Manipulating tables with `dplyr`

Data Wrangling, Session 3 (contd)

Kieran Healy
Code Horizons

January 2026

Manipulating Tables with dplyr (contd)

Window functions and moving averages

Load our libraries

```r
library(here)      # manage file paths
library(socviz)    # data and some useful functions
library(tidyverse) # your friend and mine
```

dplyr's window functions

Ranking and cumulation within groups.

```r
## Data on COVID-19
library(covdata)

covnat_weekly
```

```
# A tibble: 4,966 × 11
   date       year_week cname   iso3      pop cases deaths cu_cases cu_deaths
   <date>     <chr>     <chr>   <chr>   <dbl> <dbl>  <dbl>    <dbl>     <dbl>
 1 2019-12-30 2020-01   Austria AUT   8932664    NA     NA       NA        NA
 2 2020-01-06 2020-02   Austria AUT   8932664    NA     NA       NA        NA
 3 2020-01-13 2020-03   Austria AUT   8932664    NA     NA       NA        NA
 4 2020-01-20 2020-04   Austria AUT   8932664    NA     NA       NA        NA
 5 2020-01-27 2020-05   Austria AUT   8932664    NA     NA       NA        NA
 6 2020-02-03 2020-06   Austria AUT   8932664    NA     NA       NA        NA
 7 2020-02-10 2020-07   Austria AUT   8932664    NA     NA       NA        NA
 8 2020-02-17 2020-08   Austria AUT   8932664    NA     NA       NA        NA
 9 2020-02-24 2020-09   Austria AUT   8932664    12      0       12         0
10 2020-03-02 2020-10   Austria AUT   8932664   115      0      127         0
# ℹ 4,956 more rows
# ℹ 2 more variables: r14_cases <dbl>, r14_deaths <dbl>
```

dplyr's window functions

cumsum() gives cumulative sums

```r
covnat_weekly |>
  filter(iso3 == "FRA") |>
  select(date, cname, iso3, cases) |>
  mutate(cases = ifelse(is.na(cases), 0, cases), # convert NA vals in `cases` to 0
         cumulative = cumsum(cases))
```

```
# A tibble: 159 × 5
   date       cname  iso3  cases cumulative
   <date>     <chr>  <chr> <dbl>      <dbl>
 1 2019-12-30 France FRA       0          0
 2 2020-01-06 France FRA       0          0
 3 2020-01-13 France FRA       0          0
 4 2020-01-20 France FRA       3          3
 5 2020-01-27 France FRA       3          6
 6 2020-02-03 France FRA       6         12
 7 2020-02-10 France FRA       0         12
 8 2020-02-17 France FRA       4         16
 9 2020-02-24 France FRA     133        149
10 2020-03-02 France FRA     981       1130
# ℹ 149 more rows
```

dplyr's window functions

`cume_dist()` gives the proportion of values <= to the current value.

```
covnat_weekly |>
  select(date, cname, iso3, deaths) |>
  filter(iso3 == "FRA") |>
  filter(cume_dist(desc(deaths)) < 0.1) # i.e. Top 10%
```

```
# A tibble: 15 × 4
   date       cname  iso3  deaths
   <date>     <chr>  <chr>  <dbl>
 1 2020-04-06 France FRA     3348
 2 2020-10-26 France FRA     3517
 3 2020-11-02 France FRA     5281
 4 2020-11-09 France FRA     6018
 5 2020-11-16 France FRA     6208
 6 2020-11-23 France FRA     5215
 7 2020-11-30 France FRA     4450
 8 2020-12-07 France FRA     4257
 9 2020-12-14 France FRA     3786
10 2020-12-21 France FRA     3560
11 2021-01-04 France FRA     3851
12 2021-01-11 France FRA     3833
13 2021-01-18 France FRA     3754
14 2021-01-25 France FRA     3535
15 2021-02-01 France FRA     3431
```

The `dplyr` vignette on Window functions is good.

An application

```r
covus |>
  filter(measure == "death") |>
  group_by(state) |>
  arrange(state, desc(date)) |>
  filter(state %in% "NY")
```

```
# A tibble: 371 × 7
# Groups:   state [1]
   date       state fips  data_quality_grade measure count measure_label
   <date>     <chr> <chr> <lgl>              <chr>   <dbl> <chr>
 1 2021-03-07 NY    36    NA                 death   39029 Deaths
 2 2021-03-06 NY    36    NA                 death   38970 Deaths
 3 2021-03-05 NY    36    NA                 death   38891 Deaths
 4 2021-03-04 NY    36    NA                 death   38796 Deaths
 5 2021-03-03 NY    36    NA                 death   38735 Deaths
 6 2021-03-02 NY    36    NA                 death   38660 Deaths
 7 2021-03-01 NY    36    NA                 death   38577 Deaths
 8 2021-02-28 NY    36    NA                 death   38497 Deaths
 9 2021-02-27 NY    36    NA                 death   38407 Deaths
10 2021-02-26 NY    36    NA                 death   38321 Deaths
# ℹ 361 more rows
```

Here the `count` measure is *cumulative* deaths. What if we want to recover the daily count for all the states in the data?

An application

`dplyr` has **`lead()`** and **`lag()`** functions. These allow you to access the previous and next values in a vector. You can calculate offsets this way.

```
my_vec <- c(1:20)
my_vec
```

```
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
lag(my_vec) # first element has no lag
```

```
[1] NA  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```

```
my_vec - lag(my_vec)
```

```
[1] NA  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

An application

We can write the expression directly:

```r
covus |>
  select(-data_quality_grade) |>
  filter(measure == "death") |>
  group_by(state) |>
  arrange(date) |>
  mutate(deaths_daily = count - lag(count, order_by = date)) |>
  arrange(state, desc(date)) |>
  filter(state %in% "NY")
```

```
# A tibble: 371 × 7
# Groups:   state [1]
   date       state fips  measure count measure_label deaths_daily
   <date>     <chr> <chr> <chr>   <dbl> <chr>                <dbl>
 1 2021-03-07 NY    36    death   39029 Deaths                  59
 2 2021-03-06 NY    36    death   38970 Deaths                  79
 3 2021-03-05 NY    36    death   38891 Deaths                  95
 4 2021-03-04 NY    36    death   38796 Deaths                  61
 5 2021-03-03 NY    36    death   38735 Deaths                  75
 6 2021-03-02 NY    36    death   38660 Deaths                  83
 7 2021-03-01 NY    36    death   38577 Deaths                  80
 8 2021-02-28 NY    36    death   38497 Deaths                  90
 9 2021-02-27 NY    36    death   38407 Deaths                  86
10 2021-02-26 NY    36    death   38321 Deaths                  94
# ℹ 361 more rows
```

Writing our own functions

We write functions using the special `function()` function.*

```r
my_fun <- function(x) {
  x + 1
}

my_fun # we've created the function; it's just an object
```

```
function (x)
{
    x + 1
}
```

```r
my_fun(x = 1) # But we can supply it with an input!
```

```
[1] 2
```

```r
my_fun(10)
```

```
[1] 11
```

*Nerds love this sort of stuff.

Writing our own functions

We write our function. It's just the expression we originally wrote, wrapped up.

```
get_daily_count <- function(count, date){
  count - lag(count, order_by = date)
}
```

This function has no generality, error-handling, or anything else. It's a once-off.

Writing our own functions

Now we can use it like any other:

```r
covus |>
  filter(measure == "death") |>
  select(-data_quality_grade) |>
  group_by(state) |>
  arrange(date) |>
  mutate(deaths_daily = get_daily_count(count, date)) |>
  arrange(state, desc(date)) |>
  filter(state %in% "NY")
```

```
# A tibble: 371 × 7
# Groups:   state [1]
   date       state fips  measure count measure_label deaths_daily
   <date>     <chr> <chr> <chr>   <dbl> <chr>                <dbl>
 1 2021-03-07 NY    36    death   39029 Deaths                  59
 2 2021-03-06 NY    36    death   38970 Deaths                  79
 3 2021-03-05 NY    36    death   38891 Deaths                  95
 4 2021-03-04 NY    36    death   38796 Deaths                  61
 5 2021-03-03 NY    36    death   38735 Deaths                  75
 6 2021-03-02 NY    36    death   38660 Deaths                  83
 7 2021-03-01 NY    36    death   38577 Deaths                  80
 8 2021-02-28 NY    36    death   38497 Deaths                  90
 9 2021-02-27 NY    36    death   38407 Deaths                  86
10 2021-02-26 NY    36    death   38321 Deaths                  94
# ℹ 361 more rows
```

Not super-useful quite yet, but if our task had more steps …

The `slider` package

Tidy moving averages with `slider`

`dplyr`'s window functions don't include moving averages.

There are several options, notably `RcppRoll`

We'll use the `slider` package.

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

Tidy moving averages with slider

```r
covus |>
  filter(measure == "death") |>
  select(-data_quality_grade) |>
  group_by(state) |>
  arrange(date) |>
  mutate(
    deaths_daily = get_daily_count(count, date),
    deaths7 = slide_mean(deaths_daily,
                         before = 7,
                         na_rm = TRUE)) |>
  arrange(state, desc(date)) |>
  filter(state %in% "NY")
```

```
# A tibble: 371 × 8
# Groups:   state [1]
   date       state fips  measure count measure_label deaths_daily deaths7
   <date>     <chr> <chr> <chr>   <dbl> <chr>                <dbl>   <dbl>
 1 2021-03-07 NY    36    death   39029 Deaths                  59    77.8
 2 2021-03-06 NY    36    death   38970 Deaths                  79    81.1
 3 2021-03-05 NY    36    death   38891 Deaths                  95    83
 4 2021-03-04 NY    36    death   38796 Deaths                  61    82.6
 5 2021-03-03 NY    36    death   38735 Deaths                  75    88
 6 2021-03-02 NY    36    death   38660 Deaths                  83    89.9
 7 2021-03-01 NY    36    death   38577 Deaths                  80    90.8
 8 2021-02-28 NY    36    death   38497 Deaths                  90    90.1
 9 2021-02-27 NY    36    death   38407 Deaths                  86    91.5
10 2021-02-26 NY    36    death   38321 Deaths                  94    95.6
# ℹ 361 more rows
```

Tidy moving averages with `slider`

```r
deaths7 = slide_mean(deaths_daily,
                     before = 7,
                     na_rm = TRUE))
```

Notice the Tidyverse-style `na_rm` argument rather than the usual base `na.rm`

The package provides a lot of different functions, from general-purpose **`slide_max()`**, **`slide_min()`** to more specialized sliding functions. In particular note e.g. **`slide_index_mean()`** that addresses some subtleties in averaging over dates with gaps.

Move columns with relocate()

```
gss_sm
```

```
# A tibble: 2,867 × 32
    year    id ballot       age childs sibs   degree race  sex   region income16
   <dbl> <dbl> <labelled> <dbl>  <dbl> <labe> <fct>  <fct> <fct> <fct>  <fct>
 1  2016     1 1             47      3 2      Bache… White Male  New E… $170000…
 2  2016     2 2             61      0 3      High … White Male  New E… $50000 …
 3  2016     3 3             72      2 3      Bache… White Male  New E… $75000 …
 4  2016     4 1             43      4 3      High … White Fema… New E… $170000…
 5  2016     5 3             55      2 2      Gradu… White Fema… New E… $170000…
 6  2016     6 2             53      2 2      Junio… White Fema… New E… $60000 …
 7  2016     7 1             50      2 2      High … White Male  New E… $170000…
 8  2016     8 3             23      3 6      High … Other Fema… Middl… $30000 …
 9  2016     9 1             45      3 5      High … Black Male  Middl… $60000 …
10  2016    10 3             71      4 1      Junio… White Male  Middl… $60000 …
# ℹ 2,857 more rows
# ℹ 21 more variables: relig <fct>, marital <fct>, padeg <fct>, madeg <fct>,
#   partyid <fct>, polviews <fct>, happy <fct>, partners <fct>, grass <fct>,
#   zodiac <fct>, pres12 <labelled>, wtssall <dbl>, income_rc <fct>,
#   agegrp <fct>, ageq <fct>, siblings <fct>, kids <fct>, religion <fct>,
#   bigregion <fct>, partners_rc <fct>, obama <dbl>
```

Shuffle columns around

```r
gss_sm
```

```
# A tibble: 2,867 × 32
    year    id ballot       age childs sibs   degree race  sex    region income16
   <dbl> <dbl> <labelled> <dbl>  <dbl> <labe> <fct>  <fct> <fct>  <fct>  <fct>
 1  2016     1 1             47      3 2      Bache… White Male   New E… $170000…
 2  2016     2 2             61      0 3      High … White Male   New E… $50000 …
 3  2016     3 3             72      2 3      Bache… White Male   New E… $75000 …
 4  2016     4 1             43      4 3      High … White Fema… New E… $170000…
 5  2016     5 3             55      2 2      Gradu… White Fema… New E… $170000…
 6  2016     6 2             53      2 2      Junio… White Fema… New E… $60000 …
 7  2016     7 1             50      2 2      High … White Male   New E… $170000…
 8  2016     8 3             23      3 6      High … Other Fema… Middl… $30000 …
 9  2016     9 1             45      3 5      High … Black Male   Middl… $60000 …
10  2016    10 3             71      4 1      Junio… White Male   Middl… $60000 …
# ℹ 2,857 more rows
# ℹ 21 more variables: relig <fct>, marital <fct>, padeg <fct>, madeg <fct>,
#   partyid <fct>, polviews <fct>, happy <fct>, partners <fct>, grass <fct>,
#   zodiac <fct>, pres12 <labelled>, wtssall <dbl>, income_rc <fct>,
#   agegrp <fct>, ageq <fct>, siblings <fct>, kids <fct>, religion <fct>,
#   bigregion <fct>, partners_rc <fct>, obama <dbl>
```

Shuffle columns around

```r
gss_sm |>
  select(region, bigregion, year,
         id:region,
         starts_with("p"),
         contains("income"))
```

```
# A tibble: 2,867 × 19
   region     bigregion  year    id ballot   age childs sibs  degree race  sex
   <fct>      <fct>     <dbl> <dbl> <labe> <dbl>  <dbl> <lab> <fct>  <fct> <fct>
 1 New Engla… Northeast  2016     1 1         47      3 2     Bache… White Male
 2 New Engla… Northeast  2016     2 2         61      0 3     High … White Male
 3 New Engla… Northeast  2016     3 3         72      2 3     Bache… White Male
 4 New Engla… Northeast  2016     4 1         43      4 3     High … White Fema…
 5 New Engla… Northeast  2016     5 3         55      2 2     Gradu… White Fema…
 6 New Engla… Northeast  2016     6 2         53      2 2     Junio… White Fema…
 7 New Engla… Northeast  2016     7 1         50      2 2     High … White Male
 8 Middle At… Northeast  2016     8 3         23      3 6     High … Other Fema…
 9 Middle At… Northeast  2016     9 1         45      3 5     High … Black Male
10 Middle At… Northeast  2016    10 3         71      4 1     Junio… White Male
# ℹ 2,857 more rows
# ℹ 8 more variables: padeg <fct>, partyid <fct>, polviews <fct>,
#   partners <fct>, pres12 <labelled>, partners_rc <fct>, income16 <fct>,
#   income_rc <fct>
```

Shuffle columns around

```
gss_sm ▷
  select(region, bigregion, year,
         id:region,
         starts_with("p"),
         contains("income")) ▷
  rename(children = childs,
         siblings = sibs)
```

```
# A tibble: 2,867 × 19
   region      bigregion  year    id ballot   age children siblings degree race 
   <fct>       <fct>     <dbl> <dbl> <labe> <dbl>    <dbl> <labell> <fct>  <fct>
 1 New England Northeast  2016     1 1         47        3 2        Bache… White
 2 New England Northeast  2016     2 2         61        0 3        High … White
 3 New England Northeast  2016     3 3         72        2 3        Bache… White
 4 New England Northeast  2016     4 1         43        4 3        High … White
 5 New England Northeast  2016     5 3         55        2 2        Gradu… White
 6 New England Northeast  2016     6 2         53        2 2        Junio… White
 7 New England Northeast  2016     7 1         50        2 2        High … White
 8 Middle Atl… Northeast  2016     8 3         23        3 6        High … Other
 9 Middle Atl… Northeast  2016     9 1         45        3 5        High … Black
10 Middle Atl… Northeast  2016    10 3         71        4 1        Junio… White
# ℹ 2,857 more rows
# ℹ 9 more variables: sex <fct>, padeg <fct>, partyid <fct>, polviews <fct>,
#   partners <fct>, pres12 <labelled>, partners_rc <fct>, income16 <fct>,
#   income_rc <fct>
```

Shuffle columns around

```r
gss_sm ▷
  select(region, bigregion, year,
         id:region,
         starts_with("p"),
         contains("income")) ▷
  rename(children = childs,
         siblings = sibs) ▷
  relocate(id)
```

```
# A tibble: 2,867 × 19
      id region      bigregion  year ballot   age children siblings degree race
   <dbl> <fct>       <fct>     <dbl> <labe> <dbl>    <dbl> <labell> <fct>  <fct>
 1     1 New England Northeast  2016 1         47        3 2        Bache… White
 2     2 New England Northeast  2016 2         61        0 3        High … White
 3     3 New England Northeast  2016 3         72        2 3        Bache… White
 4     4 New England Northeast  2016 1         43        4 3        High … White
 5     5 New England Northeast  2016 3         55        2 2        Gradu… White
 6     6 New England Northeast  2016 2         53        2 2        Junio… White
 7     7 New England Northeast  2016 1         50        2 2        High … White
 8     8 Middle Atl… Northeast  2016 3         23        3 6        High … Other
 9     9 Middle Atl… Northeast  2016 1         45        3 5        High … Black
10    10 Middle Atl… Northeast  2016 3         71        4 1        Junio… White
# ℹ 2,857 more rows
# ℹ 9 more variables: sex <fct>, padeg <fct>, partyid <fct>, polviews <fct>,
#   partners <fct>, pres12 <labelled>, partners_rc <fct>, income16 <fct>,
#   income_rc <fct>
```

Shuffle columns around

```
gss_sm ▷
  select(region, bigregion, year,
         id:region,
         starts_with("p"),
         contains("income")) ▷
  rename(children = childs,
         siblings = sibs) ▷
  relocate(id) ▷
  select(-ballot)
```

```
# A tibble: 2,867 × 18
      id region bigregion  year   age children siblings degree race  sex   padeg
   <dbl> <fct>  <fct>     <dbl> <dbl>    <dbl> <labell> <fct>  <fct> <fct> <fct>
 1     1 New E… Northeast  2016    47        3 2        Bache… White Male  Grad…
 2     2 New E… Northeast  2016    61        0 3        High … White Male  Lt H…
 3     3 New E… Northeast  2016    72        2 3        Bache… White Male  High…
 4     4 New E… Northeast  2016    43        4 3        High … White Fema… <NA>
 5     5 New E… Northeast  2016    55        2 2        Gradu… White Fema… Bach…
 6     6 New E… Northeast  2016    53        2 2        Junio… White Fema… <NA>
 7     7 New E… Northeast  2016    50        2 2        High … White Male  High…
 8     8 Middl… Northeast  2016    23        3 6        High … Other Fema… Lt H…
 9     9 Middl… Northeast  2016    45        3 5        High … Black Male  Lt H…
10    10 Middl… Northeast  2016    71        4 1        Junio… White Male  High…
# ℹ 2,857 more rows
# ℹ 7 more variables: partyid <fct>, polviews <fct>, partners <fct>,
#   pres12 <labelled>, partners_rc <fct>, income16 <fct>, income_rc <fct>
```

Shuffle columns around

```
gss_sm ▷
  select(region, bigregion, year,
         id:region,
         starts_with("p"),
         contains("income")) ▷
  rename(children = childs,
         siblings = sibs) ▷
  relocate(id) ▷
  select(-ballot) ▷
  relocate(where(is.numeric),
           .before = where(is.factor))
```

```
# A tibble: 2,867 × 18
      id  year   age children siblings   pres12    region bigregion degree race
   <dbl> <dbl> <dbl>    <dbl> <labelled> <labelle> <fct>  <fct>     <fct>  <fct>
 1     1  2016    47        3 2             3       New E… Northeast Bache… White
 2     2  2016    61        0 3             1       New E… Northeast High … White
 3     3  2016    72        2 3             2       New E… Northeast Bache… White
 4     4  2016    43        4 3             2       New E… Northeast High … White
 5     5  2016    55        2 2             1       New E… Northeast Gradu… White
 6     6  2016    53        2 2             1       New E… Northeast Junio… White
 7     7  2016    50        2 2            NA       New E… Northeast High … White
 8     8  2016    23        3 6            NA       Middl… Northeast High … Other
 9     9  2016    45        3 5            NA       Middl… Northeast High … Black
10    10  2016    71        4 1             2       Middl… Northeast Junio… White
# ℹ 2,857 more rows
# ℹ 8 more variables: sex <fct>, padeg <fct>, partyid <fct>, polviews <fct>,
#   partners <fct>, partners_rc <fct>, income16 <fct>, income_rc <fct>
```

Shuffle columns around

```r
gss_sm ▷
  select(region, bigregion, year,
         id:region,
         starts_with("p"),
         contains("income")) ▷
  rename(children = childs,
         siblings = sibs) ▷
  relocate(id) ▷
  select(-ballot) ▷
  relocate(where(is.numeric),
           .before = where(is.factor)) ▷
  relocate(contains("region"),
           .after = year)
```

```
# A tibble: 2,867 × 18
      id  year region      bigregion   age children siblings pres12 degree race
   <dbl> <dbl> <fct>       <fct>     <dbl>    <dbl> <labell> <labe> <fct>  <fct>
 1     1  2016 New England Northeast    47        3 2          3     Bache… White
 2     2  2016 New England Northeast    61        0 3          1     High … White
 3     3  2016 New England Northeast    72        2 3          2     Bache… White
 4     4  2016 New England Northeast    43        4 3          2     High … White
 5     5  2016 New England Northeast    55        2 2          1     Gradu… White
 6     6  2016 New England Northeast    53        2 2          1     Junio… White
 7     7  2016 New England Northeast    50        2 2         NA     High … White
 8     8  2016 Middle Atl… Northeast    23        3 6         NA     High … Other
 9     9  2016 Middle Atl… Northeast    45        3 5         NA     High … Black
10    10  2016 Middle Atl… Northeast    71        4 1          2     Junio… White
# ℹ 2,857 more rows
# ℹ 8 more variables: sex <fct>, padeg <fct>, partyid <fct>, polviews <fct>,
#   partners <fct>, partners_rc <fct>, income16 <fct>, income_rc <fct>
```

Example: UK Election Data

```r
library(ukelection2019)

ukvote2019
```

```
# A tibble: 3,320 × 13
   cid     constituency electorate party_name candidate votes vote_share_percent
   <chr>   <chr>             <int> <chr>      <chr>     <int>              <dbl>
 1 W07000… Aberavon          50747 Labour     Stephen … 17008               53.8
 2 W07000… Aberavon          50747 Conservat… Charlott…  6518               20.6
 3 W07000… Aberavon          50747 The Brexi… Glenda D…  3108                9.8
 4 W07000… Aberavon          50747 Plaid Cym… Nigel Hu…  2711                8.6
 5 W07000… Aberavon          50747 Liberal D… Sheila K…  1072                3.4
 6 W07000… Aberavon          50747 Independe… Captain …   731                2.3
 7 W07000… Aberavon          50747 Green      Giorgia …   450                1.4
 8 W07000… Aberconwy         44699 Conservat… Robin Mi… 14687               46.1
 9 W07000… Aberconwy         44699 Labour     Emily Ow… 12653               39.7
10 W07000… Aberconwy         44699 Plaid Cym… Lisa Goo…  2704                8.5
# ℹ 3,310 more rows
# ℹ 6 more variables: vote_share_change <dbl>, total_votes_cast <int>,
#   vrank <int>, turnout <dbl>, fname <chr>, lname <chr>
```

Example: UK Election Data

Use `sample_n()` to sample `n` rows of your tibble.

```
library(ukelection2019)

ukvote2019 ▷
  sample_n(10)
```

```
# A tibble: 10 × 13
   cid     constituency electorate party_name candidate votes vote_share_percent
   <chr>   <chr>             <int> <chr>      <chr>     <int>              <dbl>
 1 E14000… Norfolk Sou…      78455 Green      Pallavi …  1645                3.2
 2 E14000… Norfolk Sou…      78455 Liberal D… Josie Ra…  4166                8.1
 3 E14000… Stafford          72572 Conservat… Theo Cla… 29992               58.6
 4 E14001… Wycombe           78094 Green      Peter Si…  1454                2.7
 5 S14000… East Kilbri…      81224 UKIP       David Ma…   559                1
 6 E14000… High Peak         74343 Green      Robert H…  1148                2.1
 7 E14000… Hammersmith       74759 Liberal D… Jessie V…  6947               13.4
 8 E14001… Wansbeck          63339 Liberal D… Stephen …  2539                6.3
 9 E14000… Bosworth          81537 Liberal D… Michael …  9096               16.1
10 E14000… Crewe & Nan…      80321 Labour     Laura Sm… 20196               37.4
# ℹ 6 more variables: vote_share_change <dbl>, total_votes_cast <int>,
#   vrank <int>, turnout <dbl>, fname <chr>, lname <chr>
```

Example: UK Election Data

A one-column tibble of unique constituency names

```
ukvote2019 |>
  distinct(constituency)
```

```
# A tibble: 650 × 1
   constituency
   <chr>
 1 Aberavon
 2 Aberconwy
 3 Aberdeen North
 4 Aberdeen South
 5 Aberdeenshire West & Kincardine
 6 Airdrie & Shotts
 7 Aldershot
 8 Aldridge-Brownhills
 9 Altrincham & Sale West
10 Alyn & Deeside
# ℹ 640 more rows
```

Example: UK Election Data

Tally them up

```r
ukvote2019 |>
  distinct(constituency) |>
  tally()
```

```
# A tibble: 1 × 1
      n
  <int>
1   650
```

```r
# Base R / non-pipeline version

length(unique(ukvote2019$constituency))
```

```
[1] 650
```

Example: UK Election Data

Which parties fielded the most candidates?

```
ukvote2019 |>
  count(party_name) |>
  arrange(desc(n))
```

```
# A tibble: 69 × 2
   party_name                    n
   <chr>                     <int>
 1 Conservative                636
 2 Labour                      631
 3 Liberal Democrat            611
 4 Green                       497
 5 The Brexit Party            275
 6 Independent                 224
 7 Scottish National Party      59
 8 UKIP                         44
 9 Plaid Cymru                  36
10 Christian Peoples Alliance   29
# ℹ 59 more rows
```

Example: UK Election Data

Top 5

```r
ukvote2019 |>
  count(party_name) |>
  slice_max(order_by = n, n = 5)
```

```
# A tibble: 5 × 2
  party_name           n
  <chr>            <int>
1 Conservative       636
2 Labour             631
3 Liberal Democrat   611
4 Green              497
5 The Brexit Party   275
```

Example: UK Election Data

Top 5

```r
ukvote2019 |>
  count(party_name) |>
  slice_max(order_by = n, n = 5)
```

```
# A tibble: 5 × 2
  party_name          n
  <chr>           <int>
1 Conservative      636
2 Labour            631
3 Liberal Democrat  611
4 Green             497
5 The Brexit Party  275
```

Bottom 5

```r
ukvote2019 |>
  count(party_name) |>
  slice_min(order_by = n, n = 5)
```

```
# A tibble: 25 × 2
   party_name                                n
   <chr>                                 <int>
 1 Ashfield Independents                     1
 2 Best for Luton                            1
 3 Birkenhead Social Justice Party           1
 4 British National Party                    1
 5 Burnley & Padiham Independent Party       1
 6 Church of the Militant Elvis Party        1
 7 Citizens Movement Party UK                1
 8 CumbriaFirst                              1
 9 Heavy Woollen District Independents       1
10 Independent Network                       1
# ℹ 15 more rows
```

Example: UK Election Data

How many constituencies are there?

```r
ukvote2019 ▷
  count(constituency)
```

```
# A tibble: 650 × 2
   constituency                          n
   <chr>                             <int>
 1 Aberavon                              7
 2 Aberconwy                             4
 3 Aberdeen North                        6
 4 Aberdeen South                        4
 5 Aberdeenshire West & Kincardine       4
 6 Airdrie & Shotts                      5
 7 Aldershot                             4
 8 Aldridge-Brownhills                   5
 9 Altrincham & Sale West                6
10 Alyn & Deeside                        5
# ℹ 640 more rows
```

```r
ukvote2019 ▷
  distinct(constituency) ▷
  count()
```

```
# A tibble: 1 × 1
      n
  <int>
1   650
```

```r
# Base R style ...
length(unique(ukvote2019$constituency))
```

```
[1] 650
```

Counting Twice Over

```r
ukvote2019 |>
  count(constituency) |>
  count(n)
```

```
# A tibble: 8 × 2
      n    nn
  <int> <int>
1     3    21
2     4   194
3     5   226
4     6   139
5     7    49
6     8    18
7     9     2
8    12     1
```

Counting Twice Over

```
ukvote2019
```

```
# A tibble: 3,320 × 13
   cid     constituency electorate party_name candidate votes
vote_share_percent
   <chr>   <chr>             <int> <chr>      <chr>     <int>
<dbl>
 1 W07000… Aberavon          50747 Labour     Stephen … 17008
53.8
 2 W07000… Aberavon          50747 Conservat… Charlott…  6518
20.6
 3 W07000… Aberavon          50747 The Brexi… Glenda D…  3108
9.8
 4 W07000… Aberavon          50747 Plaid Cym… Nigel Hu…  2711
8.6
 5 W07000… Aberavon          50747 Liberal D… Sheila K…  1072
3.4
 6 W07000… Aberavon          50747 Independe… Captain …   731
2.3
 7 W07000… Aberavon          50747 Green      Giorgia …   450
1.4
 8 W07000… Aberconwy         44699 Conservat… Robin Mi… 14687
46.1
 9 W07000… Aberconwy         44699 Labour     Emily Ow… 12653
39.7
10 W07000… Aberconwy         44699 Plaid Cym… Lisa Goo…  2704
8.5
```

Counting Twice Over

```r
ukvote2019 |>
  count(constituency, name = "n_cands")
```

```
# A tibble: 650 × 2
   constituency                     n_cands
   <chr>                              <int>
 1 Aberavon                               7
 2 Aberconwy                              4
 3 Aberdeen North                         6
 4 Aberdeen South                         4
 5 Aberdeenshire West & Kincardine        4
 6 Airdrie & Shotts                       5
 7 Aldershot                              4
 8 Aldridge-Brownhills                    5
 9 Altrincham & Sale West                 6
10 Alyn & Deeside                         5
# ℹ 640 more rows
```

Counting Twice Over

```r
ukvote2019 |>
  count(constituency, name = "n_cands") |>
  count(n_cands, name = "n_const")
```

```
# A tibble: 8 × 2
  n_cands n_const
    <int>   <int>
1       3      21
2       4     194
3       5     226
4       6     139
5       7      49
6       8      18
7       9       2
8      12       1
```

Recap and Looking Ahead

Recap and Looking Ahead

Coding as gardening

Working in RStudio with RMarkdown documents

Core `dplyr` verbs

Subset your table: `filter()` rows, `select()` columns

Logically `group_by()` one or more columns

Add columns with `mutate()`

Summarize (by group, or the whole table) with `summarize()`

Expand your `dplyr` actions

Count up rows with `n()`, `tally()` or `count()`

Calculate quantities with `sum()`, `mean()`, `min()`, etc

Subset rows with logical expressions or `slice` functions

Conditionally select columns by name directly, with `%in%` or `%nin%`, or with tidy selectors like `starts_with()`, `ends_with()`, `contains()`

Conditionally select columns by *type* with `where()` and some criterion, e.g. `where(is.numeric)`

Conditionally select and then *act* on columns with `across(where(<condition>), <action>)`

Expand your `dplyr` actions

Tidy up columns with `relocate()` and `rename()`

Tidy up rows with `arrange()`

A dplyr shortcut

A dplyr shortcut

So far we have been writing, e.g.,

```r
gss_sm ▷
  group_by(bigregion, religion) ▷
  summarize(total = n())
```

```
# A tibble: 24 × 3
# Groups:   bigregion [4]
   bigregion religion   total
   <fct>     <fct>      <int>
 1 Northeast Protestant   158
 2 Northeast Catholic     162
 3 Northeast Jewish        27
 4 Northeast None         112
 5 Northeast Other         28
 6 Northeast <NA>           1
 7 Midwest   Protestant   325
 8 Midwest   Catholic     172
 9 Midwest   Jewish         3
10 Midwest   None         157
# ℹ 14 more rows
```

A dplyr shortcut

Or

```
gss_sm ▷
  group_by(bigregion, religion) ▷
  tally()
```

```
# A tibble: 24 × 3
# Groups:   bigregion [4]
   bigregion religion        n
   <fct>     <fct>       <int>
 1 Northeast Protestant    158
 2 Northeast Catholic      162
 3 Northeast Jewish         27
 4 Northeast None          112
 5 Northeast Other          28
 6 Northeast <NA>            1
 7 Midwest   Protestant    325
 8 Midwest   Catholic      172
 9 Midwest   Jewish          3
10 Midwest   None          157
# ℹ 14 more rows
```

A dplyr shortcut

Or

```
gss_sm ▷
  count(bigregion, religion)
```

```
# A tibble: 24 × 3
   bigregion religion        n
   <fct>     <fct>       <int>
 1 Northeast Protestant    158
 2 Northeast Catholic      162
 3 Northeast Jewish         27
 4 Northeast None          112
 5 Northeast Other          28
 6 Northeast <NA>            1
 7 Midwest   Protestant    325
 8 Midwest   Catholic      172
 9 Midwest   Jewish          3
10 Midwest   None          157
# ℹ 14 more rows
```

With this last one the final result is *ungrouped*, no matter how many levels of grouping there are going in.

A dplyr shortcut

But we can also write this:

```
gss_sm |>
  summarize(total = n(), .by = c(bigregion, religion))
```

```
# A tibble: 24 × 3
   bigregion religion   total
   <fct>     <fct>      <int>
 1 Northeast None         112
 2 Northeast Catholic     162
 3 Northeast Protestant   158
 4 Northeast Other         28
 5 Northeast Jewish        27
 6 West      Jewish        10
 7 West      None         180
 8 West      Other         48
 9 West      Protestant   238
10 West      Catholic     155
# ℹ 14 more rows
```

By default the result is an *ungrouped* tibble, whereas with `group_by()` … `summarize()` the result would still be grouped by `bigregion` at the end. To prevent unexpected results, you can't use `.by` on tibble that's already grouped.

Data as implicitly first

This code:

```
gss_sm |>
  summarize(total = n(), .by = c(bigregion, religion))
```

```
# A tibble: 24 × 3
   bigregion religion   total
   <fct>     <fct>      <int>
 1 Northeast None         112
 2 Northeast Catholic     162
 3 Northeast Protestant   158
 4 Northeast Other         28
 5 Northeast Jewish        27
 6 West      Jewish        10
 7 West      None         180
 8 West      Other         48
 9 West      Protestant   238
10 West      Catholic     155
# ℹ 14 more rows
```

Data as implicitly first

… is equivalent to this:

```
summarize(gss_sm, total = n(), .by = c(bigregion, religion))
```

```
# A tibble: 24 × 3
   bigregion religion   total
   <fct>     <fct>      <int>
 1 Northeast None         112
 2 Northeast Catholic     162
 3 Northeast Protestant   158
 4 Northeast Other         28
 5 Northeast Jewish        27
 6 West      Jewish        10
 7 West      None         180
 8 West      Other         48
 9 West      Protestant   238
10 West      Catholic     155
# ℹ 14 more rows
```

This is true of Tidyverse pipelines in general. Let's look at the help for `summarize()` to see why.

Two dplyr gotchas

Comparisons filtering on proportions

Let’s say you are working with proportions …

```
df
```

```
# A tibble: 4 × 3
  id    prop1 prop2
  <chr> <dbl> <dbl>
1 A       0.1   0.2
2 B       0.1   0.21
3 C       0.11  0.2
4 D       0.1   0.1
```

Comparisons filtering on proportions

And you want to focus on cases where `prop1` *plus* `prop2` is greater than 0.3:

```
df |>
  filter(prop1 + prop2 > 0.3)
```

```
# A tibble: 3 × 3
  id    prop1 prop2
  <chr> <dbl> <dbl>
1 A      0.1   0.2
2 B      0.1   0.21
3 C      0.11  0.2
```

Comparisons filtering on proportions

And you want to focus on cases where `prop1` *plus* `prop2` is greater than 0.3:

```
df |>
  filter(prop1 + prop2 > 0.3)
```

```
# A tibble: 3 × 3
  id    prop1 prop2
  <chr> <dbl> <dbl>
1 A      0.1   0.2
2 B      0.1   0.21
3 C      0.11  0.2
```

The row with `id A` shouldn't have been included there.

This is not dplyr's fault. It's our floating point friend again.

Comparisons filtering on proportions

```r
df |>
  filter(prop1 + prop2 == 0.3)
```

```
# A tibble: 0 × 3
# ℹ 3 variables: id <chr>, prop1 <dbl>, prop2 <dbl>
```

The row with `id` `A` *should* have been included here!

Comparisons filtering on proportions

This won't give the right behavior either:

```r
df |>
  mutate(prop3 = prop1 + prop2) |>
  filter(prop3 == 0.3)
```

```
# A tibble: 0 × 4
# ℹ 4 variables: id <chr>, prop1 <dbl>, prop2 <dbl>, prop3 <dbl>
```

Comparisons filtering on proportions

So, beware.

```
df |>
  filter(prop1*100 + prop2*100 == 0.3*100)
```

```
# A tibble: 1 × 3
  id    prop1 prop2
  <chr> <dbl> <dbl>
1 A       0.1   0.2
```

Better:

```
df |>
  filter(near(prop1 + prop2, 0.3))
```

```
# A tibble: 1 × 3
  id    prop1 prop2
  <chr> <dbl> <dbl>
1 A       0.1   0.2
```

Zero Counts in dplyr

```
df <- read_csv(here("data", "first_terms.csv"))

df
```

```
# A tibble: 280 × 4
     pid start_year party      sex
   <dbl> <date>     <chr>      <chr>
 1  3160 2013-01-03 Republican M
 2  3161 2013-01-03 Democratic F
 3  3162 2013-01-03 Democratic M
 4  3163 2013-01-03 Republican M
 5  3164 2013-01-03 Democratic M
 6  3165 2013-01-03 Republican M
 7  3166 2013-01-03 Republican M
 8  3167 2013-01-03 Democratic F
 9  3168 2013-01-03 Republican M
10  3169 2013-01-03 Democratic M
# ℹ 270 more rows
```

Zero Counts in dplyr

```r
df |>
    group_by(start_year, party, sex) |>
    summarize(N = n()) |>
    mutate(freq = N / sum(N))
```

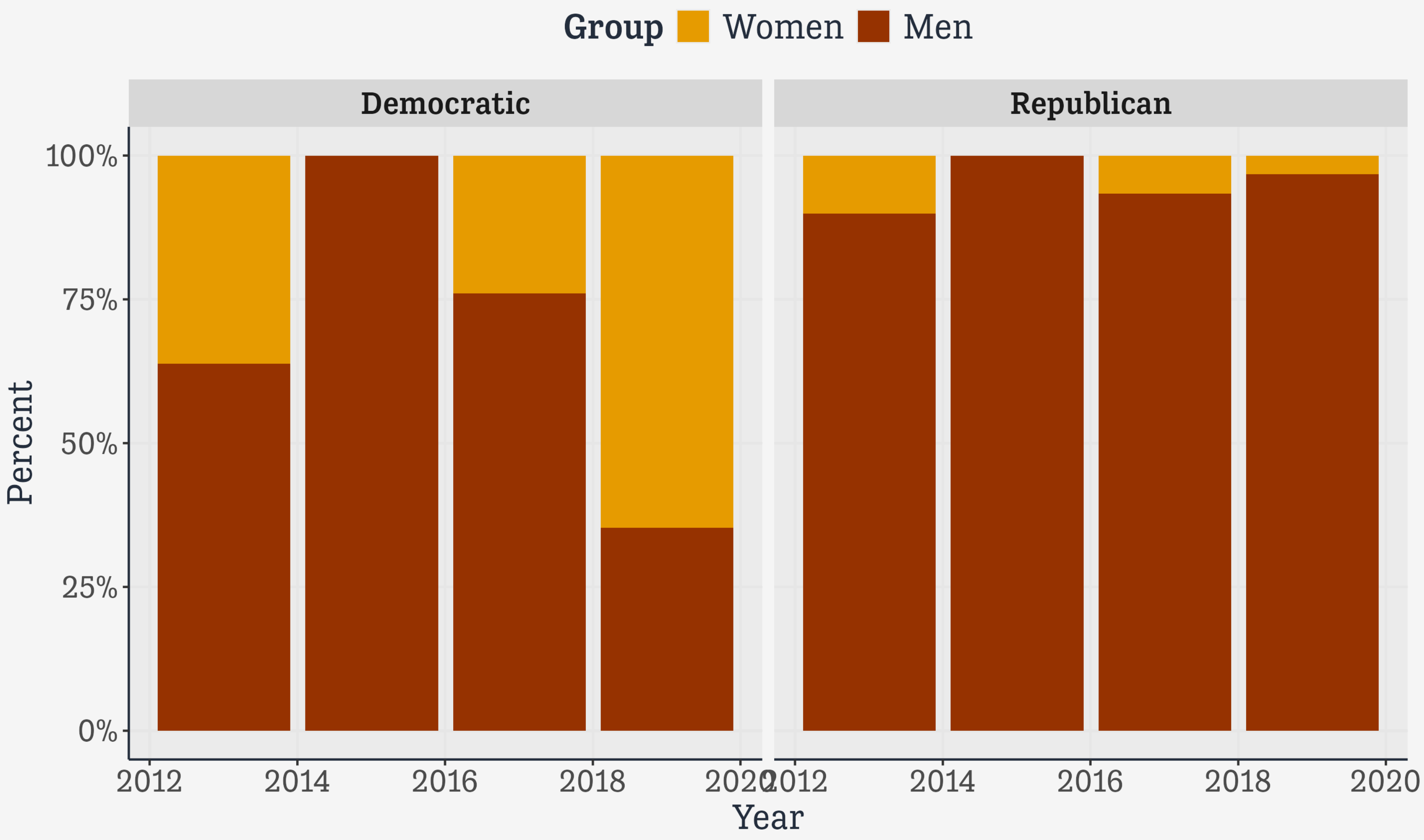
```
# A tibble: 14 × 5
# Groups:   start_year, party [8]
   start_year party      sex       N   freq
   <date>     <chr>      <chr> <int>  <dbl>
 1 2013-01-03 Democratic F        21 0.362
 2 2013-01-03 Democratic M        37 0.638
 3 2013-01-03 Republican F         8 0.101
 4 2013-01-03 Republican M        71 0.899
 5 2015-01-03 Democratic M         1 1
 6 2015-01-03 Republican M         5 1
 7 2017-01-03 Democratic F         6 0.24
 8 2017-01-03 Democratic M        19 0.76
 9 2017-01-03 Republican F         2 0.0667
10 2017-01-03 Republican M        28 0.933
11 2019-01-03 Democratic F        33 0.647
12 2019-01-03 Democratic M        18 0.353
13 2019-01-03 Republican F         1 0.0323
14 2019-01-03 Republican M        30 0.968
```

Zero Counts in dplyr

```r
p_col <- df |>
    group_by(start_year, party, sex) |>
    summarize(N = n()) |>
    mutate(freq = N / sum(N)) |>
    ggplot(aes(x = start_year,
               y = freq,
               fill = sex)) +
    geom_col() +
    scale_y_continuous(labels = scales::percent) +
    scale_fill_manual(values = sex_colors, labels = c("Women", "Men")) +
    labs(x = "Year", y = "Percent", fill = "Group") +
    facet_wrap(~ party)
```

Zero Counts in dplyr

```
p_col
```

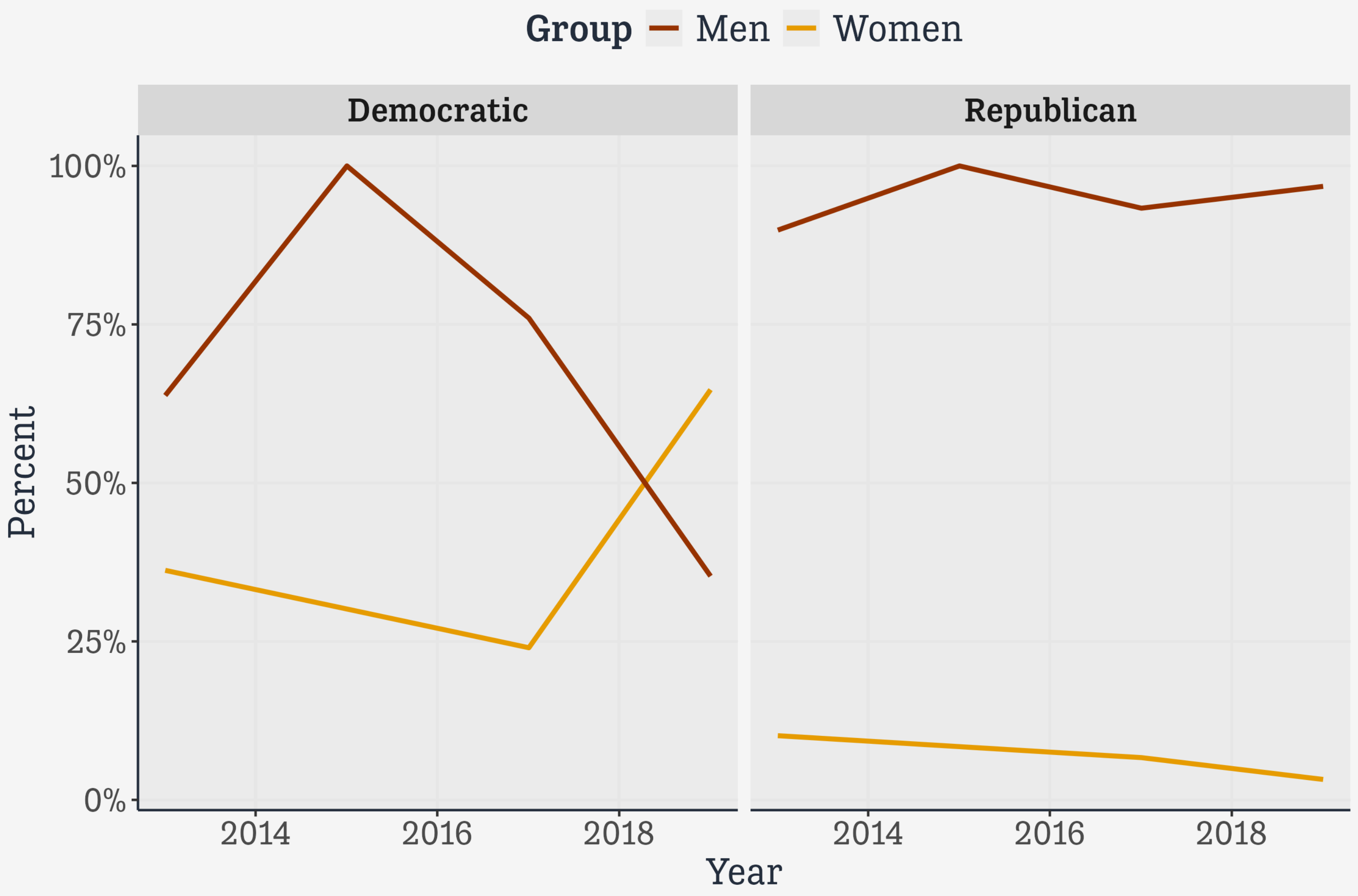


2. Zero Counts in dplyr

```r
p_line <- df |>
    group_by(start_year, party, sex) |>
    summarize(N = n()) |>
    mutate(freq = N / sum(N)) |>
    ggplot(aes(x = start_year,
               y = freq,
               color = sex)) +
    geom_line(size = 1.1) +
    scale_y_continuous(labels = scales::percent) +
    scale_color_manual(values = sex_colors, labels = c("Women", "Men")) +
    guides(color = guide_legend(reverse = TRUE)) +
    labs(x = "Year", y = "Percent", color = "Group") +
    facet_wrap(~ party)
```

Zero Counts in dplyr

```
p_line
```



Option 1: factors and .drop

Factors are for categorical variables and are stored differently from characters.

This can matter when modeling, and also now.

```
df_f <- df |>
  mutate(party_f = factor(party))

df_f
```

```
# A tibble: 280 × 5
     pid start_year party      sex   party_f
   <dbl> <date>     <chr>      <chr> <fct>
 1  3160 2013-01-03 Republican M     Republican
 2  3161 2013-01-03 Democratic F     Democratic
 3  3162 2013-01-03 Democratic M     Democratic
 4  3163 2013-01-03 Republican M     Republican
 5  3164 2013-01-03 Democratic M     Democratic
 6  3165 2013-01-03 Republican M     Republican
 7  3166 2013-01-03 Republican M     Republican
 8  3167 2013-01-03 Democratic F     Democratic
 9  3168 2013-01-03 Republican M     Republican
10  3169 2013-01-03 Democratic M     Democratic
# ℹ 270 more rows
```

Option 1: factors and `.drop`

```
df_f ▷
  group_by(party_f) ▷
  tally()
```

```
# A tibble: 2 × 2
  party_f        n
  <fct>      <int>
1 Democratic   135
2 Republican   145
```

Factors are integer values with named labels, or *levels*:

```
typeof(df_f$party_f)
```

```
[1] "integer"
```

```
levels(df_f$party_f)
```

```
[1] "Democratic" "Republican"
```

Option 1: factors and .drop

By default, unused levels won't display:

```
df_f <- df |>
  mutate(party_f = factor(party,
                        levels = c("Democratic",
                                   "Republican",
                                   "Libertarian")))
df_f |>
  group_by(party_f) |>
  tally()
```

```
# A tibble: 2 × 2
  party_f        n
  <fct>      <int>
1 Democratic   135
2 Republican   145
```

```
levels(df_f$party_f)
```

```
[1] "Democratic"  "Republican"  "Libertarian"
```

Option 1: factors and `.drop`

By default, unused levels won't display:

```
df ▷
  mutate(across(where(is.character), as_factor)) ▷
  group_by(start_year, party, sex) ▷
  summarize(N = n()) ▷
  mutate(freq = N / sum(N))
```

```
# A tibble: 14 × 5
# Groups:   start_year, party [8]
   start_year party      sex       N   freq
   <date>     <fct>      <fct> <int>  <dbl>
 1 2013-01-03 Republican M        71 0.899
 2 2013-01-03 Republican F         8 0.101
 3 2013-01-03 Democratic M        37 0.638
 4 2013-01-03 Democratic F        21 0.362
 5 2015-01-03 Republican M         5 1
 6 2015-01-03 Democratic M         1 1
 7 2017-01-03 Republican M        28 0.933
 8 2017-01-03 Republican F         2 0.0667
 9 2017-01-03 Democratic M        19 0.76
10 2017-01-03 Democratic F         6 0.24
11 2019-01-03 Republican M        30 0.968
12 2019-01-03 Republican F         1 0.0323
13 2019-01-03 Democratic M        18 0.353
14 2019-01-03 Democratic F        33 0.647
```

Option 1: factors and `.drop`

You can make `dplyr` keep empty factor levels though:

```
df ▷
  mutate(across(where(is.character), as_factor)) ▷
  group_by(start_year, party, sex, .drop = FALSE) ▷
  summarize(N = n()) ▷
  mutate(freq = N / sum(N))
```

```
# A tibble: 16 × 5
# Groups:   start_year, party [8]
   start_year party      sex       N   freq
   <date>     <fct>      <fct> <int>  <dbl>
 1 2013-01-03 Republican M        71 0.899
 2 2013-01-03 Republican F         8 0.101
 3 2013-01-03 Democratic M        37 0.638
 4 2013-01-03 Democratic F        21 0.362
 5 2015-01-03 Republican M         5 1
 6 2015-01-03 Republican F         0 0
 7 2015-01-03 Democratic M         1 1
 8 2015-01-03 Democratic F         0 0
 9 2017-01-03 Republican M        28 0.933
10 2017-01-03 Republican F         2 0.0667
11 2017-01-03 Democratic M        19 0.76
12 2017-01-03 Democratic F         6 0.24
13 2019-01-03 Republican M        30 0.968
14 2019-01-03 Republican F         1 0.0323
15 2019-01-03 Democratic M        18 0.353
```

Option 2: ungroup() and complete()

Maybe you don't want to deal with factors.

```
df_c <- df |>
    group_by(start_year, party, sex) |>
    summarize(N = n()) |>
    mutate(freq = N / sum(N)) |>
    ungroup() |>
    complete(start_year, party, sex,
             fill = list(N = 0, freq = 0))
```

Option 2: ungroup() and complete()

```
df_c
```

```
# A tibble: 16 × 5
   start_year party      sex        N   freq
   <date>     <chr>      <chr> <int>  <dbl>
 1 2013-01-03 Democratic F        21 0.362
 2 2013-01-03 Democratic M        37 0.638
 3 2013-01-03 Republican F         8 0.101
 4 2013-01-03 Republican M        71 0.899
 5 2015-01-03 Democratic F         0 0
 6 2015-01-03 Democratic M         1 1
 7 2015-01-03 Republican F         0 0
 8 2015-01-03 Republican M         5 1
 9 2017-01-03 Democratic F         6 0.24
10 2017-01-03 Democratic M        19 0.76
11 2017-01-03 Republican F         2 0.0667
12 2017-01-03 Republican M        28 0.933
13 2019-01-03 Democratic F        33 0.647
14 2019-01-03 Democratic M        18 0.353
15 2019-01-03 Republican F         1 0.0323
16 2019-01-03 Republican M        30 0.968
```

Option 2: ungroup() and complete()

```r
p_out <- df_c |>
  ggplot(aes(x = start_year,
             y = freq,
             color = sex)) +
    geom_line(size = 1.1) +
    scale_y_continuous(labels = scales::percent) +
    scale_color_manual(values = sex_colors, labels = c("Women", "Men")) +
    guides(color = guide_legend(reverse = TRUE)) +
    labs(x = "Year", y = "Percent", color = "Group") +
    facet_wrap(~ party)
```

Option 2: ungroup() and complete()

```
p_out
```