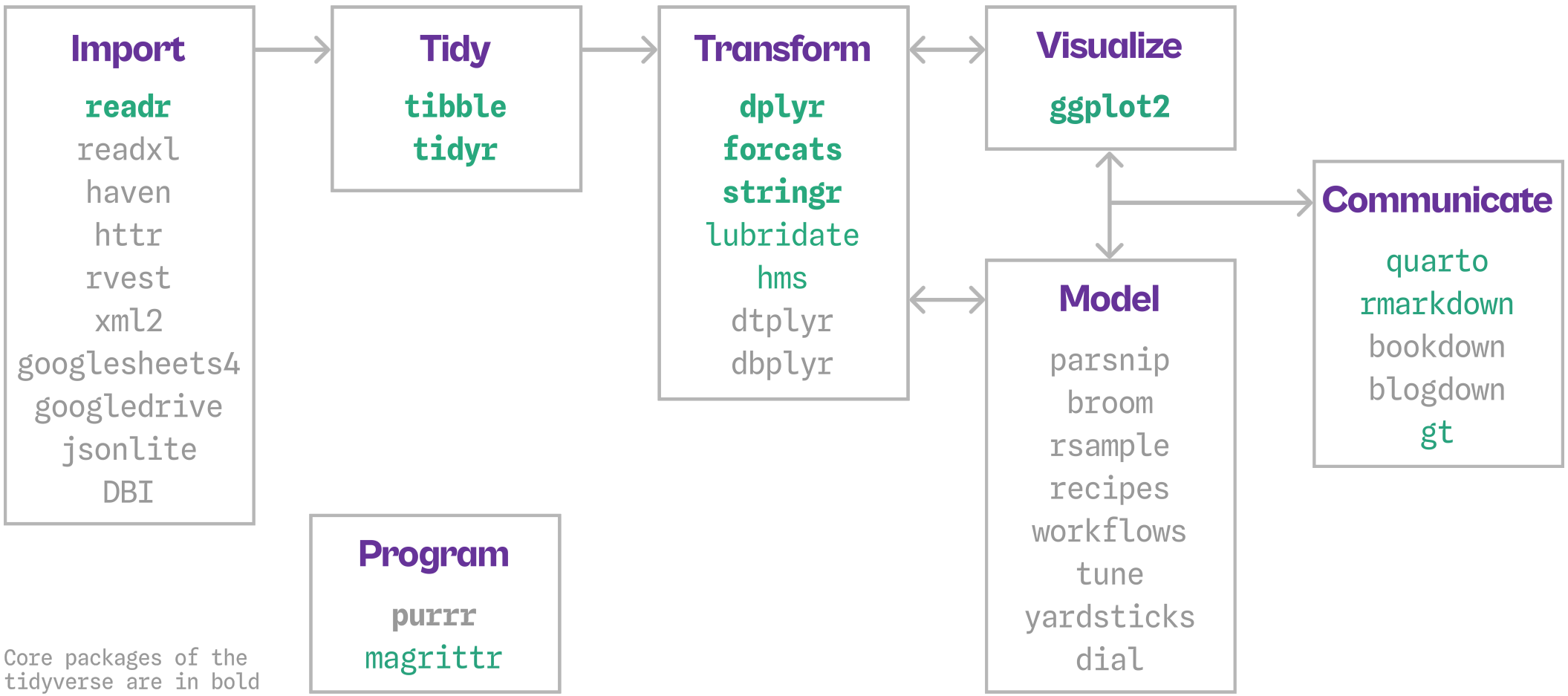


Core packages of the tidyverse are in bold

The {tidyverse}



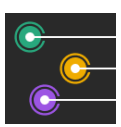
The {tidyverse}



Core packages of the tidyverse are in bold

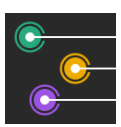
Data Wrangling

— The Setup —



Our Example Data: `babynames`

- 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 `{babynames}` package: `install.packages("babynames")` (includes 3 more data sets)
- See also `??babynames`



Our Example Data: babynames

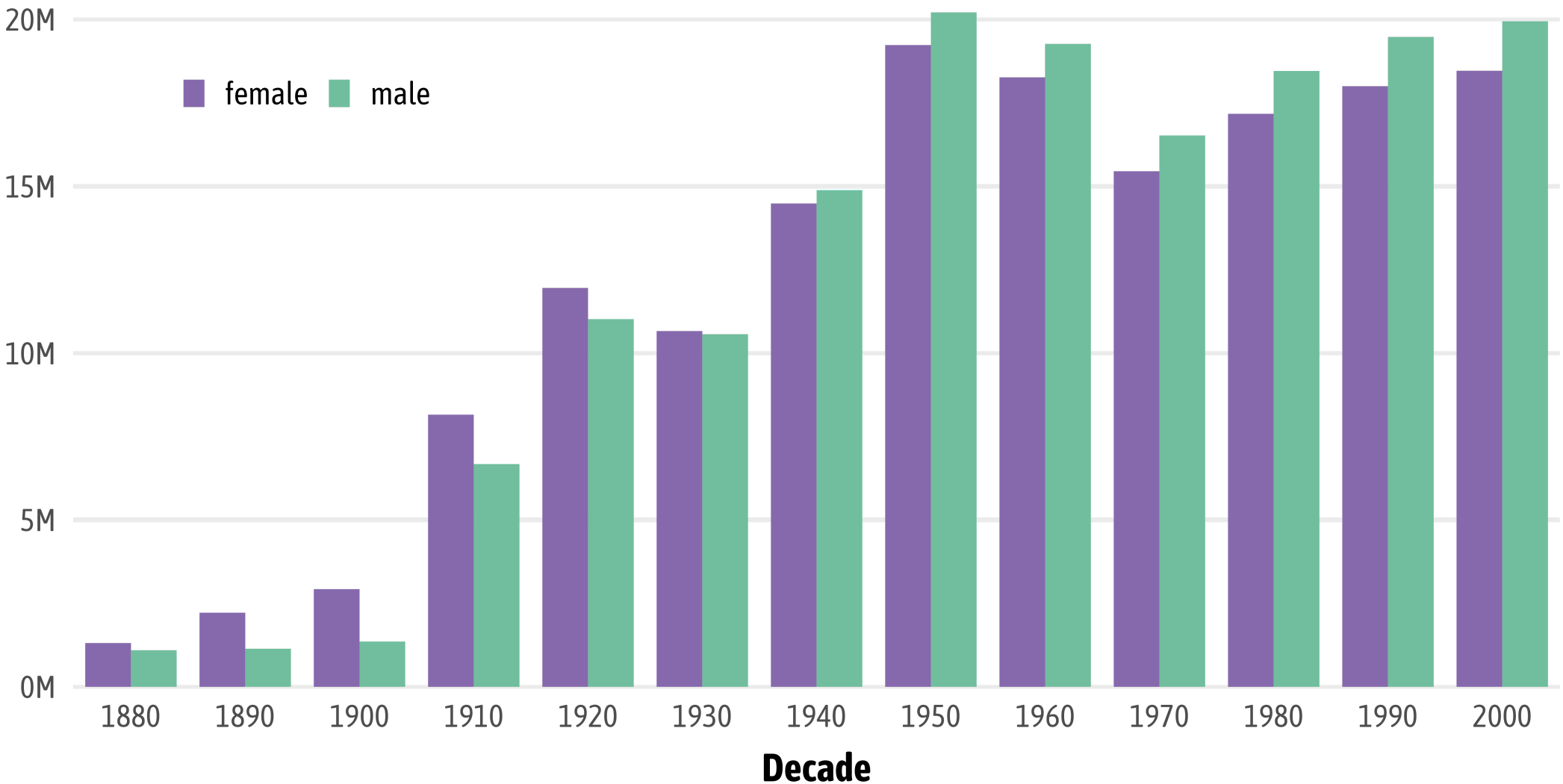
```
1 # install.packages("babynames")
2 library(babynames)
```

```
1 tibble::glimpse(babynames)
```

```
Rows: 1,924,665
Columns: 5
$ year <dbl> 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880,…
$ sex  <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", …
$ name <chr> "Mary", "Anna", "Emma", "Elizabeth", "Minnie", "Margaret", "Ida", "Alice", "Bertha", "Sarah", "…
$ n    <int> 7065, 2604, 2003, 1939, 1746, 1578, 1472, 1414, 1320, 1288, 1258, 1226, 1156, 1063, 1045, 1040,…
$ prop <dbl> 0.07238359, 0.02667896, 0.02052149, 0.01986579, 0.01788843, 0.01616720, 0.01508119, 0.01448696,…
```


Reported Babies in the US between 1880 and 2009 per Decade and Sex

While in the beginning considerably more **girls** were reported, slightly more **boys** were recorded since the '40s.



Source: US Social Security Administration (SSA)

Excourse: Integer Division

```
1 1 %/% 10
```

```
[1] 0
```

```
1 11 %/% 10
```

```
[1] 1
```

```
1 17 %/% 10
```

```
[1] 1
```

```
1 2019 %/% 10
```

```
[1] 201
```

```
1 2020 %/% 10
```

```
[1] 202
```

```
1 2019 %/% 10 * 10
```

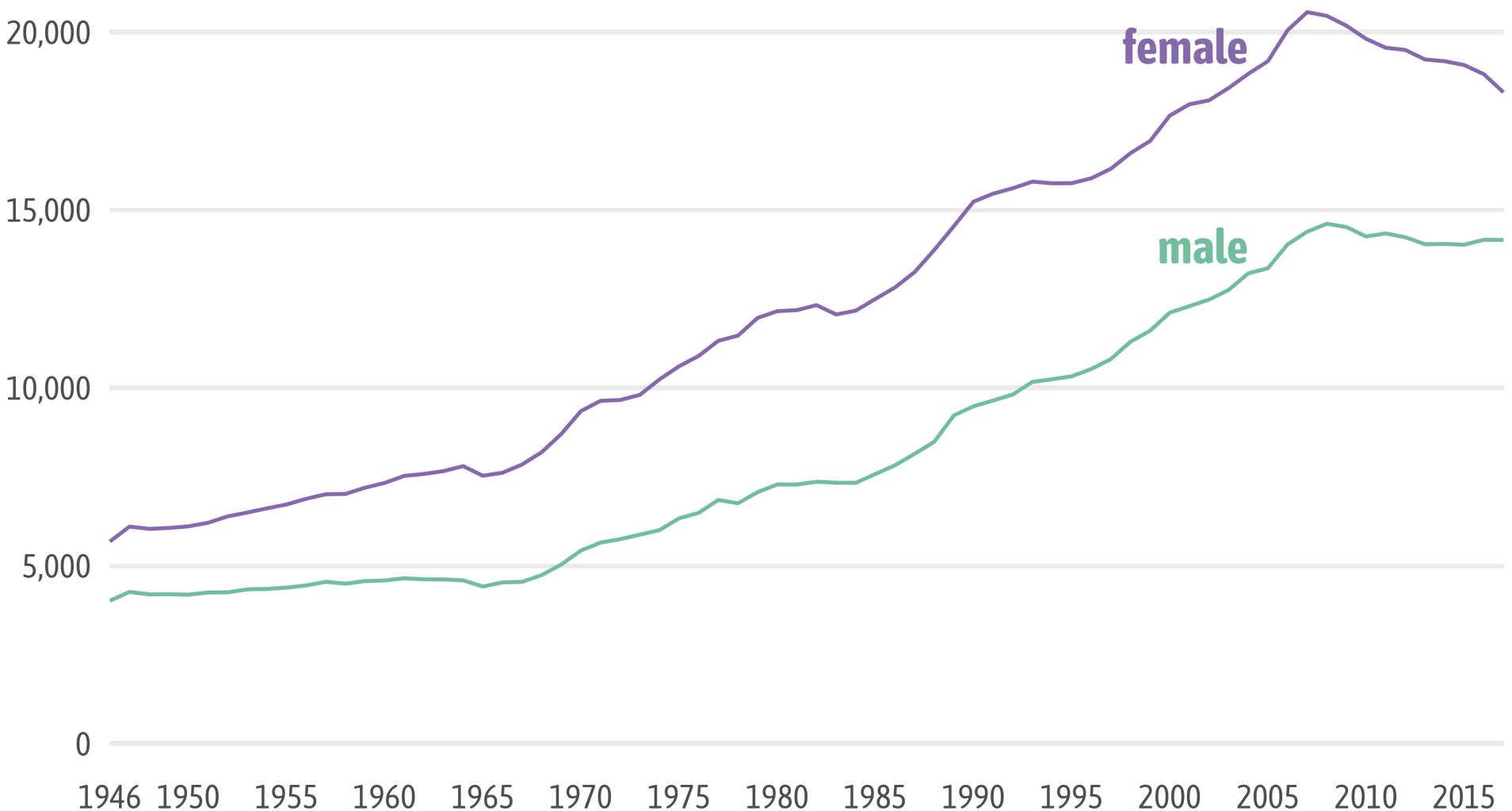
```
[1] 2010
```

```
1 2020 %/% 10 * 10
```

```
[1] 2020
```


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

Number of unique baby names in the US since 1946 with at least five records per year



Source: US Social Security Administration (SSA)

The {tidyverse} Package Collections

The **{tidyverse}** package collection installs all core packages:

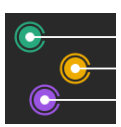
```
1 install.packages("tidyverse")
```

installs `tibble`, `tidyr`, `dplyr`, `forcats`, `stringr`, `ggplot2`, and `purrr`
and their dependency packages

→ often trouble with package versions, thus we install each package on its own

Data Wrangling

— with the {dplyr} Package —

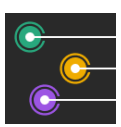


dplyr : go wrangling

Illustration by Allison Horst

Why should I care about {dplyr}?

- contains a set of convenient functions to perform common transformation and summary operations
- compared to base R functions:
  - 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}

Verb(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.

Load {dplyr}

```
1 # install.packages("dplyr")
2 library(dplyr)
```

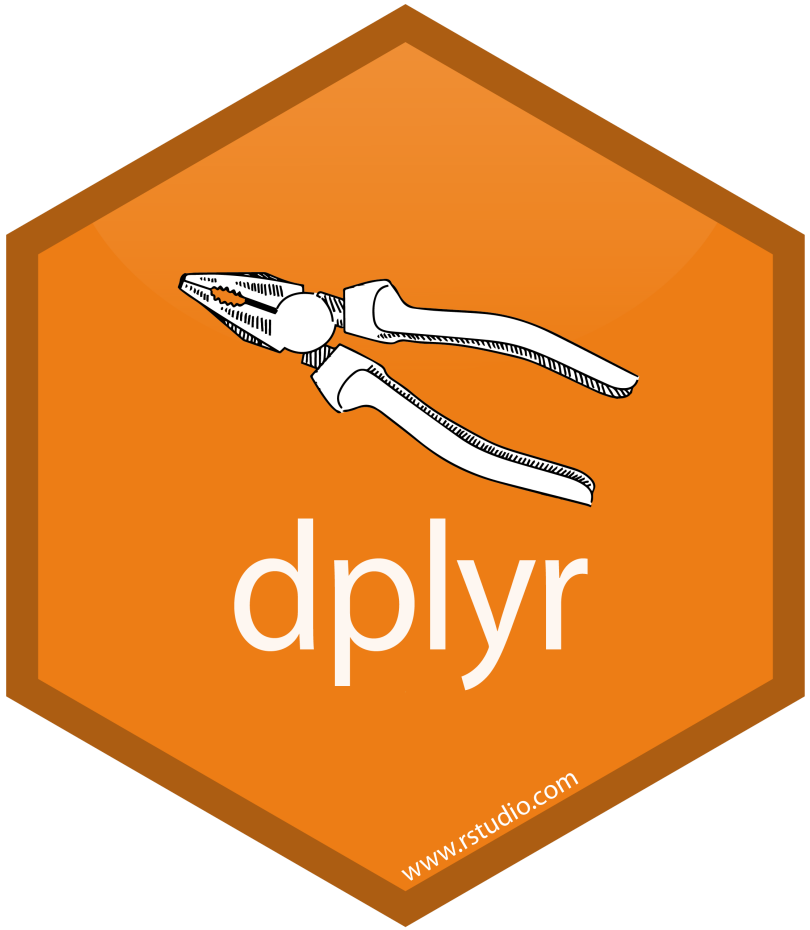
```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

filter()

— Pick Rows with Matching Criteria —

Cédric Scherer // R Course TU Dresden // Data Wrangling with the {tidyverse}

Pick Rows with Matching Criteria

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

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

```
# A tibble: 5 × 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**:

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

Equivalent code in base R:

```
1 babynames[babynames$year == 2000 & babynames$n > 25000, ]
```

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


Pick Rows with Matching Criteria

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

→ join filtering conditions with `&` (and) and `|` (or):

```
1 filter(babynames, (name == "Cedric" | name == "Clarissa"))
```

```
# A tibble: 293 × 5
    year sex   name          n      prop
   <dbl> <chr> <chr>     <int>     <dbl>
 1  1880 F     Clarissa     10 0.000102
 2  1882 F     Clarissa     13 0.000112
 3  1883 F     Clarissa     17 0.000142
 4  1884 F     Clarissa     18 0.000131
 5  1885 F     Clarissa     11 0.0000775
 6  1886 F     Clarissa     15 0.0000976
 7  1887 F     Clarissa     17 0.000109
 8  1888 F     Clarissa     12 0.0000633
 9  1889 F     Clarissa     20 0.000106
10  1890 F     Clarissa     25 0.000124
# … with 283 more rows
```


Pick Rows with Matching Criteria

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

→ join filtering conditions with `&` (and) and `|` (or):

```
1  filter(babynames, (name == "Cedric" | name == "Clarissa") & year %in% 1900:1905)
```

```
# A tibble: 9 × 5
   year sex   name          n       prop
  <dbl> <chr> <chr>     <int>      <dbl>
1  1900 F     Clarissa     20 0.0000629
2  1901 F     Clarissa     33 0.000130
3  1902 F     Clarissa     25 0.0000892
4  1903 F     Clarissa     21 0.0000755
5  1903 M     Cedric        7 0.0000541
6  1904 F     Clarissa     32 0.000109
7  1904 M     Cedric        8 0.0000578
8  1905 F     Clarissa     15 0.0000484
9  1905 M     Cedric        5 0.0000349
```


Pick Rows with Matching Criteria

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

→ filter conditions out with `!=` (not equal):

```
1 filter(babynames, name != "Mary")
```

```
# A tibble: 1,924,397 × 5
    year sex   name           n   prop
   <dbl> <chr> <chr>      <int>  <dbl>
 1  1880 F     Anna        2604 0.0267
 2  1880 F     Emma        2003 0.0205
 3  1880 F     Elizabeth   1939 0.0199
 4  1880 F     Minnie      1746 0.0179
 5  1880 F     Margaret    1578 0.0162
 6  1880 F     Ida         1472 0.0151
 7  1880 F     Alice       1414 0.0145
 8  1880 F     Bertha      1320 0.0135
 9  1880 F     Sarah       1288 0.0132
10  1880 F     Annie       1258 0.0129
# … with 1,924,387 more rows
```

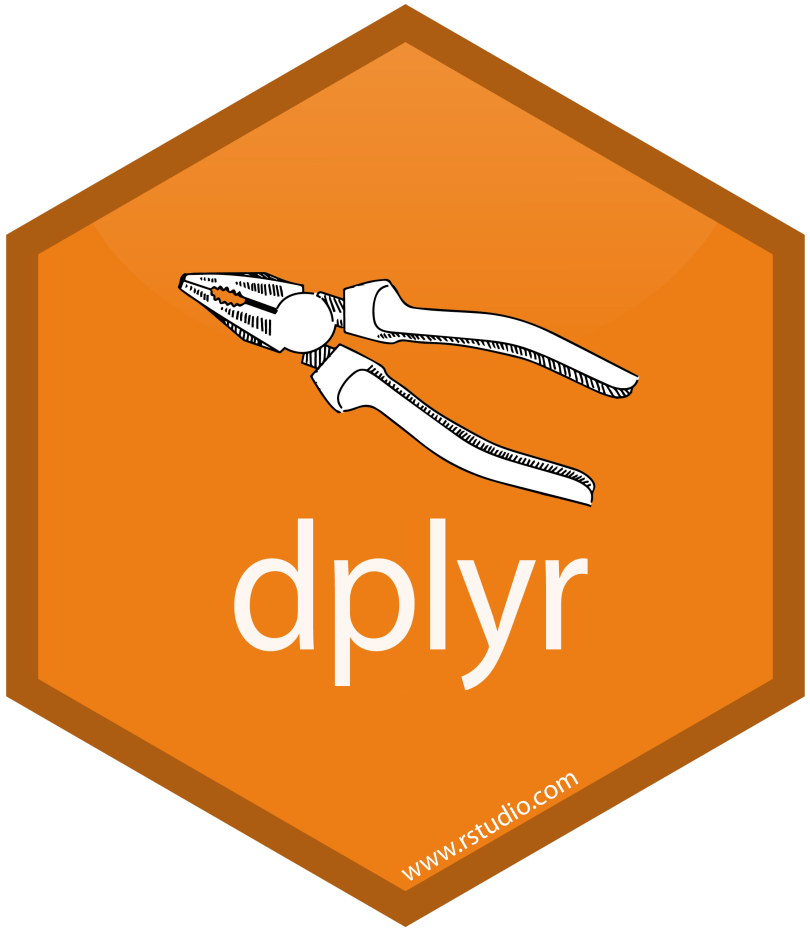

Pick Rows with Matching Criteria

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

→ reverse filter conditions with `!`:

```
1 filter(babynames, !name %in% c("Anna", "Emma", "Elizabeth"))
```

```
# A tibble: 1,923,865 × 5
    year sex   name          n   prop
   <dbl> <chr> <chr>     <int>  <dbl>
 1  1880 F     Mary       7065 0.0724
 2  1880 F     Minnie     1746 0.0179
 3  1880 F     Margaret   1578 0.0162
 4  1880 F     Ida        1472 0.0151
 5  1880 F     Alice      1414 0.0145
 6  1880 F     Bertha     1320 0.0135
 7  1880 F     Sarah      1288 0.0132
 8  1880 F     Annie      1258 0.0129
 9  1880 F     Clara      1226 0.0126
10  1880 F     Ella       1156 0.0118
# … with 1,923,855 more rows
```

`select()`

— Pick Columns with Matching Criteria —

Cédric Scherer // R Course TU Dresden // Data Wrangling with the {tidyverse}

Pick Columns with Matching Criteria

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

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

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


Pick Columns with Matching Criteria

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

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

Equivalent code in base R:

```
1 babynames[, c("year", "name", "n")]
```

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


Pick Columns with Matching Criteria

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

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

```
# A tibble: 1,924,665 × 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
```

```
1 select(babynames, -prop, -sex)
```

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


Pick Columns with Matching Criteria

`select(data, condition)` can also be used to **reorder columns**:

```
1 select(babynames, name, n, prop, sex, year)
```

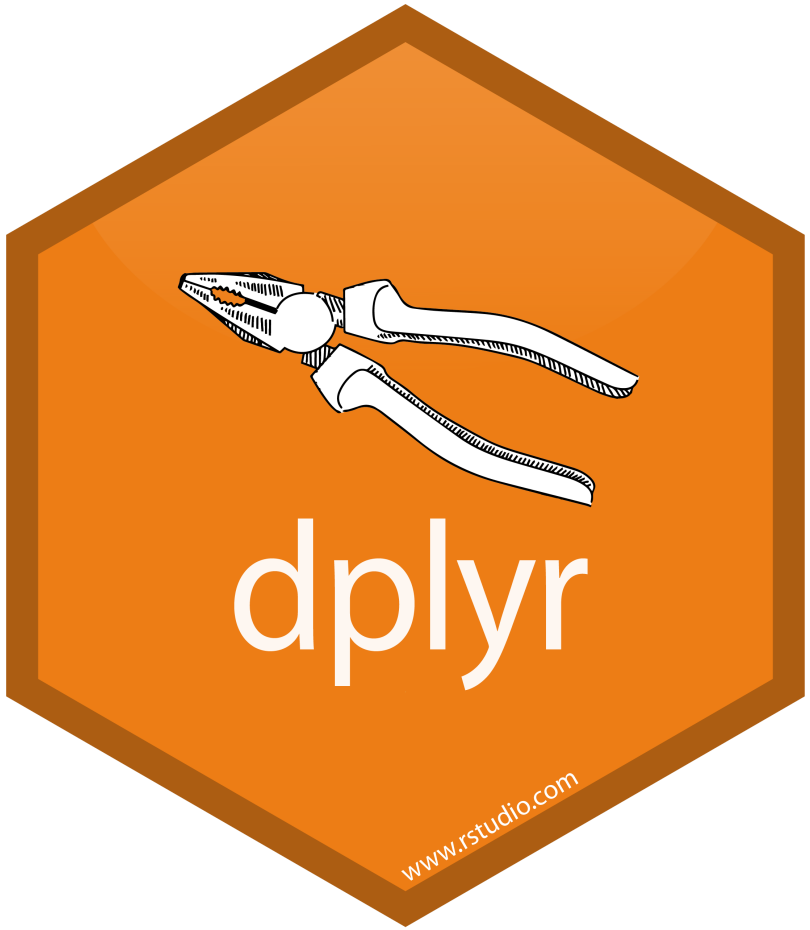
```
# A tibble: 1,924,665 × 5
   name           n   prop sex    year
   <chr>      <int>  <dbl> <chr> <dbl>
 1 Mary        7065 0.0724 F      1880
 2 Anna        2604 0.0267 F      1880
 3 Emma        2003 0.0205 F      1880
 4 Elizabeth   1939 0.0199 F      1880
 5 Minnie      1746 0.0179 F      1880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


Pick Columns with Matching Criteria

`select(data, condition)` can also be used to **reorder columns**:

```
1 select(babynames, n, everything())
```

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

arrange()

— Reorder Rows —

Cédric Scherer // R Course TU Dresden // Data Wrangling with the {tidyverse}

Reorder Rows

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

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

```
# A tibble: 1,924,665 × 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:

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

Equivalent code in base R:

```
1 babynames[order(babynames$prop), ]
```

```
# A tibble: 1,924,665 × 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:

```
1  arrange(babynames, -year, -prop)  ## only works for numeric
```

```
# A tibble: 1,924,665 × 5
    year sex   name          n    prop
   <dbl> <chr> <chr>     <int>   <dbl>
 1  2017 F     Emma      19738 0.0105
 2  2017 F     Olivia    18632 0.00994
 3  2017 M     Liam      18728 0.00954
 4  2017 M     Noah      18326 0.00933
 5  2017 F     Ava       15902 0.00848
 6  2017 F     Isabella  15100 0.00805
 7  2017 F     Sophia    14831 0.00791
 8  2017 M     William   14904 0.00759
 9  2017 M     James     14232 0.00725
10  2017 F     Mia       13437 0.00717
# … with 1,924,655 more rows
```


Reorder Rows

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

```
1  arrange(babynames, desc(year), desc(prop))  ## the "official" way
```

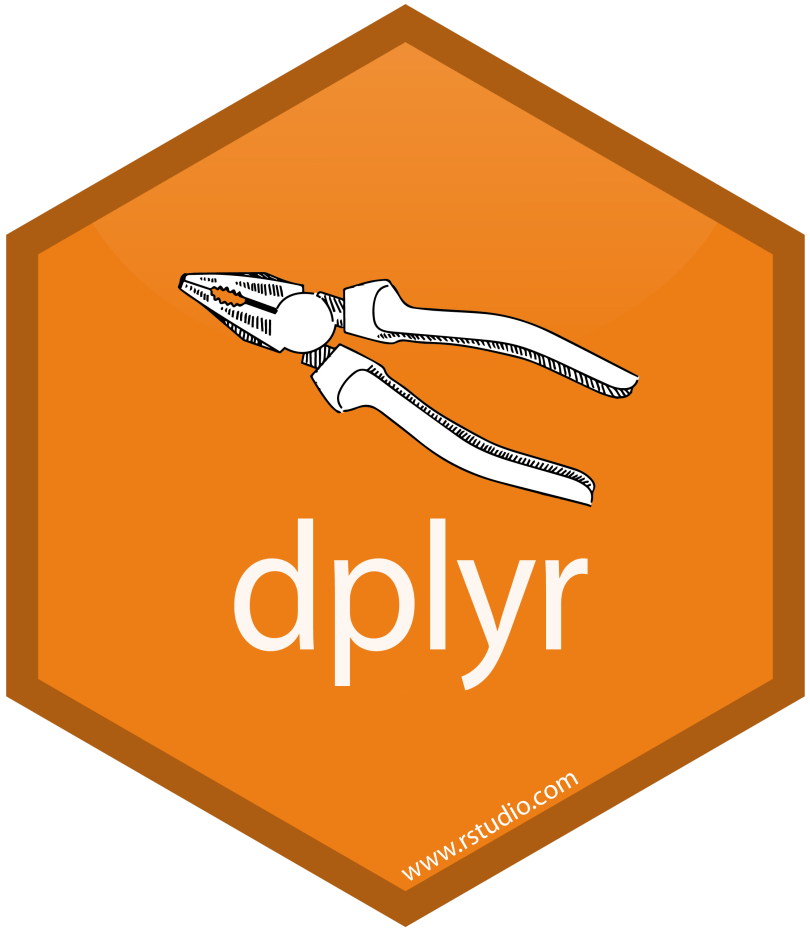
```
# A tibble: 1,924,665 × 5
    year sex   name          n    prop
   <dbl> <chr> <chr>     <int>   <dbl>
 1  2017 F     Emma      19738 0.0105
 2  2017 F     Olivia    18632 0.00994
 3  2017 M     Liam      18728 0.00954
 4  2017 M     Noah      18326 0.00933
 5  2017 F     Ava       15902 0.00848
 6  2017 F     Isabella  15100 0.00805
 7  2017 F     Sophia    14831 0.00791
 8  2017 M     William   14904 0.00759
 9  2017 M     James     14232 0.00725
10  2017 F     Mia       13437 0.00717
# … with 1,924,655 more rows
```


Reorder Rows

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

```
1  arrange(babynames, desc(name))
```

```
# A tibble: 1,924,665 × 5
    year sex   name           n       prop
   <dbl> <chr> <chr>      <int>      <dbl>
 1  2010 M     Zzyzx          5 0.00000244
 2  2014 M     Zyyon          6 0.00000293
 3  2010 F     Zyyanna        6 0.00000306
 4  2015 M     Zyvon          7 0.00000343
 5  2009 M     Zyvion         5 0.00000236
 6  2016 F     Zyva           8 0.00000415
 7  2017 F     Zyva           9 0.0000048
 8  2015 M     Zyus           5 0.00000245
 9  2002 M     Zytavious      6 0.0000029
10  2004 M     Zytavious      6 0.00000284
# … with 1,924,655 more rows
```

mutate()

— Create New Variables —

Cédric Scherer // R Course TU Dresden // Data Wrangling with the {tidyverse}

Illustration by Allison Horst

Create New Variables

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

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

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


Create New Variables

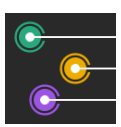
`mutate(data, condition)` allows you to create new variables (columns) **based on existing columns** or not based on your data:

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

Equivalent code in base R:

```
1 head(transform(babynames, decade = year %/% 10 * 10))
```

```
  year sex      name    n       prop decade
1 1880   F      Mary 7065 0.07238359   1880
2 1880   F      Anna 2604 0.02667896   1880
3 1880   F      Emma 2003 0.02052149   1880
4 1880   F Elizabeth 1939 0.01986579   1880
5 1880   F    Minnie 1746 0.01788843   1880
6 1880   F  Margaret 1578 0.01616720   1880
```

Create New Variables

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

```
1  mutate(babynames, name = stringr::str_to_lower(name))
```

```
# A tibble: 1,924,665 × 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 on existing columns or **not based on your data**:

```
1 mutate(babynames, note = "Please check!")
```

```
# A tibble: 1,924,665 × 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 on existing columns** or not based on your data:

```
1 mutate(babynames, note = if_else(year < 1900, "Please check!", "Ok"))
```

```
# A tibble: 1,924,665 × 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 on existing columns or ~~not~~ **based on your data**:

```
1 mutate(babynames, id = row_number())
```

```
# A tibble: 1,924,665 × 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 on existing and even newly created columns in the same step:

```
1 # install.packages("ggplot2")  ## contains the `mpg` data set
2 mpg <- ggplot2::mpg  ## we need some more numeric columns
```

```
1 mpg
```

```
# A tibble: 234 × 11
   manufacturer model      displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 audi         a4           1.8  1999     4 auto(l5)   f        18    29 p     compact
 2 audi         a4           1.8  1999     4 manual(m5) f        21    29 p     compact
 3 audi         a4           2    2008     4 manual(m6) f        20    31 p     compact
 4 audi         a4           2    2008     4 auto(av)   f        21    30 p     compact
 5 audi         a4           2.8  1999     6 auto(l5)   f        16    26 p     compact
 6 audi         a4           2.8  1999     6 manual(m5) f        18    26 p     compact
 7 audi         a4           3.1  2008     6 auto(av)   f        18    27 p     compact
 8 audi         a4 quattro   1.8  1999     4 manual(m5) 4        18    26 p     compact
 9 audi         a4 quattro   1.8  1999     4 auto(l5)   4        16    25 p     compact
10 audi         a4 quattro   2    2008     4 manual(m6) 4        20    28 p     compact
# … with 224 more rows
```


Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) **based on existing and even newly created columns** in the same step:

```
1  mpg2 <- select(mpg, model, hwy, cty) ## only to fit my slide
```

```
1  mpg2
```

```
# A tibble: 234 × 3
   model        hwy   cty
   <chr>      <int> <int>
 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(data, condition)` allows you to create new variables (columns) **based on existing and even newly created columns** in the same step:

```
1 mutate(mpg2, diff = hwy - cty, perc = diff / hwy)
```

```
# A tibble: 234 × 5
   model        hwy   cty  diff  perc
   <chr>      <int> <int> <int> <dbl>
 1 a4            29    18    11 0.379
 2 a4            29    21     8 0.276
 3 a4            31    20    11 0.355
 4 a4            30    21     9 0.3
 5 a4            26    16    10 0.385
 6 a4            26    18     8 0.308
 7 a4            27    18     9 0.333
 8 a4 quattro    26    18     8 0.308
 9 a4 quattro    25    16     9 0.36
10 a4 quattro    28    20     8 0.286
# … with 224 more rows
```


Create New Variables

`mutate(data, condition)` allows you to create new variables (columns) **based on existing and even newly created columns** in the same step:

```
1 mutate(mpg2, diff = hwy - cty, perc = diff / hwy)
```

Equivalent code in base R:

```
1 mpg_diff <- transform(mpg2, diff = hwy - cty)
2 head(transform(mpg_diff, perc = diff / hwy))
```

```
  model hwy cty diff      perc
1    a4  29  18   11 0.3793103
2    a4  29  21    8 0.2758621
3    a4  31  20   11 0.3548387
4    a4  30  21    9 0.3000000
5    a4  26  16   10 0.3846154
6    a4  26  18    8 0.3076923
```


Create New Variables

`mutate()` in combination with `across()` allows you to transform all columns in one step:

```
1  mutate(mpg2, across(everything(), as.double))
```

```
# A tibble: 234 × 3
   model   hwy   cty
   <dbl> <dbl> <dbl>
 1    NA    29    18
 2    NA    29    21
 3    NA    31    20
 4    NA    30    21
 5    NA    26    16
 6    NA    26    18
 7    NA    27    18
 8    NA    26    18
 9    NA    25    16
10    NA    28    20
# … with 224 more rows
```


Create New Variables

`mutate()` in combination with `across()` and `where()` allows you to conditionally create new columns in one step:

```
1 mutate(mpg2, across(where(is.numeric), as.double))
```

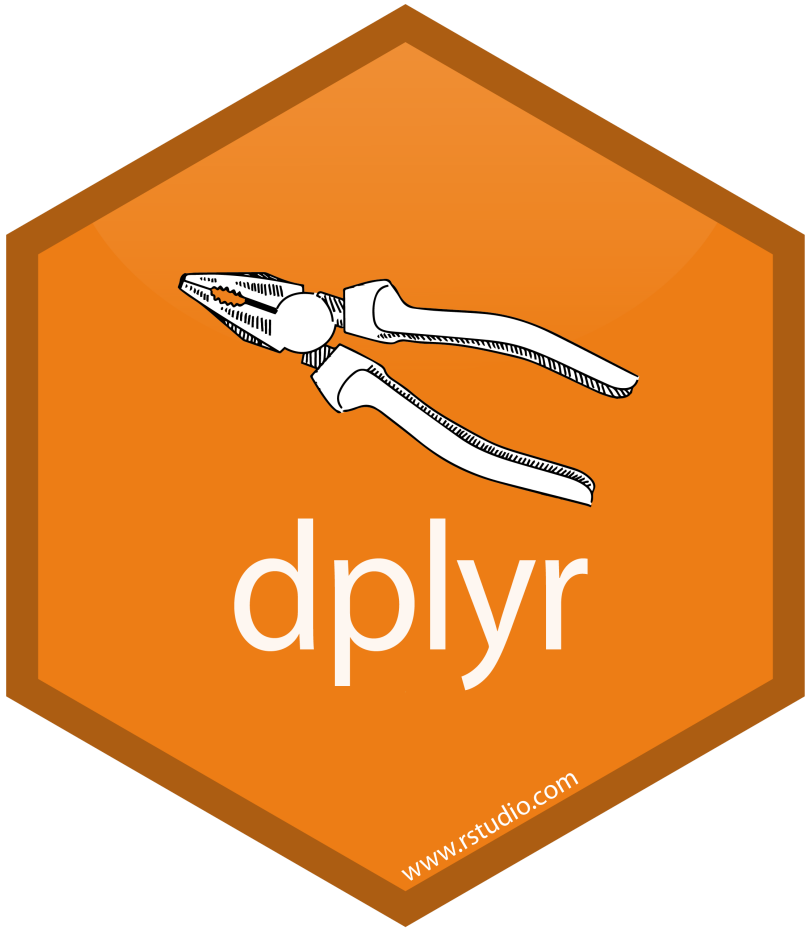
```
# A tibble: 234 × 3
   model        hwy   cty
   <chr>      <dbl> <dbl>
 1 a4            29    18
 2 a4            29    21
 3 a4            31    20
 4 a4            30    21
 5 a4            26    16
 6 a4            26    18
 7 a4            27    18
 8 a4 quattro    26    18
 9 a4 quattro    25    16
10 a4 quattro    28    20
# … with 224 more rows
```


Create New Variables

`mutate()` in combination with `across()` and `where()` allows you to conditionally create new columns in one step:

```
1  mutate(mpg2, across(where(is.numeric), list(avg = mean, log_2 = log2)))
```

```
# A tibble: 234 × 7
   model        hwy   cty hwy_avg hwy_log_2 cty_avg cty_log_2
   <chr>      <int> <int>   <dbl>     <dbl>   <dbl>     <dbl>
 1 a4            29    18    23.4      4.86    16.9      4.17
 2 a4            29    21    23.4      4.86    16.9      4.39
 3 a4            31    20    23.4      4.95    16.9      4.32
 4 a4            30    21    23.4      4.91    16.9      4.39
 5 a4            26    16    23.4      4.70    16.9      4
 6 a4            26    18    23.4      4.70    16.9      4.17
 7 a4            27    18    23.4      4.75    16.9      4.17
 8 a4 quattro    26    18    23.4      4.70    16.9      4.17
 9 a4 quattro    25    16    23.4      4.64    16.9      4
10 a4 quattro    28    20    23.4      4.81    16.9      4.32
# … with 224 more rows
```

summarize()

— Sum Up Variables —

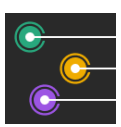
Cédric Scherer // R Course TU Dresden // Data Wrangling with the {tidyverse}

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 × 1
  unique_children
            <int>
1       348120517
```

Sum Up Variables

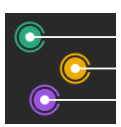
`summarize(data, condition)` allows you to create new variables (columns) based on existing columns or not based on your data:

```
1  summarize(babynames, unique_children = sum(n, na.rm = TRUE))
```

Equivalent code in base R:

```
1  sum(babynames$n, na.rm = TRUE)
```

```
[1] 348120517
```

Sum Up Variables

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

```
summarize(babynames, n = n())
```

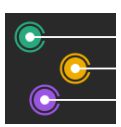
```
# A tibble: 1 × 1
        n
    <int>
1 1924665
```


Sum Up Variables

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

```
1 summarize(babynames, across(is.numeric, mean))
```

```
# A tibble: 1 × 3
   year     n     prop
  <dbl> <dbl>    <dbl>
1 1975.  181. 0.000136
```

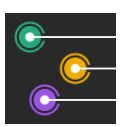



Sum Up Variables

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

```
summarize(babynames, across(c("n", "prop"), mean))
```

```
# A tibble: 1 × 2
      n     prop
  <dbl>    <dbl>
1  181. 0.000136
```

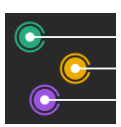



Sum Up Variables

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

```
summarize(babynames, across(c("n", "prop"), mean, na.rm = TRUE))
```

```
# A tibble: 1 × 2
      n     prop
  <dbl>    <dbl>
1  181. 0.000136
```

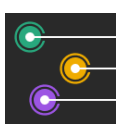



Sum Up Variables

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

```
summarize(babynames, across(c("n", "prop"), ~mean(.x, na.rm = TRUE)))
```

```
# A tibble: 1 × 2
      n     prop
  <dbl>    <dbl>
1  181. 0.000136
```

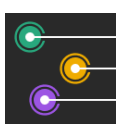



Sum Up Variables

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

```
summarize(mpg2, across(where(is.numeric), list(avg = mean, max = max)))
```

```
# A tibble: 1 × 4
  hwy_avg hwy_max cty_avg cty_max
    <dbl>   <int>   <dbl>   <int>
1    23.4      44    16.9      35
```

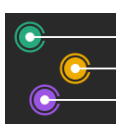



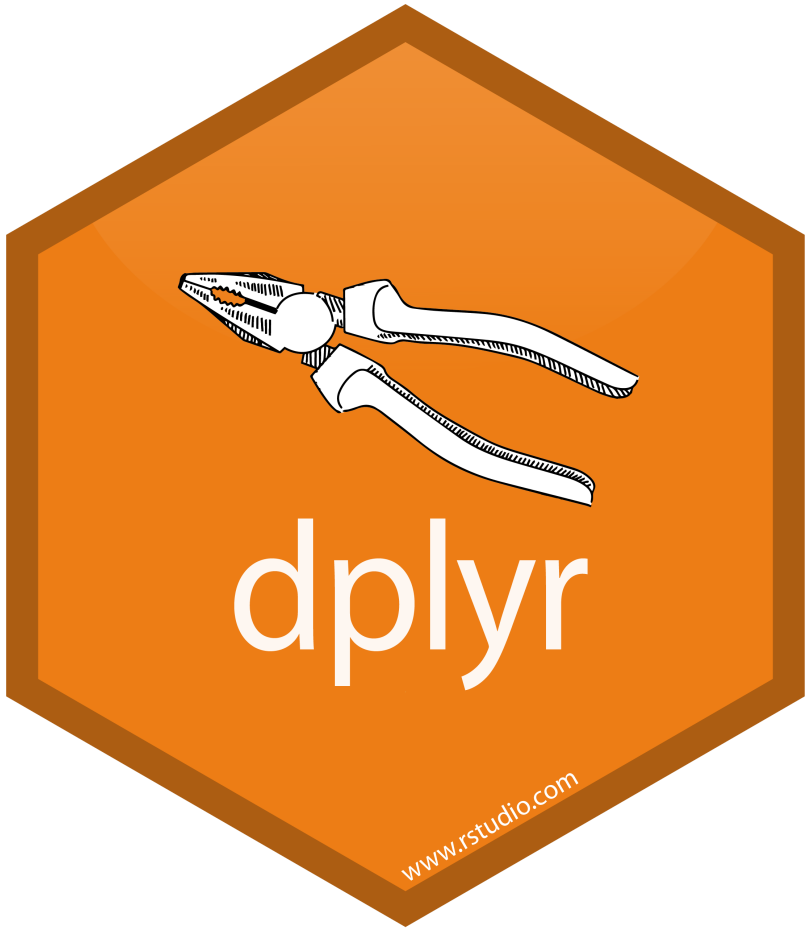
Sum Up Variables

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

```
summarize(mpg2, across(where(is.numeric), list(avg = ~mean(., na.rm = TRUE), max = ~max(.))))
```

```
# A tibble: 1 × 4
  hwy_avg hwy_max cty_avg cty_max
    <dbl>   <int>   <dbl>   <int>
1    23.4      44    16.9      35
```



group_by()

— Create Subsets —

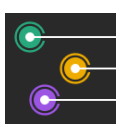
Cédric Scherer // R Course TU Dresden // Data Wrangling with the {tidyverse}

Create Subsets

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

```
1 group_by(babynames, sex)
```

```
# A tibble: 1,924,665 × 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:

```
1 group_by(babynames, sex, year)
```

```
# A tibble: 1,924,665 × 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:

```
1 grouped <- group_by(babynames, sex, year)
2 ungroup(grouped)
```

```
# A tibble: 1,924,665 × 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
```

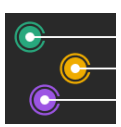



Image Source: blog.finaldraft.com/five-steps-to-writing-your-own-superhero


group_by()
SUPERPOWER!
Verbs will be automatically applied “by group”
Image Source: blog.finaldraft.com/five-steps-to-writing-your-own-superhero

Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
1 (names_group <- group_by(babynames, sex))
```

```
# A tibble: 1,924,665 × 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
```

```
1 summarize(names_group, sum = sum(n))
```

```
# A tibble: 2 × 2
  sex         sum
  <chr>     <int>
1 F     172371079
2 M     175749438
```


Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
babies1 <- filter(babynames, year %in% range(year))
```

```
babies2 <- group_by(babies1, year)
```

```
babies2
```

```
# A tibble: 34,469 × 5
# Groups:   year [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 34,459 more rows
```


Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
babies1 <- filter(babynames, year %in% range(year))
```

```
babies2 <- group_by(babies1, year)
```

```
babies3 <- mutate(babies2, n_year = sum(n))
```

```
babies3
```

```
# A tibble: 34,469 × 6
# Groups:   year [2]
    year sex   name           n   prop n_year
   <dbl> <chr> <chr>      <int>  <dbl>  <int>
 1  1880 F     Mary        7065 0.0724 201484
 2  1880 F     Anna        2604 0.0267 201484
 3  1880 F     Emma        2003 0.0205 201484
 4  1880 F     Elizabeth   1939 0.0199 201484
 5  1880 F     Minnie      1746 0.0179 201484
 6  1880 F     Margaret    1578 0.0162 201484
 7  1880 F     Ida         1472 0.0151 201484
 8  1880 F     Alice       1414 0.0145 201484
 9  1880 F     Bertha      1320 0.0135 201484
10  1880 F     Sarah       1288 0.0132 201484
# … with 34,459 more rows
```


Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
1  babies1 <- filter(babynames, year %in% range(year))
```

```
1  babies2 <- group_by(babies1, sex, year)
```

```
1  babies3 <- mutate(babies2, n_year = sum(n))
```

```
1  summarize(babies2, n_year = sum(n))
```

```
# A tibble: 2 × 2
   year  n_year
  <dbl>   <int>
1  1880  201484
2  2017 3546301
```


Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
1 babies4 <- group_by(babies3, sex, year)
```

```
1 babies4
```

```
# A tibble: 34,469 × 6
# Groups:   sex, year [4]
    year sex   name           n   prop n_year
   <dbl> <chr> <chr>      <int>  <dbl>  <int>
 1  1880 F     Mary        7065 0.0724 201484
 2  1880 F     Anna        2604 0.0267 201484
 3  1880 F     Emma        2003 0.0205 201484
 4  1880 F     Elizabeth   1939 0.0199 201484
 5  1880 F     Minnie      1746 0.0179 201484
 6  1880 F     Margaret    1578 0.0162 201484
 7  1880 F     Ida         1472 0.0151 201484
 8  1880 F     Alice       1414 0.0145 201484
 9  1880 F     Bertha      1320 0.0135 201484
10  1880 F     Sarah       1288 0.0132 201484
# … with 34,459 more rows
```


Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
1 babies4 <- group_by(babies3, sex, year)
```

```
1 babies5 <- mutate(babies4, n_sex_year = sum(n), prop_sex_year = n_sex_year / n_year)
```

```
1 babies5
```

```
# A tibble: 34,469 × 8
# Groups:   sex, year [4]
    year sex   name           n    prop n_year n_sex_year prop_sex_year
   <dbl> <chr> <chr>      <int>   <dbl>  <int>      <int>         <dbl>
 1  1880 F     Mary        7065 0.0724 201484      90993         0.452
 2  1880 F     Anna        2604 0.0267 201484      90993         0.452
 3  1880 F     Emma        2003 0.0205 201484      90993         0.452
 4  1880 F     Elizabeth   1939 0.0199 201484      90993         0.452
 5  1880 F     Minnie      1746 0.0179 201484      90993         0.452
 6  1880 F     Margaret    1578 0.0162 201484      90993         0.452
 7  1880 F     Ida         1472 0.0151 201484      90993         0.452
 8  1880 F     Alice       1414 0.0145 201484      90993         0.452
 9  1880 F     Bertha      1320 0.0135 201484      90993         0.452
10  1880 F     Sarah       1288 0.0132 201484      90993         0.452
# … with 34,459 more rows
```


Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
1 babies4 <- group_by(babies3, sex, year)
```

```
1 babies5 <- summarize(babies4, n_sex_year = sum(n), prop_sex_year = n_sex_year / n_year)
```

```
1 babies5
```

```
# A tibble: 34,469 × 4
# Groups:   sex, year [4]
   sex    year n_sex_year prop_sex_year
   <chr> <dbl>      <int>         <dbl>
 1 F      1880      90993         0.452
 2 F      1880      90993         0.452
 3 F      1880      90993         0.452
 4 F      1880      90993         0.452
 5 F      1880      90993         0.452
 6 F      1880      90993         0.452
 7 F      1880      90993         0.452
 8 F      1880      90993         0.452
 9 F      1880      90993         0.452
10 F      1880      90993         0.452
# … with 34,459 more rows
```


Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
1 babies4 <- group_by(babies3, sex, year)
```

```
1 babies5 <- summarize(babies4, n_sex_year = sum(n), prop_sex_year = n_sex_year / unique(n_year))
```

```
1 babies5
```

```
# A tibble: 4 × 4
# Groups:   sex [2]
  sex    year n_sex_year prop_sex_year
  <chr> <dbl>      <int>         <dbl>
1 F      1880      90993         0.452
2 F      2017    1711811         0.483
3 M      1880     110491         0.548
4 M      2017    1834490         0.517
```


Applying Verbs Across Groups

`group_by()` in combination with other **`{dplyr}`** verbs allows you to calculate summary statistics for specified groups without the need to create new subsets:

```
1 babies4 <- group_by(babies3, sex, year)
```

```
1 babies5 <- unique(summarize(babies4, n_sex_year = sum(n), prop_sex_year = n_sex_year / n_year))
```

```
1 babies5
```

```
# A tibble: 4 × 4
# Groups:   sex, year [4]
  sex    year n_sex_year prop_sex_year
  <chr> <dbl>      <int>         <dbl>
1 F      1880      90993         0.452
2 F      2017    1711811         0.483
3 M      1880     110491         0.548
4 M      2017    1834490         0.517
```

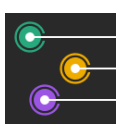

Your Turn: {dplyr} Verbs

- Search for your own name in the `babynames` data set
- Find the year when your name was most popular
- Report the number of babies with your name born in your birth year
- Count the number of babies born in your birth year
- Calculate the proportion of babies born with the name “Mary” in the 1880s versus the 2000s

Your Turn: {dplyr} Verbs

```
1 filter(babynames, name == "Cedric")
```

```
# A tibble: 152 × 5
    year sex   name        n      prop
   <dbl> <chr> <chr>   <int>     <dbl>
 1  1890 M     Cedric      5 0.0000418
 2  1893 M     Cedric      9 0.0000744
 3  1895 M     Cedric      5 0.0000395
 4  1896 M     Cedric      7 0.0000542
 5  1897 M     Cedric      8 0.0000656
 6  1899 M     Cedric      5 0.0000434
 7  1903 M     Cedric      7 0.0000541
 8  1904 M     Cedric      8 0.0000578
 9  1905 M     Cedric      5 0.0000349
10  1906 M     Cedric      8 0.0000555
# … with 142 more rows
```

Your Turn: {dplyr} Verbs

```
cedrics <- filter(babynames, name == "Cedric")
```

```
filter(cedrics, n == max(n))
```

```
# A tibble: 1 × 5
   year sex   name      n     prop
  <dbl> <chr> <chr>  <int>    <dbl>
1  1982 M     Cedric   976 0.000517
```

```
arrange(cedrics, -n)  ## sorting by `-n` provides the answer as well
```

```
# A tibble: 152 × 5
    year sex   name      n     prop
   <dbl> <chr> <chr>  <int>    <dbl>
 1  1982 M     Cedric   976 0.000517
 2  1981 M     Cedric   910 0.000489
 3  1980 M     Cedric   907 0.000489
 4  1977 M     Cedric   903 0.000528
 5  1974 M     Cedric   901 0.000553
 6  1972 M     Cedric   895 0.000534
 7  1984 M     Cedric   884 0.000471
 8  1973 M     Cedric   883 0.000547
 9  1970 M     Cedric   879 0.000461
10  1975 M     Cedric   871 0.000537
# … with 142 more rows
```


Your Turn: {dplyr} Verbs

```
filter(babynames, name == "Cedric", year == 1986)
```

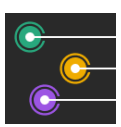
```
# A tibble: 2 × 5
   year sex   name        n       prop
  <dbl> <chr> <chr>   <int>      <dbl>
1  1986 F     Cedric     11 0.00000596
2  1986 M     Cedric    706 0.000368
```

```
filter(babynames, name == "Cedric" & year == 1986)  ## using `&` instead
```

```
# A tibble: 2 × 5
   year sex   name        n       prop
  <dbl> <chr> <chr>   <int>      <dbl>
1  1986 F     Cedric     11 0.00000596
2  1986 M     Cedric    706 0.000368
```

```
filter(cedrics, year == 1986)  ## using the subset from the task before
```

```
# A tibble: 2 × 5
   year sex   name        n       prop
  <dbl> <chr> <chr>   <int>      <dbl>
1  1986 F     Cedric     11 0.00000596
2  1986 M     Cedric    706 0.000368
```

Your Turn: {dplyr} Verbs

```
1  names_1986 <- filter(babynames, year == 1986)
```

```
1  summarize(names_1986, n = sum(n))
```

```
# A tibble: 1 × 1
        n
    <int>
1 3555871
```

```
1  sum(names_1986$n)  ## basic sum -> returns an integer
```

```
[1] 3555871
```

```
1  summarize(filter(babynames, year == 1986), n = sum(n))  ## nested version
```

```
# A tibble: 1 × 1
        n
    <int>
1 3555871
```

```
1  sum(filter(babynames, year == 1986)$n)  ## you should rather not do this
```

```
[1] 3555871
```


Your Turn: {dplyr} Verbs

```
1  maries_selected <- filter(babynames, name == "Mary", year %in% c(1880:1889, 2000:2009))
```

```
1  maries_selected
```

```
# A tibble: 38 × 5
    year sex   name      n     prop
   <dbl> <chr> <chr> <int>    <dbl>
 1  1880 F     Mary   7065 0.0724
 2  1880 M     Mary     27 0.000228
 3  1881 F     Mary   6919 0.0700
 4  1881 M     Mary     29 0.000268
 5  1882 F     Mary   8148 0.0704
 6  1882 M     Mary     30 0.000246
 7  1883 F     Mary   8012 0.0667
 8  1883 M     Mary     32 0.000284
 9  1884 F     Mary   9217 0.0670
10  1884 M     Mary     36 0.000293
# … with 28 more rows
```


Your Turn: {dplyr} Verbs

```r
maries_selected <- filter(babynames, name == "Mary", year %in% c(1880:1889, 2000:2009))
```

```r
maries_grouped <- group_by(maries_selected, decade = year %/% 10 * 10)
```

```r
maries_grouped
```

```
# A tibble: 38 × 6
# Groups:   decade [2]
    year sex   name      n      prop decade
   <dbl> <chr> <chr> <int>     <dbl>  <dbl>
 1  1880 F     Mary   7065 0.0724      1880
 2  1880 M     Mary     27 0.000228    1880
 3  1881 F     Mary   6919 0.0700      1880
 4  1881 M     Mary     29 0.000268    1880
 5  1882 F     Mary   8148 0.0704      1880
 6  1882 M     Mary     30 0.000246    1880
 7  1883 F     Mary   8012 0.0667      1880
 8  1883 M     Mary     32 0.000284    1880
 9  1884 F     Mary   9217 0.0670      1880
10  1884 M     Mary     36 0.000293    1880
# … with 28 more rows
```

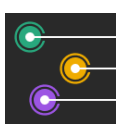

Your Turn: {dplyr} Verbs

```
1  maries_selected <- filter(babynames, name == "Mary", year %in% c(1880:1889, 2000:2009))
```

```
1  maries_grouped <- group_by(maries_selected, decade = year %/% 10 * 10)
```

```
1  summarize(maries_grouped, prop_avg = mean(prop))
```

```
# A tibble: 2 × 2
  decade prop_avg
   <dbl>    <dbl>
1   1880  0.0333
2   2000  0.00127
```

Readability of Code

Have a look again at this chunk of code:

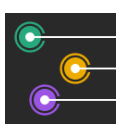
```
babies1 <- filter(babynames, year %in% range(year))
babies2 <- group_by(babies1, year)
babies3 <- mutate(babies2, n_year = sum(n))
babies4 <- group_by(babies3, sex, year)
babies5 <- summarize(babies4, n_sex_year = sum(n), prop_sex_year = n_sex_year / unique(n_year))
```

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:

```
babies3 <- mutate(group_by(filter(babynames, year %in% range(year)), year), n_year = sum(n))
babies5 <- summarize(group_by(babies3, sex, year), n_sex_year = sum(n), prop_sex_year = n_sex_year
```

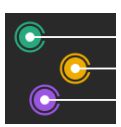



Readability of Code

One could rewrite the code like this:

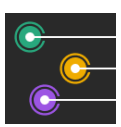
```
babies3 <-
  mutate(
    group_by(
      filter(
        babynames, year %in% range(year)
      ),
      year
    ),
    n_year = sum(n)
  )
```

```
babies5 <-
  summarize(
    group_by(
      babies3, sex, year
    ),
    n_sex_year = sum(n),
    prop_sex_year = n_sex_year / unique(n_year)
  )
```

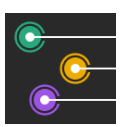



Data Wrangling

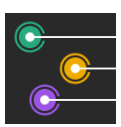
— Ceci n'est pas une pipe —



```
teach(
  dvb(
    breakfast(
      shower(
        wake_up(
          Cédric,
          7.0
        ),
        temp == 38
      ),
      c(“cereals”, “tea”),
    ),
    price = “EUR2.70”,
    line = 12,
    depart = “08:10”
  ),
  course = “Reproducible Data Analysis with R”
)
```

```
Cédric %>%
  wake_up(7.0) %>%
  shower(temp == 38) %>%
  breakfast(c("cereals", "tea")) %>%
  dvb(
    price = "EUR2.70",
    train = 12,
    depart = "08:10"
  ) %>%
  teach(course = "Reproducible Data Analysis with R")
```

"The Treachery of Images" by Réne Magritte

"The Treachery of Images" by Réne Magritte & the inspired hex logo

Readability of Code: The Pipe

With the pipe one can rewrite the code like this:

```
babies5 <-
  babynames %>%
  filter(year %in% range(year)) %>%
  group_by(year) %>%
  mutate(n_year = sum(n)) %>%
  group_by(sex, year) %>%
  summarize(
    n_sex_year = sum(n),
    prop_sex_year = n_sex_year / unique(n_year)
  )
```

```
babies5
```

```
# A tibble: 4 × 4
# Groups:   sex [2]
  sex    year n_sex_year prop_sex_year
  <chr> <dbl>      <int>         <dbl>
1 F      1880      90993         0.452
2 F      2017    1711811         0.483
3 M      1880     110491         0.548
4 M      2017    1834490         0.517
```


Readability of Code: The Pipe

Putting the object and data on their own lines allows you to inspect the output:

```
# babies5 <-
  babynames %>%
  filter(year %in% range(year)) %>%
  group_by(year) %>%
  mutate(n_year = sum(n)) %>%
  group_by(sex, year) %>%
  summarize(
    n_sex_year = sum(n),
    prop_sex_year = n_sex_year / unique(n_year)
  )
```

```
# A tibble: 4 × 4
# Groups:   sex [2]
  sex    year n_sex_year prop_sex_year
  <chr> <dbl>      <int>         <dbl>
1 F      1880      90993         0.452
2 F      2017    1711811         0.483
3 M      1880     110491         0.548
4 M      2017    1834490         0.517
```


Readability of Code: The Pipe

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

```r
# babies5 <-
  babynames %>%
  filter(year %in% range(year)) %>%
  group_by(year) %>%
  mutate(n_year = sum(n)) # %>%
  # group_by(sex, year) %>%
  # summarize(
  #   n_sex_year = sum(n),
  #   prop_sex_year = n_sex_year / unique(n_year)
  # )
```

```
# A tibble: 34,469 × 6
# Groups:   year [2]
    year sex   name           n   prop n_year
   <dbl> <chr> <chr>      <int>  <dbl>  <int>
 1  1880 F     Mary        7065 0.0724 201484
 2  1880 F     Anna        2604 0.0267 201484
 3  1880 F     Emma        2003 0.0205 201484
 4  1880 F     Elizabeth   1939 0.0199 201484
 5  1880 F     Minnie      1746 0.0179 201484
 6  1880 F     Margaret    1578 0.0162 201484
 7  1880 F     Ida         1472 0.0151 201484
 8  1880 F     Alice       1414 0.0145 201484
 9  1880 F     Bertha      1320 0.0135 201484
10  1880 F     Sarah       1288 0.0132 201484
# … with 34,459 more rows
```


Readability of Code: The Pipe

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

```r
# babies5 <-
  babynames %>%
  filter(year %in% range(year)) %>%
  group_by(year) %>%
  mutate(n_year = sum(n)) %>%
  # group_by(sex, year) %>%
  # summarize(
  #   n_sex_year = sum(n),
  #   prop_sex_year = n_sex_year / unique(n_year)
  # ) %>%
  {}  ## some love it—some hate it
```

```
# A tibble: 34,469 × 6
# Groups:   year [2]
    year sex   name           n   prop n_year
   <dbl> <chr> <chr>      <int>  <dbl>  <int>
 1  1880 F     Mary        7065 0.0724 201484
 2  1880 F     Anna        2604 0.0267 201484
 3  1880 F     Emma        2003 0.0205 201484
 4  1880 F     Elizabeth   1939 0.0199 201484
 5  1880 F     Minnie      1746 0.0179 201484
 6  1880 F     Margaret    1578 0.0162 201484
 7  1880 F     Ida         1472 0.0151 201484
 8  1880 F     Alice       1414 0.0145 201484
 9  1880 F     Bertha      1320 0.0135 201484
```


Readability of Code: The Pipe

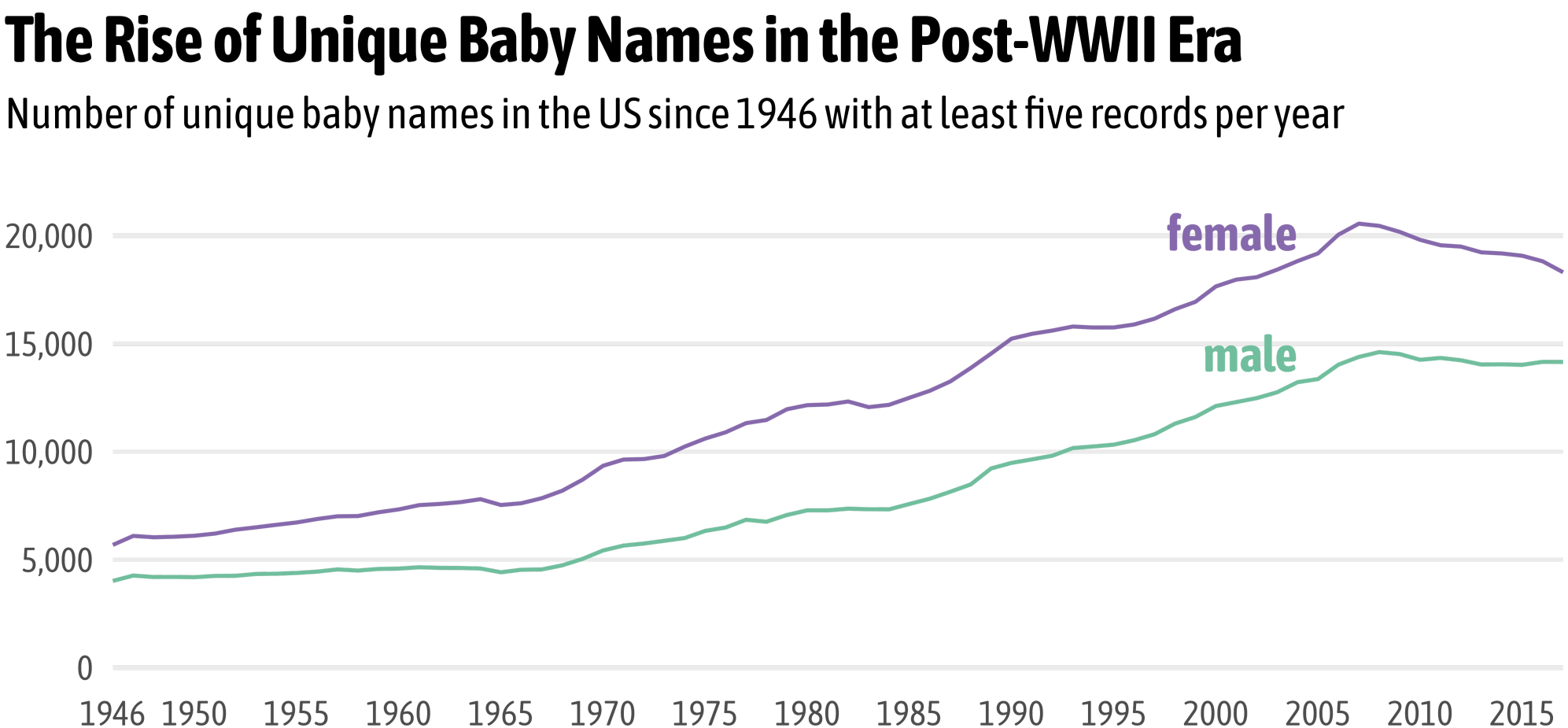
This way, you can also easily inactivate verbs or single conditions:

```
#babies5 <-
  babynames %>%
  filter(year %in% range(year)) %>%
  group_by(year) %>%
  mutate(n_year = sum(n)) %>%
  #group_by(sex, year) %>%
  summarize(
    n_sex_year = sum(n),
    #prop_sex_year = n_sex_year / unique(n_year)
  )
```

```
# A tibble: 2 × 2
   year n_sex_year
  <dbl>      <int>
1  1880     201484
2  2017    3546301
```


Your Turn: Data Wrangling

- Shape the `babynames` data as necessary for the Post-WWII visualization:



Your Turn: Data Wrangling

- Inspect the `flights` data set from the **`{nycflights13}`** package.
- Count the number of flights recorded 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: Store delays with 1 digit only.

Shape the babynames Data

```
babynames %>%
  filter(year > 1945) %>%
  group_by(year, sex) %>%
  summarize(unique_names = n_distinct(name))
```

```
# A tibble: 144 × 3
# Groups:   year [72]
    year sex   unique_names
   <dbl> <chr>        <int>
 1  1946 F             5686
 2  1946 M             4019
 3  1947 F             6103
 4  1947 M             4268
 5  1948 F             6040
 6  1948 M             4199
 7  1949 F             6065
 8  1949 M             4203
 9  1950 F             6111
10  1950 M             4191
# … with 134 more rows
```


Shape the babynames Data

```
babynames %>%
  filter(year > 1945) %>%
  group_by(year, sex) %>%
  summarize(unique_names = n_distinct(name)) %>%
  ungroup() %>%
  mutate(sex = if_else(sex == "M", "male", "female"))
```

```
# A tibble: 144 × 3
    year sex    unique_names
   <dbl> <chr>         <int>
 1  1946 female         5686
 2  1946 male           4019
 3  1947 female         6103
 4  1947 male           4268
 5  1948 female         6040
 6  1948 male           4199
 7  1949 female         6065
 8  1949 male           4203
 9  1950 female         6111
10  1950 male           4191
# … with 134 more rows
```


The {nycflights} Data Package

```r
# install.packages("nycflights)
library(nycflights13)
```

```r
tibble::glimpse(flights)
```

```
Rows: 336,776
Columns: 19
$ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2…
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
$ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
$ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, 558, 558, 558, 559, 559, 559, …
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, 600, 600, 600, 600, 559, 600, …
$ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1, 0, -1, 0, 0, 1, -8, -3, -4, …
$ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849, 853, 924, 923, 941, 702, 854,…
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851, 856, 917, 937, 910, 706, 902,…
$ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -14, 31, -4, -8, -7, 12, -6, -8,…
$ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "AA", "B6", "B6", "UA", "UA", "…
$ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 49, 71, 194, 1124, 707, 1806, 1…
$ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N39463", "N516JB", "N829AS", "N593…
$ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA", "JFK", "LGA", "JFK", "JFK", "…
$ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD", "MCO", "ORD", "PBI", "TPA", "…
$ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 158, 345, 361, 257, 44, 337, 15…
$ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, 1028, 1005, 2475, 2565, 1389, …
$ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6…
```


Flights in June and July 2013

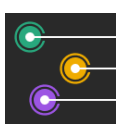
```
filter(flights, month == 6 | month == 7)
```

```
# A tibble: 57,668 × 19
    year month   day dep_time sched_dep_…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier flight tailnum origin dest
   <int> <int> <int>    <int>        <int>   <dbl>   <int>   <int>   <dbl> <chr>    <int> <chr>   <chr>  <chr>
 1  2013     6     1        2         2359       3     341     350      -9 B6         739 N618JB  JFK    PSE
 2  2013     6     1      451          500      -9     624     640     -16 US        1431 N538UW  EWR    CLT
 3  2013     6     1      506          515      -9     715     800     -45 UA        1686 N35407  EWR    IAH
 4  2013     6     1      534          545     -11     800     829     -29 UA        1451 N27724  LGA    IAH
 5  2013     6     1      538          545      -7     925     922       3 B6         725 N806JB  JFK    BQN
 6  2013     6     1      539          540      -1     832     840      -8 AA         701 N5EAAA  JFK    MIA
 7  2013     6     1      546          600     -14     850     910     -20 UA         540 N492UA  EWR    RSW
 8  2013     6     1      551          600      -9     828     850     -22 AA         707 N3EUAA  LGA    DFW
 9  2013     6     1      552          600      -8     647     655      -8 US        1911 N946UW  LGA    PHL
10  2013     6     1      553          600      -7     700     711     -11 EV        5716 N835AS  JFK    IAD
# … with 57,658 more rows, 5 more variables: air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>, and abbreviated variable names ¹sched_dep_time, ²dep_delay, ³arr_time, ⁴sched_arr_time,
#   ⁵arr_delay
```


Flights in June and July 2013

```
filter(flights, month %in% c(6, 7))
```

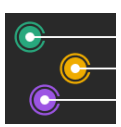
```
# A tibble: 57,668 × 19
    year month   day dep_time sched_dep_…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier flight tailnum origin dest
   <int> <int> <int>    <int>        <int>   <dbl>   <int>   <int>   <dbl> <chr>    <int> <chr>   <chr>  <chr>
 1  2013     6     1        2         2359       3     341     350      -9 B6         739 N618JB  JFK    PSE
 2  2013     6     1      451          500      -9     624     640     -16 US        1431 N538UW  EWR    CLT
 3  2013     6     1      506          515      -9     715     800     -45 UA        1686 N35407  EWR    IAH
 4  2013     6     1      534          545     -11     800     829     -29 UA        1451 N27724  LGA    IAH
 5  2013     6     1      538          545      -7     925     922       3 B6         725 N806JB  JFK    BQN
 6  2013     6     1      539          540      -1     832     840      -8 AA         701 N5EAAA  JFK    MIA
 7  2013     6     1      546          600     -14     850     910     -20 UA         540 N492UA  EWR    RSW
 8  2013     6     1      551          600      -9     828     850     -22 AA         707 N3EUAA  LGA    DFW
 9  2013     6     1      552          600      -8     647     655      -8 US        1911 N946UW  LGA    PHL
10  2013     6     1      553          600      -7     700     711     -11 EV        5716 N835AS  JFK    IAD
# … with 57,658 more rows, 5 more variables: air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>, and abbreviated variable names ¹sched_dep_time, ²dep_delay, ³arr_time, ⁴sched_arr_time,
#   ⁵arr_delay
```

Catch-Up Delay

```
1 filter(flights, dep_delay > 0, arr_delay <= 0)
```

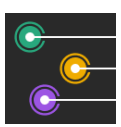
```
# A tibble: 35,442 × 19
    year month   day dep_time sched_dep_…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier flight tailnum origin dest
   <int> <int> <int>    <int>        <int>   <dbl>   <int>   <int>   <dbl> <chr>    <int> <chr>   <chr>  <chr>
 1  2013     1     1      601          600       1     844     850      -6 B6         343 N644JB  EWR    PBI
 2  2013     1     1      644          636       8     931     940      -9 UA        1701 N75435  EWR    FLL
 3  2013     1     1      646          645       1     910     916      -6 UA         883 N569UA  LGA    DEN
 4  2013     1     1      646          645       1    1023    1030      -7 UA        1496 N38727  EWR    SNA
 5  2013     1     1      701          700       1    1123    1154     -31 UA        1203 N77296  EWR    SJU
 6  2013     1     1      752          750       2    1025    1029      -4 UA         477 N511UA  LGA    DEN
 7  2013     1     1      803          800       3    1132    1144     -12 UA         223 N510UA  JFK    SFO
 8  2013     1     1      826          817       9    1145    1158     -13 UA        1480 N76522  EWR    SFO
 9  2013     1     1      846          845       1    1138    1205     -27 B6         553 N564JB  EWR    RSW
10  2013     1     1      856          855       1    1140    1203     -23 UA        1296 N75426  EWR    PBI
# … with 35,432 more rows, 5 more variables: air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>, and abbreviated variable names ¹sched_dep_time, ²dep_delay, ³arr_time, ⁴sched_arr_time,
#   ⁵arr_delay
```

Average Delays per Origin and Month

```
flights %>%
  group_by(month, origin) %>%
  summarize(
    dep_delay_avg = mean(dep_delay),
    arr_delay_avg = mean(arr_delay)
  ) %>%
  arrange(-arr_delay_avg)
```

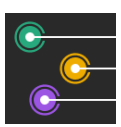
```
# A tibble: 36 × 4
# Groups:   month [12]
   month origin dep_delay_avg arr_delay_avg
   <int> <chr>          <dbl>         <dbl>
 1     1 EWR               NA            NA
 2     1 JFK               NA            NA
 3     1 LGA               NA            NA
 4     2 EWR               NA            NA
 5     2 JFK               NA            NA
 6     2 LGA               NA            NA
 7     3 EWR               NA            NA
 8     3 JFK               NA            NA
 9     3 LGA               NA            NA
10     4 EWR               NA            NA
# … with 26 more rows
```

Average Delays per Origin and Month

```
 1 flights %>%
 2   group_by(month, origin) %>%
 3   summarize(
 4     dep_delay_avg = mean(dep_delay, na.rm = TRUE),
 5     arr_delay_avg = mean(arr_delay, na.rm = TRUE)
 6   ) %>%
 7   arrange(-arr_delay_avg)
```

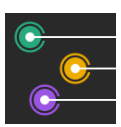
```
# A tibble: 36 × 4
# Groups:   month [12]
   month origin dep_delay_avg arr_delay_avg
   <int> <chr>          <dbl>         <dbl>
 1     7 JFK             23.8          20.2
 2    12 EWR             21.0          19.6
 3     6 JFK             20.5          17.6
 4     6 EWR             22.5          16.9
 5     7 EWR             22.0          15.5
 6     6 LGA             19.3          14.8
 7     7 LGA             19.0          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

```
 1 flights %>%
 2   group_by(month, origin) %>%
 3   summarize(
 4     dep_delay_avg = round(mean(dep_delay, na.rm = TRUE), 1),
 5     arr_delay_avg = round(mean(arr_delay, na.rm = TRUE), 1)
 6   ) %>%
 7   arrange(-arr_delay_avg)
```

```
# A tibble: 36 × 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

```
 1  flights %>%
 2    select(month, origin, ends_with("delay")) %>%
 3    group_by(month, origin) %>%
 4    summarize(across(ends_with("delay"), ~round(mean(., na.rm = TRUE), 1))) %>%
 5    arrange(-arr_delay)
```

```
# A tibble: 36 × 4
# Groups:   month [12]
   month origin dep_delay arr_delay
   <int> <chr>      <dbl>     <dbl>
 1     7 JFK         23.8      20.2
 2    12 EWR         21        19.6
 3     6 JFK         20.5      17.6
 4     6 EWR         22.5      16.9
 5     7 EWR         22        15.5
 6     6 LGA         19.3      14.8
 7     7 LGA         19        14.2
 8     4 EWR         17.4      14.1
 9     1 EWR         14.9      12.8
10    12 JFK         14.8      12.7
# … with 26 more rows
```


Other Helpful {tidyverse} Functions

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

Find Unique Values

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

```
1 distinct(mpg, manufacturer)
```

```
# A tibble: 15 × 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):

```
1 distinct(mpg, hwy)
```

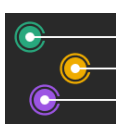
```
# A tibble: 27 × 1
     hwy
   <int>
 1    29
 2    31
 3    30
 4    26
 5    27
 6    25
 7    28
 8    24
 9    23
10    20
# … with 17 more rows
```


Count Unique Values

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

```
1  n_distinct(mpg$model)  ## only works with vectors, no 1st data argument
```

```
[1] 38
```

Extract Rows by Ordinal Position

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

```
1  slice(mpg, 100:120)
```

```
# A tibble: 21 × 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(m5) f        28    33 r     subcompact
 2 honda        civic    1.6  1999     4 auto(l4)   f        24    32 r     subcompact
 3 honda        civic    1.6  1999     4 manual(m5) f        25    32 r     subcompact
 4 honda        civic    1.6  1999     4 manual(m5) f        23    29 p     subcompact
 5 honda        civic    1.6  1999     4 auto(l4)   f        24    32 r     subcompact
 6 honda        civic    1.8  2008     4 manual(m5) f        26    34 r     subcompact
 7 honda        civic    1.8  2008     4 auto(l5)   f        25    36 r     subcompact
 8 honda        civic    1.8  2008     4 auto(l5)   f        24    36 c     subcompact
 9 honda        civic    2    2008     4 manual(m6) f        21    29 p     subcompact
10 hyundai      sonata   2.4  1999     4 auto(l4)   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:

```
1 mpg %>% group_by(manufacturer) %>% slice(1)
```

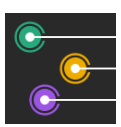
```
# A tibble: 15 × 11
# Groups:   manufacturer [15]
   manufacturer model              displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>              <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 audi         a4                   1.8  1999     4 auto(l5)   f        18    29 p     compact
 2 chevrolet    c1500 suburban 2wd   5.3  2008     8 auto(l4)   r        14    20 r     suv
 3 dodge        caravan 2wd          2.4  1999     4 auto(l3)   f        18    24 r     minivan
 4 ford         expedition 2wd       4.6  1999     8 auto(l4)   r        11    17 r     suv
 5 honda        civic                1.6  1999     4 manual(m5) f        28    33 r     subcompact
 6 hyundai      sonata               2.4  1999     4 auto(l4)   f        18    26 r     midsize
 7 jeep         grand cherokee 4wd   3    2008     6 auto(l5)   4        17    22 d     suv
 8 land rover   range rover          4    1999     8 auto(l4)   4        11    15 p     suv
 9 lincoln      navigator 2wd        5.4  1999     8 auto(l4)   r        11    17 r     suv
10 mercury      mountaineer 4wd      4    1999     6 auto(l5)   4        14    17 r     suv
11 nissan       altima               2.4  1999     4 manual(m5) f        21    29 r     compact
12 pontiac      grand prix           3.1  1999     6 auto(l4)   f        18    26 r     midsize
13 subaru       forester awd         2.5  1999     4 manual(m5) 4        18    25 r     suv
14 toyota       4runner 4wd          2.7  1999     4 manual(m5) 4        15    20 r     suv
15 volkswagen   gti                  2    1999     4 manual(m5) f        21    29 r     compact
```


Select Rows Randomly

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

```
1 slice_sample(mpg, n = 5)
```

```
# A tibble: 5 × 11
  manufacturer model                displ  year   cyl trans      drv     cty   hwy fl    class
  <chr>        <chr>                <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
1 hyundai      tiburon                2    1999     4 manual(m5) f        19    29 r     subcompact
2 volkswagen   gti                    2    1999     4 manual(m5) f        21    29 r     compact
3 toyota       4runner 4wd            2.7  1999     4 auto(l4)   4        16    20 r     suv
4 honda        civic                  1.8  2008     4 manual(m5) f        26    34 r     subcompact
5 toyota       land cruiser wagon 4wd 5.7  2008     8 auto(s6)   4        13    18 r     suv
```

Select Rows Randomly

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

```
1  mpg %>% group_by(year, manufacturer) %>% slice_sample(n = 1)
```

```
# A tibble: 30 × 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                   2.8  1999     6 auto(l5)   f        16    26 p     compact
 2 chevrolet    corvette             5.7  1999     8 manual(m6) r        16    26 p     2seater
 3 dodge        durango 4wd          5.2  1999     8 auto(l4)   4        11    16 r     suv
 4 ford         expedition 2wd       5.4  1999     8 auto(l4)   r        11    17 r     suv
 5 honda        civic                1.6  1999     4 manual(m5) f        28    33 r     subcompact
 6 hyundai      sonata               2.4  1999     4 auto(l4)   f        18    26 r     midsize
 7 jeep         grand cherokee 4wd   4    1999     6 auto(l4)   4        15    20 r     suv
 8 land rover   range rover          4    1999     8 auto(l4)   4        11    15 p     suv
 9 lincoln      navigator 2wd        5.4  1999     8 auto(l4)   r        11    17 r     suv
10 mercury      mountaineer 4wd      4    1999     6 auto(l5)   4        14    17 r     suv
# … with 20 more rows
```


Select Rows Randomly

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

```
1  slice_sample(mpg, prop = .05)
```

```
# A tibble: 11 × 11
   manufacturer model              displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>              <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 ford         mustang              4.6  1999     8 auto(l4)   r        15    21 r     subcompact
 2 dodge        dakota pickup 4wd    3.7  2008     6 manual(m6) 4        15    19 r     pickup
 3 hyundai      tiburon              2    1999     4 manual(m5) f        19    29 r     subcompact
 4 dodge        caravan 2wd          3.3  2008     6 auto(l4)   f        11    17 e     minivan
 5 toyota       4runner 4wd          3.4  1999     6 auto(l4)   4        15    19 r     suv
 6 chevrolet    k1500 tahoe 4wd      5.3  2008     8 auto(l4)   4        11    14 e     suv
 7 chevrolet    malibu               2.4  1999     4 auto(l4)   f        19    27 r     midsize
 8 dodge        ram 1500 pickup 4wd  4.7  2008     8 manual(m6) 4         9    12 e     pickup
 9 toyota       corolla              1.8  1999     4 manual(m5) f        26    35 r     compact
10 toyota       corolla              1.8  2008     4 auto(l4)   f        26    35 r     compact
11 volkswagen   new beetle           2    1999     4 auto(l4)   f        19    26 r     subcompact
```


Pick Top or Bottom Values

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

```
1  slice_max(mpg, n = 5, order_by = hwy)
```

```
# A tibble: 6 × 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 manual(m5) f        33    44 d     compact
2 volkswagen   new beetle   1.9  1999     4 manual(m5) f        35    44 d     subcompact
3 volkswagen   new beetle   1.9  1999     4 auto(l4)   f        29    41 d     subcompact
4 toyota       corolla      1.8  2008     4 manual(m5) f        28    37 r     compact
5 honda        civic        1.8  2008     4 auto(l5)   f        25    36 r     subcompact
6 honda        civic        1.8  2008     4 auto(l5)   f        24    36 c     subcompact
```


Pick Top or Bottom Values

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

```
1  slice_max(mpg, n = 5, order_by = hwy, with_ties = FALSE)
```

```
# A tibble: 5 × 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 manual(m5) f        33    44 d     compact
2 volkswagen   new beetle   1.9  1999     4 manual(m5) f        35    44 d     subcompact
3 volkswagen   new beetle   1.9  1999     4 auto(l4)   f        29    41 d     subcompact
4 toyota       corolla      1.8  2008     4 manual(m5) f        28    37 r     compact
5 honda        civic        1.8  2008     4 auto(l5)   f        25    36 r     subcompact
```


Pick Top or Bottom Values

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

```
1  slice_min(mpg, n = 5, order_by = hwy)
```

```
# A tibble: 5 × 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 pickup 4wd    4.7  2008     8 auto(l5)   4         9    12 e     pickup
2 dodge        durango 4wd          4.7  2008     8 auto(l5)   4         9    12 e     suv
3 dodge        ram 1500 pickup 4wd  4.7  2008     8 auto(l5)   4         9    12 e     pickup
4 dodge        ram 1500 pickup 4wd  4.7  2008     8 manual(m6) 4         9    12 e     pickup
5 jeep         grand cherokee 4wd   4.7  2008     8 auto(l5)   4         9    12 e     suv
```


Pick Top or Bottom Values

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

```
1  mpg %>% group_by(manufacturer) %>% slice_max(n = 1, order_by = hwy)
```

```
# A tibble: 25 × 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(m6) f        20    31 p     compact
 2 chevrolet    malibu        2.4  2008     4 auto(l4)   f        22    30 r     midsize
 3 dodge        caravan 2wd   2.4  1999     4 auto(l3)   f        18    24 r     minivan
 4 dodge        caravan 2wd   3    1999     6 auto(l4)   f        17    24 r     minivan
 5 dodge        caravan 2wd   3.3  2008     6 auto(l4)   f        17    24 r     minivan
 6 dodge        caravan 2wd   3.3  2008     6 auto(l4)   f        17    24 r     minivan
 7 ford         mustang       3.8  1999     6 manual(m5) r        18    26 r     subcompact
 8 ford         mustang       4    2008     6 manual(m5) r        17    26 r     subcompact
 9 honda        civic         1.8  2008     4 auto(l5)   f        25    36 r     subcompact
10 honda        civic         1.8  2008     4 auto(l5)   f        24    36 c     subcompact
# … with 15 more rows
```


Pick Top or Bottom Values

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

```
1  slice_max(mpg, prop = .01, order_by = displ)
```

```
# A tibble: 2 × 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(m6) r        15    24 p     2seater
2 chevrolet    k1500 tahoe 4wd   6.5  1999     8 auto(l4)   4        14    17 d     suv
```


Count Observations

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

```
1 count(mpg)
```

```
# A tibble: 1 × 1
      n
  <int>
1   234
```


Count Observations

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

```
1 count(mpg)
```

Equivalent code in base R:

```
1 nrow(mpg)
```

```
[1] 234
```


Count Observations

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

```
1 count(mpg)
```

```
# A tibble: 1 × 1
      n
  <int>
1   234
```

```
1 summarize(mpg, n = n())
```

```
# A tibble: 1 × 1
      n
  <int>
1   234
```


Count Observations

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

```
1  count(mpg, manufacturer)
```

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

```
1  mpg %>% group_by(manufacturer) %>% summarize(n = n())
```

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


Count Observations

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

```
1  count(mpg, manufacturer, sort = TRUE)
```

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

```
1  mpg %>% group_by(manufacturer) %>% summarize(n = n()) %>% arrange(-n)
```


Count Observations

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

```
1  add_count(mpg2)
```

```
# A tibble: 234 × 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

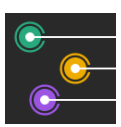
`add_count()` adds the sample size of your data as additional column:

```
1 add_count(mpg2)
```

```
# A tibble: 234 × 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
```

```
1 mutate(mpg2, n = n())
```

```
# A tibble: 234 × 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:

```
1  add_count(mpg2, model)
```

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

```
1  mpg2 %>% group_by(model) %>% mutate(n = n())
```

```
# A tibble: 234 × 4
# Groups:   model [38]
   model        hwy   cty     n
   <chr>      <int> <int> <int>
 1 a4            29    18     7
 2 a4            29    21     7
 3 a4            31    20     7
 4 a4            30    21     7
 5 a4            26    16     7
 6 a4            26    18     7
 7 a4            27    18     7
 8 a4 quattro    26    18     8
 9 a4 quattro    25    16     8
10 a4 quattro    28    20     8
# … with 224 more rows
```


Join Data Sets

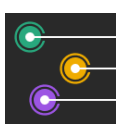
Several `*_join()` functions let you merge two data tables:

```
  1  band_members
```

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

```
  1  band_instruments
```

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

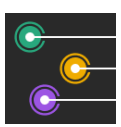



Join Data Sets

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

```
1 left_join(band_members, band_instruments)
```

```
# A tibble: 3 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
```

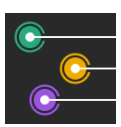



Join Data Sets

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

```
1 right_join(band_members, band_instruments)
```

```
# A tibble: 3 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
3 Keith <NA>    guitar
```

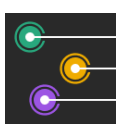



Join Data Sets

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

```
1 inner_join(band_members, band_instruments)
```

```
# A tibble: 2 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
```

Join Data Sets

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

```
1 full_join(band_members, band_instruments)
```

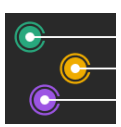
```
# A tibble: 4 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
4 Keith <NA>    guitar
```


Join Data Sets

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

```
1 full_join(band_members, band_instruments, by = "name")
```

```
# A tibble: 4 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
4 Keith <NA>    guitar
```

Join Data Sets

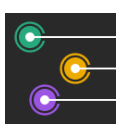
Several `*_join()` functions let you merge two data tables:

```
  1  band_members
```

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

```
  1  band_instruments2
```

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

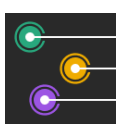



Join Data Sets

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

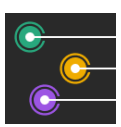
```
1 full_join(band_members, band_instruments2, by = c("name" = "artist"))
```

```
# A tibble: 4 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
4 Keith <NA>    guitar
```

Data Wrangling

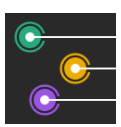
— Tidy Data —



The {tidyr} Package

The **{tidyr}** package contains tools for changing the shape (pivoting) and hierarchy (nesting and unnesting) of a data table.

```
1 # install.packages("tidyr")
2 library(tidyr)
```

Data Comes in Many Shapes

Wide format:

- variables and/or observations are spread across multiple columns
- some values are encoded in the column names

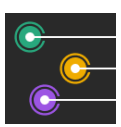
group	metric_x_22	metric_y_22	metric_x_23	metric_y_23
A	46	12	32	1
B	2	35	16	42
C	21	24	7	27

Data Comes in Many Shapes

Long format:

- rows are single measurements
- all measurements are stored in a single column

group	year	metric	value
A	2022	x	46
B	2022	x	2
C	2022	x	21
A	2023	x	32
B	2023	x	16
C	2023	x	7
A	2022	y	12
B	2022	y	35
C	2022	y	24
A	2023	y	1
B	2023	y	42
C	2023	y	27



Data Comes in Many Shapes

Long format:

- rows are single measurements
- all measurements are stored in a single column

group	year	metric	value
A	2022	x	46
B	2022	x	2
C	2022	x	21
A	2023	x	32
B	2023	x	16
C	2023	x	7
A	2022	y	12
B	2022	y	35
C	2022	y	24
A	2023	y	1
B	2023	y	42
C	2023	y	27

Tidy format:

- rows are observations, columns are variables
- all values are cell inputs

group	year	metric_x	metric_y
A	2022	10	32
B	2022	2	35
C	2022	13	10
A	2023	12	43
B	2023	16	42
C	2023	7	27

Create All Possible Combinations

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

```
1 mpg3 <- count(mpg, manufacturer, class)
2 mpg3
```

```
# A tibble: 32 × 3
   manufacturer class          n
   <chr>        <chr>      <int>
 1 audi         compact       15
 2 audi         midsize        3
 3 chevrolet    2seater        5
 4 chevrolet    midsize        5
 5 chevrolet    suv            9
 6 dodge        minivan       11
 7 dodge        pickup        19
 8 dodge        suv            7
 9 ford         pickup         7
10 ford         subcompact     9
# … with 22 more rows
```


Create All Possible Combinations

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

```
1 mpg3 <- count(mpg, manufacturer, class)
2 complete(mpg3, manufacturer, nesting(class))
```

```
# A tibble: 105 × 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

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

```
1 mpg3 <- count(mpg, manufacturer, class)
2 complete(mpg3, manufacturer, nesting(class), fill = list(n = 0))
```

```
# A tibble: 105 × 3
   manufacturer class          n
   <chr>        <chr>      <int>
 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
```


Reshaping Data

`pivot_longer()` to turn a wide data set into a long one:

```
1  mpg4 <-
2    mpg %>%
3    group_by(manufacturer, model, trans, cyl, year) %>%
4    summarize(cty = mean(cty, na.rm = TRUE))
```

```
1  mpg4
```

```
# A tibble: 183 × 6
# Groups:   manufacturer, model, trans, cyl [157]
   manufacturer model      trans        cyl  year   cty
   <chr>        <chr>      <chr>      <int> <int> <dbl>
 1 audi         a4         auto(av)       4  2008    21
 2 audi         a4         auto(av)       6  2008    18
 3 audi         a4         auto(l5)       4  1999    18
 4 audi         a4         auto(l5)       6  1999    16
 5 audi         a4         manual(m5)     4  1999    21
 6 audi         a4         manual(m5)     6  1999    18
 7 audi         a4         manual(m6)     4  2008    20
 8 audi         a4 quattro auto(l5)       4  1999    16
 9 audi         a4 quattro auto(l5)       6  1999    15
10 audi         a4 quattro auto(s6)       4  2008    19
# … with 173 more rows
```


Reshaping Data

`pivot_longer()` to turn a wide data set into a long one:

```
mpg_wide <- pivot_wider(
  mpg4,
  id_cols = c(manufacturer, model, trans, cyl),
  names_from = year,  ## use as names for columns
  values_from = cty   ## use as values in these columns
)
```

```
mpg_wide
```

```
# A tibble: 157 × 6
# Groups:   manufacturer, model, trans, cyl [157]
   manufacturer model      trans        cyl `2008` `1999`
   <chr>        <chr>      <chr>      <int>  <dbl>  <dbl>
 1 audi         a4         auto(av)       4     21     NA
 2 audi         a4         auto(av)       6     18     NA
 3 audi         a4         auto(l5)       4     NA     18
 4 audi         a4         auto(l5)       6     NA     16
 5 audi         a4         manual(m5)     4     NA     21
 6 audi         a4         manual(m5)     6     NA     18
 7 audi         a4         manual(m6)     4     20     NA
 8 audi         a4 quattro auto(l5)       4     NA     16
 9 audi         a4 quattro auto(l5)       6     NA     15
10 audi         a4 quattro auto(s6)       4     19     NA
# … with 147 more rows
```


Reshaping Data

`pivot_longer()` to turn a wide data set into a long one:

```
mpg_wide <- pivot_wider(
  mpg4,
  id_cols = c(manufacturer, model, trans, cyl),
  names_from = year,
  values_from = cty,
  names_prefix = "cty_"  ## ad "cty_" to each new column name (-> year)
)
```

```
mpg_wide
```

```
# A tibble: 157 × 6
# Groups:   manufacturer, model, trans, cyl [157]
   manufacturer model      trans        cyl cty_2008 cty_1999
   <chr>        <chr>      <chr>      <int>    <dbl>    <dbl>
 1 audi         a4         auto(av)       4       21       NA
 2 audi         a4         auto(av)       6       18       NA
 3 audi         a4         auto(l5)       4       NA       18
 4 audi         a4         auto(l5)       6       NA       16
 5 audi         a4         manual(m5)     4       NA       21
 6 audi         a4         manual(m5)     6       NA       18
 7 audi         a4         manual(m6)     4       20       NA
 8 audi         a4 quattro auto(l5)       4       NA       16
 9 audi         a4 quattro auto(l5)       6       NA       15
10 audi         a4 quattro auto(s6)       4       19       NA
# … with 147 more rows
```


Reshaping Data

`pivot_wider()` to turn a long data set into a wide one:

```
pivot_wider(
  mpg4,
  id_cols = c(manufacturer, trans, cyl), ## removed model
  names_from = year,
  values_from = cty,
  names_prefix = "cty_"
)
```

```
# A tibble: 95 × 5
# Groups:   manufacturer, trans, cyl [95]
   manufacturer trans        cyl cty_2008  cty_1999
   <chr>        <chr>      <int> <list>    <list>
 1 audi         auto(av)       4 <dbl [1]> <NULL>
 2 audi         auto(av)       6 <dbl [1]> <NULL>
 3 audi         auto(l5)       4 <NULL>    <dbl [2]>
 4 audi         auto(l5)       6 <NULL>    <dbl [3]>
 5 audi         manual(m5)     4 <NULL>    <dbl [2]>
 6 audi         manual(m5)     6 <NULL>    <dbl [2]>
 7 audi         manual(m6)     4 <dbl [2]> <NULL>
 8 audi         auto(s6)       4 <dbl [1]> <NULL>
 9 audi         auto(s6)       6 <dbl [2]> <NULL>
10 audi         manual(m6)     6 <dbl [1]> <NULL>
# … with 85 more rows
```


Reshaping Data

`pivot_longer()` to turn a wide data set into a long one:

```
mpg_wide %>%
  pivot_longer(
    cols = starts_with("cty_"),
    names_to = "year",  ## column names as values in new column "year"
    values_to = "cty",  ## values from these column in new column "cty"
    names_prefix = "cty_"  ## remove "cty_" from the column names
  )
```

```
# A tibble: 314 × 6
# Groups:   manufacturer, model, trans, cyl [157]
   manufacturer model trans        cyl year    cty
   <chr>        <chr> <chr>      <int> <chr> <dbl>
 1 audi         a4    auto(av)       4 2008     21
 2 audi         a4    auto(av)       4 1999     NA
 3 audi         a4    auto(av)       6 2008     18
 4 audi         a4    auto(av)       6 1999     NA
 5 audi         a4    auto(l5)       4 2008     NA
 6 audi         a4    auto(l5)       4 1999     18
 7 audi         a4    auto(l5)       6 2008     NA
 8 audi         a4    auto(l5)       6 1999     16
 9 audi         a4    manual(m5)     4 2008     NA
10 audi         a4    manual(m5)     4 1999     21
# … with 304 more rows
```

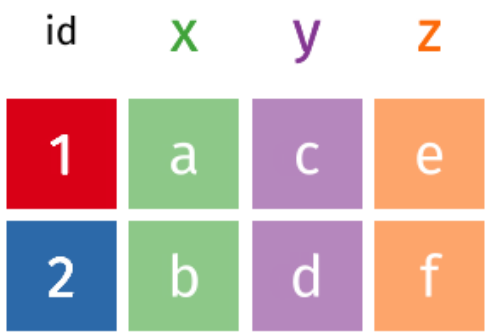

Reshaping Data

`pivot_longer()` to turn a wide data set into a long one:

```
mpg_wide %>%
  pivot_longer(
    cols = starts_with("cty_"),
    names_to = "year",
    values_to = "cty",
    names_prefix = "cty_",
    values_drop_na = TRUE
  )
```

```
# A tibble: 183 × 6
# Groups:   manufacturer, model, trans, cyl [157]
   manufacturer model      trans        cyl year    cty
   <chr>        <chr>      <chr>      <int> <chr> <dbl>
 1 audi         a4         auto(av)       4 2008     21
 2 audi         a4         auto(av)       6 2008     18
 3 audi         a4         auto(l5)       4 1999     18
 4 audi         a4         auto(l5)       6 1999     16
 5 audi         a4         manual(m5)     4 1999     21
 6 audi         a4         manual(m5)     6 1999     18
 7 audi         a4         manual(m6)     4 2008     20
 8 audi         a4 quattro auto(l5)       4 1999     16
 9 audi         a4 quattro auto(l5)       6 1999     15
10 audi         a4 quattro auto(s6)       4 2008     19
# … with 173 more rows
```

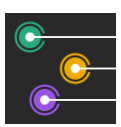

wide



Created by Garrick Aden-Buie

Data Wrangling

— Working with Factors —



The {forcats} Package

The **{forcats}** package provides helpers for working with categorical, ordered data.

```
# install.packages("forcats")
library(forcats)
```

```
mpg_fct <- mutate(mpg, man_fct = factor(manufacturer))
```

```
levels(mpg_fct$man_fct)
```

```
[1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"       "land rover"
[9] "lincoln"    "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen"
```


Working with Factors

The `fct_*` functions simplify the process of reordering factor levels:

```
1 f1 <- mpg_fct %>% mutate(man_fct = fct_inorder(man_fct))
2 levels(f1$man_fct)
```

```
[1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"       "land rover"
[9] "lincoln"    "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen"
```

```
1 f2 <- mpg_fct %>% mutate(man_fct = fct_infreq(man_fct))
2 levels(f2$man_fct)
```

```
[1] "dodge"      "toyota"     "volkswagen" "ford"       "chevrolet"  "audi"       "hyundai"    "subaru"
[9] "nissan"     "honda"      "jeep"       "pontiac"    "land rover" "mercury"    "lincoln"
```

```
1 f3 <- mpg_fct %>% mutate(man_fct = fct_rev(man_fct))
2 levels(f3$man_fct)
```

```
[1] "volkswagen" "toyota"     "subaru"     "pontiac"    "nissan"     "mercury"    "lincoln"    "land rover"
[9] "jeep"       "hyundai"    "honda"      "ford"       "dodge"      "chevrolet"  "audi"
```


Working with Factors

The `fct_*` functions simplify the process of reordering factor levels:

```
1 f4 <- mpg_fct %>% mutate(man_fct = fct_relevel(man_fct, "volkswagen", after = 1))
2 levels(f4$man_fct)
```

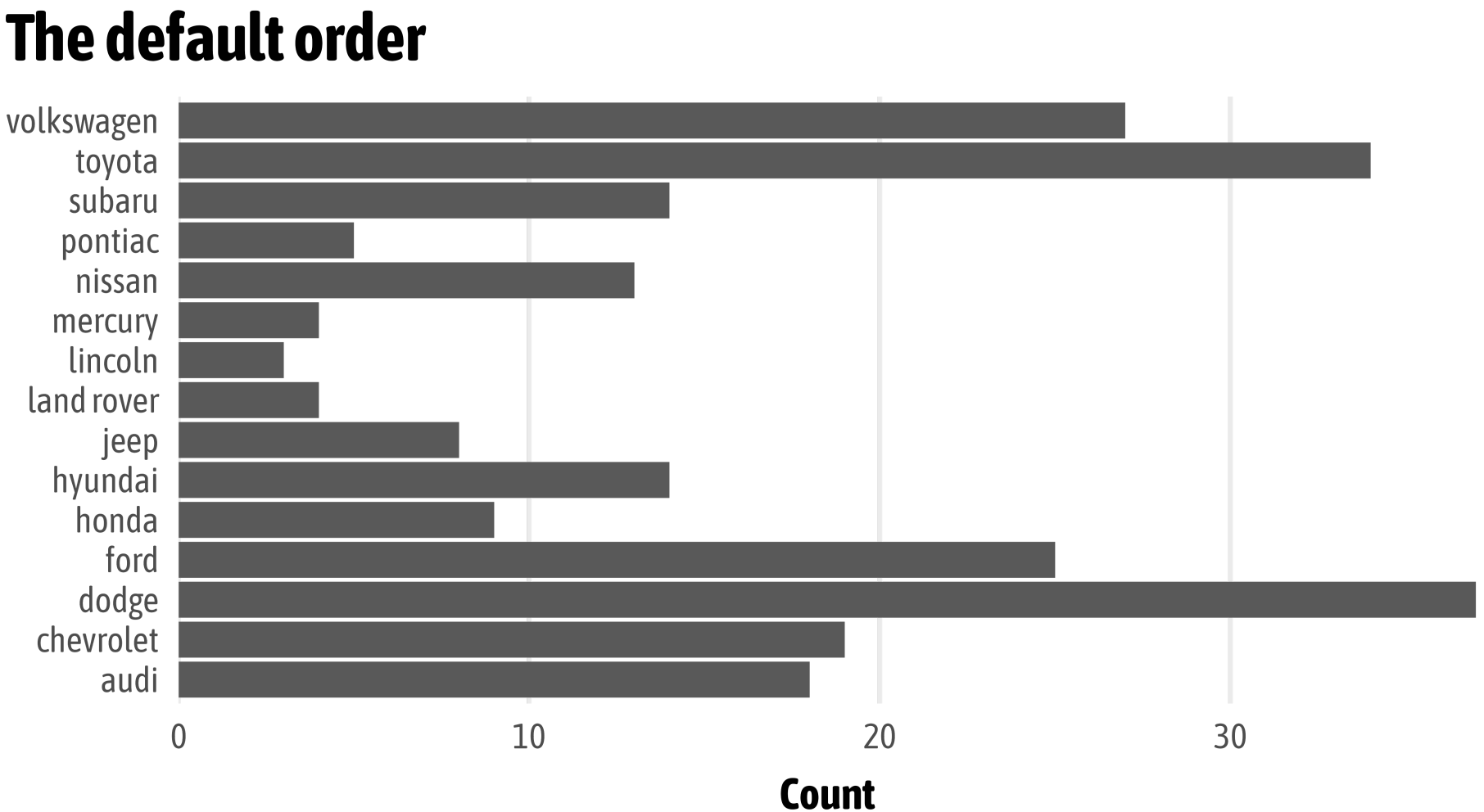
```
[1] "audi"       "volkswagen" "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"
[9] "land rover" "lincoln"    "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"
```

```
1 f5 <- mpg_fct %>% mutate(man_fct = fct_relevel(man_fct, "audi", after = Inf))
2 levels(f5$man_fct)
```

```
[1] "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"       "land rover" "lincoln"
[9] "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen" "audi"
```

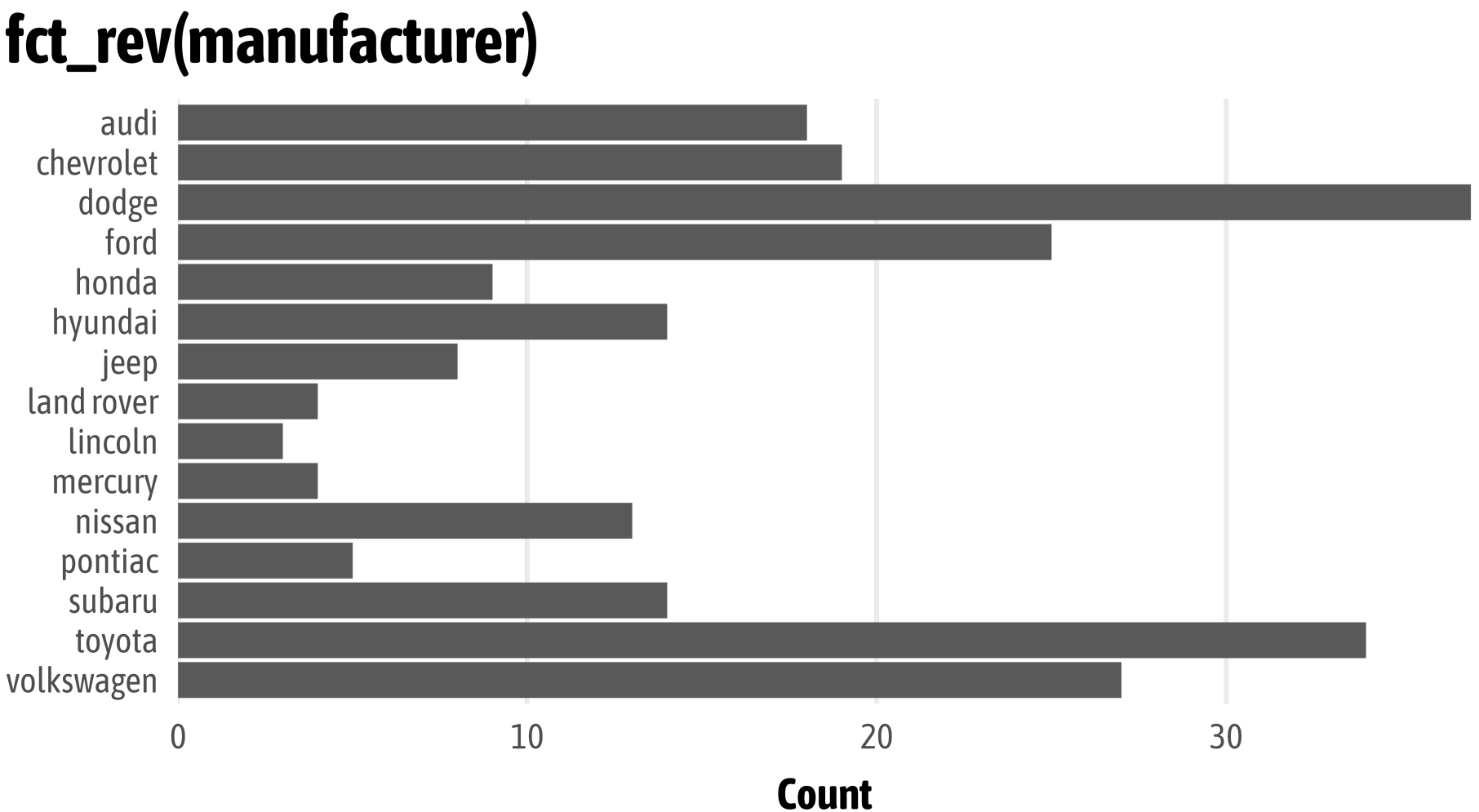

Working with Factors

The `fct_*` functions are especially helpful when plotting categorical variables:



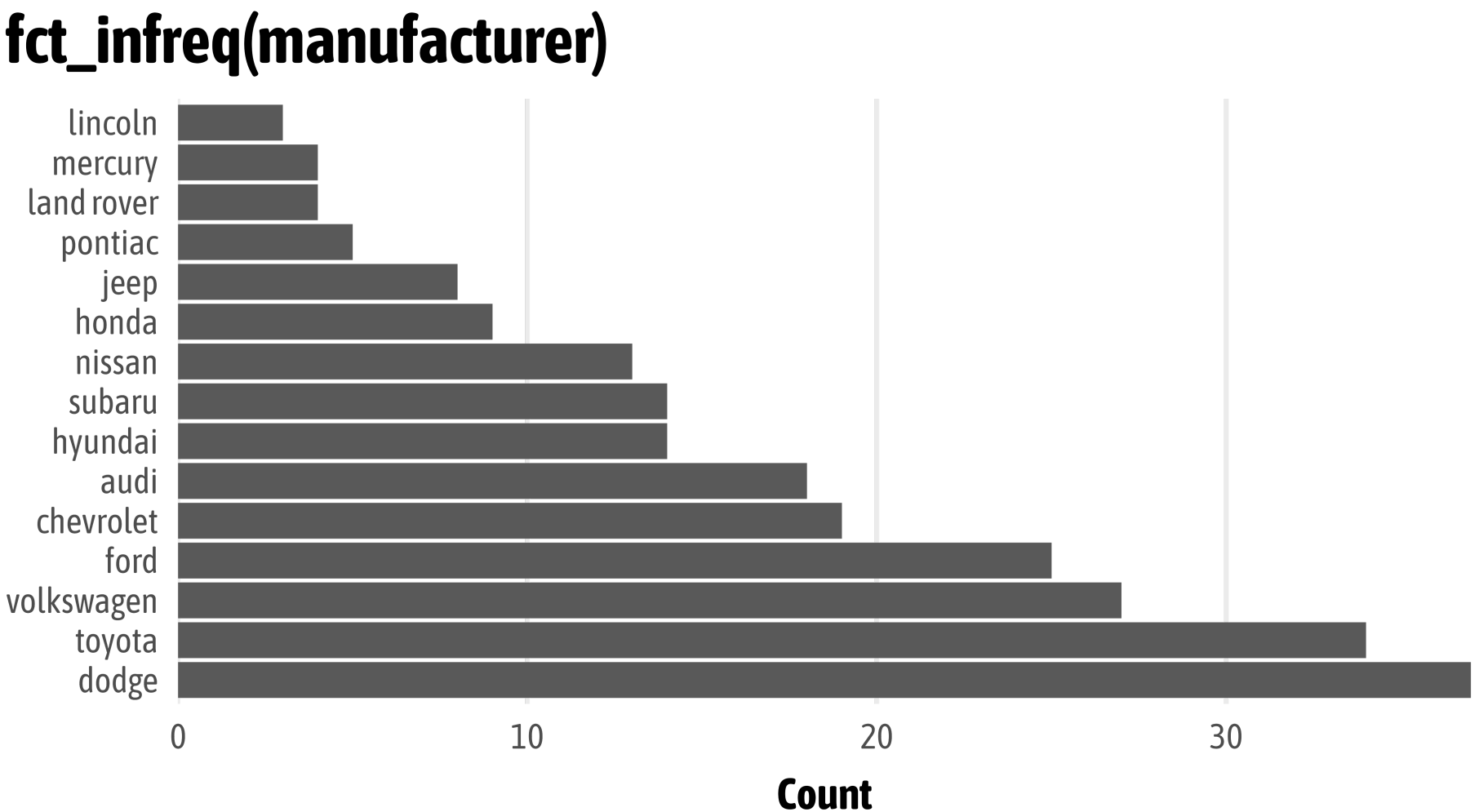
Working with Factors

The `fct_*` functions are especially helpful when plotting categorical variables:



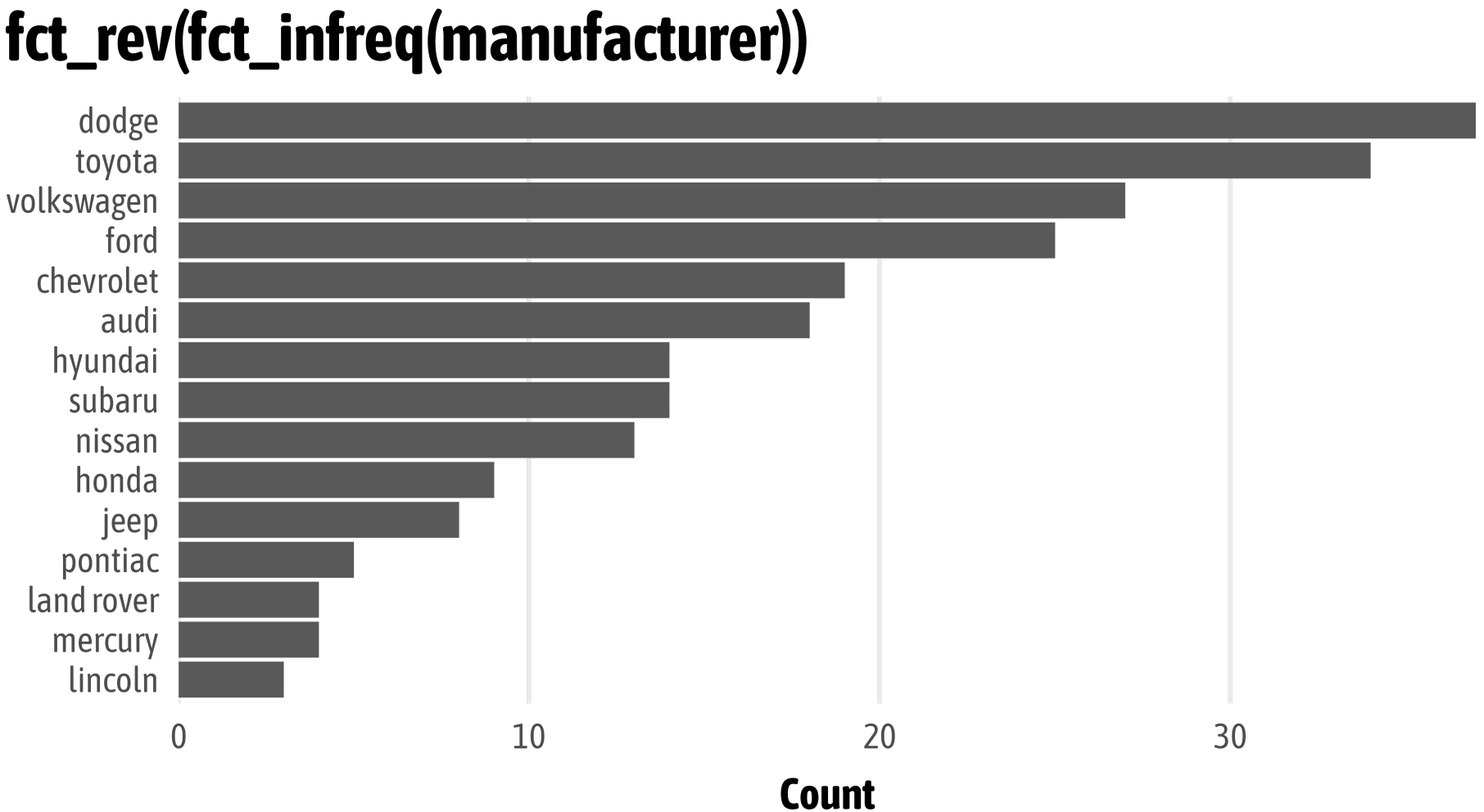
Working with Factors

The `fct_*` functions are especially helpful when plotting categorical variables:



Working with Factors

The `fct_*` functions are especially helpful when plotting categorical variables:



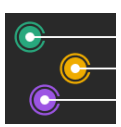
Working with Factors

You can also order factor levels based on one or two weights:

```
mpg_fct <- mpg_fct %>% group_by(man_fct) %>% mutate(avg_displ = mean(displ)) %>% ungroup()
```

```
mpg_fct
```

```
# A tibble: 234 × 13
   manufacturer model      displ  year   cyl trans      drv     cty   hwy fl    class   man_fct avg_displ
   <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>   <fct>       <dbl>
 1 audi         a4           1.8  1999     4 auto(l5)   f        18    29 p     compact audi         2.54
 2 audi         a4           1.8  1999     4 manual(m5) f        21    29 p     compact audi         2.54
 3 audi         a4           2    2008     4 manual(m6) f        20    31 p     compact audi         2.54
 4 audi         a4           2    2008     4 auto(av)   f        21    30 p     compact audi         2.54
 5 audi         a4           2.8  1999     6 auto(l5)   f        16    26 p     compact audi         2.54
 6 audi         a4           2.8  1999     6 manual(m5) f        18    26 p     compact audi         2.54
 7 audi         a4           3.1  2008     6 auto(av)   f        18    27 p     compact audi         2.54
 8 audi         a4 quattro   1.8  1999     4 manual(m5) 4        18    26 p     compact audi         2.54
 9 audi         a4 quattro   1.8  1999     4 auto(l5)   4        16    25 p     compact audi         2.54
10 audi         a4 quattro   2    2008     4 manual(m6) 4        20    28 p     compact audi         2.54
# … with 224 more rows
```

Working with Factors

You can also order factor levels based on one or two weights:

```
mpg_fct <- mpg_fct %>% group_by(man_fct) %>% mutate(avg_displ = mean(displ)) %>% ungroup()
```

```
f6 <- mpg_fct %>% mutate(man_fct = fct_reorder(man_fct, avg_displ))  ### also `fct_reorder2()`
levels(f6$man_fct)
```

```
[1] "honda"      "volkswagen" "hyundai"    "subaru"     "audi"       "toyota"     "nissan"     "pontiac"
[9] "land rover" "dodge"      "mercury"    "ford"       "jeep"       "chevrolet"  "lincoln"
```


Working with Factors

The **{forcats}** package also provides functions to "lump" levels:

```
1 f7 <- mpg_fct %>% mutate(man_fct = fct_lump_n(man_fct, 10))
2 levels(f7$man_fct)
```

```
[1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "nissan"     "subaru"
[9] "toyota"     "volkswagen" "Other"
```

```
1 f8 <- mpg_fct %>% mutate(man_fct = fct_lump_n(man_fct, 10, other_level = "other manufacturers"))
2 levels(f8$man_fct)
```

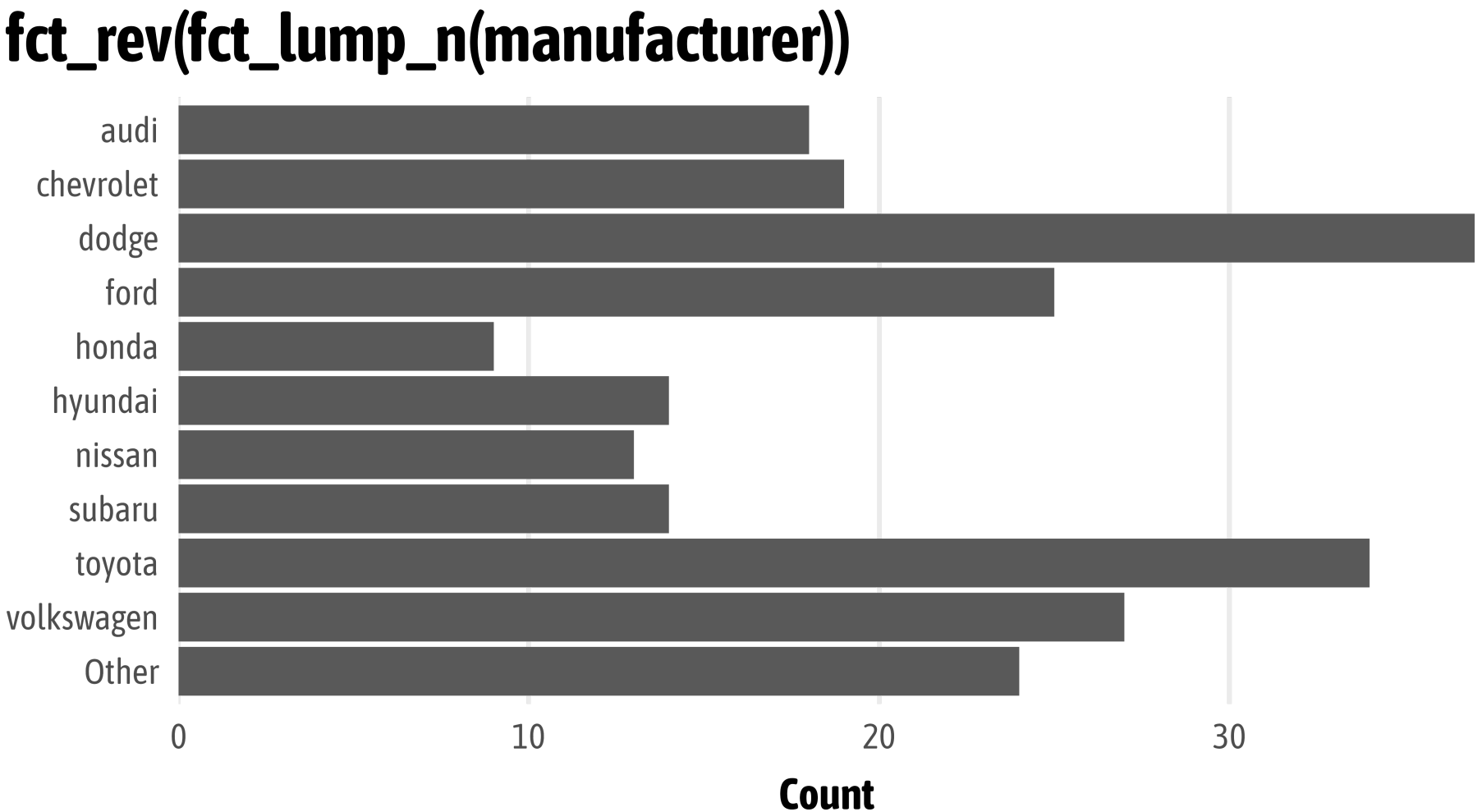
```
[1] "audi"                "chevrolet"           "dodge"               "ford"
[5] "honda"               "hyundai"             "nissan"              "subaru"
[9] "toyota"              "volkswagen"          "other manufacturers"
```

```
1 f8 <- mpg_fct %>% mutate(man_fct = fct_lump_lowfreq(man_fct))
2 levels(f8$man_fct)
```

```
[1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"       "land rover"
[9] "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen" "Other"
```

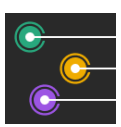

Working with Factors

The **{forcats}** package also provides functions to "lump" levels:



Data Wrangling

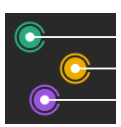
— Working with Strings —



The {stringr} Package

The **{stringr}** package contains consistent, simple and easy to use set of wrappers to work with strings.

```
1 # install.packages("stringr")
2 library(stringr)
```

Working with Strings

Functions from the **{stringr}** package operate on character strings:

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

```
manufacturers
```

```
 [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"       "land rover"
 [9] "lincoln"    "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen"
```

```
typeof(manufacturers)
```

```
[1] "character"
```


Working with Strings

The `str_*` functions offer a consistent syntax to evaluate and manipulate strings:

```
1 str_length(manufacturers)
```

```
[1]  4  9  5  4  5  7  4 10  7  7  6  7  6  6 10
```

```
1 str_sub(manufacturers, 1, 3)
```

```
[1] "aud" "che" "dod" "for" "hon" "hyu" "jee" "lan" "lin" "mer" "nis" "pon" "sub" "toy" "vol"
```

```
1 str_subset(manufacturers, "y")
```

```
[1] "hyundai" "mercury" "toyota"
```

```
1 str_subset(manufacturers, "[ac]")
```

```
[1] "audi"       "chevrolet"  "honda"      "hyundai"    "land rover" "lincoln"    "mercury"    "nissan"
[9] "pontiac"    "subaru"     "toyota"     "volkswagen"
```

```
1 str_detect(manufacturers, "[ac]")
```

```
[1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
1 str_starts(manufacturers, "[ac]")  ## also `str_ends()`
```

```
[1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```


Working with Strings

The `str_*` functions offer a consistent syntax to evaluate and manipulate strings:

```
1 str_count(manufacturers, "[aeiou]")
```

```
[1] 3 3 2 1 2 3 2 3 2 2 2 3 3 3 3
```

```
1 str_extract(manufacturers, "[aeiou]")
```

```
[1] "a" "e" "o" "o" "o" "u" "e" "a" "i" "e" "i" "o" "u" "o" "o"
```

```
1 str_extract_all(manufacturers, "[aeiou]")
```

```
[[1]]
[1] "a" "u" "i"

[[2]]
[1] "e" "o" "e"

[[3]]
[1] "o" "e"

[[4]]
[1] "o"

[[5]]
[1] "o" "a"

[[6]]
[1] "u" "a" "i"
```


Working with Strings

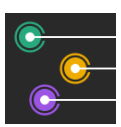
The `str_*` functions offer a consistent syntax to evaluate and manipulate strings:

```
1 str_replace(manufacturers, "[aeiou]", "X")
```

```
[1] "Xudi"       "chXvrolet"  "dXdge"      "fXrd"       "hXnda"      "hyXndai"    "jXep"       "lXnd rover"
[9] "lXncoln"    "mXrcury"    "nXssan"     "pXntiac"    "sXbaru"     "tXyota"     "vXlkswagen"
```

```
1 str_replace_all(manufacturers, "[aeiou]", "X")
```

```
[1] "XXdX"       "chXvrXlXt"  "dXdgX"      "fXrd"       "hXndX"      "hyXndXX"    "jXXp"       "lXnd rXvXr"
[9] "lXncXln"    "mXrcXry"    "nXssXn"     "pXntXXc"    "sXbXrX"     "tXyXtX"     "vXlkswXgXn"
```

Working with Strings

The `str_*` functions offer a consistent syntax to evaluate and manipulate strings:

```
1 str_remove(manufacturers, "[lrs]")
```

```
 [1] "audi"      "chevolet"  "dodge"     "fod"       "honda"     "hyundai"   "jeep"      "and rover"
 [9] "incoln"    "mecury"    "nisan"     "pontiac"   "ubaru"     "toyota"    "vokswagen"
```

```
1 str_remove_all(manufacturers, "[lrs]")
```

```
 [1] "audi"     "chevoet"  "dodge"    "fod"      "honda"    "hyundai"  "jeep"     "and ove"  "incon"
[10] "mecuy"    "nian"     "pontiac"  "ubau"     "toyota"   "vokwagen"
```


Working with Strings

The `str_*` functions offer a consistent syntax to evaluate and manipulate strings:

```
1 str_trim(c("we", "do not", "like ", " malformatted variables "))
```

```
[1] "we"                   "do not"               "like"                 "malformatted variables"
```

```
1 str_wrap(manufacturers, 8)
```

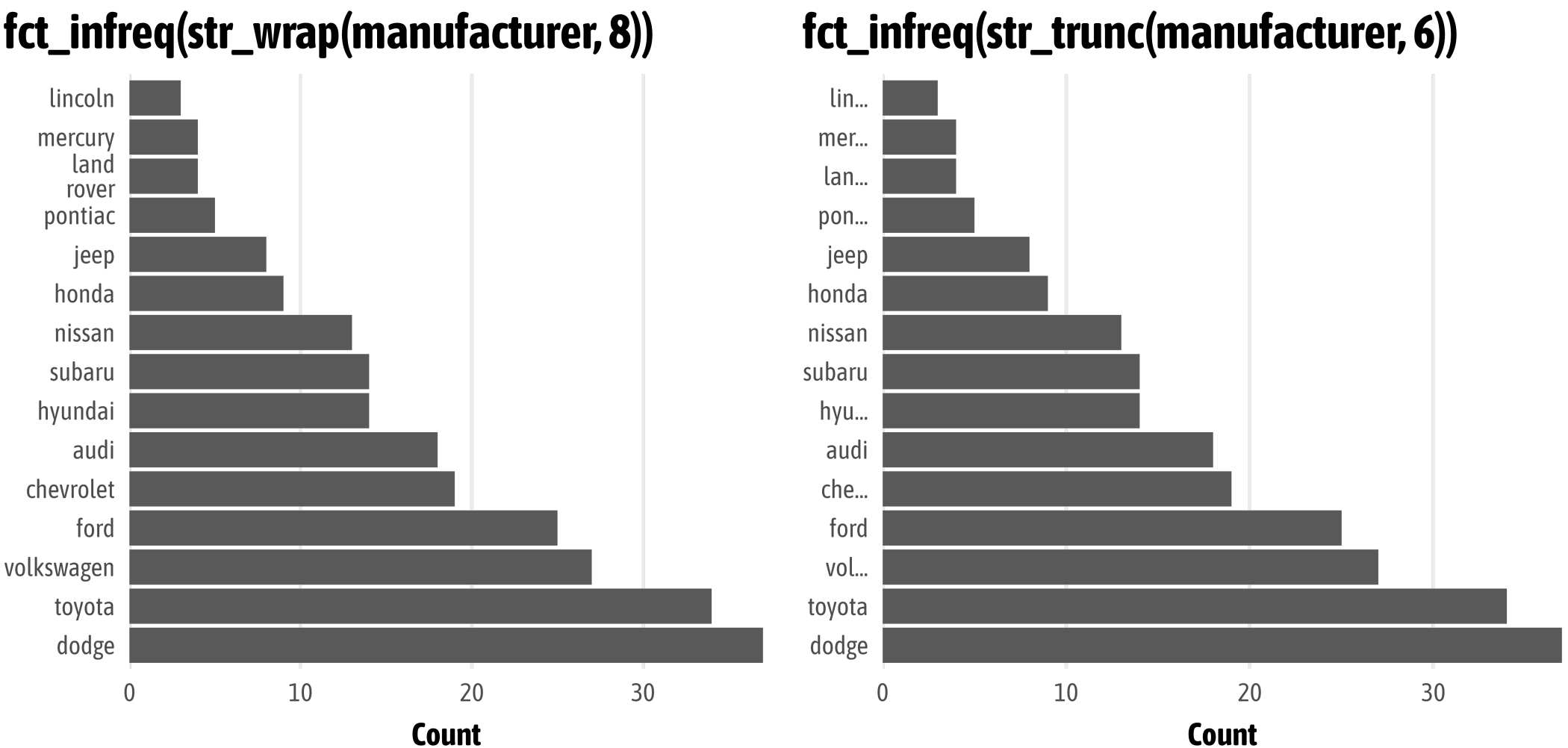
```
 [1] "audi"        "chevrolet"   "dodge"       "ford"        "honda"       "hyundai"     "jeep"
 [8] "land\nrover" "lincoln"     "mercury"     "nissan"      "pontiac"     "subaru"      "toyota"
[15] "volkswagen"
```

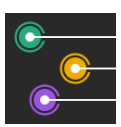
```
1 str_trunc(manufacturers, 6)
```

```
 [1] "audi"   "che..." "dodge"  "ford"   "honda"  "hyu..." "jeep"   "lan..." "lin..." "mer..." "nissan"
[12] "pon..." "subaru" "toyota" "vol..."
```


Working with Strings

Again, these `str_*` functions can be helpful when plotting character variables:





Working with Strings

Functions from the **{stringr}** package also allow to change the letter cases:

```
1 str_to_upper(manufacturers)
```

```
[1] "AUDI"       "CHEVROLET"  "DODGE"      "FORD"       "HONDA"      "HYUNDAI"    "JEEP"       "LAND ROVER"
[9] "LINCOLN"    "MERCURY"    "NISSAN"     "PONTIAC"    "SUBARU"     "TOYOTA"     "VOLKSWAGEN"
```

```
1 str_to_title(manufacturers)
```

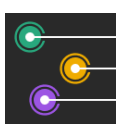
```
[1] "Audi"       "Chevrolet"  "Dodge"      "Ford"       "Honda"      "Hyundai"    "Jeep"       "Land Rover"
[9] "Lincoln"    "Mercury"    "Nissan"     "Pontiac"    "Subaru"     "Toyota"     "Volkswagen"
```

```
1 str_to_lower(c("yes", "Maybe", "NO"))
```

```
[1] "yes"   "maybe" "no"
```


Data Wrangling

— Working with Dates —



The {lubridate} Package

The **{lubridate}** package to parse, extract, and adjust date-time formats and time-spans.

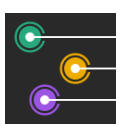
```
1 # install.packages("lubridate")
2 library(lubridate)
```

```
Loading required package: timechange


Attaching package: 'lubridate'

The following objects are masked from 'package:base':

    date, intersect, setdiff, union
```

Working with Dates

Functions from the **{lubridate}** package operate on dates:

```
flights_selected <-
  flights %>%
  mutate(
    date = format(time_hour, format = "%d.%m.%Y"),  ## turn into date as string
    datetime = str_remove_all(time_hour, ":|-")  ## turn into custom formatted datetime string
  ) %>%
  select(flight, date, datetime) %>%
  slice_sample(n = 100) %>%  ## just use random subset
  arrange(flight)
```


Working with Dates

Functions from the **{lubridate}** package operate on dates:

```
  1  flights_selected
```

```
# A tibble: 100 × 3
   flight date       datetime
    <int> <chr>      <chr>
 1      6 06.05.2013 20130506 190000
 2     27 22.04.2013 20130422 080000
 3     51 23.05.2013 20130523 100000
 4     61 02.06.2013 20130602 110000
 5     63 07.02.2013 20130207 100000
 6     63 09.05.2013 20130509 140000
 7     65 10.10.2013 20131010 160000
 8    133 13.08.2013 20130813 150000
 9    199 25.02.2013 20130225 210000
10    221 19.11.2013 20131119 170000
# … with 90 more rows
```


Working with Dates

{lubridate} allows to parse date-time objects of different formats:

```
flights_dates <- flights_selected %>%
  mutate(datetime = ymd_hms(datetime), date = dmy(date))  ## also `ymd*`, `ydm*`, `mdy*`, ...
```

```
flights_dates
```

```
# A tibble: 100 × 3
   flight date       datetime
    <int> <date>     <dttm>
 1      6 2013-05-06 2013-05-06 19:00:00.000
 2     27 2013-04-22 2013-04-22 08:00:00.000
 3     51 2013-05-23 2013-05-23 10:00:00.000
 4     61 2013-06-02 2013-06-02 11:00:00.000
 5     63 2013-02-07 2013-02-07 10:00:00.000
 6     63 2013-05-09 2013-05-09 14:00:00.000
 7     65 2013-10-10 2013-10-10 16:00:00.000
 8    133 2013-08-13 2013-08-13 15:00:00.000
 9    199 2013-02-25 2013-02-25 21:00:00.000
10    221 2013-11-19 2013-11-19 17:00:00.000
# … with 90 more rows
```


Working with Dates

{lubridate} allows to parse date-time objects of different formats:

```
flights_dates <- flights_selected %>%
  mutate(datetime = ymd_hms(datetime), date = dmy(date))  ## also `ymd()`, `ydm()`, `mdy()`, ...
```

```
head(flights_dates$datetime)
```

```
[1] "2013-05-06 19:00:00 UTC" "2013-04-22 08:00:00 UTC" "2013-05-23 10:00:00 UTC" "2013-06-02 11:00:00 UTC"
[5] "2013-02-07 10:00:00 UTC" "2013-05-09 14:00:00 UTC"
```

```
flights_dates <- flights_selected %>%
  mutate(datetime = ymd_hms(datetime, tz = "America/New_York"), date = dmy(date))
```

```
head(flights_dates$datetime)
```

```
[1] "2013-05-06 19:00:00 EDT" "2013-04-22 08:00:00 EDT" "2013-05-23 10:00:00 EDT" "2013-06-02 11:00:00 EDT"
[5] "2013-02-07 10:00:00 EST" "2013-05-09 14:00:00 EDT"
```


Working with Dates

{lubridate} also provides helper functions to work with timezones:

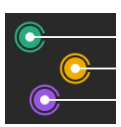
```
flights_dates %>%
  mutate(
    dt_west = with_tz(datetime, "America/Los_Angeles"),
    dt_cet  = with_tz(datetime, "CET"),
  )
```

```
# A tibble: 100 × 5
   flight date       datetime                dt_west              dt_cet
    <int> <date>     <dttm>                  <dttm>               <dttm>
 1      6 2013-05-06 2013-05-06 19:00:00.000 2013-05-06 16:00:00.000 2013-05-07 01:00:00.000
 2     27 2013-04-22 2013-04-22 08:00:00.000 2013-04-22 05:00:00.000 2013-04-22 14:00:00.000
 3     51 2013-05-23 2013-05-23 10:00:00.000 2013-05-23 07:00:00.000 2013-05-23 16:00:00.000
 4     61 2013-06-02 2013-06-02 11:00:00.000 2013-06-02 08:00:00.000 2013-06-02 17:00:00.000
 5     63 2013-02-07 2013-02-07 10:00:00.000 2013-02-07 07:00:00.000 2013-02-07 16:00:00.000
 6     63 2013-05-09 2013-05-09 14:00:00.000 2013-05-09 11:00:00.000 2013-05-09 20:00:00.000
 7     65 2013-10-10 2013-10-10 16:00:00.000 2013-10-10 13:00:00.000 2013-10-10 22:00:00.000
 8    133 2013-08-13 2013-08-13 15:00:00.000 2013-08-13 12:00:00.000 2013-08-13 21:00:00.000
 9    199 2013-02-25 2013-02-25 21:00:00.000 2013-02-25 18:00:00.000 2013-02-26 03:00:00.000
10    221 2013-11-19 2013-11-19 17:00:00.000 2013-11-19 14:00:00.000 2013-11-19 23:00:00.000
# … with 90 more rows
```

```
flights_dates
```

```
# A tibble: 100 × 3
   flight date       datetime
    <int> <date>     <dttm>
 1      6 2013-05-06 2013-05-06 19:00:00.000
```


```
 2     27 2013-04-22 2013-04-22 08:00:00.000
 3     51 2013-05-23 2013-05-23 10:00:00.000
 4     61 2013-06-02 2013-06-02 11:00:00.000
 5     63 2013-02-07 2013-02-07 10:00:00.000
 6     63 2013-05-09 2013-05-09 14:00:00.000
 7     65 2013-10-10 2013-10-10 16:00:00.000
 8    133 2013-08-13 2013-08-13 15:00:00.000
 9    199 2013-02-25 2013-02-25 21:00:00.000
10    221 2013-11-19 2013-11-19 17:00:00.000
# … with 90 more rows
```

Working with Dates

{lubridate} also provides helper functions to work with timezones:

```
1  flights_dates %>%
2    mutate(
3      dt_west = force_tz(datetime, "America/Los_Angeles"),
4      dt_cet  = force_tz(datetime, "CET"),
5    )
```

```
# A tibble: 100 × 5
   flight date       datetime                dt_west                 dt_cet
    <int> <date>     <dttm>                  <dttm>                  <dttm>
 1      6 2013-05-06 2013-05-06 19:00:00.000 2013-05-06 19:00:00.000 2013-05-06 19:00:00.000
 2     27 2013-04-22 2013-04-22 08:00:00.000 2013-04-22 08:00:00.000 2013-04-22 08:00:00.000
 3     51 2013-05-23 2013-05-23 10:00:00.000 2013-05-23 10:00:00.000 2013-05-23 10:00:00.000
 4     61 2013-06-02 2013-06-02 11:00:00.000 2013-06-02 11:00:00.000 2013-06-02 11:00:00.000
 5     63 2013-02-07 2013-02-07 10:00:00.000 2013-02-07 10:00:00.000 2013-02-07 10:00:00.000
 6     63 2013-05-09 2013-05-09 14:00:00.000 2013-05-09 14:00:00.000 2013-05-09 14:00:00.000
 7     65 2013-10-10 2013-10-10 16:00:00.000 2013-10-10 16:00:00.000 2013-10-10 16:00:00.000
 8    133 2013-08-13 2013-08-13 15:00:00.000 2013-08-13 15:00:00.000 2013-08-13 15:00:00.000
 9    199 2013-02-25 2013-02-25 21:00:00.000 2013-02-25 21:00:00.000 2013-02-25 21:00:00.000
10    221 2013-11-19 2013-11-19 17:00:00.000 2013-11-19 17:00:00.000 2013-11-19 17:00:00.000
# … with 90 more rows
```


Working with Dates

... and functions to extract certain components of date-time objects:

```
1  flights_dates %>%
2    mutate(year = year(date), month = month(date))
```

```
# A tibble: 100 × 5
   flight date       datetime                 year month
    <int> <date>     <dttm>                  <dbl> <dbl>
 1      6 2013-05-06 2013-05-06 19:00:00.000  2013     5
 2     27 2013-04-22 2013-04-22 08:00:00.000  2013     4
 3     51 2013-05-23 2013-05-23 10:00:00.000  2013     5
 4     61 2013-06-02 2013-06-02 11:00:00.000  2013     6
 5     63 2013-02-07 2013-02-07 10:00:00.000  2013     2
 6     63 2013-05-09 2013-05-09 14:00:00.000  2013     5
 7     65 2013-10-10 2013-10-10 16:00:00.000  2013    10
 8    133 2013-08-13 2013-08-13 15:00:00.000  2013     8
 9    199 2013-02-25 2013-02-25 21:00:00.000  2013     2
10    221 2013-11-19 2013-11-19 17:00:00.000  2013    11
# … with 90 more rows
```


Working with Dates

... and functions to extract certain components of date-time objects:

```
1  flights_dates %>%
2    mutate(yday = yday(date), week = week(date), wday = wday(date))
```

```
# A tibble: 100 × 6
   flight date       datetime                yday  week  wday
    <int> <date>     <dttm>                 <dbl> <dbl> <dbl>
 1      6 2013-05-06 2013-05-06 19:00:00.000   126    18     2
 2     27 2013-04-22 2013-04-22 08:00:00.000   112    16     2
 3     51 2013-05-23 2013-05-23 10:00:00.000   143    21     5
 4     61 2013-06-02 2013-06-02 11:00:00.000   153    22     1
 5     63 2013-02-07 2013-02-07 10:00:00.000    38     6     5
 6     63 2013-05-09 2013-05-09 14:00:00.000   129    19     5
 7     65 2013-10-10 2013-10-10 16:00:00.000   283    41     5
 8    133 2013-08-13 2013-08-13 15:00:00.000   225    33     3
 9    199 2013-02-25 2013-02-25 21:00:00.000    56     8     2
10    221 2013-11-19 2013-11-19 17:00:00.000   323    47     3
# … with 90 more rows
```


Working with Dates

... and functions to extract certain components of date-time objects:

```
1  flights_dates %>%
2    mutate(month = month(date, label = TRUE), wday = wday(date, label = TRUE))
```

```
# A tibble: 100 × 5
   flight date       datetime                month wday
    <int> <date>     <dttm>                  <ord> <ord>
 1      6 2013-05-06 2013-05-06 19:00:00.000 May   Mon
 2     27 2013-04-22 2013-04-22 08:00:00.000 Apr   Mon
 3     51 2013-05-23 2013-05-23 10:00:00.000 May   Thu
 4     61 2013-06-02 2013-06-02 11:00:00.000 Jun   Sun
 5     63 2013-02-07 2013-02-07 10:00:00.000 Feb   Thu
 6     63 2013-05-09 2013-05-09 14:00:00.000 May   Thu
 7     65 2013-10-10 2013-10-10 16:00:00.000 Oct   Thu
 8    133 2013-08-13 2013-08-13 15:00:00.000 Aug   Tue
 9    199 2013-02-25 2013-02-25 21:00:00.000 Feb   Mon
10    221 2013-11-19 2013-11-19 17:00:00.000 Nov   Tue
# … with 90 more rows
```


Working with Dates

... and functions to extract certain components of date-time objects:

```
1  flights_dates %>%
2    mutate(month = month(date, label = TRUE), wday = wday(date, label = TRUE, abbr = FALSE))
```

```
# A tibble: 100 × 5
   flight date       datetime                month wday
    <int> <date>     <dttm>                  <ord> <ord>
 1      6 2013-05-06 2013-05-06 19:00:00.000 May   Monday
 2     27 2013-04-22 2013-04-22 08:00:00.000 Apr   Monday
 3     51 2013-05-23 2013-05-23 10:00:00.000 May   Thursday
 4     61 2013-06-02 2013-06-02 11:00:00.000 Jun   Sunday
 5     63 2013-02-07 2013-02-07 10:00:00.000 Feb   Thursday
 6     63 2013-05-09 2013-05-09 14:00:00.000 May   Thursday
 7     65 2013-10-10 2013-10-10 16:00:00.000 Oct   Thursday
 8    133 2013-08-13 2013-08-13 15:00:00.000 Aug   Tuesday
 9    199 2013-02-25 2013-02-25 21:00:00.000 Feb   Monday
10    221 2013-11-19 2013-11-19 17:00:00.000 Nov   Tuesday
# … with 90 more rows
```


Working with Dates

To change the language used for the labels, change the `locale`:

```
[1] "de_DE.UTF-8"
```

```
1 Sys.setlocale("LC_TIME", "de_DE.UTF-8") ## on Mac
2 # Sys.setlocale("LC_TIME", "German")    ## on Win
```

```
1 flights_dates %>%
2   mutate(month = month(date, label = TRUE), wday = wday(date, label = TRUE, abbr = FALSE))
```

```
# A tibble: 100 × 5
   flight date       datetime                 month wday
    <int> <date>     <dttm>                   <ord> <ord>
 1      6 2013-05-06 2013-05-06 19:00:00.000 Mai   Montag
 2     27 2013-04-22 2013-04-22 08:00:00.000 Apr   Montag
 3     51 2013-05-23 2013-05-23 10:00:00.000 Mai   Donnerstag
 4     61 2013-06-02 2013-06-02 11:00:00.000 Jun   Sonntag
 5     63 2013-02-07 2013-02-07 10:00:00.000 Feb   Donnerstag
 6     63 2013-05-09 2013-05-09 14:00:00.000 Mai   Donnerstag
 7     65 2013-10-10 2013-10-10 16:00:00.000 Okt   Donnerstag
 8    133 2013-08-13 2013-08-13 15:00:00.000 Aug   Dienstag
 9    199 2013-02-25 2013-02-25 21:00:00.000 Feb   Montag
10    221 2013-11-19 2013-11-19 17:00:00.000 Nov   Dienstag
# … with 90 more rows
```


Working with Dates

You can also do arithmetic operations on date-time objects:

```
1  flights_dates %>%
2    mutate(date_ahead = date + 30)
```

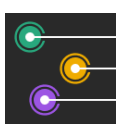
```
# A tibble: 100 × 4
   flight date       datetime                date_ahead
    <int> <date>     <dttm>                  <date>
 1      6 2013-05-06 2013-05-06 19:00:00.000 2013-06-05
 2     27 2013-04-22 2013-04-22 08:00:00.000 2013-05-22
 3     51 2013-05-23 2013-05-23 10:00:00.000 2013-06-22
 4     61 2013-06-02 2013-06-02 11:00:00.000 2013-07-02
 5     63 2013-02-07 2013-02-07 10:00:00.000 2013-03-09
 6     63 2013-05-09 2013-05-09 14:00:00.000 2013-06-08
 7     65 2013-10-10 2013-10-10 16:00:00.000 2013-11-09
 8    133 2013-08-13 2013-08-13 15:00:00.000 2013-09-12
 9    199 2013-02-25 2013-02-25 21:00:00.000 2013-03-27
10    221 2013-11-19 2013-11-19 17:00:00.000 2013-12-19
# … with 90 more rows
```


Working with Dates

You can also do arithmetic operations on date-time objects:

```
1  flights_dates %>%
2    mutate(date_ahead = date + weeks(6))
```

```
# A tibble: 100 × 4
   flight date       datetime                date_ahead
    <int> <date>     <dttm>                  <date>
 1      6 2013-05-06 2013-05-06 19:00:00.000 2013-06-17
 2     27 2013-04-22 2013-04-22 08:00:00.000 2013-06-03
 3     51 2013-05-23 2013-05-23 10:00:00.000 2013-07-04
 4     61 2013-06-02 2013-06-02 11:00:00.000 2013-07-14
 5     63 2013-02-07 2013-02-07 10:00:00.000 2013-03-21
 6     63 2013-05-09 2013-05-09 14:00:00.000 2013-06-20
 7     65 2013-10-10 2013-10-10 16:00:00.000 2013-11-21
 8    133 2013-08-13 2013-08-13 15:00:00.000 2013-09-24
 9    199 2013-02-25 2013-02-25 21:00:00.000 2013-04-08
10    221 2013-11-19 2013-11-19 17:00:00.000 2013-12-31
# … with 90 more rows
```

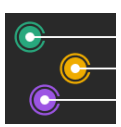
Working with Times

Functions from the **{hms}** package operate on time stamps:

```
1  # install.packages("hms")
```

```
1  data.frame(hours = 1:3, hms = hms::hms(hours = 1:3))  ## do not confuse with `lubridate::hms()`
```

```
  hours      hms
1     1 01:00:00
2     2 02:00:00
3     3 03:00:00
```

Working with Times

Functions from the **{hms}** package operate on time stamps:

```
1  flights_time <- flights_dates %>%
2    mutate(time = hms::as_hms(datetime))
```

```
1  flights_time
```

```
# A tibble: 100 × 4
   flight date       datetime                time
    <int> <date>     <dttm>                  <time>
 1      6 2013-05-06 2013-05-06 19:00:00.000 19:00
 2     27 2013-04-22 2013-04-22 08:00:00.000 08:00
 3     51 2013-05-23 2013-05-23 10:00:00.000 10:00
 4     61 2013-06-02 2013-06-02 11:00:00.000 11:00
 5     63 2013-02-07 2013-02-07 10:00:00.000 10:00
 6     63 2013-05-09 2013-05-09 14:00:00.000 14:00
 7     65 2013-10-10 2013-10-10 16:00:00.000 16:00
 8    133 2013-08-13 2013-08-13 15:00:00.000 15:00
 9    199 2013-02-25 2013-02-25 21:00:00.000 21:00
10    221 2013-11-19 2013-11-19 17:00:00.000 17:00
# … with 90 more rows
```


Working with Times

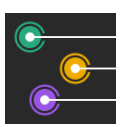
Functions from the **{hms}** package operate on time stamps:

```
1  flights_time %>%
2    mutate(time_num = as.numeric(time))
```

```
# A tibble: 100 × 5
   flight date       datetime                time    time_num
    <int> <date>     <dttm>                  <time>     <dbl>
 1      6 2013-05-06 2013-05-06 19:00:00.000 19:00      68400
 2     27 2013-04-22 2013-04-22 08:00:00.000 08:00      28800
 3     51 2013-05-23 2013-05-23 10:00:00.000 10:00      36000
 4     61 2013-06-02 2013-06-02 11:00:00.000 11:00      39600
 5     63 2013-02-07 2013-02-07 10:00:00.000 10:00      36000
 6     63 2013-05-09 2013-05-09 14:00:00.000 14:00      50400
 7     65 2013-10-10 2013-10-10 16:00:00.000 16:00      57600
 8    133 2013-08-13 2013-08-13 15:00:00.000 15:00      54000
 9    199 2013-02-25 2013-02-25 21:00:00.000 21:00      75600
10    221 2013-11-19 2013-11-19 17:00:00.000 17:00      61200
# … with 90 more rows
```


Data Wrangling

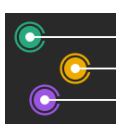
— String Interpolation —



The {glue} Package

The **{glue}** package offers simple and fast evaluation of string literals.

```
1 # install.packages("glue")
2 library(glue)
```

Combining Strings (and Values)

`glue()` allows you to combine strings and values:

```
mpg %>%
  select(manufacturer, model, year, trans) %>%
  mutate(
    text = glue("The model {manufacturer} {model} with {trans} as transmission was build in {year}.
  )
```

```
# A tibble: 234 × 5
   manufacturer model       year trans      text
   <chr>        <chr>      <int> <chr>      <glue>
 1 audi         a4          1999 auto(l5)   The model audi a4 with auto(l5) as transmission was build in 1999.
 2 audi         a4          1999 manual(m5) The model audi a4 with manual(m5) as transmission was build in 19…
 3 audi         a4          2008 manual(m6) The model audi a4 with manual(m6) as transmission was build in 20…
 4 audi         a4          2008 auto(av)   The model audi a4 with auto(av) as transmission was build in 2008.
 5 audi         a4          1999 auto(l5)   The model audi a4 with auto(l5) as transmission was build in 1999.
 6 audi         a4          1999 manual(m5) The model audi a4 with manual(m5) as transmission was build in 19…
 7 audi         a4          2008 auto(av)   The model audi a4 with auto(av) as transmission was build in 2008.
 8 audi         a4 quattro  1999 manual(m5) The model audi a4 quattro with manual(m5) as transmission was bui…
 9 audi         a4 quattro  1999 auto(l5)   The model audi a4 quattro with auto(l5) as transmission was build…
10 audi         a4 quattro  2008 manual(m6) The model audi a4 quattro with manual(m6) as transmission was bui…
# … with 224 more rows
```


Combining Strings (and Values)

`glue()` in combination with functions from **{stringr}**:

```
mpg %>%
  select(manufacturer, model, year, trans) %>%
  mutate(text =
    glue("The {str_to_title(manufacturer)} {str_to_title(model)} with {trans} as transmission was b
  )
```

```
# A tibble: 234 × 5
   manufacturer model      year trans      text
   <chr>        <chr>     <int> <chr>      <glue>
 1 audi         a4         1999 auto(l5)   The Audi A4 with auto(l5) as transmission was build in 1999.
 2 audi         a4         1999 manual(m5) The Audi A4 with manual(m5) as transmission was build in 1999.
 3 audi         a4         2008 manual(m6) The Audi A4 with manual(m6) as transmission was build in 2008.
 4 audi         a4         2008 auto(av)   The Audi A4 with auto(av) as transmission was build in 2008.
 5 audi         a4         1999 auto(l5)   The Audi A4 with auto(l5) as transmission was build in 1999.
 6 audi         a4         1999 manual(m5) The Audi A4 with manual(m5) as transmission was build in 1999.
 7 audi         a4         2008 auto(av)   The Audi A4 with auto(av) as transmission was build in 2008.
 8 audi         a4 quattro 1999 manual(m5) The Audi A4 Quattro with manual(m5) as transmission was build in …
 9 audi         a4 quattro 1999 auto(l5)   The Audi A4 Quattro with auto(l5) as transmission was build in 19…
10 audi         a4 quattro 2008 manual(m6) The Audi A4 Quattro with manual(m6) as transmission was build in …
# … with 224 more rows
```


Alternatively: paste Functions

`paste` and() `paste0()` also allow you to combine strings and values:

```
1 date <- Sys.Date()
2 wday <- wday(date, label = TRUE, abbr = FALSE)
```

```
1 paste("Today is", wday, ",", date)
```

```
[1] "Today is Tuesday , 2023-02-28"
```

```
1 paste("Today is", wday, ",", date, sep = "*")
```

```
[1] "Today is*Tuesday*,*2023-02-28"
```

```
1 paste("Today is ", wday, ", ", date, sep = "")
```

```
[1] "Today is Tuesday, 2023-02-28"
```

```
1 paste0("Today is ", wday, ", ", date)
```

```
[1] "Today is Tuesday, 2023-02-28"
```


Alternatively: paste Functions

`paste` and() `paste0()` also allow you to combine strings and values:

```
mpg %>%
  select(manufacturer, model, year, trans) %>%
  mutate(text =
    paste0("The ", str_to_title(manufacturer), " ", str_to_title(model), " with ", trans, " as tran
  )
```

```
# A tibble: 234 × 5
   manufacturer model      year trans      text
   <chr>        <chr>     <int> <chr>      <chr>
 1 audi         a4         1999 auto(l5)   The Audi A4 with auto(l5) as transmission was build in 1999.
 2 audi         a4         1999 manual(m5) The Audi A4 with manual(m5) as transmission was build in 1999.
 3 audi         a4         2008 manual(m6) The Audi A4 with manual(m6) as transmission was build in 2008.
 4 audi         a4         2008 auto(av)   The Audi A4 with auto(av) as transmission was build in 2008.
 5 audi         a4         1999 auto(l5)   The Audi A4 with auto(l5) as transmission was build in 1999.
 6 audi         a4         1999 manual(m5) The Audi A4 with manual(m5) as transmission was build in 1999.
 7 audi         a4         2008 auto(av)   The Audi A4 with auto(av) as transmission was build in 2008.
 8 audi         a4 quattro 1999 manual(m5) The Audi A4 Quattro with manual(m5) as transmission was build in …
 9 audi         a4 quattro 1999 auto(l5)   The Audi A4 Quattro with auto(l5) as transmission was build in 19…
10 audi         a4 quattro 2008 manual(m6) The Audi A4 Quattro with manual(m6) as transmission was build in …
# … with 224 more rows
```


Your Turn: More Data Wrangling

- Explore the `billboard` data set from the **`{tidyr}`** package.
- Turn it into a tidy format.
- Drop all rows that do not contain information on the rank.
- Shorten the track title to 20 characters, preferably indicating that the title has been shortened.
- Add a column with the date in a German format: `dd.mm.yyyy`.
- Create a new summary table with 1 line per track containing
  - the number of weeks a track was ranked in the Top 100
  - the average rank of each track across weeks
  - **Bonus:** the artist column as a factor ordered by the overall number of weeks and number of tracks and a column with the explicit rank

Your Turn: Inspect Data

```
1 df_bill <- tidyr::billboard
```

```
1 glimpse(df_bill)
```

```
Rows: 317
Columns: 79
$ artist       <chr> "2 Pac", "2Ge+her", "3 Doors Down", "3 Doors Down", "504 Boyz", "98^0", "A*Teens", "Aal…
$ track        <chr> "Baby Don't Cry (Keep...", "The Hardest Part Of ...", "Kryptonite", "Loser", "Wobble Wo…
$ date.entered <date> 2000-02-26, 2000-09-02, 2000-04-08, 2000-10-21, 2000-04-15, 2000-08-19, 2000-07-08, 20…
$ wk1          <dbl> 87, 91, 81, 76, 57, 51, 97, 84, 59, 76, 84, 57, 50, 71, 79, 80, 99, 92, 82, 77, 70, 62,…
$ wk2          <dbl> 82, 87, 70, 76, 34, 39, 97, 62, 53, 76, 84, 47, 39, 51, 65, 78, 99, NA, 76, 54, 62, 32,…
$ wk3          <dbl> 72, 92, 68, 72, 25, 34, 96, 51, 38, 74, 75, 45, 30, 28, 53, 76, 96, NA, 76, 50, 56, 30,…
$ wk4          <dbl> 77, NA, 67, 69, 17, 26, 95, 41, 28, 69, 73, 29, 28, 18, 48, 77, 96, 95, 70, 43, 43, 23,…
$ wk5          <dbl> 87, NA, 66, 67, 17, 26, 100, 38, 21, 68, 73, 23, 21, 13, 45, 92, 100, NA, 82, 30, 39, 2…
$ wk6          <dbl> 94, NA, 57, 65, 31, 19, NA, 35, 18, 67, 69, 18, 19, 13, 36, NA, 93, NA, 81, 27, 33, 30,…
$ wk7          <dbl> 99, NA, 54, 55, 36, 2, NA, 35, 16, 61, 68, 11, 20, 11, 34, NA, 93, NA, 74, 21, 26, 35, …
$ wk8          <dbl> NA, NA, 53, 59, 49, 2, NA, 38, 14, 58, 65, 9, 17, 1, 29, NA, 96, NA, 80, 18, 26, 32, 21…
$ wk9          <dbl> NA, NA, 51, 62, 53, 3, NA, 38, 12, 57, 73, 9, 17, 1, 27, NA, NA, NA, 76, 15, 26, 32, 18…
$ wk10         <dbl> NA, NA, 51, 61, 57, 6, NA, 36, 10, 59, 83, 11, 17, 2, 30, NA, NA, NA, 76, 13, 31, 25, 1…
$ wk11         <dbl> NA, NA, 51, 61, 64, 7, NA, 37, 9, 66, 92, 1, 17, 2, 36, NA, 99, NA, 73, 13, 32, 23, 19,…
$ wk12         <dbl> NA, NA, 51, 59, 70, 22, NA, 37, 8, 68, NA, 1, 3, 3, 37, NA, NA, 97, 74, 13, 31, 28, 15,…
$ wk13         <dbl> NA, NA, 47, 61, 75, 29, NA, 38, 6, 61, NA, 1, 3, 3, 39, NA, 96, NA, 87, 13, 38, 27, 18,…
$ wk14         <dbl> NA, NA, 44, 66, 76, 36, NA, 49, 1, 67, NA, 1, 7, 4, 49, NA, 96, NA, 83, 5, 38, 29, 13, …
```

Of which format is the `billboard` data set?

Your Turn: Tidy Data

```
df_bill %>%
  tidyr::pivot_longer(
    cols = -c(artist, track, date.entered),
    names_to = "week",
    values_to = "rank"
  )
```

```
# A tibble: 24,092 × 5
   artist track                   date.entered week   rank
   <chr>  <chr>                   <date>       <chr> <dbl>
 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk1      87
 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk2      82
 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk3      72
 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk4      77
 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk5      87
 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk6      94
 7 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk7      99
 8 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk8      NA
 9 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk9      NA
10 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk10     NA
# … with 24,082 more rows
```


Your Turn: Tidy Data

```
df_bill %>%
  tidyr::pivot_longer(
    cols = -c(artist, track, date.entered),
    names_to = "week",
    values_to = "rank",
    names_prefix = "wk"
  )
```

```
# A tibble: 24,092 × 5
   artist track                   date.entered week   rank
   <chr>  <chr>                   <date>       <chr> <dbl>
 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   1        87
 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   2        82
 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   3        72
 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   4        77
 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   5        87
 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   6        94
 7 2 Pac  Baby Don't Cry (Keep... 2000-02-26   7        99
 8 2 Pac  Baby Don't Cry (Keep... 2000-02-26   8        NA
 9 2 Pac  Baby Don't Cry (Keep... 2000-02-26   9        NA
10 2 Pac  Baby Don't Cry (Keep... 2000-02-26   10       NA
# … with 24,082 more rows
```


Your Turn: Tidy Data

```
df_bill %>%
  tidyr::pivot_longer(
    cols = -c(artist, track, date.entered),
    names_to = "week",
    values_to = "rank",
    names_prefix = "wk",
    values_drop_na = TRUE
  )
```

```
# A tibble: 5,307 × 5
   artist  track                   date.entered week   rank
   <chr>   <chr>                   <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   1        87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   2        82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   3        72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   4        77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   5        87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   6        94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   7        99
 8 2Ge+her The Hardest Part Of ... 2000-09-02   1        91
 9 2Ge+her The Hardest Part Of ... 2000-09-02   2        87
10 2Ge+her The Hardest Part Of ... 2000-09-02   3        92
# … with 5,297 more rows
```


Your Turn: Tidy Data

```
df_bill %>%
  tidyr::pivot_longer(
    cols = -c(artist, track, date.entered),
    names_to = "week",
    values_to = "rank",
    names_prefix = "wk"
  ) %>%
  dplyr::filter(!is.na(rank))
```

```
# A tibble: 5,307 × 5
   artist  track                   date.entered week   rank
   <chr>   <chr>                   <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   1        87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   2        82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   3        72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   4        77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   5        87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   6        94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   7        99
 8 2Ge+her The Hardest Part Of ... 2000-09-02   1        91
 9 2Ge+her The Hardest Part Of ... 2000-09-02   2        87
10 2Ge+her The Hardest Part Of ... 2000-09-02   3        92
# … with 5,297 more rows
```


Your Turn: Tidy Data

```
df_bill_tidy <-
   df_bill %>%
  tidyr::pivot_longer(
    cols = -c(artist, track, date.entered),
    names_to = "week",
    values_to = "rank",
    names_prefix = "wk",
    values_drop_na = TRUE
  )
```


Your Turn: Shorten Track Names

```
df_bill_tidy %>%
  dplyr::mutate(
    track = stringr::str_trunc(track, 20)
  )
```

```
# A tibble: 5,307 × 5
   artist  track                date.entered week   rank
   <chr>   <chr>                <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (K... 2000-02-26   1        87
 2 2 Pac   Baby Don't Cry (K... 2000-02-26   2        82
 3 2 Pac   Baby Don't Cry (K... 2000-02-26   3        72
 4 2 Pac   Baby Don't Cry (K... 2000-02-26   4        77
 5 2 Pac   Baby Don't Cry (K... 2000-02-26   5        87
 6 2 Pac   Baby Don't Cry (K... 2000-02-26   6        94
 7 2 Pac   Baby Don't Cry (K... 2000-02-26   7        99
 8 2Ge+her The Hardest Part ... 2000-09-02   1        91
 9 2Ge+her The Hardest Part ... 2000-09-02   2        87
10 2Ge+her The Hardest Part ... 2000-09-02   3        92
# … with 5,297 more rows
```


Your Turn: Shorten Track Names

```
1  stringr::str_trunc(df_bill$track, 20)
```

```
 [1] "Baby Don't Cry (K..." "The Hardest Part ..." "Kryptonite"           "Loser"
 [5] "Wobble Wobble"        "Give Me Just One ..." "Dancing Queen"        "I Don't Wanna"
 [9] "Try Again"            "Open My Heart"        "More"                 "Come On Over Baby..."
[13] "I Turn To You"        "What A Girl Wants"    "Better Off Alone"     "Smoke Rings In Th..."
[17] "Sexual"               "I'm Outta Love"       "My Baby You"          "You Sang To Me"
[21] "My First Love"        "Separated"            "Back Here"            "Shape Of My Heart"
[25] "Show Me The Meani..." "The One"              "Bag Lady"             "Who Let The Dogs Out"
[29] "Pinch Me"             "Girls Dem Sugar"      "Monica"               "Tricky Tricky"
[33] "It's So Hard"         "Whoa!"                "Been There"           "Bring It All To Me"
[37] "Deep Inside"          "Give Me You"          "All The Small Things" "The Bad Touch"
[41] "It's My Life"         "He Wasn't Man Enough" "Just Be A Man Abo..." "Spanish Guitar"
[45] "A Country Boy Can..." "Yes!"                "You'll Always Be ..." "Do What You Gotta Do"
[49] "Put Your Hand In ..." "My Love Goes On A..." "What Means The Wo..." "Crybaby"
[53] "Thank God I Found..." "Aaron's Party (Co..." "Take That"           "That Other Woman"
[57] "I Lost It"            "What I Need To Do"    "Meanwhile Back At..." "A Little Gasoline"
[61] "The Light"            "Hanginaround"         "Higher"               "With Arms Wide Open"
[65] "You Won't Be Lone..." "Left & Right"         "Untitled (How Doe..." "Party Up (Up In H..."
[69] "What You Want"        "What's My Name"       "That's What I'm L..." "What'Chu Like"
[73] "Unconditional"        "All Good?"            "Independent Women..." "Jumpin' Jumpin'"
```


Your Turn: Change Date Format

```
df_bill_tidy %>%
  dplyr::mutate(
    track = stringr::str_trunc(track, 20),
    date.entered.german = format(date.entered, format = "%d.%m.%Y")
  )
```

```
# A tibble: 5,307 × 6
   artist  track                date.entered week   rank date.entered.german
   <chr>   <chr>                <date>       <chr> <dbl> <chr>
 1 2 Pac   Baby Don't Cry (K... 2000-02-26   1        87 26.02.2000
 2 2 Pac   Baby Don't Cry (K... 2000-02-26   2        82 26.02.2000
 3 2 Pac   Baby Don't Cry (K... 2000-02-26   3        72 26.02.2000
 4 2 Pac   Baby Don't Cry (K... 2000-02-26   4        77 26.02.2000
 5 2 Pac   Baby Don't Cry (K... 2000-02-26   5        87 26.02.2000
 6 2 Pac   Baby Don't Cry (K... 2000-02-26   6        94 26.02.2000
 7 2 Pac   Baby Don't Cry (K... 2000-02-26   7        99 26.02.2000
 8 2Ge+her The Hardest Part ... 2000-09-02   1        91 02.09.2000
 9 2Ge+her The Hardest Part ... 2000-09-02   2        87 02.09.2000
10 2Ge+her The Hardest Part ... 2000-09-02   3        92 02.09.2000
# … with 5,297 more rows
```


Your Turn: Change Date Format

```
(df_bill_custom <-
  df_bill_tidy %>%
  dplyr::mutate(
    track = stringr::str_trunc(track, 20),
    date.entered.german = format(date.entered, format = "%d.%m.%Y")
  ) %>%
  dplyr::select(artist, track, week, rank, starts_with("date.")))
```

```
# A tibble: 5,307 × 6
   artist  track                week   rank date.entered date.entered.german
   <chr>   <chr>                <chr> <dbl> <date>       <chr>
 1 2 Pac   Baby Don't Cry (K... 1        87 2000-02-26   26.02.2000
 2 2 Pac   Baby Don't Cry (K... 2        82 2000-02-26   26.02.2000
 3 2 Pac   Baby Don't Cry (K... 3        72 2000-02-26   26.02.2000
 4 2 Pac   Baby Don't Cry (K... 4        77 2000-02-26   26.02.2000
 5 2 Pac   Baby Don't Cry (K... 5        87 2000-02-26   26.02.2000
 6 2 Pac   Baby Don't Cry (K... 6        94 2000-02-26   26.02.2000
 7 2 Pac   Baby Don't Cry (K... 7        99 2000-02-26   26.02.2000
 8 2Ge+her The Hardest Part ... 1        91 2000-09-02   02.09.2000
 9 2Ge+her The Hardest Part ... 2        87 2000-09-02   02.09.2000
10 2Ge+her The Hardest Part ... 3        92 2000-09-02   02.09.2000
# … with 5,297 more rows
```


Your Turn: Summary Table

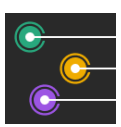
```
1 df_bill_custom %>%
2   dplyr::count(artist, track)
```

```
# A tibble: 317 × 3
   artist         track                    n
   <chr>          <chr>                <int>
 1 2 Pac          Baby Don't Cry (K...     7
 2 2Ge+her        The Hardest Part ...     3
 3 3 Doors Down   Kryptonite              53
 4 3 Doors Down   Loser                   20
 5 504 Boyz       Wobble Wobble           18
 6 98^0           Give Me Just One ...    20
 7 A*Teens        Dancing Queen            5
 8 Aaliyah        I Don't Wanna           20
 9 Aaliyah        Try Again               32
10 Adams, Yolanda Open My Heart           20
# … with 307 more rows
```


Your Turn: Check for Duplicates

```
1  df_bill_custom %>%
2    dplyr::count(artist, track, week) %>%
3    dplyr::filter(n > 1)
```

```
# A tibble: 0 × 4
# … with 4 variables: artist <chr>, track <chr>, week <chr>, n <int>
```

Your Turn: Summary Table

```
df_bill_custom %>%
  ## same as `count()`...
  dplyr::group_by(artist, track) %>%
  dplyr::summarize(
    weeks = dplyr::n()
    ## ... but we need to add more here
  )
```

```
# A tibble: 317 × 3
# Groups:   artist [228]
   artist         track                weeks
   <chr>          <chr>                <int>
 1 2 Pac          Baby Don't Cry (K...     7
 2 2Ge+her        The Hardest Part ...     3
 3 3 Doors Down   Kryptonite              53
 4 3 Doors Down   Loser                   20
 5 504 Boyz       Wobble Wobble           18
 6 98^0           Give Me Just One ...    20
 7 A*Teens        Dancing Queen            5
 8 Aaliyah        I Don't Wanna           20
 9 Aaliyah        Try Again               32
10 Adams, Yolanda Open My Heart           20
# … with 307 more rows
```

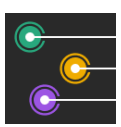

Your Turn: Summary Table

```
df_bill_custom %>%
  dplyr::group_by(artist, track) %>%
  dplyr::summarize(
    weeks = dplyr::n(),
    avg_rank = mean(rank)
  )
```

```
# A tibble: 317 × 4
# Groups:   artist [228]
   artist         track                weeks avg_rank
   <chr>          <chr>                <int>    <dbl>
 1 2 Pac          Baby Don't Cry (K...     7     85.4
 2 2Ge+her        The Hardest Part ...     3     90
 3 3 Doors Down   Kryptonite              53     26.5
 4 3 Doors Down   Loser                   20     67.1
 5 504 Boyz       Wobble Wobble           18     56.2
 6 98^0           Give Me Just One ...    20     37.6
 7 A*Teens        Dancing Queen            5     97
 8 Aaliyah        I Don't Wanna           20     52.0
 9 Aaliyah        Try Again               32     16.7
10 Adams, Yolanda Open My Heart           20     67.8
# … with 307 more rows
```


Your Turn: Summary Table

```
df_bill_sum <-
  df_bill_custom %>%
  dplyr::group_by(artist, track) %>%
  dplyr::summarize(
    weeks = dplyr::n(),
    avg_rank = mean(rank)
  )
```

Your Turn: Summary Table

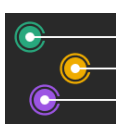
```
1  df_bill_sum %>%
2    add_count(artist)
```

```
# A tibble: 317 × 5
# Groups:   artist [228]
   artist         track                weeks avg_rank     n
   <chr>          <chr>                <int>    <dbl> <int>
 1 2 Pac          Baby Don't Cry (K...     7     85.4     1
 2 2Ge+her        The Hardest Part ...     3     90       1
 3 3 Doors Down   Kryptonite              53     26.5     2
 4 3 Doors Down   Loser                   20     67.1     2
 5 504 Boyz       Wobble Wobble           18     56.2     1
 6 98^0           Give Me Just One ...    20     37.6     1
 7 A*Teens        Dancing Queen            5     97       1
 8 Aaliyah        I Don't Wanna           20     52.0     2
 9 Aaliyah        Try Again               32     16.7     2
10 Adams, Yolanda Open My Heart           20     67.8     1
# … with 307 more rows
```


Your Turn: Summary Table

```
df_bill_sum %>%
  ## still grouped--otherwise group by artist first!
  mutate(
    weeks_artist = sum(weeks),
    tracks_artist = dplyr::n()
  )
```

```
# A tibble: 317 × 6
# Groups:   artist [228]
   artist         track                weeks avg_rank weeks_artist tracks_artist
   <chr>          <chr>                <int>    <dbl>        <int>         <int>
 1 2 Pac          Baby Don't Cry (K...     7     85.4            7             1
 2 2Ge+her        The Hardest Part ...     3     90              3             1
 3 3 Doors Down   Kryptonite              53     26.5           73             2
 4 3 Doors Down   Loser                   20     67.1           73             2
 5 504 Boyz       Wobble Wobble           18     56.2           18             1
 6 98^0           Give Me Just One ...    20     37.6           20             1
 7 A*Teens        Dancing Queen            5     97              5             1
 8 Aaliyah        I Don't Wanna           20     52.0           52             2
 9 Aaliyah        Try Again               32     16.7           52             2
10 Adams, Yolanda Open My Heart           20     67.8           20             1
# … with 307 more rows
```

Your Turn: Summary Table

```
df_bill_sum %>%
  dplyr::mutate(
    weeks_artist = sum(weeks),
    tracks_artist = dplyr::n()
  ) %>%
  dplyr::arrange(-weeks_artist, -tracks_artist)
```

```
# A tibble: 317 × 6
# Groups:   artist [228]
   artist          track                weeks avg_rank weeks_artist tracks_artist
   <chr>           <chr>                <int>    <dbl>        <int>         <int>
 1 Creed           Higher                  57     36.9          104             2
 2 Creed           With Arms Wide Open     47     33.8          104             2
 3 Lonestar        Amazed                  55     26.7           95             3
 4 Lonestar        Smile                   20     60.3           95             3
 5 Lonestar        What About Now          20     49.9           95             3
 6 Destiny's Child Independent Women...    28     14.8           92             3
 7 Destiny's Child Jumpin' Jumpin'         32     22.9           92             3
 8 Destiny's Child Say My Name             32     20.9           92             3
 9 N'Sync          Bye Bye Bye             23     14.3           74             3
10 N'Sync          It's Gonna Be Me        25     21.7           74             3
# … with 307 more rows
```


Your Turn: Summary Table

```
df_bill_sum_fct <-
  df_bill_sum %>%
  dplyr::mutate(
    weeks_artist = sum(weeks),
    tracks_artist = dplyr::n()
  ) %>%
  dplyr::mutate(artist = forcats::fct_reorder2(
    artist, weeks_artist, tracks_artist
  ))
```

```
head(levels(df_bill_sum_fct$artist), 10)
```

```
[1] "2 Pac"              "2Ge+her"            "3 Doors Down"       "504 Boyz"
[5] "98^0"               "A*Teens"            "Aaliyah"            "Adams, Yolanda"
[9] "Adkins, Trace"      "Aguilera, Christina"
```

Factor levels not correct–what's wrong?

Your Turn: Summary Table

```
df_bill_sum_fct <-
  df_bill_sum %>%
  dplyr::mutate(
    weeks_artist = sum(weeks),
    tracks_artist = dplyr::n()
  ) %>%
  dplyr::ungroup() %>%
  dplyr::mutate(artist = forcats::fct_reorder2(
    artist, weeks_artist, tracks_artist
  ))
```

```
head(levels(df_bill_sum_fct$artist), 10)
```

```
[1] "Jay-Z"                "Dixie Chicks, The"    "Houston, Whitney"     "Aguilera, Christina"
[5] "Backstreet Boys, The" "Braxton, Toni"        "Destiny's Child"      "DMX"
[9] "Eminem"               "Jackson, Alan"
```

Factor levels not correct–what's wrong?

Your Turn: Summary Table

```
df_bill_sum_fct <-
  df_bill_sum %>%
  mutate(
    tracks_artist = dplyr::n(),
    weeks_artist = sum(weeks)
  ) %>%
  dplyr::ungroup() %>%
  dplyr::mutate(artist = forcats::fct_reorder2(
    artist, weeks_artist, tracks_artist, .fun = unique
  ))
```

```
head(levels(df_bill_sum_fct$artist), 10)
```

```
[1] "Creed"              "Lonestar"           "Destiny's Child"    "N'Sync"
[5] "Sisqo"              "3 Doors Down"       "Jay-Z"              "Aguilera, Christina"
[9] "Hill, Faith"        "Houston, Whitney"
```

```
tail(levels(df_bill_sum_fct$artist), 10)
```

```
[1] "De La Soul"       "Larrieux, Amel"   "Spencer, Tracie"  "LL Cool J"        "Tuesday"
[6] "Zombie Nation"    "Estefan, Gloria"  "Fragma"           "Ghostface Killah" "Master P"
```

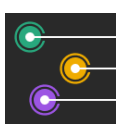

Your Turn: Summary Table

```
df_bill_sum_fct %>%
  dplyr::mutate(
    rank_artist = as.numeric(artist)
  )
```

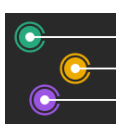
```
# A tibble: 317 × 7
   artist         track                weeks avg_rank tracks_artist weeks_artist rank_artist
   <fct>          <chr>                <int>    <dbl>         <int>        <int>       <dbl>
 1 2 Pac          Baby Don't Cry (K...     7     85.4             1            7         189
 2 2Ge+her        The Hardest Part ...     3     90               1            3         216
 3 3 Doors Down   Kryptonite              53     26.5             2           73           6
 4 3 Doors Down   Loser                   20     67.1             2           73           6
 5 504 Boyz       Wobble Wobble           18     56.2             1           18         135
 6 98^0           Give Me Just One ...    20     37.6             1           20          89
 7 A*Teens        Dancing Queen            5     97               1            5         205
 8 Aaliyah        I Don't Wanna           20     52.0             2           52          20
 9 Aaliyah        Try Again               32     16.7             2           52          20
10 Adams, Yolanda Open My Heart           20     67.8             1           20          90
# … with 307 more rows
```

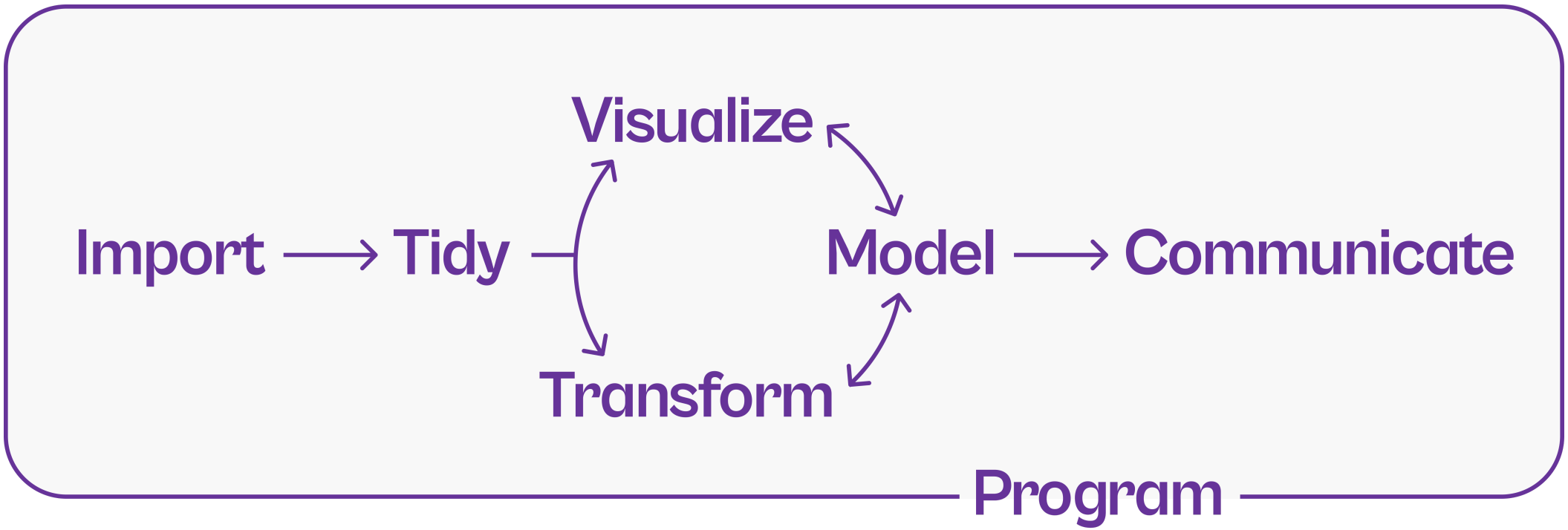

Resources

- "R for Data Science" by Hadley Wickham and Garrett Grolemund
- tidvyerse Webpage
- {dplyr} cheatsheet
- {tidyr} cheatsheet
- {forcats} cheatsheet
- {stringr} cheatsheet
- {lubridate} cheatsheet
- {readr} cheatsheet (incl. {readxl} and {googlesheets4r})

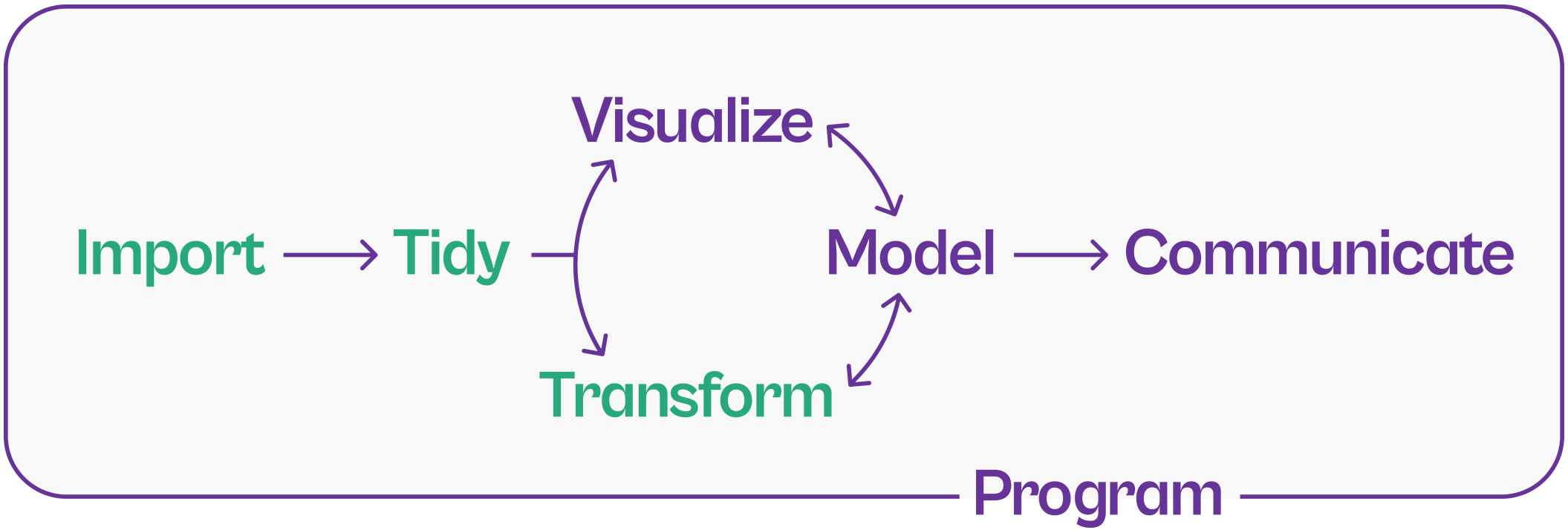


What's Next?

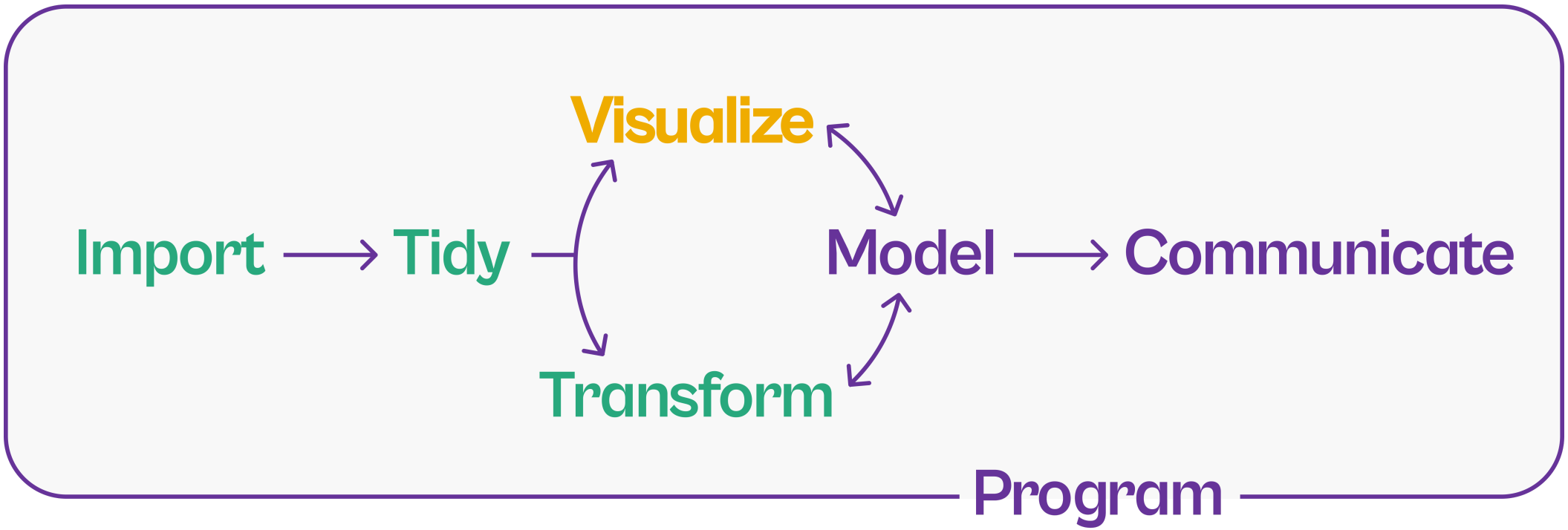




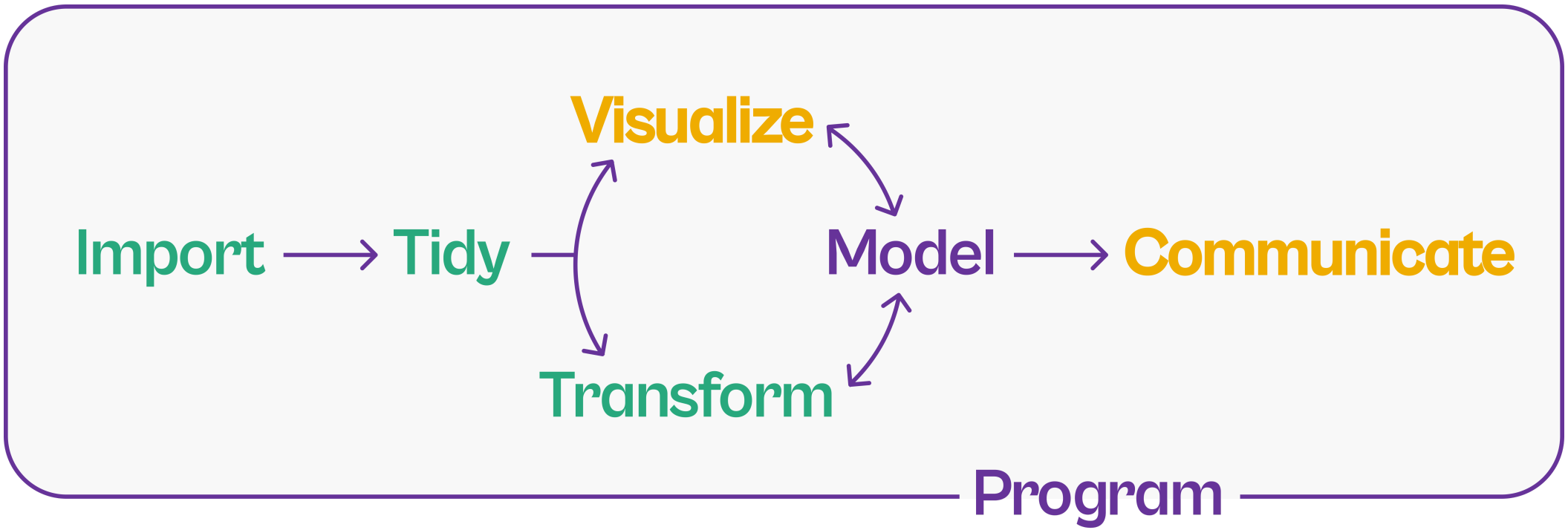
The data science workflow, modified from "R for Data Science"



The data science workflow, modified from "R for Data Science"



The data science workflow, modified from "R for Data Science"



The data science workflow, modified from "R for Data Science"

That's it Folks...

— Thank you! —