4.4 Introduction to R: Manipulation

Julia Gallucci

Data Science Institute, University of Toronto

2023-06-03

Acknowledgements

Slides are adapted from Anjali Silva, originally from Amy Farrow under the supervision of Rohan Alexander, University of Toronto. Slides have been modified by Julia Gallucci, 2023.

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()` is like a transposed version of `print()`: columns run down the page, and data runs across. This makes it possible to see every column in a data frame.

Take a look

```
glimpse(ads_data)
```

```
Rows: 1,460
Columns: 52
$ StartDate <dttm> 2019-06-14 09:43:20,
#2019-06-14 09:43:11, 2019-~
$ EndDate <dttm> 2019-06-14 09:44:30, 2019-06-14
#09:44:57, 2019-~
$ Status <dbl+lbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
#0, 0, 0, 0, 0,~
$ Progress <dbl> 100, 100, 100, 100, 100, 100,
#100, 100, 100, 100~
$ Duration__in_seconds_ <dbl> 70, 105, 88, 109,
#109, 70, 99, 105, 124, 100, 96~
$ Finished <dbl+lbl> 1, 1, 1, 1, 1, 1, 1, 1, 1,
#1, 1, 1, 1, 1, 1,~
$ RecordedDate <dttm> 2019-06-14 09:44:31,
#2019-06-14 09:44:58, 2019-~
$ ResponseId <chr> "R_11dq3s9btLX57LD",
#"R_DRWZdBOugPUKqGt", "R_3QD~
$ DistributionChannel <chr> "anonymous",
```

Filtering

`filter()` is used to subset a data frame, retaining only rows that satisfy your conditions.

To be retained, the row must produce a value of `TRUE` for the condition(s).

Filtering

For example, lets say we want to filter the data to only view rows with duration in seconds less than 100.

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

```
# A tibble: 41 x 52
StartDate EndDate Status Progress
<dttm> <dttm> <dbl+lbl> <dbl>
1 2019-06-14 09:43:20 2019-06-14 09:44:30 0 [IP
#Address] 100
2 2019-06-14 09:43:29 2019-06-14 09:44:58 0 [IP
#Address] 100
3 2019-06-14 09:44:00 2019-06-14 09:45:11 0 [IP
#Address] 100
4 2019-06-14 09:43:32 2019-06-14 09:45:12 0 [IP
#Address] 100
5 2019-06-14 09:43:48 2019-06-14 09:45:25 0 [IP
#Address] 100
6 2019-06-14 09:44:24 2019-06-14 09:45:26 0 [IP
#Address] 100
7 2019-06-14 09:43:50 2019-06-14 09:45:29 0 [IP
```

Arranging

`arrange()` orders the rows of a data frame by the values of selected columns

Arranging

For example, lets say we want to arrange the data to view rows with duration in seconds ascending.

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

```
# A tibble: 1,460 x 52
StartDate EndDate Status Progress
<dttm> <dttm> <dbl+lbl> <dbl>
1 2019-06-14 09:58:11 2019-06-14 09:59:01 0 [IP
#Address] 100
2 2019-06-14 09:44:24 2019-06-14 09:45:26 0 [IP
#Address] 100
3 2019-06-14 09:43:20 2019-06-14 09:44:30 0 [IP
#Address] 100
4 2019-06-14 09:44:00 2019-06-14 09:45:11 0 [IP
#Address] 100
5 2019-06-14 09:52:10 2019-06-14 09:53:26 0 [IP
#Address] 100
6 2019-06-14 09:45:57 2019-06-14 09:47:13 0 [IP
#Address] 100
7 2019-06-14 09:50:37 2019-06-14 09:51:53 0 [IP
```

Selecting

`select()` allows us to pick certain columns from a dataframe, using their names

`select()` allows us to remove columns from a dataframe, using their names

Selecting

For example, lets say we want to select the column "RecordedDate" from the data set.

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

```
# A tibble: 1,460 x 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
# i 1,450 more rows
```

Selecting

For example, lets say we want to remove the columns "Consent" and "Distribution Channel" from the data set.

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

```
# A tibble: 1,460 x 50
StartDate EndDate Status Progress
<dttm> <dttm> <dbl+lbl> <dbl>
1 2019-06-14 09:43:20 2019-06-14 09:44:30 0 [IP
#Address] 100
2 2019-06-14 09:43:11 2019-06-14 09:44:57 0 [IP
#Address] 100
3 2019-06-14 09:43:29 2019-06-14 09:44:58 0 [IP
#Address] 100
4 2019-06-14 09:43:10 2019-06-14 09:45:00 0 [IP
#Address] 100
5 2019-06-14 09:43:11 2019-06-14 09:45:00 0 [IP
#Address] 100
6 2019-06-14 09:44:00 2019-06-14 09:45:11 0 [IP
#Address] 100
7 2019-06-14 09:43:32 2019-06-14 09:45:12 0 [IP
```

The pipe

So far, we have written our code like this:

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

But what if we want to perform multiple operations in one go?

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) %>%
#filter the data set to only view rows with
#duration in seconds less than 100
  arrange(Duration__in_seconds_) %>%
#arrange the data set to view rows with duration
#in seconds ascending
  select(RecordedDate, Duration__in_seconds_)
#select the column "RecordedDate" and "Duration
#in seconds" from the data set
```

The pipe

```
# A tibble: 41 x 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
# i 31 more rows
```

Mutating

`mutate()` creates new columns that are functions of existing variables.

It can also modify (if the name is the same as an existing column)

Mutating

For example, lets say we want to create a new column named "Birthyear_add_day" that takes the column Birthyear and joins it with a provided string ("07-01").

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

#modify the data to convert to date-time format
```

Mutating

```
# A tibble: 1,460 x 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
# i 1,450 more rows
```

Mutating

For example, lets say we want to create a new column named "age" that takes the column EndDate and subtracts the column "Birthyear_add_day"

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

Mutating

```
# A tibble: 1,460 x 4
EndDate Birthyear Birthyear_add_day age
<dttm> <dbl> <dttm> <drtn>
1 2019-06-14 09:44:30 1993 1993-07-01 00:00:00
#9479.406 days
2 2019-06-14 09:44:57 1978 1978-07-01 00:00:00
#14958.406 days
3 2019-06-14 09:44:58 1993 1993-07-01 00:00:00
#9479.406 days
4 2019-06-14 09:45:00 1983 1983-07-01 00:00:00
#13132.406 days
5 2019-06-14 09:45:00 1990 1990-07-01 00:00:00
#10575.406 days
6 2019-06-14 09:45:11 1980 1980-07-01 00:00:00
#14227.406 days
7 2019-06-14 09:45:12 1996 1996-07-01 00:00:00
#8383.406 days
8 2019-06-14 09:45:12 1986 1986-07-01 00:00:00
#12036.406 days
9 2019-06-14 09:45:13 2000 2000-07-01 00:00:00
#6922.406 days
```

Summary

summary is a function used to provide a concise overview of the central tendency, dispersion, and distribution of the data.

```
summary(ads_data)
```

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

    Progress   Duration__in_seconds_     Finished
 Min.    :100   Min.    :  50.0        Min.    :1
```

Pulling a variable for calculations

`pull()` is similar to $, used to extract a single column from a data frame as a vector.

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

```
[1] 70 105 88 109 109 70 99 105 124 100 96 102 61
#98
[15] 120 86 119 120 143 115 131 164 140 126 88
#127 146 88
[29] 134 163 111 164 123 176 102 119 187 179 140
#144 183 139
[43] 123 162 152 184 160 181 163 168 101 190 178
#144 194 123
[57] 133 135 185 121 163 192 210 167 139 204 117
#170 170 199
[71] 95 126 208 178 207 146 118 170 110 172 226
#78 160 185
[85] 186 222 212 185 168 213 76 213 165 173 218
#207 214 203
[99] 206 213 228 186 240 248 208 176 217 142 190
```

Using the pulled variable for descriptive statistics

You can combine the `pull()` function with summary statistics and the pipe operator `%>%` to calculate summary statistics on a specific column of a data frame in a concise manner.

I.e., Median

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
#extracts the column named
#"Duration__in_seconds_" from the data frame as a
#vector.
  median(na.rm = TRUE)
```

```
[1] 237
```

```
#median() function applied to the vector obtained
#above
```

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

Using the pulled variable for descriptive statistics

I.e., Mean

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
#extracts the column named
#"Duration__in_seconds_" from the data frame as a
#vector
  mean(na.rm = TRUE)
```

```
[1] 283.261
```

```
#mean() function applied to the vector obtained
#above
```

Using the pulled variable for descriptive statistics

I.e., Minimum and maximum can be calculated using the `range()` function.

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
#extracts the column named
#"Duration__in_seconds_" from the data frame as a
#vector
  range(na.rm = TRUE)
```

```
[1]   50 1575
```

```
#range() function applied to the vector obtained
#above
```

Using the pulled variable for descriptive statistics

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

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
#extracts the column named
#"Duration__in_seconds_" from the data frame as a
#vector
  var(na.rm = TRUE)
```

```
[1] 29487.81
```

```
##var() function applied to the vector obtained
#above
```

Using the pulled variable for descriptive statistics

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

```
ads_data %>%
  pull(Duration__in_seconds_) %>%
#extracts the column named
#"Duration__in_seconds_" from the data frame as a
#vector
  sd(na.rm = TRUE)
```

```
[1] 171.7202
```

```
#sd() function applied to the vector obtained
#above
```

Summarise

`summarise()` creates a new data frame. It returns a single row summarising all observations in the input.

```
ads_data %>%
summarise(mean_time = mean(Duration__in_seconds_,
#na.rm = TRUE),
#mean of the "Duration__in_seconds_" column using
#the mean() function
sd_time = sd(Duration__in_seconds_, na.rm =
#TRUE))
```

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

```
#standard deviation of "Duration__in_seconds_"
#column using the sd() function
```

Grouping

Before summarising, we can group by a categorical variable. `summarise()` will create a new data frame and return one row for each combination of grouping variables;

```
ads_data %>%
  group_by(Gender) %>%
#groups the data frame by the "Gender" column
#using the group_by() function
#perform subsequent calculations separately for
#each unique value in the "Gender" column.
  summarise(count = n(),
#count of observations within each group,
mean_time = mean(Duration__in_seconds_, na.rm =
#TRUE),
#mean of the "Duration__in_seconds_" column using
#the mean() function
sd_time = sd(Duration__in_seconds_, na.rm =
#TRUE))
#standard deviation of "Duration__in_seconds_"
#column using the sd() function
```

Grouping

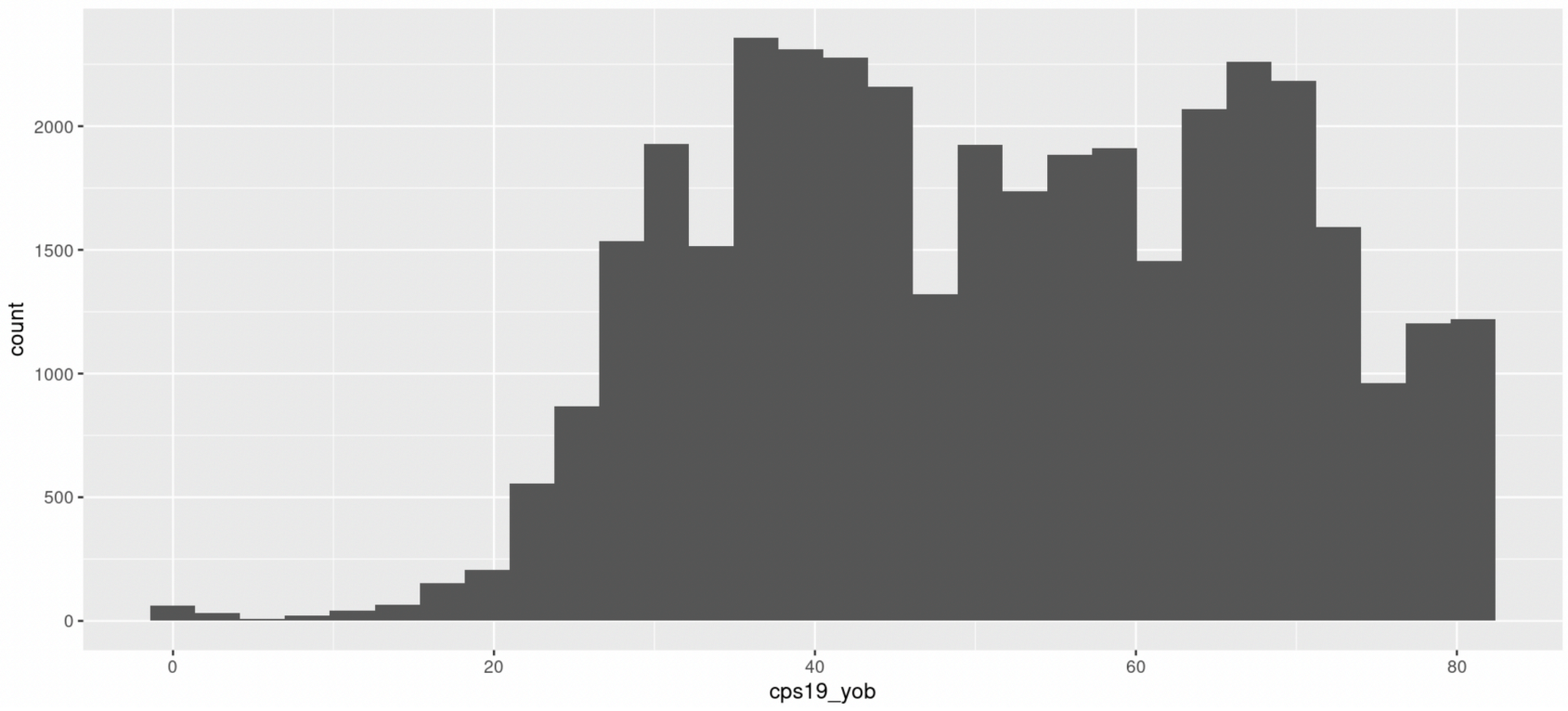
```
# A tibble: 3 x 4
Gender count mean_time sd_time
<dbl+lbl> <int> <dbl> <dbl>
1 1 [Male] 758 269.  162.
2 2 [Female] 698 299.  181.
3 3 [Prefer a third option/Other] 4 229 37.7
```

Questions?

Manipulation application: data cleaning

Data cleaning

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



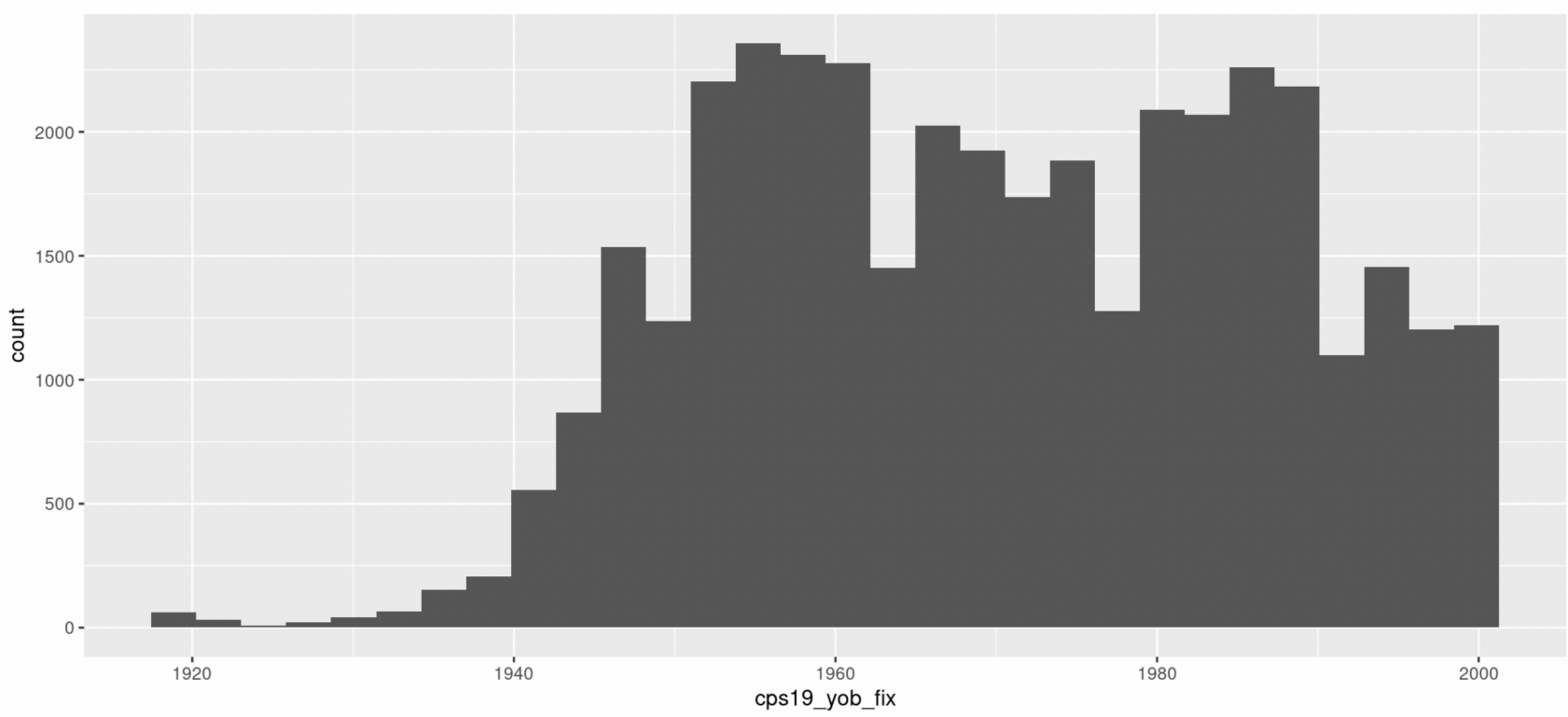
Data cleaning

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.

- create new column called "cps19_yob_fix" where 1919 is added to the value in the column cps19

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

Data cleaning



Better!

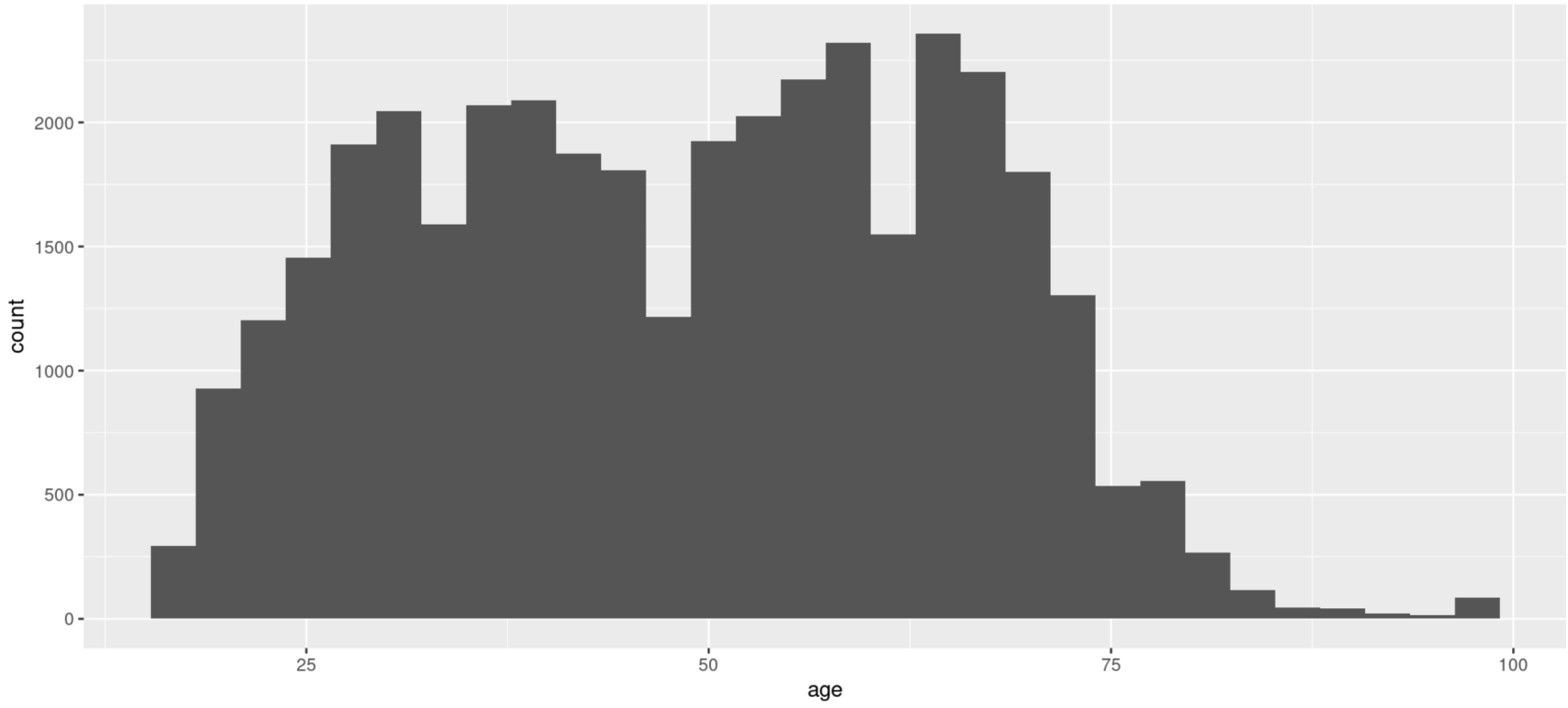
Add a variable for age

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

- create a new column "age" which is the current year of the data (2019) subtracted by the year of birth column "cps19_yob_fix"

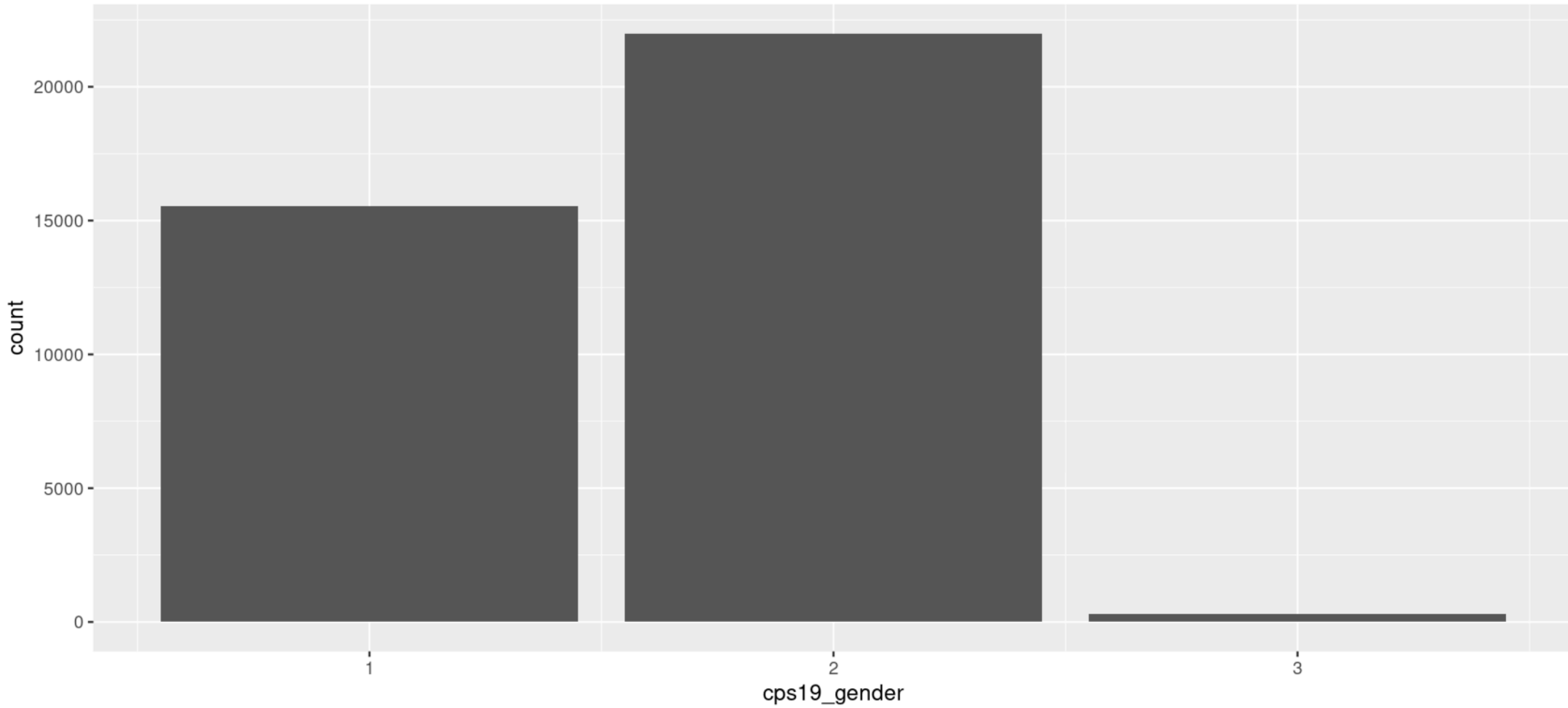
```
CES_data <- CES_data %>%
  mutate(age = 2019 - cps19_yob_fix)
```

Add a variable for age



Recoding the gender variable

Graphing gender shows that it is labelled as “1”, “2” or “3”



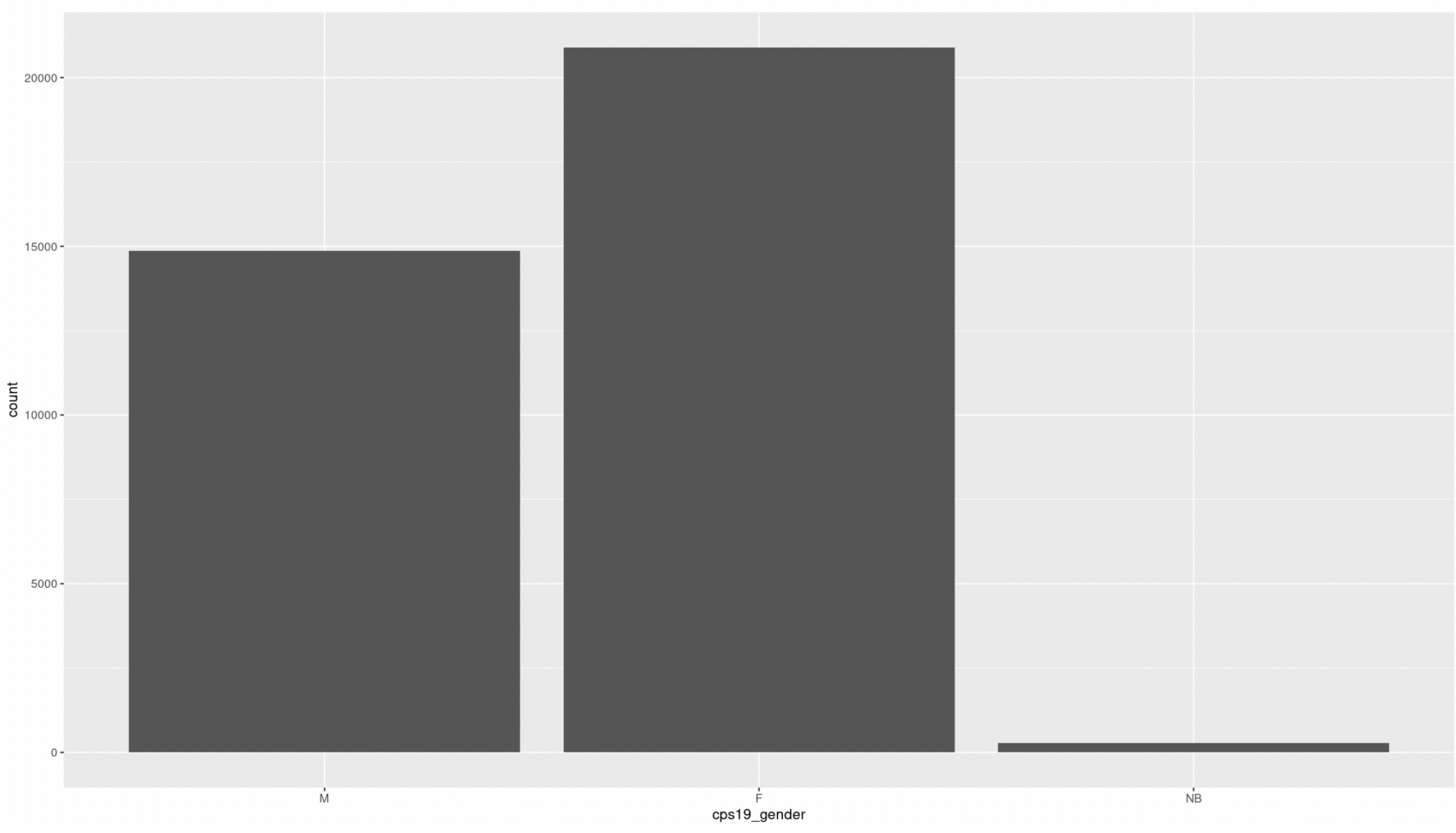
Recoding the gender variable

Lets recode it to make it more intuitive!

- create a new column called “cps19_gender_fix” that takes the original column “cps19_gender” and recodes it into a factor
- modify the column “cp19_gender_fix” so that 1 –> M, 2-> F, 3 –> NB

```
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



Fixing household counts

- filter data frame to only display household counts above 10
- arrange household counts in descending order (-)
- extract household counts from the data frame as a vector

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

Fixing household counts

- modify cps19_household so that if household is > 15 replace with NA, or else leave unchanged
- filter the data frame to keep only the rows where "cps19_household" is greater than 10
- extracts the "cps19_household" column from the filtered data frame and return vector

```
CES_data <- CES_data %>%
mutate(cps19_household = ifelse(cps19_household >
15, NA, cps19_household))
CES_data %>%
  filter(cps19_household > 10) %>%
  pull(cps19_household)
```

Fixing income

- filter the data frame to keep only the rows where "cps19_income_number" is greater than 1 million
- arrange "cps19_income_number" in descending order
- extracts the "cps19_income_number" column from the filtered data frame and return vector

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

Fixing income

- modify cps19_income number so that if income is > or equal to 100 million replace with NA, or else leave unchanged
- filter the data frame to keep only the rows where "cps19_income_number" is greater than 1 million
- extracts the "cps19_income_number" column from the filtered data frame and return vector

```
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)
```

Manipulation application: Summarising data

Summarising data

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

- filter the data frame to keep only the rows where the province is Ontario

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

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.

- filter the data frame to keep only the rows where the province is Ontario
- select only columns on interest from the data frame

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

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

- filter the data frame to keep only the rows where the province is Ontario
- select only columns on interest from the data frame
- group data by provincial identification (i.e., Liberal, NDP etc.)
- median income of each group using the median() function
- count of observations within each group

```
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())
```

Grouping

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

- filter the data frame to keep only the rows where the province is Ontario
- select only columns on interest from the data frame
- modify provincial self ID column so that it is releveled as factors in order
- group data by provincial identification (i.e., Liberal, NDP etc.)
- median income of each group using the median() function
- count of observations within each group

Grouping

```
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 Conservative",
"NDP",
"Green",
"Another party",
"None",
"Don't know/prefer not to answer"))) %>%
  group_by(cps19_prov_id) %>%
  summarise(median_income =
              median(cps19_income_number,
                     na.rm = TRUE),
            count =  n())
```

Grouping

What happens if we group by political satisfaction instead?

- filter the data frame to keep only the rows where the province is Ontario
- select only columns on interest from the data frame
- group data by provincial satisfaction
- median income of each group using the median() function
- count of observations within each group

```
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())
```

Grouping

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

- filter the data frame to keep only the rows where the province is Ontario
- select only columns on interest from the data frame
- group data by provincial identification (i.e., Liberal, NDP etc.)
- median income of each group using the median() function
- count of observations within each group
- arrange dataframe so that median income is in descending order

Grouping

```
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)
```

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:

- filter the data frame to keep only the rows where the province is Ontario
- select only columns on interest from the data frame
- modify provincial self ID column so that it is releveled as factors in order
- modify provincial satisfaction column so that it is releveled as factors in order
- group data by provincial identification AND satisfaction
- median income of each group using the median() function

Grouping

```
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 Conservative",
"NDP",
"Green",
"Another party",
"None",
"Don't know/prefer not to answer"))) %>%
```

Grouping

```
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 to answer"))) %>%
  group_by(cps19_prov_gov_sat, cps19_prov_id) %>%
  summarise(median_income =
              median(cps19_income_number,
                     na.rm = TRUE))
```

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:

- filter the data frame to keep only the rows where the province is Ontario
- select only columns on interest from the data frame
- modify provincial self ID column so that it is releveled as factors in order
- modify provincial satisfaction column so that it is releveled as factors in order
- group data by provincial identification AND satisfaction
- median income of each group using the median() function
- function spreads the values from the key into separate columns, resulting in a wide format data frame

Grouping

```
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 Conservative",
"NDP",
"Green",
"Another party",
"None",
"Don't know/prefer not to answer"))) %>%
```

Grouping

```
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 to answer"))) %>%
  group_by(cps19_prov_gov_sat, cps19_prov_id) %>%
    summarise(median_income =
                median(cps19_income_number,
                       na.rm = TRUE)) %>%
  spread(key = cps19_prov_gov_sat,
         value = median_income)
```

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 to answer"))
```

- 4 - Fix this error:

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

Questions?