Module 3: R

Manipulation

Instructor: Anjali Silva, PhD

Data Sciences Institute, University of Toronto

2022

Course Documents

- Visit: https://github.com/anjalisilva/IntroductionToR
- All course material will be available via IntroductionToR GitHub repository (https://github.com/anjalisilva/IntroductionToR). Folder structure is as follows:
  - Lessons - All files: This folder contains all files.
  - Lessons - Data only: This folder contains data only.
  - Lessons - Lesson Plans only: This folder contains lesson plans only.
  - Lessons - PDF only: This folder contains slide PDFs only.
  - README - README file
  - .gitignore - Files to ignore specified by instructor

Course Contacts

- Instructor: Anjali Silva Email: a.silva@utoronto.ca (Must use the subject line DSI-IntroR. E.g., DSI-IntroR: Inquiry about Lecture I.)
- TA: see GitHub

Overview

- Filtering (Wickham and Grolemund, 2017 Chapter 5, Timbers et al. 2021, Chapter 3.6)
- Arranging (Wickham and Grolemund, 2017 Chapter 5)
- Selecting (Wickham and Grolemund, 2017 Chapter 5, Timbers et al. 2021, Chapter 3.5)
- The pipe (Wickham and Grolemund, 2017 Chapter 5 & 18; Timbers et al. 2021, Chapter 3.8)
- Mutating (Wickham and Grolemund, 2017 Chapter 5, Timbers et al. 2021, Chapter 3.7, 3.10)
- Summarising (Wickham and Grolemund, 2017 Chapter 5, Timbers et al. 2021, Chapter 3.9)
- Grouping (Wickham and Grolemund, 2017 Chapter 5)
- Cleaning (Alexander, 2022, Chapter 11)

Take a look

```
glimpse(ads_data)
```

```
## Rows: 1,460
## Columns: 52
## $ StartDate             <dttm> 2019-06-14 09:43:20,…
## $ EndDate               <dttm> 2019-06-14 09:44:30,…
## $ Status                <dbl+lbl> 0, 0, 0, 0, 0, 0,…
## $ Progress              <dbl> 100, 100, 100, 100, 1…
## $ Duration__in_seconds_ <dbl> 70, 105, 88, 109, 109…
## $ Finished              <dbl+lbl> 1, 1, 1, 1, 1, 1,…
## $ RecordedDate          <dttm> 2019-06-14 09:44:31,…
## $ ResponseId            <chr> "R_11dq3s9btLX57LD", …
## $ DistributionChannel   <chr> "anonymous", "anonymo…
## $ UserLanguage          <chr> "EN", "EN", "EN", "EN…
## $ Consent               <dbl+lbl> 1, 1, 1, 1, 1, 1,…
## $ Pol_7                 <dbl+lbl> 5, 3, 1, 2, 6, 4,…
## $ W2_Knowledge          <dbl+lbl> 2, 2, 4, 1, 3, 2,…
## $ Gender                <dbl+lbl> 2, 1, 2, 1, 1, 1,…
## $ Race                  <dbl+lbl> 1, 1, 1, 1, 1, 3,…
## $ W1_Feeling_1          <dbl> 2, 1, 4, 3, 3, 3, 6, …
## $ W1_Actions_1_1        <dbl+lbl> NA, NA, NA, NA, N…
## $ W1_Actions_1_2        <dbl+lbl>  1, NA, NA,  1, N…
## $ W1_Actions_1_3        <dbl+lbl> NA, NA,  1, NA, N…
```

Filtering

Filtering allows us to select rows based on specific traits

```
filter(ads_data, Duration__in_seconds_ < 100)
```

```
## # A tibble: 41 × 52
##    StartDate           EndDate             Status
##    <dttm>              <dttm>              <dbl+lbl>
##  1 2019-06-14 09:43:20 2019-06-14 09:44:30 0 [IP Add…
##  2 2019-06-14 09:43:29 2019-06-14 09:44:58 0 [IP Add…
##  3 2019-06-14 09:44:00 2019-06-14 09:45:11 0 [IP Add…
##  4 2019-06-14 09:43:32 2019-06-14 09:45:12 0 [IP Add…
##  5 2019-06-14 09:43:48 2019-06-14 09:45:25 0 [IP Add…
##  6 2019-06-14 09:44:24 2019-06-14 09:45:26 0 [IP Add…
##  7 2019-06-14 09:43:50 2019-06-14 09:45:29 0 [IP Add…
##  8 2019-06-14 09:44:15 2019-06-14 09:45:42 0 [IP Add…
##  9 2019-06-14 09:44:30 2019-06-14 09:45:58 0 [IP Add…
## 10 2019-06-14 09:44:36 2019-06-14 09:46:05 0 [IP Add…
## # … with 31 more rows, and 49 more variables:
## #   Progress <dbl>, Duration__in_seconds_ <dbl>,
## #   Finished <dbl+lbl>, RecordedDate <dttm>,
## #   ResponseId <chr>, DistributionChannel <chr>,
## #   UserLanguage <chr>, Consent <dbl+lbl>,
## #   Pol_7 <dbl+lbl>, W2_Knowledge <dbl+lbl>,
```

Arranging

Arranging allows us to sort the order of the table by a certain column

```r
arrange(ads_data, Duration__in_seconds_)
```

```
## # A tibble: 1,460 × 52
##    StartDate           EndDate             Status
##    <dttm>              <dttm>              <dbl+lbl>
##  1 2019-06-14 09:58:11 2019-06-14 09:59:01 0 [IP Add…
##  2 2019-06-14 09:44:24 2019-06-14 09:45:26 0 [IP Add…
##  3 2019-06-14 09:43:20 2019-06-14 09:44:30 0 [IP Add…
##  4 2019-06-14 09:44:00 2019-06-14 09:45:11 0 [IP Add…
##  5 2019-06-14 09:52:10 2019-06-14 09:53:26 0 [IP Add…
##  6 2019-06-14 09:45:57 2019-06-14 09:47:13 0 [IP Add…
##  7 2019-06-14 09:50:37 2019-06-14 09:51:53 0 [IP Add…
##  8 2019-06-14 09:45:49 2019-06-14 09:47:08 0 [IP Add…
##  9 2019-06-14 10:10:25 2019-06-14 10:11:45 0 [IP Add…
## 10 2019-06-14 09:53:33 2019-06-14 09:54:54 0 [IP Add…
## # … with 1,450 more rows, and 49 more variables:
## #   Progress <dbl>, Duration__in_seconds_ <dbl>,
## #   Finished <dbl+lbl>, RecordedDate <dttm>,
## #   ResponseId <chr>, DistributionChannel <chr>,
## #   UserLanguage <chr>, Consent <dbl+lbl>,
## #   Pol_7 <dbl+lbl>, W2_Knowledge <dbl+lbl>,
```

Selecting

Selecting allows us to pick certain columns

```
select(ads_data, RecordedDate)
```

```
## # A tibble: 1,460 × 1
##    RecordedDate
##    <dttm>
##  1 2019-06-14 09:44:31
##  2 2019-06-14 09:44:58
##  3 2019-06-14 09:44:59
##  4 2019-06-14 09:45:00
##  5 2019-06-14 09:45:01
##  6 2019-06-14 09:45:12
##  7 2019-06-14 09:45:12
##  8 2019-06-14 09:45:13
##  9 2019-06-14 09:45:13
## 10 2019-06-14 09:45:16
## # … with 1,450 more rows
```

Selecting

We can also remove columns

```
select(ads_data, -Consent, -DistributionChannel)
```

```
## # A tibble: 1,460 × 50
##    StartDate           EndDate             Status
##    <dttm>              <dttm>              <dbl+lbl>
##  1 2019-06-14 09:43:20 2019-06-14 09:44:30 0 [IP Add…
##  2 2019-06-14 09:43:11 2019-06-14 09:44:57 0 [IP Add…
##  3 2019-06-14 09:43:29 2019-06-14 09:44:58 0 [IP Add…
##  4 2019-06-14 09:43:10 2019-06-14 09:45:00 0 [IP Add…
##  5 2019-06-14 09:43:11 2019-06-14 09:45:00 0 [IP Add…
##  6 2019-06-14 09:44:00 2019-06-14 09:45:11 0 [IP Add…
##  7 2019-06-14 09:43:32 2019-06-14 09:45:12 0 [IP Add…
##  8 2019-06-14 09:43:27 2019-06-14 09:45:12 0 [IP Add…
##  9 2019-06-14 09:43:08 2019-06-14 09:45:13 0 [IP Add…
## 10 2019-06-14 09:43:36 2019-06-14 09:45:16 0 [IP Add…
## # … with 1,450 more rows, and 47 more variables:
## #   Progress <dbl>, Duration__in_seconds_ <dbl>,
## #   Finished <dbl+lbl>, RecordedDate <dttm>,
## #   ResponseId <chr>, UserLanguage <chr>,
## #   Pol_7 <dbl+lbl>, W2_Knowledge <dbl+lbl>,
## #   Gender <dbl+lbl>, Race <dbl+lbl>,
```

The pipe

So far, we have written our code like this:

```
filter(ads_data, Duration__in_seconds_ < 100)
```

```
## # A tibble: 41 × 52
##    StartDate           EndDate             Status
##    <dttm>              <dttm>              <dbl+lbl>
##  1 2019-06-14 09:43:20 2019-06-14 09:44:30 0 [IP Add…
##  2 2019-06-14 09:43:29 2019-06-14 09:44:58 0 [IP Add…
##  3 2019-06-14 09:44:00 2019-06-14 09:45:11 0 [IP Add…
##  4 2019-06-14 09:43:32 2019-06-14 09:45:12 0 [IP Add…
##  5 2019-06-14 09:43:48 2019-06-14 09:45:25 0 [IP Add…
##  6 2019-06-14 09:44:24 2019-06-14 09:45:26 0 [IP Add…
##  7 2019-06-14 09:43:50 2019-06-14 09:45:29 0 [IP Add…
##  8 2019-06-14 09:44:15 2019-06-14 09:45:42 0 [IP Add…
##  9 2019-06-14 09:44:30 2019-06-14 09:45:58 0 [IP Add…
## 10 2019-06-14 09:44:36 2019-06-14 09:46:05 0 [IP Add…
## # … with 31 more rows, and 49 more variables:
## #   Progress <dbl>, Duration__in_seconds_ <dbl>,
## #   Finished <dbl+lbl>, RecordedDate <dttm>,
## #   ResponseId <chr>, DistributionChannel <chr>,
## #   UserLanguage <chr>, Consent <dbl+lbl>,
## #   Pol_7 <dbl+lbl>, W2_Knowledge <dbl+lbl>,
```

The pipe

We can use the pipe %>%, which passes what we wrote on the previous line into the next function as the first argument:

```
ads_data %>%
  filter(Duration__in_seconds_ < 100) %>%
  arrange(Duration__in_seconds_) %>%
  select(RecordedDate, Duration__in_seconds_)
```

```
## # A tibble: 41 × 2
##    RecordedDate        Duration__in_seconds_
##    <dttm>                              <dbl>
##  1 2019-06-14 09:59:02                    50
##  2 2019-06-14 09:45:26                    61
##  3 2019-06-14 09:44:31                    70
##  4 2019-06-14 09:45:12                    70
##  5 2019-06-14 09:53:26                    75
##  6 2019-06-14 09:47:13                    76
##  7 2019-06-14 09:51:54                    76
##  8 2019-06-14 09:47:08                    78
##  9 2019-06-14 10:11:46                    79
## 10 2019-06-14 09:54:54                    80
## # … with 31 more rows
```

The pipe

```
ads_data %>%
  filter(Duration__in_seconds_ < 100) %>%
  arrange(Duration__in_seconds_) %>%
  select(RecordedDate, Duration__in_seconds_)
```

You can think of this like:

- Take the ADS data
- Filter so we only have the rows where the survey duration is less than 100 seconds
- Arrange so we go from lowest duration to highest
- Select only the date recorded and the duration

Mutating

Mutating can be used to create new columns or change existing columns.

```
ads_data <- ads_data %>%
  mutate(Birthyear_add_day = str_c(Birthyear, "07-01")) %>%
  mutate(Birthyear_add_day = as_datetime(Birthyear_add_day))
```

```
## # A tibble: 1,460 × 3
##    EndDate             Birthyear Birthyear_add_day
##    <dttm>                  <dbl> <dttm>
##  1 2019-06-14 09:44:30      1993 1993-07-01 00:00:00
##  2 2019-06-14 09:44:57      1978 1978-07-01 00:00:00
##  3 2019-06-14 09:44:58      1993 1993-07-01 00:00:00
##  4 2019-06-14 09:45:00      1983 1983-07-01 00:00:00
##  5 2019-06-14 09:45:00      1990 1990-07-01 00:00:00
##  6 2019-06-14 09:45:11      1980 1980-07-01 00:00:00
##  7 2019-06-14 09:45:12      1996 1996-07-01 00:00:00
##  8 2019-06-14 09:45:12      1986 1986-07-01 00:00:00
##  9 2019-06-14 09:45:13      2000 2000-07-01 00:00:00
## 10 2019-06-14 09:45:16      1988 1988-07-01 00:00:00
## # … with 1,450 more rows
```

Mutating

```r
ads_data %>%
  mutate(age = EndDate - Birthyear_add_day)
```

```
## # A tibble: 1,460 × 4
##    EndDate             Birthyear Birthyear_add_day  
##    <dttm>                  <dbl> <dttm>             
##  1 2019-06-14 09:44:30      1993 1993-07-01 00:00:00
##  2 2019-06-14 09:44:57      1978 1978-07-01 00:00:00
##  3 2019-06-14 09:44:58      1993 1993-07-01 00:00:00
##  4 2019-06-14 09:45:00      1983 1983-07-01 00:00:00
##  5 2019-06-14 09:45:00      1990 1990-07-01 00:00:00
##  6 2019-06-14 09:45:11      1980 1980-07-01 00:00:00
##  7 2019-06-14 09:45:12      1996 1996-07-01 00:00:00
##  8 2019-06-14 09:45:12      1986 1986-07-01 00:00:00
##  9 2019-06-14 09:45:13      2000 2000-07-01 00:00:00
## 10 2019-06-14 09:45:16      1988 1988-07-01 00:00:00
## # … with 1,450 more rows, and 1 more variable:
## #   age <drtn>
```

Summary

```
summary(ads_data)
```

```
##    StartDate
##  Min.   :2019-06-14 09:43:03.00
##  1st Qu.:2019-06-14 09:46:47.50
##  Median :2019-06-14 09:52:50.00
##  Mean   :2019-06-14 09:57:40.11
##  3rd Qu.:2019-06-14 10:06:28.25
##  Max.   :2019-06-14 11:19:45.00
##
##     EndDate                           Status
##  Min.   :2019-06-14 09:44:30.00   Min.   :0
##  1st Qu.:2019-06-14 09:51:29.00   1st Qu.:0
##  Median :2019-06-14 09:57:57.00   Median :0
##  Mean   :2019-06-14 10:02:23.89   Mean   :0
##  3rd Qu.:2019-06-14 10:11:19.50   3rd Qu.:0
##  Max.   :2019-06-14 11:27:10.00   Max.   :0
##
##     Progress   Duration__in_seconds_    Finished
##  Min.   :100   Min.   :  50.0         Min.   :1
##  1st Qu.:100   1st Qu.: 178.0         1st Qu.:1
##  Median :100   Median : 237.0         Median :1
##  Mean   :100   Mean   : 283.3         Mean   :1
```

Pulling a variable for calculations

```
ads_data %>%
  pull(Duration__in_seconds_)
```

```
##    [1]   70  105   88  109  109   70   99  105  124
##   [10]  100   96  102   61   98  120   86  119  120
##   [19]  143  115  131  164  140  126   88  127  146
##   [28]   88  134  163  111  164  123  176  102  119
##   [37]  187  179  140  144  183  139  123  162  152
##   [46]  184  160  181  163  168  101  190  178  144
##   [55]  194  123  133  135  185  121  163  192  210
##   [64]  167  139  204  117  170  170  199   95  126
##   [73]  208  178  207  146  118  170  110  172  226
##   [82]   78  160  185  186  222  212  185  168  213
##   [91]   76  213  165  173  218  207  214  203  206
##  [100]  213  228  186  240  248  208  176  217  142
##  [109]  190  215  247  163  239  251  185  176  217
##  [118]  193  171  159  239  252  178  168  101  213
##  [127]  227  122  217  225  239  182  178  165  248
##  [136]  190  272  222  101  173  270  121  191  275
##  [145]  210  227  283  188  194  275  236  169  151
##  [154]  295  262  257  234  119  287  276  264  286
```

Using the pulled variable for descriptive statistics

Median

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
  median(na.rm = TRUE)
```

```
## [1] 237
```

We have to tell the mean() function to disregard NAs by writing `na.rm = TRUE`

Using the pulled variable for descriptive statistics

Mean

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
  mean(na.rm = TRUE)
```

```
## [1] 283.261
```

Using the pulled variable for descriptive statistics

Range can be calculated using the `range()` function.

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
  range(na.rm = TRUE)
```

```
## [1]   50 1575
```

Variance can be calculated using the `var()` function.

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
  var(na.rm = TRUE)
```

```
## [1] 29487.81
```

Using the pulled variable for descriptive statistics

Standard Deviation can be calculated using the `sd()` function.

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
  sd(na.rm = TRUE)
```

```
## [1] 171.7202
```

Summarise

```
ads_data %>%
  summarise(mean_time = mean(Duration__in_seconds_, na.rm = TRUE),
            sd_time = sd(Duration__in_seconds_, na.rm = TRUE))
```

```
## # A tibble: 1 × 2
##   mean_time sd_time
##       <dbl>   <dbl>
## 1      283.    172.
```

Grouping

Before summarising, we can group by a categorical variable

```
ads_data %>%
  group_by(Gender) %>%
  summarise(count = n(),
            mean_time = mean(Duration__in_seconds_, na.rm = TRUE),
            sd_time = sd(Duration__in_seconds_, na.rm = TRUE))
```

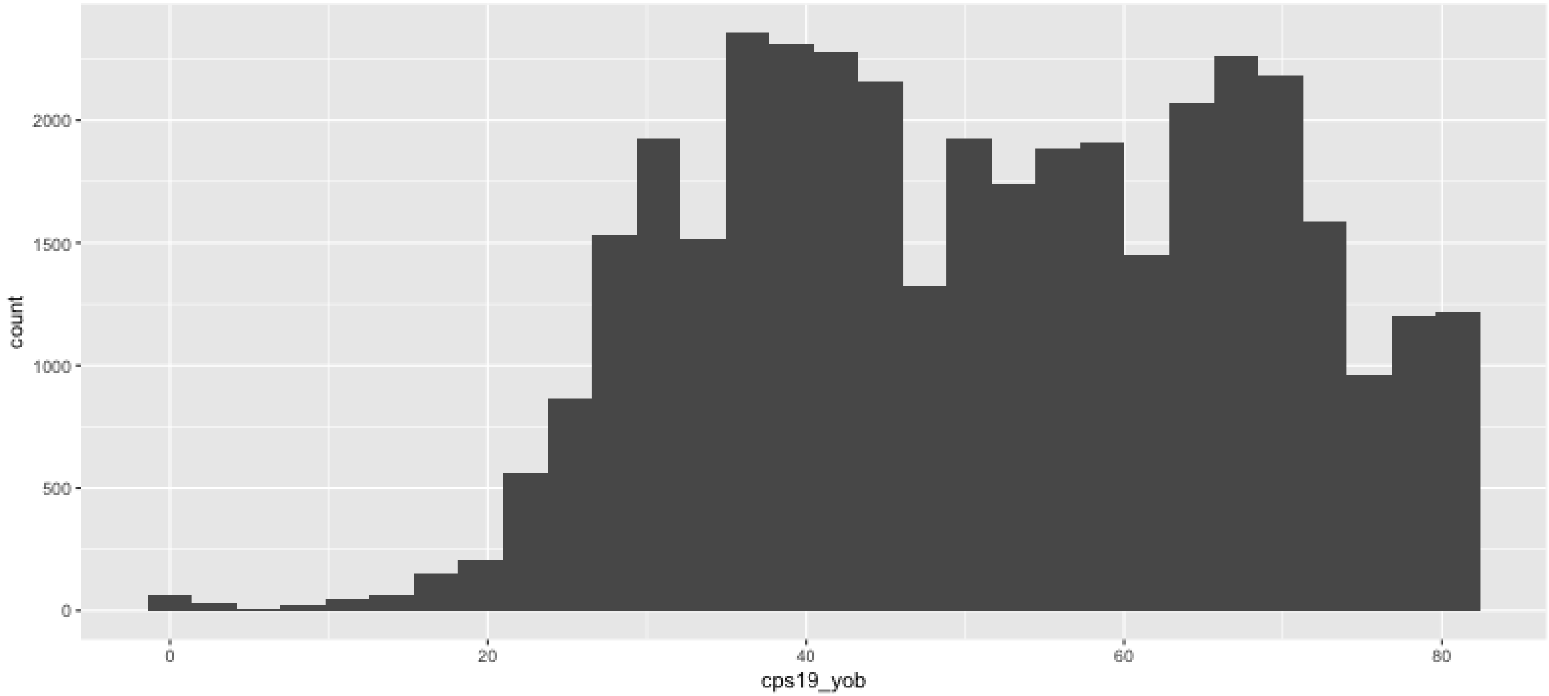
```
## # A tibble: 3 × 4
##   Gender                               count mean_…¹ sd_time
##   <dbl+lbl>                            <int>   <dbl>   <dbl>
## 1 1 [Male]                               758    269.    162.
## 2 2 [Female]                             698    299.    181.
## 3 3 [Prefer a third option/Oth…            4    229      37.7
## # … with abbreviated variable name ¹mean_time
```

Manipulation application: data cleaning

Data cleaning

Graphing year of birth shows that it goes from 1 to about 80.

```
ces_2019_raw %>%
  ggplot(aes(x = cps19_yob)) +
  geom_histogram()
```



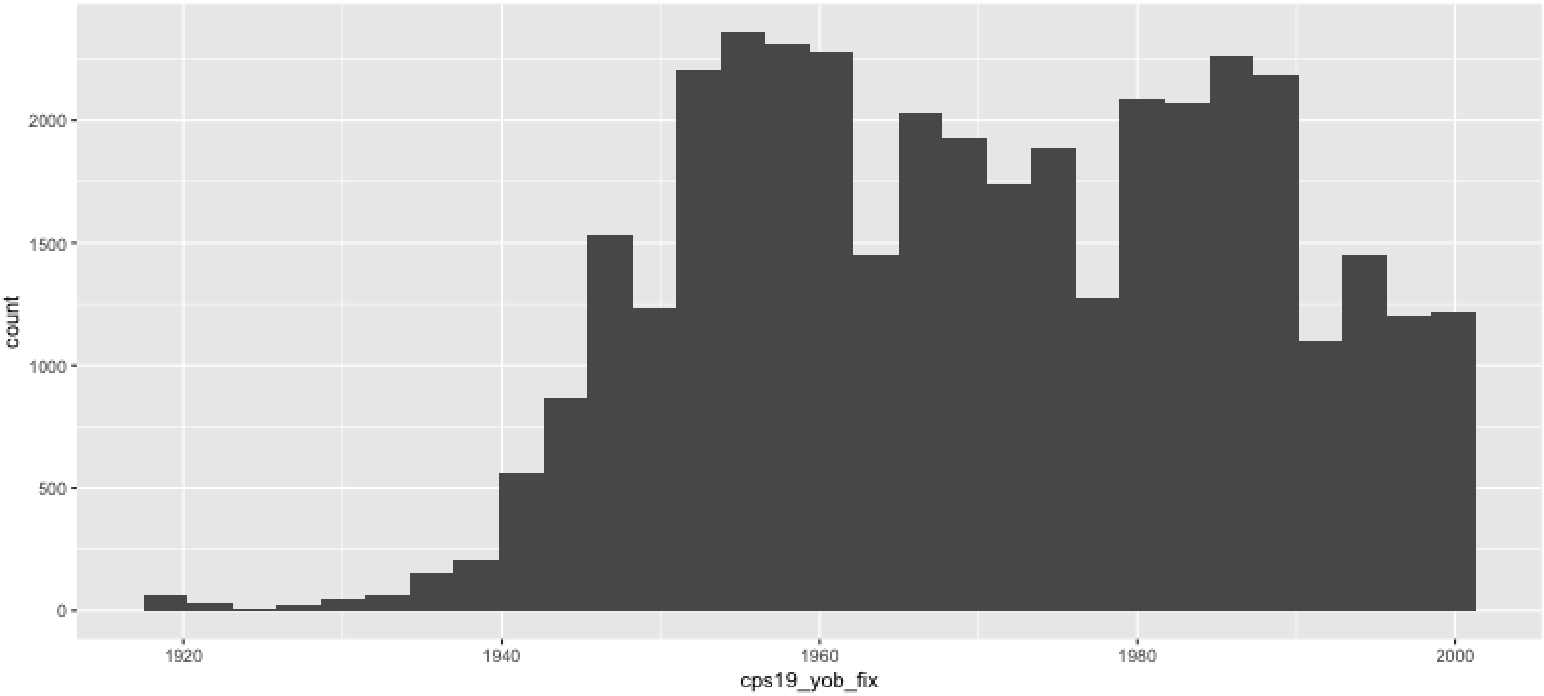
Data cleaning

The codebook says that a value of 1 corresponds to a birth year of 1920, value of 2 to a birth year of 1921, and so on. We can create a new variable that reads more intuitively.

```
CES_data <- ces_2019_raw %>%
  mutate(cps19_yob_fix = cps19_yob + 1919)
```

Data cleaning

```
CES_data %>%
  ggplot(aes(x = cps19_yob_fix)) +
  geom_histogram()
```



Better!

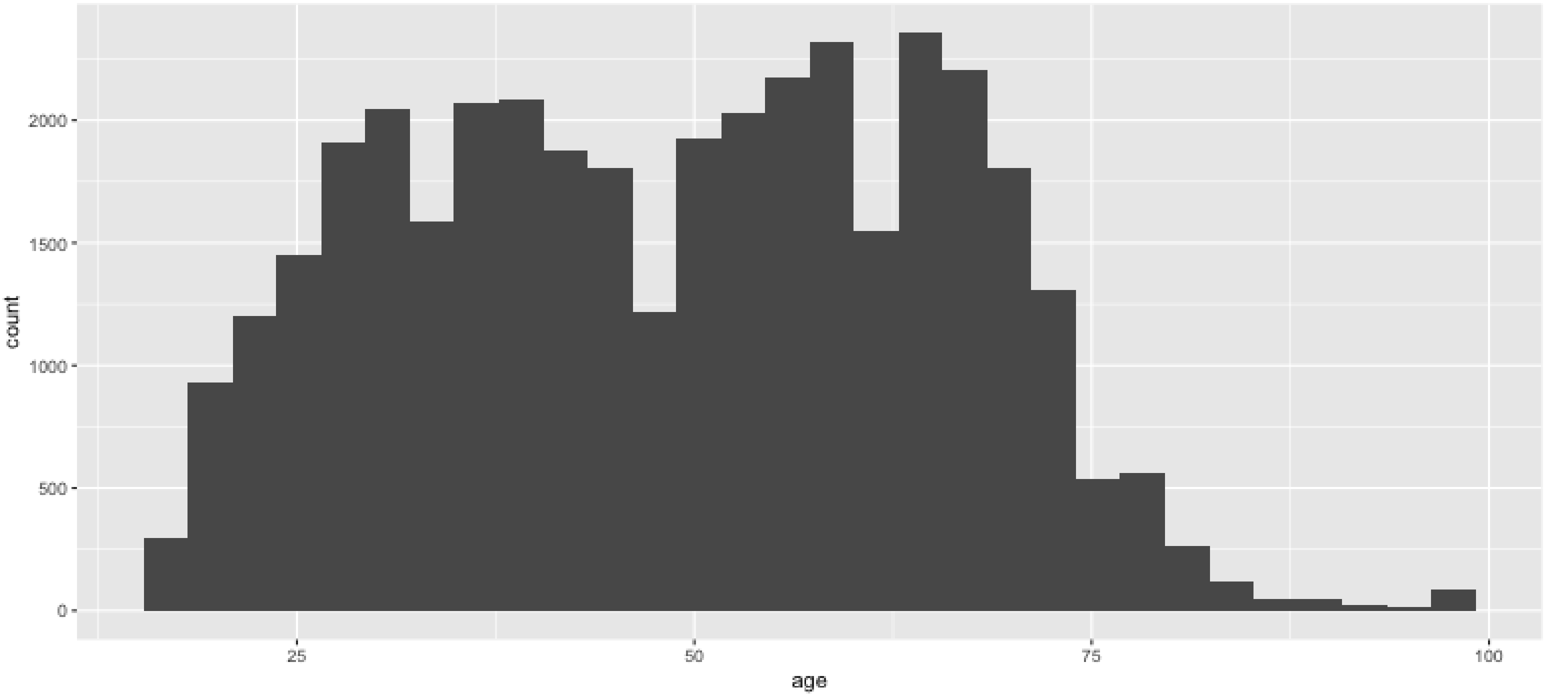
Add a variable for age

Now that we have an accurate birth year, maybe we would like to have the age of the individual as well.

```r
CES_data <- CES_data %>%
  mutate(age = 2019 - cps19_yob_fix)
```

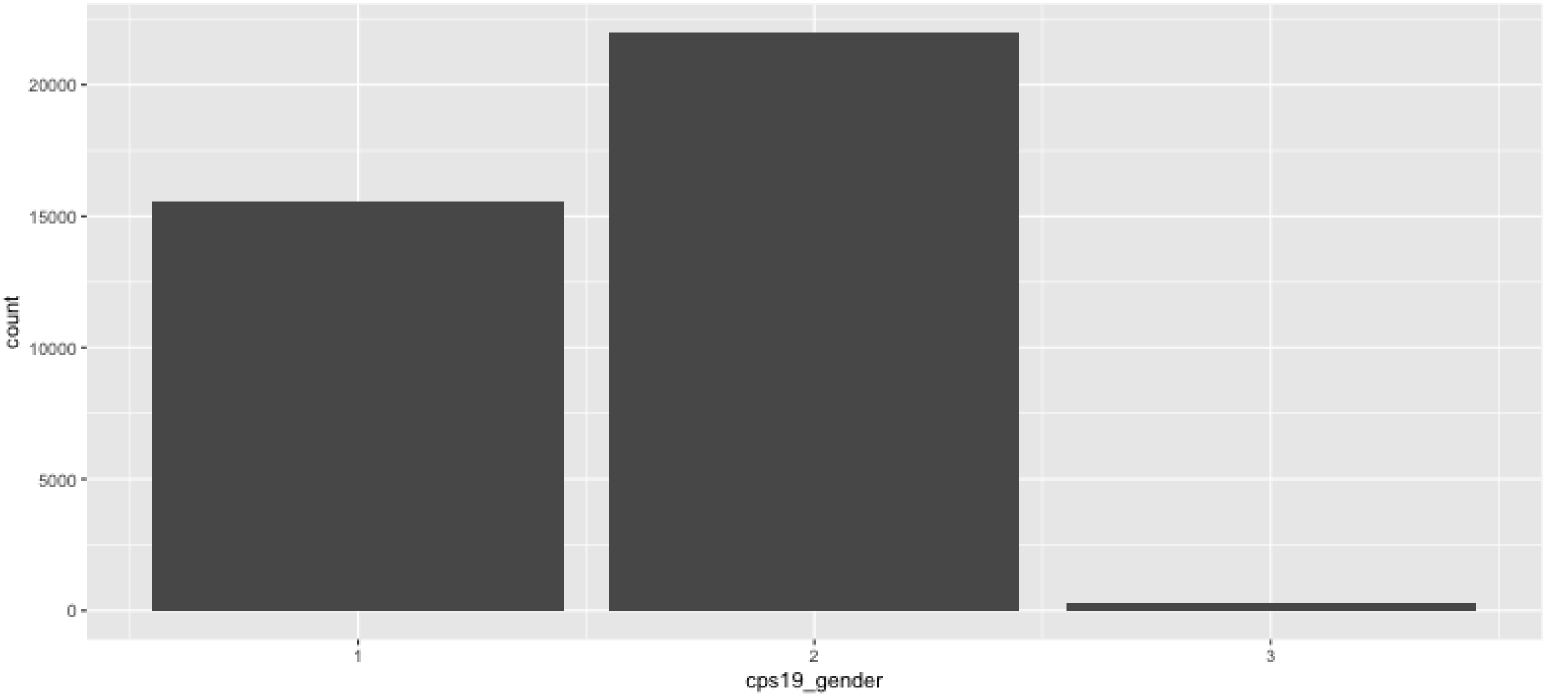
Add a variable for age

```
CES_data %>%
  ggplot(aes(x = age)) +
  geom_histogram()
```



Recoding the gender variable

```
CES_data %>%
  ggplot(aes(x = cps19_gender)) +
  geom_bar()
```

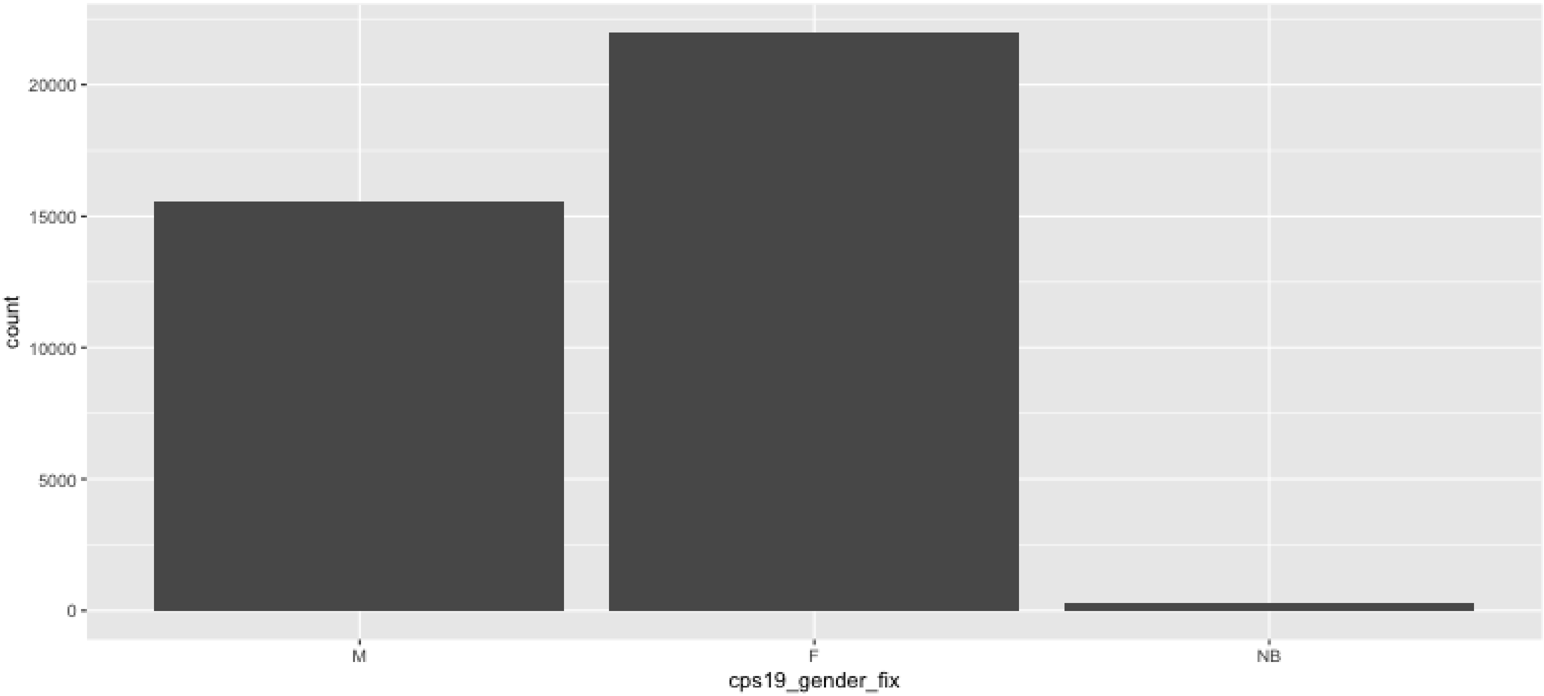


Recoding the gender variable

```
CES_data <- CES_data %>%
  mutate(cps19_gender_fix = factor(cps19_gender)) %>%
  mutate(cps19_gender_fix = fct_recode(cps19_gender_fix,
                                       "M" = "1",
                                       "F" = "2",
                                       "NB" = "3"))
```

Recoding the gender variable

```
CES_data %>%
  ggplot(aes(x = cps19_gender_fix)) +
  geom_bar()
```



Fixing household counts

```
CES_data %>%
  filter(cps19_household > 10) %>%
  arrange(-cps19_household) %>%
  pull(cps19_household)
```

```
##  [1] 7766666   72000   50000   20000   10000    5667
##  [7]    2000     501     321      99      89      87
## [13]      69      54      54      50      44      40
## [19]      34      33      29      27      23      22
## [25]      22      20      20      20      15      15
## [31]      13      13      12      12      12      11
## [37]      11      11      11      11      11      11
## [43]      11
```

Fixing household counts

```
CES_data <- CES_data %>%
  mutate(cps19_household = ifelse(cps19_household > 15,
                                  NA,
                                  cps19_household))

CES_data %>%
  filter(cps19_household > 10) %>%
  pull(cps19_household)
```

```
##  [1] 12 11 15 12 11 13 11 11 11 15 13 12 11 11 11
```

Fixing income

```
CES_data %>%
  filter(cps19_income_number > 1000000) %>%
  arrange(-cps19_income_number) %>%
  pull(cps19_income_number)
```

```
##  [1] 6.747658e+60 1.000000e+21 1.000000e+15
##  [4] 8.769655e+10 8.889899e+09 3.062936e+09
##  [7] 1.000000e+09 1.000000e+09 6.788765e+08
## [10] 3.000000e+08 7.245600e+07 3.454534e+07
## [13] 3.000000e+07 1.000000e+07 9.999999e+06
## [16] 8.900000e+06 7.696588e+06 7.440000e+06
## [19] 6.848382e+06 6.787145e+06 6.782800e+06
## [22] 6.500100e+06 4.500000e+06 3.000000e+06
## [25] 2.332100e+06 2.000000e+06 2.000000e+06
## [28] 1.872717e+06 1.800000e+06 1.650000e+06
## [31] 1.500000e+06 1.500000e+06 1.450000e+06
## [34] 1.300000e+06 1.290000e+06 1.250000e+06
## [37] 1.250000e+06 1.250000e+06 1.150000e+06
```

Fixing income

```
CES_data <- CES_data %>%
  mutate(cps19_income_number = ifelse(cps19_income_number >= 1000000000
                                  NA,
                                  cps19_income_number))

CES_data %>%
  filter(cps19_income_number > 1000000) %>%
  pull(cps19_income_number)
```

```
##  [1]   2000000   1500000   4500000   3000000
##  [5]   6848382   7696588   6787145   1250000
##  [9]   1650000   1872717 678876545   1300000
## [13]   1150000   1250000   9999999   1450000
## [17]   1500000   6500100  30000000   8900000
## [21] 300000000   7440000   6782800   2332100
## [25]   1800000   2000000  10000000   1290000
## [29]  72456000  34545345   1250000
```

Manipulation application: Summarising data

Summarising data

First we can select only data for Ontario using `filter()`:

```
CES_data %>%
  filter(cps19_province == "Ontario")
```

```
## # A tibble: 14,160 × 620
##    cps19_StartDate     cps19_EndDate       cps19_Re…¹
##    <dttm>              <dttm>              <chr>
##  1 2019-09-13 10:01:19 2019-09-13 10:27:29 R_USWDAPc…
##  2 2019-09-13 10:05:37 2019-09-13 10:50:53 R_3IQaeDX…
##  3 2019-09-13 10:05:52 2019-09-13 10:32:53 R_27WeMQ1…
##  4 2019-09-13 10:10:20 2019-09-13 10:29:45 R_3LiGZcC…
##  5 2019-09-13 10:14:47 2019-09-13 10:32:32 R_1Iu8R1U…
##  6 2019-09-13 10:15:39 2019-09-13 10:30:59 R_2EcS26h…
##  7 2019-09-13 10:15:48 2019-09-13 10:37:45 R_3yrt44w…
##  8 2019-09-13 10:16:08 2019-09-13 10:40:14 R_10OBmXJ…
##  9 2019-09-13 10:16:24 2019-09-13 10:41:24 R_2e5nvu0…
## 10 2019-09-13 10:17:06 2019-09-13 10:35:47 R_2OJdv16…
## # … with 14,150 more rows, 617 more variables:
## #   cps19_consent <dbl>, cps19_citizenship <dbl>,
## #   cps19_yob <dbl>, cps19_yob_2001_age <dbl>,
## #   cps19_gender <fct>, cps19_province <fct>,
## #   cps19_education <dbl>, cps19_demsat <dbl>,
```

Summarising data

We don't need to be dealing with all the columns. We can specifically select the ones we want using `select()`:

"How satisfied are you with the performance of your provincial government under ${e://Field/premier}?", "In provincial politics, do you usually think of yourself as a:", and income.

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number)
```

```
## # A tibble: 14,160 × 3
##    cps19_prov_gov_sat   cps19_prov_id            cps19…¹
##    <fct>                <fct>                      <dbl>
##  1 Not very satisfied   Liberal                       NA
##  2 Fairly satisfied     Progressive Conserva…         NA
##  3 Fairly satisfied     Liberal                    56000
##  4 Not at all satisfied NDP                           NA
##  5 Not at all satisfied NDP                            0
##  6 Not at all satisfied None                          NA
##  7 Not at all satisfied NDP                           NA
```

Summarising data

Now that our data looks like what we would like it to, we can start creating a summary table. Since we have the income for each participant, we can look at median incomes. We also want to know how many participants are in each category.

First, we can group the data by provincial political self-ID. To do this, we use `group_by()` to group the data and `summarise()` to produce values for each group we have created. We will start with calculating the `median()` for the incomes. We can add multiple arguments to the `summarise()` argument. `n()` adds a count for each group.

Summarising data

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number) %>%
  group_by(cps19_prov_gov_sat) %>%
  summarise(median_income = median(cps19_income_number,
                                   na.rm = TRUE),
            count = n())
```

```
## # A tibble: 5 × 3
##   cps19_prov_gov_sat              median_income count
##   <fct>                                   <dbl> <int>
## 1 Very satisfied                          80000   872
## 2 Fairly satisfied                        80000  2738
## 3 Not very satisfied                      75000  3212
## 4 Not at all satisfied                    72000  6853
## 5 Don't know/prefer not to answer         50000   485
```

Grouping

In our table, the satisfaction ratings are ordered alphabetically. We would like them to be ordered logically. We can do this by ordering the factor variable.

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number) %>%
  mutate(cps19_prov_id = factor(cps19_prov_id,
                                     levels = c("Liberal",
                                                "Progressive Conservati
                                                "NDP",
                                                "Green",
                                                "Another party",
                                                "None",
                                                "Don't know/prefer not 
```

Grouping

```
## # A tibble: 14,160 × 3
##    cps19_prov_gov_sat   cps19_prov_id         cps19…¹
##    <fct>                <fct>                   <dbl>
##  1 Not very satisfied   Liberal                    NA
##  2 Fairly satisfied     Progressive Conserva…      NA
##  3 Fairly satisfied     Liberal                 56000
##  4 Not at all satisfied NDP                        NA
##  5 Not at all satisfied NDP                         0
##  6 Not at all satisfied None                       NA
##  7 Not at all satisfied NDP                        NA
##  8 Not very satisfied   Liberal                    NA
##  9 Not very satisfied   NDP                        NA
## 10 Not at all satisfied Liberal                    NA
## # … with 14,150 more rows, and abbreviated variable
## #   name ¹cps19_income_number
```

Grouping

And combine this with our table from before:

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number) %>%
  mutate(cps19_prov_gov_sat = factor(cps19_prov_gov_sat,
                                     levels = c("Not at all satisfied",
                                                "Not very satisfied",
                                                "Fairly satisfied",
                                                "Very satisfied",
                                                "Don't know/prefer not t

  group_by(cps19_prov_gov_sat) %>%
  summarise(median_income = median(cps19_income_number,
                                   na.rm = TRUE),
            count = n())
```

```
## # A tibble: 5 × 3
##   cps19_prov_gov_sat              median_income count
##   <fct>                                   <dbl> <int>
## 1 Not at all satisfied                    72000  6853
## 2 Not very satisfied                      75000  3212
```

Grouping

What happens if we group by political identification instead?

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number) %>%
  group_by(cps19_prov_id) %>%
  summarise(median_income = median(cps19_income_number,
                                   na.rm = TRUE),
            count = n())
```

```
## # A tibble: 7 × 3
##   cps19_prov_id                  median_income count
##   <fct>                                  <dbl> <int>
## 1 Liberal                                80000  4607
## 2 NDP                                    65000  2413
## 3 Green                                  60000   812
## 4 Progressive Conservative               80000  3629
## 5 Another party                          50000    90
## 6 None                                   68000  1367
## 7 Don't know/prefer not to answer        60000  1242
```

Grouping

We could order the parties in a way that makes more sense:

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number) %>%
  mutate(cps19_prov_id = factor(cps19_prov_id,
                                    levels = c("Liberal",
                                               "Progressive Conservati
                                               "NDP",
                                               "Green",
                                               "Another party",
                                               "None",
                                               "Don't know/prefer not t
  group_by(cps19_prov_id) %>%
  summarise(median_income = median(cps19_income_number,
                                  na.rm = TRUE),
            count = n())
```

Grouping

```
## # A tibble: 7 × 3
##   cps19_prov_id                   median_income count
##   <fct>                                   <dbl> <int>
## 1 Liberal                                 80000  4607
## 2 Progressive Conservative                80000  3629
## 3 NDP                                     65000  2413
## 4 Green                                   60000   812
## 5 Another party                           50000    90
## 6 None                                    68000  1367
## 7 Don't know/prefer not to answer         60000  1242
```

Grouping

Or we could sort by median income. We can do that using `arrange()`:

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number) %>%
  group_by(cps19_prov_id) %>%
  summarise(median_income = median(cps19_income_number,
                                   na.rm = TRUE),
            count = n()) %>%
  arrange(-median_income)
```

```
## # A tibble: 7 × 3
##   cps19_prov_id                    median_income count
##   <fct>                                    <dbl> <int>
## 1 Liberal                                  80000  4607
## 2 Progressive Conservative                 80000  3629
## 3 None                                     68000  1367
## 4 NDP                                      65000  2413
## 5 Green                                    60000   812
## 6 Don't know/prefer not to answer          60000  1242
## 7 Another party                            50000    90
```

Grouping

`group_by()` can also have multiple arguments, so we can group by `cps19_prov_gov_sat` and `cps19_prov_id` at the same time:

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number) %>%
  mutate(cps19_prov_id = factor(cps19_prov_id,
                                     levels = c("Liberal",
                                                "Progressive Conservativ
                                                "NDP",
                                                "Green",
                                                "Another party",
                                                "None",
                                                "Don't know/prefer not t
  mutate(cps19_prov_gov_sat = factor(cps19_prov_gov_sat,
                                     levels = c("Not at all satisfied",
                                                "Not very satisfied",
                                                "Fairly satisfied",
                                                "Very satisfied",
                                                "Don't know/prefer not t
  group_by(cps19_prov_gov_sat, cps19_prov_id) %>%
```

Grouping

This table is less easy to read, though. `spread()` can make a table that is wide rather than long. We specify the `key`, the variable that will become our column names, and the `value`, which will become the values in those columns:

```
CES_data %>%
  filter(cps19_province == "Ontario") %>%
  select(cps19_prov_gov_sat,
         cps19_prov_id,
         cps19_income_number) %>%
  mutate(cps19_prov_id = factor(cps19_prov_id,
                                     levels = c("Liberal",
                                                "Progressive Conservati
                                                "NDP",
                                                "Green",
                                                "Another party",
                                                "None",
                                                "Don't know/prefer not t
  mutate(cps19_prov_gov_sat = factor(cps19_prov_gov_sat,
                                     levels = c("Not at all satisfied",
                                                "Not very satisfied",
                                                "Fairly satisfied",
                                                "Very satisfied",
                                                "Don't know/prefer not t
```

Grouping

```
## # A tibble: 7 × 6
##   cps19_pro…¹ Not a…² Not v…³ Fairl…⁴ Very …⁵ Don't…⁶
##   <fct>         <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Liberal       80000   80000   79999   79876   60000
## 2 Progressiv…   85000   78000   82000   84000   72000
## 3 NDP           65000   65000   76888   80000   37000
## 4 Green         60000   60000   72750   66000   32500
## 5 Another pa…   40000   48500   73500  150000   52000
## 6 None          62000   74000   69000   66000   43000
## 7 Don't know…   68500   59500   70000   85000   50000
## # … with abbreviated variable names ¹cps19_prov_id,
## #   ²`Not at all satisfied`, ³`Not very satisfied`,
## #   ⁴`Fairly satisfied`, ⁵`Very satisfied`,
## #   ⁶`Don't know/prefer not to answer`
```

Exercises

Exercises

1. Filter the rows in the CES_data dataset where the survey-taker is between 30 and 50 (cps19_age).
2. Filter the rows in the CES_data dataset where the survey-taker answered the cps19_votechoice question (i.e. the cps19_votechoice variable is not NA).
3. Select the variables cps19_age and cps19_province from the CES_data dataset.
4. Select all variables except cps19_province from the CES_data dataset.

Exercises

1. Create a variable in the dataset CES_data that states if a person consumes news content or not (i.e. cps19_news_cons is equal to "0 minutes" or it is not).
2. Modify the variable cps19_income_number in the dataset CES_data so that it is measured in thousands (i.e. divide the income number by 1000).

Exercises

1. Use the CES_data dataset. Group by cps19_votechoice. Find both the median and mean rating of Trudeau (cps19_lead_rating_23):
2. Use the CES_data dataset. Group by cps19_imm and cps19_spend_educ. Find the count for each group.

Exercises

- 1 - Fix this error:

```
CES_data %>%
  summarise(mean = mean(cps19_age)) %>%
  group_by(cps19_gender)
```

- 2 - Fix this error:

```
CES_data %>%
  filter(cps19_vote_choice == "Green Party")
```

Exercises

- 3 - Fix this error:

```
CES_data %>%
  mutate(cps19_fed_donate = factor(cps19_fed_donate,
                                     levels = c("Yes",
                                                "No",
                                                "Don't know/ Prefer not
```

- 4 - Fix this error:

```
CES_data %>%
  select(cps19_province
         cps19_age
         cps19_gender)
```

Any questions?