HOUSEKEEPING ITEMS

Midterm Project things to mention:

- Expect a “Thanks for sending!” reply when you send your RPubs link

TODAY'S CLASS

6:00PM – 7:30PM: Joining data (Not with SQL! In R!)

7:45PM – 8:45PM: Leveraging the Tidyverse to Simplify Data Wrangling

9:00PM – 9:50PM: Leveraging %>% and the Tidyverse for your project

THIS HOUR: WRANGLING WITH THE TIDYVERSE

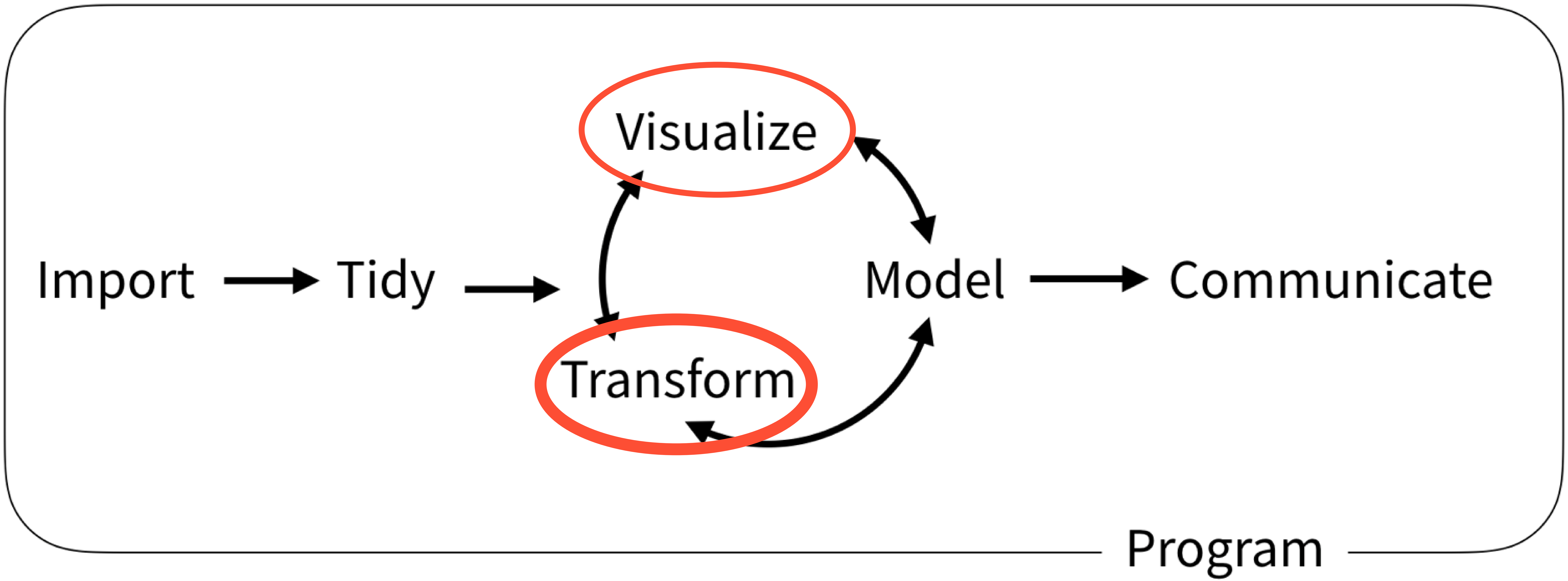
Intro: Logicals and Tibbles

1: Strings

2: Factors

3: Dates/Times

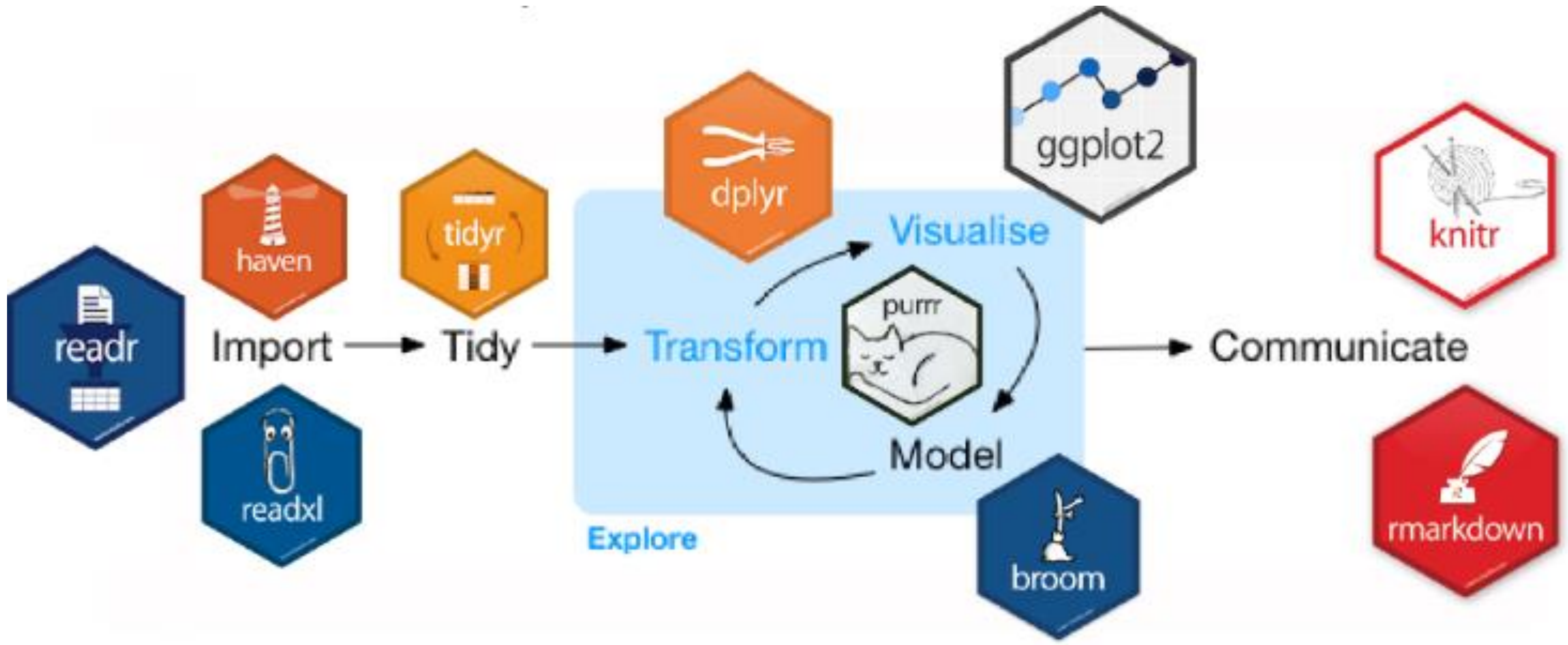
HOW THIS IMPROVES DATA SCIENCE PROJECTS



WHAT IS THE TIDYVERSE?

An opinionated collection of packages…

designed to simplify data analysis.



PREREQUISITES

PACKAGE PREREQUISITE

```
library(tidyverse) # core tidyverse includes dplyr, stringr, and forcats

# may need to install the following packages first
library(lubridate)
library(glue)
```

DATA PREREQUISITE

```
# go ahead and set your working directory to this week’s folder you downloaded
crime <- read_csv("cincinnati_crimes_20190812.csv")
```

INTRO: LOGICALS

dplyr

www.rstudio.com

CREATING BOOLEAN VALUES

Operator	Description
>	a > b
>=	a >= b
<	a< b
<=	a <= b
== (check for equality)	a == b
!= (check for not equal)	a != b
%in% (check for group membership)	a %in% c(a, b, c)
is.na()	is.na(tailnum)
!is.na()	!is.na(tailnum)

```
# comparison operators create Boolean values
# i.e., TRUE and FALSE


# create Boolean values
2 <= 3
## [1] TRUE

# create a Boolean vector
!is.na(letters[1:15])
## [1]  TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [10] TRUE TRUE TRUE TRUE TRUE TRUE
```

LOGICAL VALUES AND DATA TYPES

R's data type for Boolean values

```
# values can be logical
typeof(TRUE)
## [1] "logical"
typeof(FALSE)
## [1] "logical"


## vectors can be logical
x <- c(TRUE, NA, FALSE)
typeof(x)
## [1] "logical"
```

Creating a logical variable (vector) in your data set

```
# generation z
crime %>%
  select(INCIDENT_NO, SUSPECT_AGE) %>%
  mutate(gen_z = SUSPECT_AGE %in% c("UNDER 18", "18-25"))
# A tibble: 21,153 x 3
   INCIDENT_NO SUSPECT_AGE gen_z
   <chr>       <chr>       <lgl>
 1 199003291   26-30       FALSE
 2 199006697   UNKNOWN     FALSE
 3 199002974   18-25       TRUE
 4 199002942   UNKNOWN     FALSE
 5 199003557   UNKNOWN     FALSE
 6 199001482   UNKNOWN     FALSE
 7 199005210   31-40       FALSE
 8 199006079   UNKNOWN     FALSE
 9 199006287   26-30       FALSE
10 199000792   UNKNOWN     FALSE
# ... with 21,143 more rows
```

GENERATING INSIGHTS FROM LOGICALS

Count TRUEs by summing a logical vector

```
# quick example
x <- c(8, 4, 5, 1)
x
## [1] TRUE TRUE TRUE FALSE

# How many elements
# satisfy the condition?
sum(x)
## [1] 3
```

Find proportion of TRUEs by taking the mean of a logical vector

```
# generation z
crime %>%
  select(INCIDENT_NO, SUSPECT_AGE) %>%
  mutate(gen_z = SUSPECT_AGE %in% c("UNDER 18", "18-25")) %>%
  summarize(pct_gen_z = mean(gen_z, na.rm = TRUE))
# A tibble: 1 x 1
  pct_gen_z
      <dbl>
1     0.176
```

YOUR TURN!

Using our *crimes* data set:

After grouping by the DAYOFWEEK variable,

1. *How many records occurred in the SNA_NEIGHBORHOOD of Clifton?*
2. *What percentage is this for each group?*

BONUS! Can you calculate the counts and percentages without a mutate statement?

SOLUTION

```
crime %>%
  group_by(DAYOFWEEK) %>%
  mutate(clifton = SNA_NEIGHBORHOOD == "CLIFTON") %>%
  summarize(
    num_clifton = sum(clifton, na.rm = TRUE),
    num_total = n(),
    pct_clifton = mean(clifton, na.rm = TRUE)
  )
# A tibble: 8 x 4
  DAYOFWEEK num_clifton num_total pct_clifton
  <chr>           <int>     <int>       <dbl>
1 FRIDAY             72      3062      0.0235
2 MONDAY             53      3020      0.0175
3 SATURDAY           34      2925      0.0116
4 SUNDAY             39      2883      0.0135
5 THURSDAY           30      2925      0.0103
6 TUESDAY            57      3048      0.0187
7 WEDNESDAY          46      2927      0.0157
8 NA                 26       363      0.0716
```

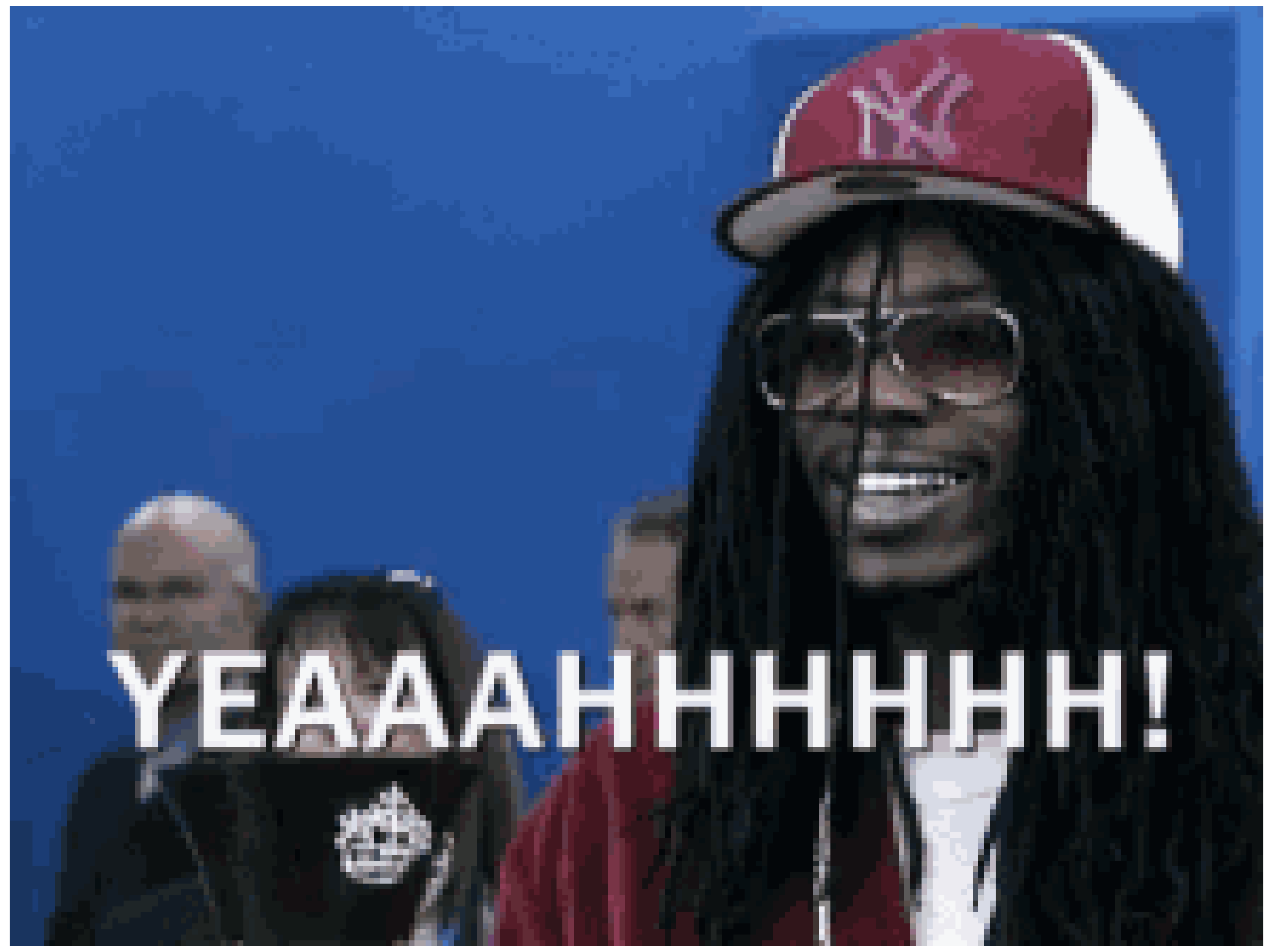
SOLUTION WITH BONUS

```
crime %>%
  group_by(DAYOFWEEK) %>%
  summarize(
    num_clifton = sum(SNA_NEIGHBORHOOD == "CLIFTON", na.rm = TRUE),
    num_total = n(),
    pct_clifton = mean(SNA_NEIGHBORHOOD == "CLIFTON", na.rm = TRUE)
  )
```

INTRO: TIBBLES

TIBBLES ARE UBIQUITOUS!

You’ve worked with tibbles before!




```
crime %>%
  group_by(DAYOFWEEK) %>%
  mutate(clifton = SNA_NEIGHBORHOOD == "CLIFTON") %>%
  summarize(
    num_clifton = sum(clifton, na.rm = TRUE),
    num_total = n(),
    pct_clifton = mean(clifton, na.rm = TRUE)
  )
# A tibble: 8 x 4
  DAYOFWEEK num_clifton num_total pct_clifton
  <chr>           <int>     <int>       <dbl>
1 FRIDAY             72      3062      0.0235
2 MONDAY             53      3020      0.0175
3 SATURDAY           34      2925      0.0116
4 SUNDAY             39      2883      0.0135
5 THURSDAY           30      2925      0.0103
6 TUESDAY            57      3048      0.0187
7 WEDNESDAY          46      2927      0.0157
8 NA                 26       363      0.0716
```

WHAT ARE TIBBLES?

From the Tidyverse website:
"A **tibble**, or tbl_df, is a modern reimagining of the data.frame, keeping what time has proven to be effective, and throwing out what is not.

Tibbles:

- Are data frames, but with edited behaviors
- Never change input data types (e.g., strings to factors, characters to numeric)
- Never change variable names
- Never create row names
- ~~Never gonna give you up~~
- Allow non-syntactic variable names

```
crime %>%
    head(10)

# A tibble: 10 x 40
   INSTANCEID INCIDENT_NO DATE_REPORTED DATE_FROM DATE_TO CLSD   UCR DST   BEAT
   <chr>      <chr>       <chr>         <chr>     <chr>   <chr> <dbl> <chr> <chr>
 1 92A296AB-~ 199003291   2/16/2019 10~ 2/16/201~ 2/16/2~ J--C~   201 4     5
 2 44ACB102-~ 199006697   4/4/2019 16:~ 4/4/2019~ 4/4/20~ Z--E~  1151 2     1
 3 2CED4B80-~ 199002974   2/12/2019 17~ 2/5/2019~ 2/7/20~ D--V~   201 4     4
 4 EEB41765-~ 199002942   2/12/2019 10~ 2/6/2019~ 2/6/20~ J--C~   201 5     2
 5 F4622DF5-~ 199003557   2/20/2019 15~ 2/19/201~ 2/19/2~ J--C~   600 4     3
 6 EF456ED0-~ 199001482   1/21/2019 11~ 1/20/201~ 1/21/2~ Z--E~   600 4     2
 7 0859E5C0-~ 199005210   3/15/2019 14~ 3/12/201~ 3/12/2~ H--W~  1493 2     2
 8 9B091265-~ 199006079   3/27/2019 4:~ 3/27/201~ 3/27/2~ Z--E~  1400 1     3
 9 D2DAF74C-~ 199006287   3/29/2019 15~ 3/29/201~ 3/29/2~ Z--E~   600 4     2
10 43EEB437-~ 199000792   1/10/2019 13~ 1/9/2019~ 1/9/20~ J--C~   600 5     1
# ... with 31 more variables: OFFENSE <chr>, LOCATION <chr>, THEFT_CODE <chr>,
```

CREATING TIBBLES

Create or coerce into tibble with **as_tibble()**

```
as_tibble(iris)
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# ... with 140 more rows
```

Create tibbles from individual vectors (recycling occurs)

```
tibble(
  division = c("Columbus",
               "Nashville",
               "Atlanta"),
  test_group = 1,
  # use backticks for non-syntactical name
  `:)_order` = 1:3
)

# A tibble: 3 x 3
  division  test_group `:)_order`
  <chr>          <dbl>      <int>
1 Columbus           1          1
2 Nashville          1          2
3 Atlanta            1          3
```

DIFFERENCES BETWEEN TIBBLES AND DATA FRAMES: PRINT METHOD

Tibbles ☺

```
as_tibble(iris)
# A tibble: 150 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          5.1         3.5          1.4         0.2 setosa
2          4.9         3            1.4         0.2 setosa
3          4.7         3.2          1.3         0.2 setosa
4          4.6         3.1          1.5         0.2 setosa
5          5           3.6          1.4         0.2 setosa
6          5.4         3.9          1.7         0.4 setosa
7          4.6         3.4          1.4         0.3 setosa
8          5           3.4          1.5         0.2 setosa
9          4.4         2.9          1.4         0.2 setosa
10         4.9         3.1          1.5         0.1 setosa
# ... with 140 more rows
```

Base R ☹

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
11          5.4         3.7          1.5         0.2  setosa
12          4.8         3.4          1.6         0.2  setosa
13          4.8         3.0          1.4         0.1  setosa
14          4.3         3.0          1.1         0.1  setosa
15          5.8         4.0          1.2         0.2  setosa
16          5.7         4.4          1.5         0.4  setosa
17          5.4         3.9          1.3         0.4  setosa
18          5.1         3.5          1.4         0.3  setosa
19          5.7         3.8          1.7         0.3  setosa
20          5.1         3.8          1.5         0.3  setosa
21          5.4         3.4          1.7         0.2  setosa
22          5.1         3.7          1.5         0.4  setosa
23          4.6         3.6          1.0         0.2  setosa
24          5.1         3.3          1.7         0.5  setosa
25          4.8         3.4          1.9         0.2  setosa
26          5.0         3.0          1.6         0.2  setosa
27          5.0         3.4          1.6         0.4  setosa
```

(and it automatically prints 1000 rows)

REVIEW: SELECTING COLUMNS FROM DATA FRAMES

- **Preserve** the structure of the output to be the same as the input with data_frame[column]
  - Can use a column name in quotes or a column index
- **Simplify** the structure of the output with data_frame[[column]]
  - Can use a column name in quotes or a column index
- **Simplify** the structure of the output to be a smaller structure than the input with data_frame$column
  - Must use a column name with a $

DIFFERENCES BETWEEN TIBBLES AND DATA FRAMES:
SUBSETTING AND SIMPLIFYING OUTPUT

Base R: Subsetting data frames with square brackets sometimes returns a vector

```
# matrix subsetting simplifies
cars[, “speed”]
 [1]  4  4  7  7  8  9 10 10 10 11 11
[12] 12 12 12 12 13 13 13 13 14 14 14
[23] 14 15 15 15 16 16 17 17 17 18 18
[34] 18 18 19 19 19 20 20 20 20 20 22
[45] 23 24 24 24 24 25

# list subsetting doesn’t simplify
cars[“speed”]
 speed
1     4
2     4
3     7
4     7
5     8
6     9
```

Tibbles always return another tibble when subsetting with square brackets

```
cars %>%
  as_tibble() %>%
  # use the placeholder .
  # when piping into [ ] or [[ ]] or $
  .[, "speed"]
# A tibble: 50 x 1
   speed
   <dbl>
 1     4
 2     4
 3     7
 4     7
 5     8
 6     9
 7    10
 8    10
 9    10
10    11
# ... with 40 more rows
```

FOR MORE INFORMATION

https://tibble.tidyverse.org/

01/ STRINGS

stringr

www.rstudio.com

WORKING WITH CHARACTER STRINGS

- Often, we have character strings in our data that are long (e.g., description fields), messy (e.g., manual user input), and/or inconsistent

- Working with strings in Base R can be frustrating because of syntax inconsistencies

- The **stringr** package allows you to work with strings easily

COMMON STRING TASKS WE'RE COVERING

Matching patterns

Leveraging (easier) regular expressions

Extracting characters

Finding lengths

Padding strings

Changing case

Replacing patterns

… and so much more that's not in this training because strings are crazy

stringr FUNCTIONS

Every stringr function begins with str_

str_sub()

str_count()

str_replace()

str_detect()

str_remove()

…

Check out all the options with stringr::str_ + tab !

MATCHING PATTERNS WITH **str_detect()**

str_detect() checks if elements of a character vector match a pattern, returning a logical vector

```
# str_detect() searches
# for the pattern
# anywhere in the string
x <- c("apple", "pineapple",
     "crabapple", NA, "peach")

# returns one boolean
# value for each element
str_detect(x, "app")
[1]  TRUE  TRUE  TRUE    NA FALSE
```

Creating variables with **str_detect()**

```
crime %>%
  select(HATE_BIAS) %>%
  mutate(hate_toward_group = str_detect(HATE_BIAS, "ANTI-"))
# A tibble: 21,153 x 2
   HATE_BIAS                  hate_toward_group
   <chr>                      <lgl>
 1 N--NO BIAS/NOT APPLICABLE FALSE
 2 N--NO BIAS/NOT APPLICABLE FALSE
 3 N--NO BIAS/NOT APPLICABLE FALSE
 4 N--NO BIAS/NOT APPLICABLE FALSE
 5 N--NO BIAS/NOT APPLICABLE FALSE
 6 N--NO BIAS/NOT APPLICABLE FALSE
 7 N--NO BIAS/NOT APPLICABLE FALSE
 8 N--NO BIAS/NOT APPLICABLE FALSE
 9 N--NO BIAS/NOT APPLICABLE FALSE
10 N--NO BIAS/NOT APPLICABLE FALSE
# ... with 21,143 more rows
```

YOUR TURN!

Using our *crimes* data set and the CLSD variable:

1. *How many records have “CLOSED” in the CLSD variable, meaning the case is closed?*
2. *What is the proportion of records that are closed?*

SOLUTION

```
crime %>%
  select(CLSD) %>%
  mutate(closed_case = str_detect(CLSD, "CLOSED")) %>%
  summarize(num_closed = sum(closed_case, na.rm = TRUE),
            pct_closed = mean(closed_case, na.rm = TRUE))
# A tibble: 1 x 2
  num_closed pct_closed
       <int>      <dbl>
1      10269      0.497
```

SOLUTION PART 2

Question: How do I ignore case?

Answer: Use **stringr::regex()** (or other **stringr** functions) to ignore case!

```
crime %>%
  select(CLSD) %>%
  mutate(closed_case = str_detect(CLSD,
                          regex("cLoSeD", ignore_case = TRUE))) %>%
  summarize(num_closed = sum(closed_case, na.rm = TRUE),
        pct_closed = mean(closed_case, na.rm = TRUE))
```

YOUR FIRST REGULAR EXPRESSION

- "Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.

- Regular expressions are sequences of characters that define a search pattern, and can become very complicated quickly. The **stringr** package helps to avoid complicated regular expressions like:

```
email_pat = "^([a-z0-9_\\.-]+)@([\\da-z\\.-]+)\\.([a-z\\.]{2,6})$"
```

- However, regular expressions are convenient sometimes.

YOUR FIRST REGULAR EXPRESSION

Anchors

Characters	Description
^	string begins with
$	string ends with

```
# match pattern at beginning of string
crime %>%
  filter(str_detect(SNA_NEIGHBORHOOD, "^MT.")) %>%
  count(SNA_NEIGHBORHOOD, sort = TRUE)
# A tibble: 5 x 2
  SNA_NEIGHBORHOOD     n
  <chr>            <int>
1 MT. AIRY           563
2 MT. AUBURN         419
3 MT. WASHINGTON     254
4 MT. ADAMS           77
5 MT. LOOKOUT         62
```

YOUR FIRST REGULAR EXPRESSION

Anchors

Characters	Description
^	string begins with
$	string ends with

```
# match pattern at end of string
crime %>%
  filter(str_detect(SNA_NEIGHBORHOOD, "HILL$")) %>%
  count(SNA_NEIGHBORHOOD, sort = TRUE)

# A tibble: 6 x 2
  SNA_NEIGHBORHOOD         n
  <chr>                <int>
1 EAST PRICE HILL       1348
2 WEST PRICE HILL       1197
3 COLLEGE HILL           755
4 BOND HILL              367
5 VILLAGES AT ROLL HILL  265
6 LOWER PRICE HILL        98
```

YOUR FIRST REGULAR EXPRESSION

Alternatives

Characters	Description
|	string contains one of these
[ ]	string contains any of these
[^]	string contains anything but these
[-]	string contains in range of

```
# check for multiple regular expressions
# at the same time
crime %>%
  filter(str_detect(SNA_NEIGHBORHOOD,
                    "^MT.|HILL$|SOUTH")) %>%
  count(SNA_NEIGHBORHOOD, sort = TRUE)
# A tibble: 13 x 2
   SNA_NEIGHBORHOOD        n
   <chr>               <int>
 1 EAST PRICE HILL      1348
 2 WEST PRICE HILL      1197
 3 COLLEGE HILL          755
 4 MT. AIRY              563
 5 MT. AUBURN            419
 6 SOUTH FAIRMOUNT       374
 7 BOND HILL             367
 8 VILLAGES AT ROLL HILL 265
 9 MT. WASHINGTON        254
10 LOWER PRICE HILL       98
11 MT. ADAMS              77
12 MT. LOOKOUT            62
13 SOUTH CUMMINSVILLE     53
```

YOUR FIRST REGULAR EXPRESSION

Quantifiers

Characters	Description
a?	zero or one
a*	zero or more
a+	one or more
a{n}	exactly n
a{n, }	b or more
a{n, m}	between n and m

```
## look for suspect ages in double-digits
crime %>%
  filter(str_detect(SUSPECT_AGE, "^[0-9]{2}")) %>%
  count(SUSPECT_AGE)


# A tibble: 6 x 2
  SUSPECT_AGE     n
  <chr>       <int>
1 18-25        2652
2 26-30        1724
3 31-40        2031
4 41-50         899
5 51-60         418
6 61-70         137
```

HUNGRY FOR MORE?

https://stringr.tidyverse.org/articles/regular-expressions.html

EXTRACTING CHARACTERS WITH str_sub()

Extract location code with defined start/end positions

```
crime %>%
  transmute(LOCATION,
         location_code = str_sub(string = LOCATION,
                              start = 1,
                              end = 2))
# A tibble: 21,153 x 2
   LOCATION                     location_code
   <chr>                        <chr>
 1 02-MULTI FAMILY              02
 2 01-SINGLE FAMILY HOME        01
 3 02-MULTI FAMILY APARTMENT    02
 4 29-GAS STATION               29
 5 47-STREET                    47
 6 47-STREET                    47
 7 47-STREET                    47
 8 47-STREET                    47
 9 38-VARIETY/CONVENIENCE STORE 38
10 02-MULTI FAMILY              02
```

Extract last three digits by counting backward from the last character

```
crime %>%
  transmute(ZIP,
         last_three = str_sub(ZIP, -3))
# A tibble: 21,153 x 2
     ZIP last_three
   <dbl> <chr>
 1 45237 237
 2 45206 206
 3 45229 229
 4 45225 225
 5 45229 229
 6 45202 202
 7 45227 227
 8 45202 202
 9 45206 206
10 45220 220
# ... with 21,143 more rows
```

DATA CLEANING WITH **str_length()** AND **str_pad()**

str_length() outputs the number of characters a string contains

```
crime %>%
  transmute(ZIP = as.character(ZIP),
            num_digits_zip = str_length(ZIP))
# A tibble: 21,153 x 2
   ZIP   num_digits_zip
   <chr>          <int>
 1 45237              5
 2 45206              5
 3 45229              5
 4 45225              5
 5 45229              5
 6 45202              5
 7 45227              5
 8 45202              5
 9 45206              5
10 45220              5
# ... with 21,143 more rows
```

str_pad() example: right-pad to fill in empty digits with Xs

```
crime %>%
  transmute(ZIP = as.character(ZIP),
            num_digits_zip = str_length(ZIP),
            fixed_zip = str_pad(string = ZIP,
                                width = 5,
                                side = "right",
                                pad = "X")) %>%
  filter(num_digits_zip < 5)
  ZIP   num_digits_zip fixed_zip
  <chr>          <int> <chr>
1 452                3 452XX
2 33                 2 33XXX
3 33                 2 33XXX
4 33                 2 33XXX
```

YOUR TURN!

```
# fill in the blanks!
crime %>%
  # select a few variables
  select(HOUR_FROM, ZIP) %>%
  mutate(
    # change hour_from to a character
    HOUR_FROM = as.__________(HOUR_FROM),
    # left-pad zeroes to create 24-hour time
    HOUR_FROM = str_pad(string = HOUR_FROM,
                        width = ___,
                        side = "____",
                        pad = "___"),
    # change zip to a character
    ZIP = __________________,
    # make if-then statement to right-pad zip codes less than 5 digits
    ZIP = if_else(
      # check the condition for the if_else function
      condition = ____________(ZIP) < ___,
      # if less than 5 digits, right-pad an X
      true = _______________________________,
      # otherwise keep the zip code as-is
      false = ZIP)
```

SOLUTION

```
# fill in the blanks!
crime %>%
  # select a few variables
  select(HOUR_FROM, ZIP) %>%
  mutate(
    # change hour_from to a character
    HOUR_FROM = as.character(HOUR_FROM),
    # left-pad zeroes to create 24-hour time
    HOUR_FROM = str_pad(string = HOUR_FROM,
                        width = 4,
                        side = "left",
                        pad = "0"),
    # change zip to a character
    ZIP = as.character(ZIP),
    # make if-then statement to right-pad zip codes less than 5 digits
    ZIP = if_else(
      # check the condition for the if_else function
      condition = str_length(ZIP) < 5,
      # if less than 5 digits, right-pad an X
      true = str_pad(ZIP, 5, "right", "X"),
      # otherwise keep the zip code as-is
      false = ZIP)
  )
```

OTHER USEFUL FUNCTIONS FROM **stringr**

```
# a lame example vector
x <- c("VEG SOUP", " MIXED VEG/VEG MEDLEY", "bAd NaMe 4 VeG ")

## str_to_lower()--there is also str_to_upper() and str_to_title()
str_to_lower(x)
[1] "veg soup"            " mexed veg/veg medley" "bad name 4 veg "

## str_trim removes whitespace from the side(s) you specify
str_trim(x)
[1] "VEG SOUP"            "MEXED VEG/VEG MEDLEY" "bAd NaMe 4 VeG"
```

OTHER USEFUL FUNCTIONS FROM **stringr**

Replacing patterns

```
# same lame example vector
x <- c("VEG SOUP", " MIXED VEG/VEG MEDLEY", "bAd NaMe 4 VeG ")

## str_replace replaces the first matched pattern
str_replace(x,
        pattern = "VEG",
        replacement = "VEGETABLE")
[1] "VEGETABLE SOUP"  " MIXED VEGETABLE/VEG MEDLEY" "bAd NaMe 4 VeG "

# str_replace_all replaces all matched patterns
str_replace_all(x,
          pattern = "VEG",
          replacement = "VEGETABLE")
[1] "VEGETABLE SOUP" " MIXED VEGETABLE/VEGETABLE MEDLEY" "bAd NaMe 4 VeG "
```

FOR MORE INFORMATION

https://stringr.tidyverse.org/

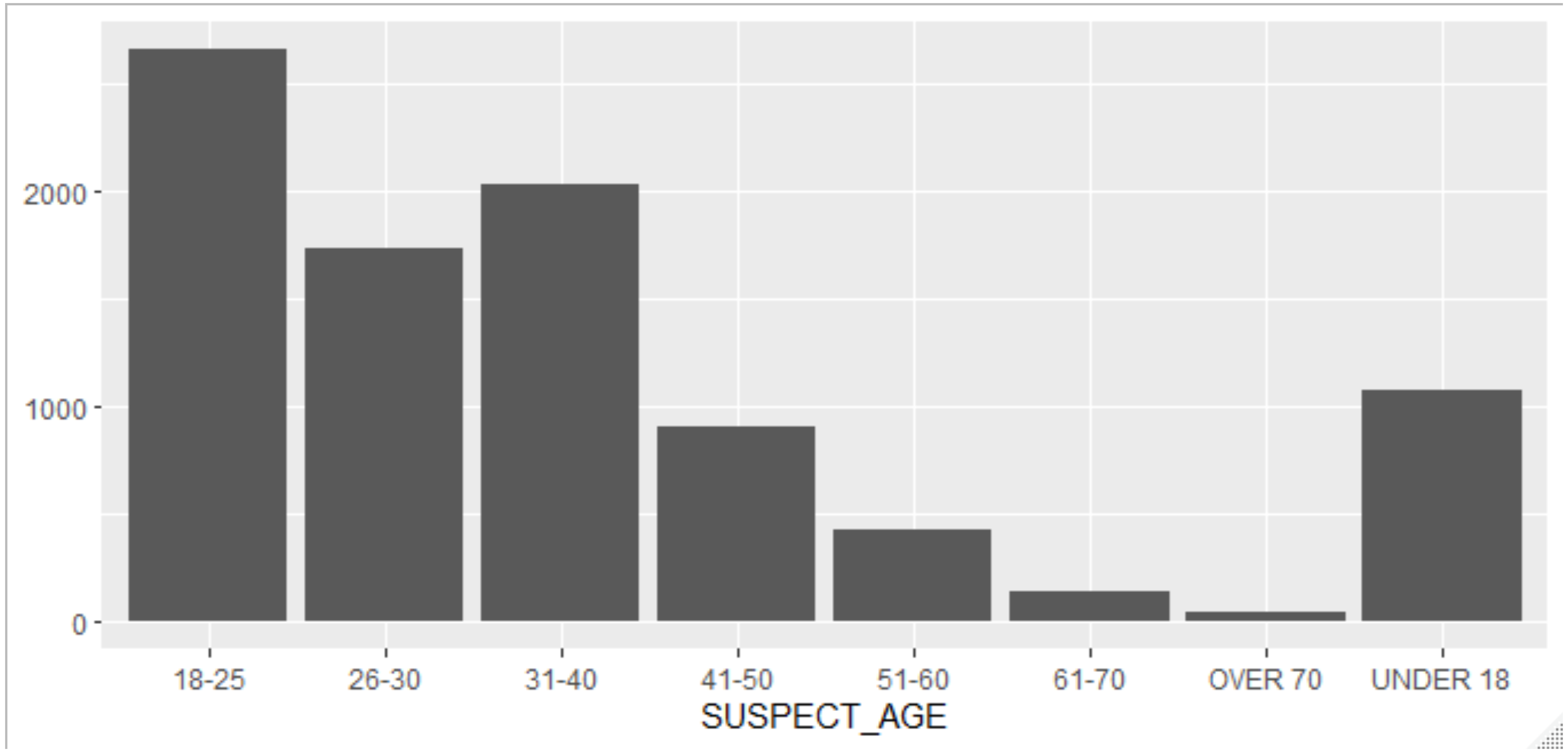
BONUS: PASTE STRINGS WITH **glue**

Love pasting strings but hate dealing with variables inside strings? Check out the glue package!

https://glue.tidyverse.org/

02/ FACTORS

WHY WE CARE ABOUT FACTORS



WORKING WITH FACTORS

- Factors are a useful data structure, particularly for modeling and visualizations, because they control the order of levels

- Working with factors in Base R can be frustrating because of syntax inconsistencies and a handful of missing tools

- The **forcats** package allows you to modify factors with minimal pain

HOW R REPRESENTS AND STORES FACTORS

Factors: R’s representation of categorical data. Consists of:

- A set of discrete values
- An ordered set of valid levels

```
(eyes <- base::factor(x = c("blue", "green", "green"),
                      levels = c("blue", "brown", "green")))
```

Stored as an integer vector with a levels attribute

```
unclass(eyes)
[1] 1 3 3
attr(,"levels")
[1] "blue"  "brown" "green"
```

forcats FUNCTIONS AND COMMON TASKS

All **forcats** functions start with **fct_**

- fct_relevel()
- fct_recode()
- fct_collapse()
- fct_unique()

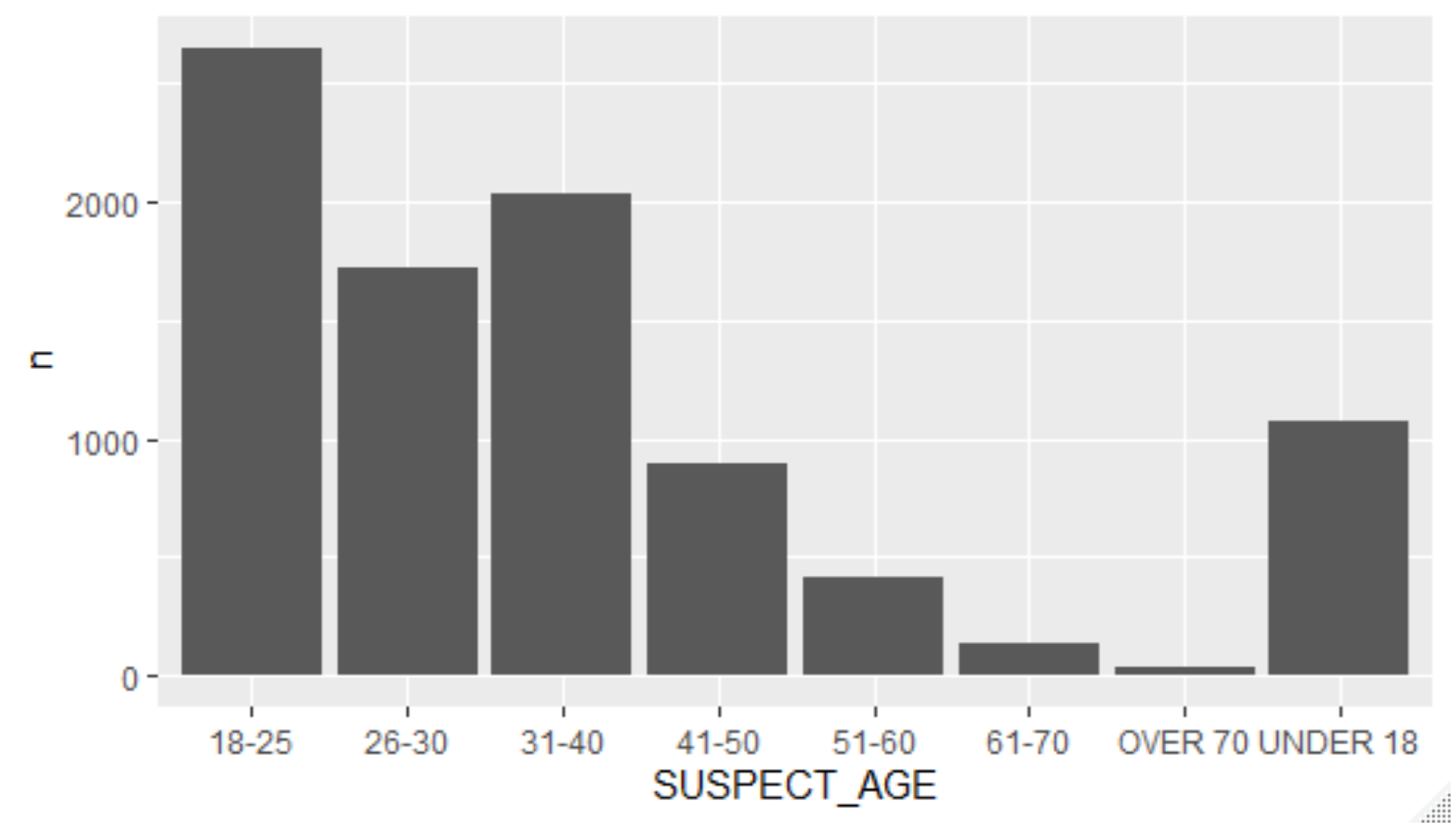
Common tasks we're covering

- Reorder levels
- Recode levels
- Collapse levels
- Temporarily reorder levels
- Reorder levels based on other variable(s)
- … and more!

GRAPHING WITHOUT REORDERING FACTOR LEVELS

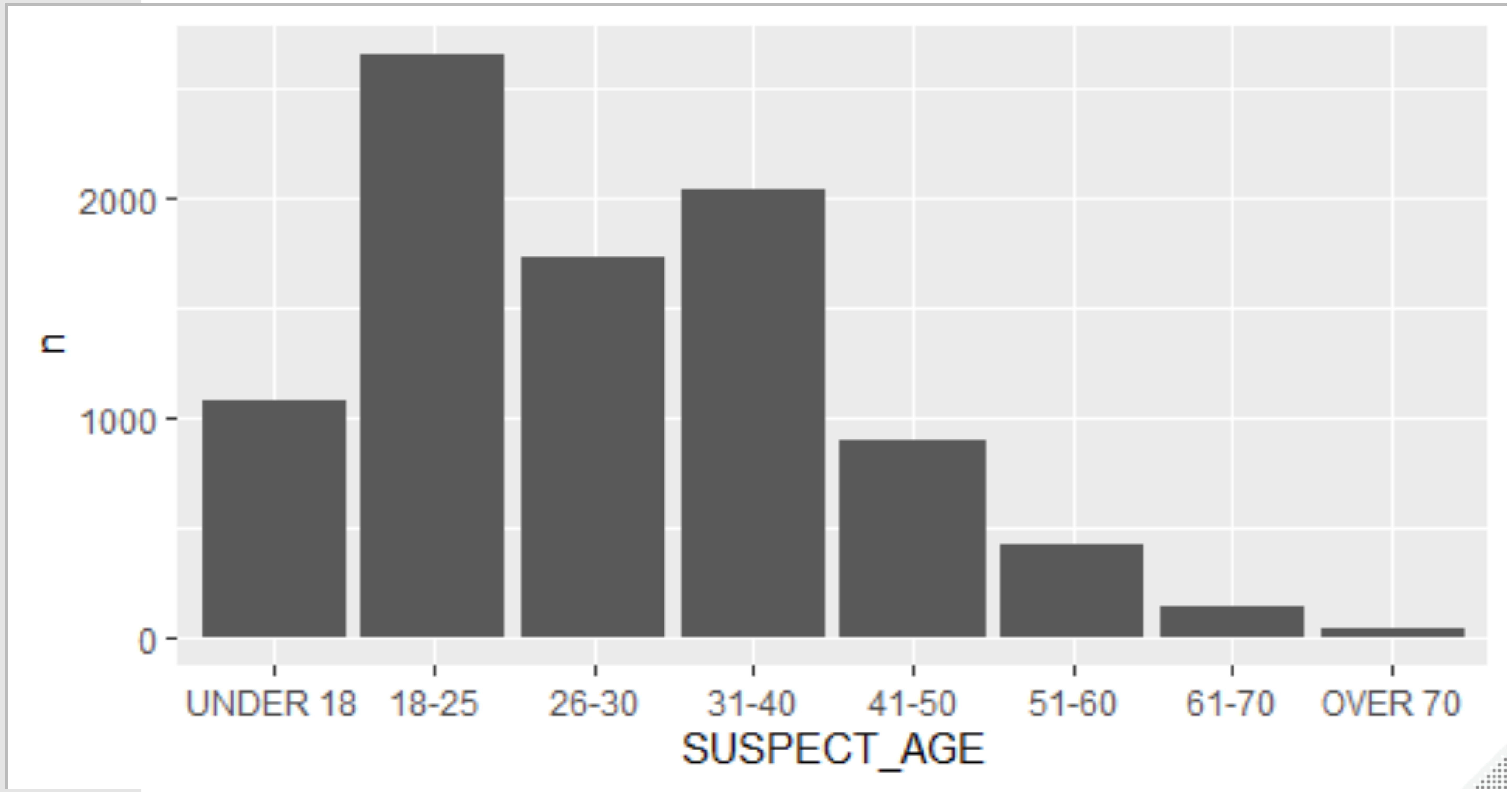
```
# create a new data set
age <- crime %>%
  # filter suspect ages simply for readability
  filter(SUSPECT_AGE != "UNKNOWN")

# notice how SUSPECT_AGE is a character variable
age %>% count(SUSPECT_AGE)
# A tibble: 8 x 2
  SUSPECT_AGE     n
  <chr>       <int>
1 18-25        2652
2 26-30        1724
3 31-40        2031
4 41-50         899
5 51-60         418
6 61-70         137
7 OVER 70        33
8 UNDER 18     1068
```



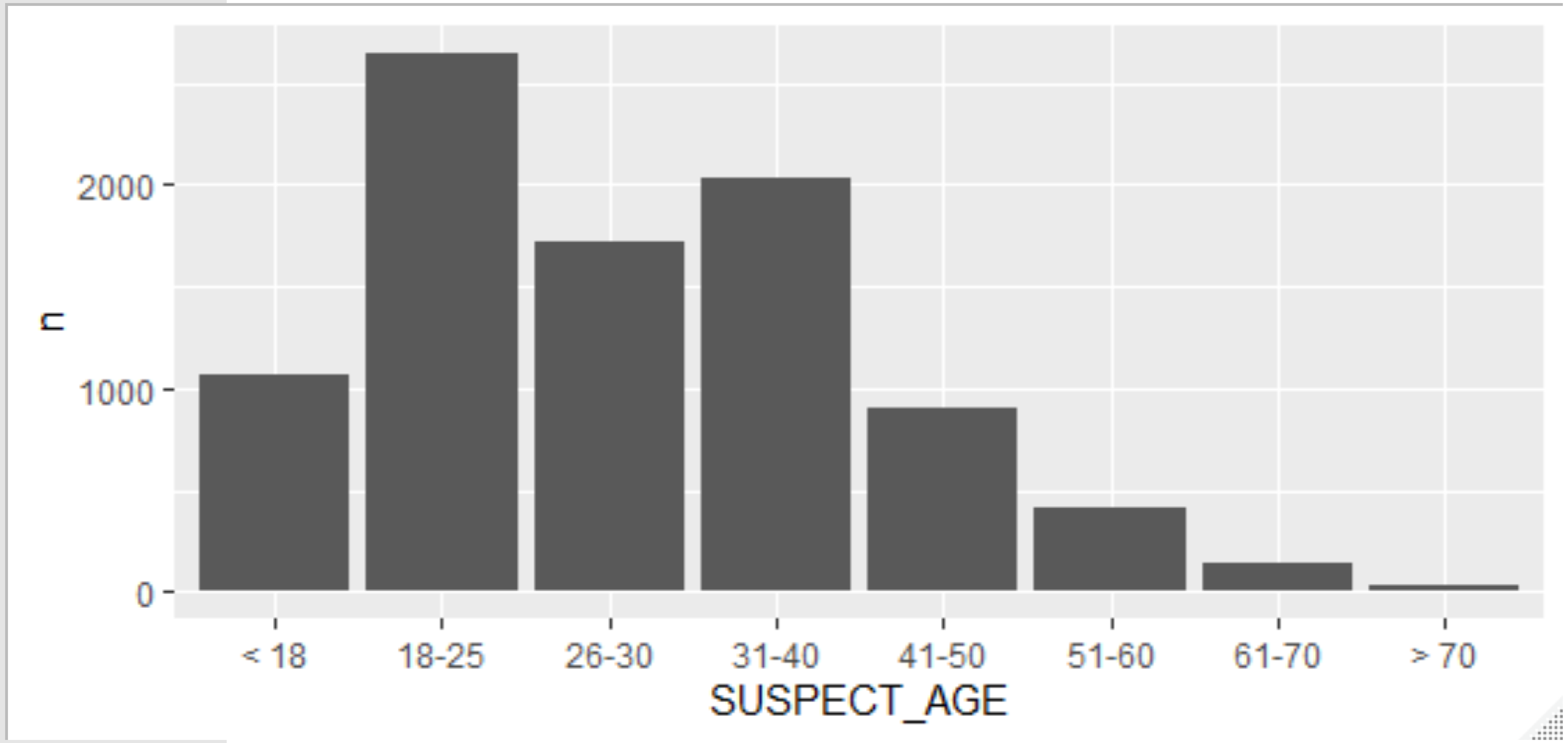
REORDER LEVELS WITH fct_relevel()

```
age_releveled <- age %>%
  # fct_relevel() converts characters to factors
  mutate(SUSPECT_AGE = fct_relevel(SUSPECT_AGE,
                                  "UNDER 18",
                                   "18-25",
                                   "26-30",
                                   "31-40",
                                   "41-50",
                                   "51-60",
                                   "61-70",
                                   "OVER 70"))
# SUSPECT_AGE is now a factor that we reordered!
age_releveled %>% count(SUSPECT_AGE)
# A tibble: 8 x 2
  SUSPECT_AGE     n
  <fct>       <int>
1 UNDER 18     1068
2 18-25        2652
3 26-30        1724
4 31-40        2031
5 41-50         899
6 51-60         418
7 61-70         137
8 OVER 70        33
```



RECODE LEVELS WITH **fct_recode()**

```
age_recoded <- age_releveled %>%
  mutate(
    SUSPECT_AGE = fct_recode(
      SUSPECT_AGE,
      #  new = old
      "< 18" = "UNDER 18",
      "> 70" = "OVER 70"
    )
  )
```



YOUR TURN!

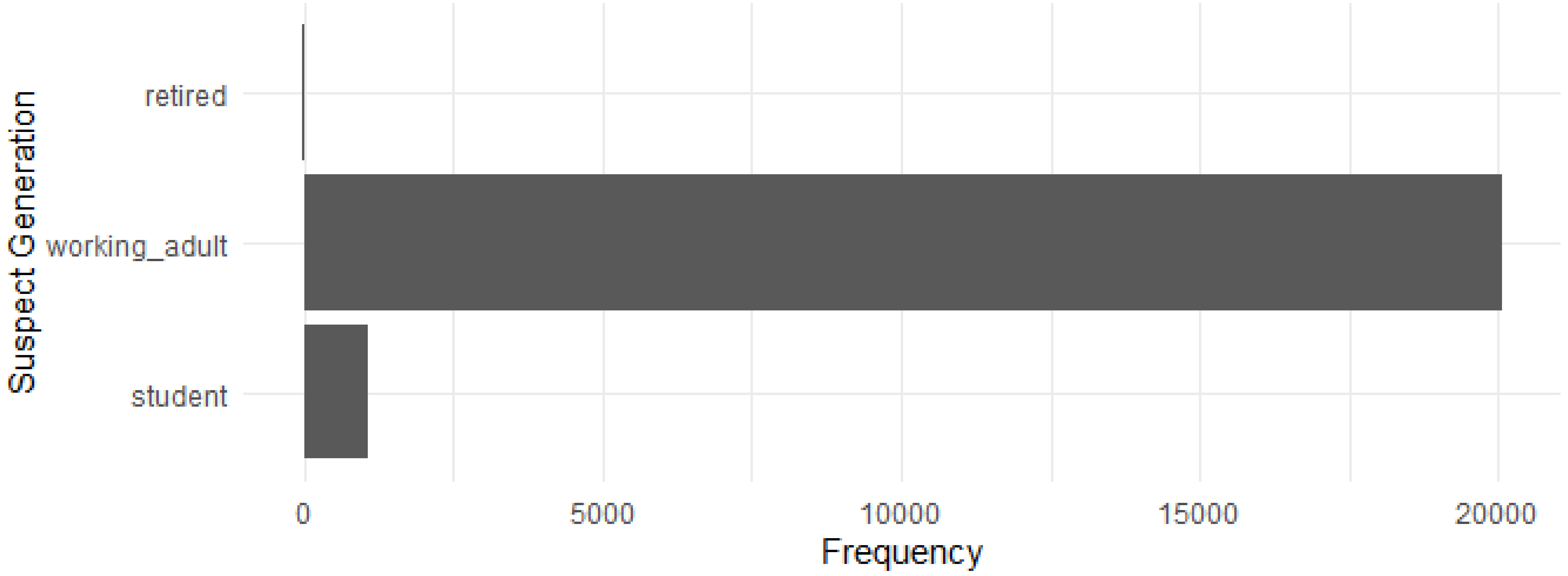
Using our *crimes* data set, fill in the blanks (in the provided R script) to:

1. *Create a variable called suspect_generation where the suspect's age*
   - *From zero to 18 is "student"*
   - *From 18 to 60 is "working_adult"*
   - *60+ is "retired"*
2. *Reorder the suspect_generation variable in student/working_adult/retired order*
3. *Make a bar chart to show the distribution of the suspect_generation variable*

SOLUTION

```
crime %>%
  mutate(suspect_generation = case_when(SUSPECT_AGE == "UNDER 18" ~ "student",
                                        SUSPECT_AGE == "OVER 70"  ~ "retired",
                                        is.na(SUSPECT_AGE)        ~ NA_character_,
                                        TRUE                      ~ "working_adult"),
         suspect_generation = fct_relevel(suspect_generation,
                                          "student", "working_adult", "retired")) %>%
  ggplot(aes(x = suspect_generation)) +
    geom_bar() +
    labs(x = "Suspect Generation",
         y = "Frequency") +
    coord_flip() +
    theme_minimal()
```

SOLUTION



Suspect Generation
retired
working_adult
student
0
5000
10000
15000
20000
Frequency

COLLAPSE FACTORS WITH **fct_collapse()**

There are 7 distinct values for DAYOFWEEK…

```
crime %>%
  distinct(DAYOFWEEK)

# A tibble: 8 x 1
  DAYOFWEEK
  <chr>
1 SATURDAY
2 THURSDAY
3 TUESDAY
4 WEDNESDAY
5 SUNDAY
6 FRIDAY
7 MONDAY
8 NA
```

…but we can collapse these into 2 levels.

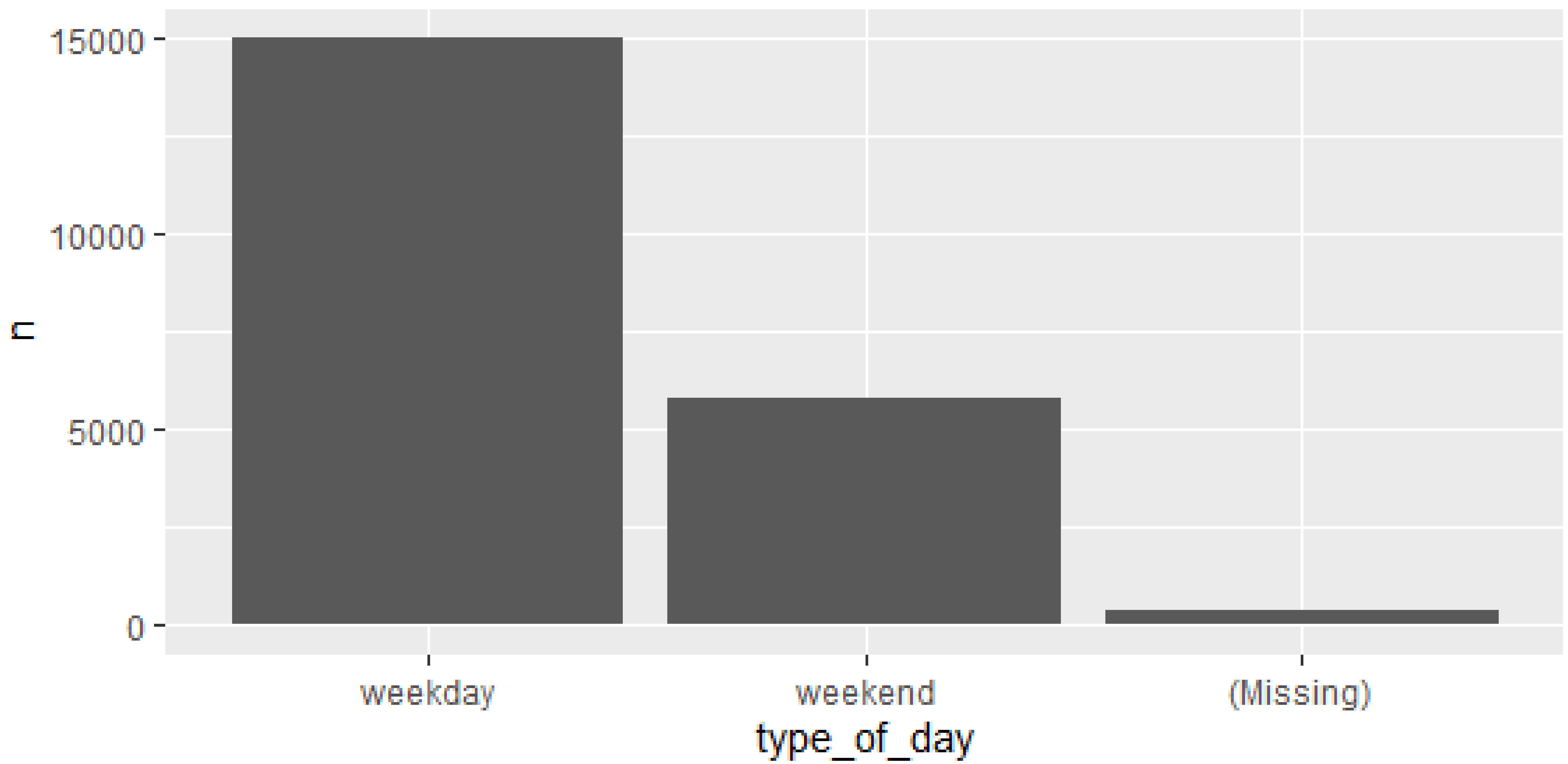
```
day <- crime %>%
  mutate(
    type_of_day = fct_collapse(
      DAYOFWEEK,
      weekday = c("MONDAY", "TUESDAY",
                  "WEDNESDAY", "THURSDAY",
                  "FRIDAY"),
      weekend = c("SATURDAY", "SUNDAY")
    ),
    # give missing values an explicit factor level
    # ensure they appear in summaries and on plots
    type_of_day = fct_explicit_na(type_of_day)
  )
```

COLLAPSE FACTORS WITH **fct_collapse()**

Our new graph reflects the changed levels!

```
day %>% count(type_of_day)

# A tibble: 3 x 2
  type_of_day     n
  <fct>       <int>
1 weekday     14982
2 weekend      5808
3 (Missing)     363
```



TEMPORARILY REORDER FACTORS

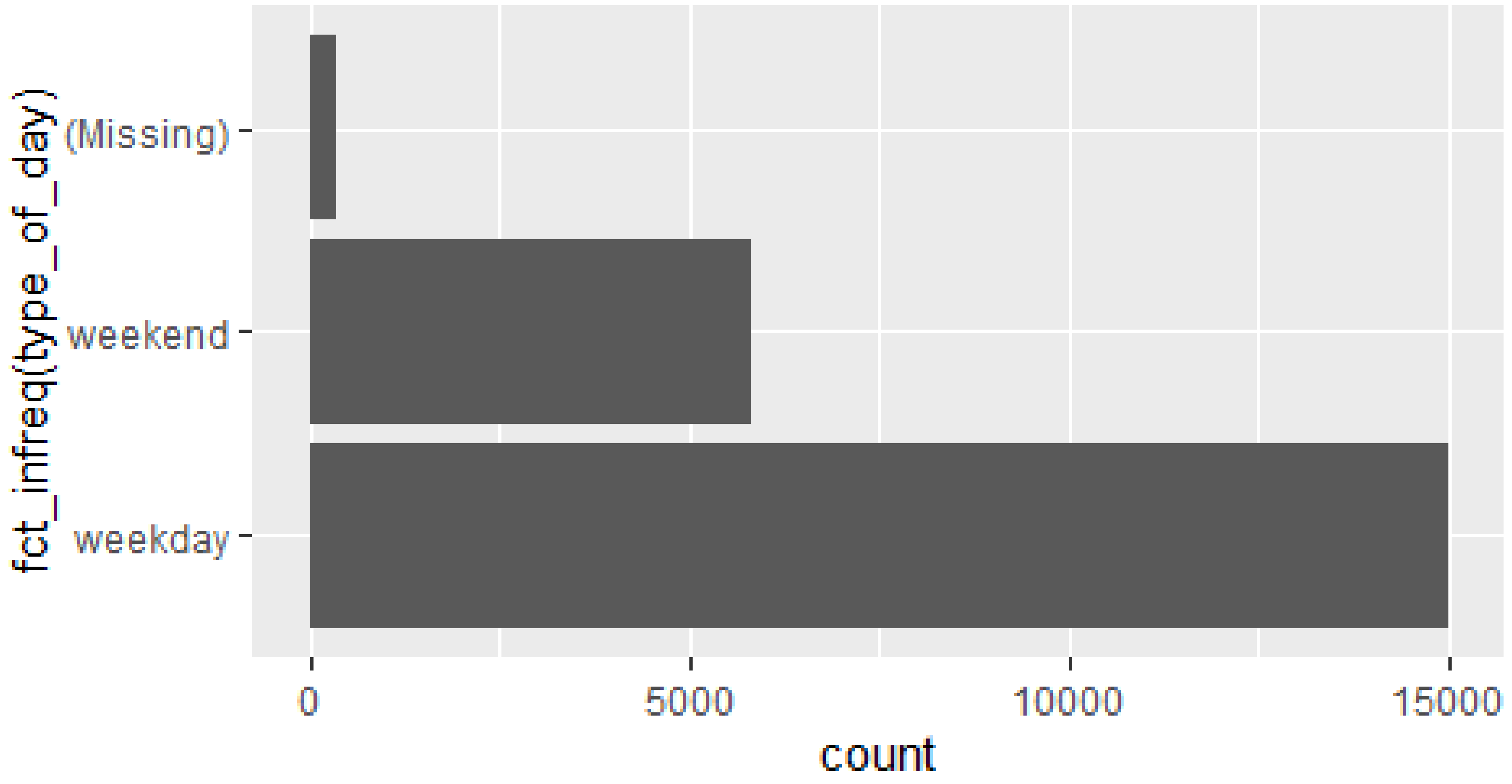
Place certain **forcats** functions inside **ggplot()** calls to temporarily reorder factors without permanently altering levels.

```
crime %>%
  distinct(DAYOFWEEK)

# A tibble: 8 x 1
  DAYOFWEEK
  <chr>
1 SATURDAY
2 THURSDAY
3 TUESDAY
4 WEDNESDAY
5 SUNDAY
6 FRIDAY
7 MONDAY
8 NA
```

fct_infreq() orders by frequency

```
day %>%
  ggplot(aes(x = fct_infreq(type_of_day))) +
    geom_bar() +
    coord_flip()
```



TEMPORARILY REORDER FACTORS

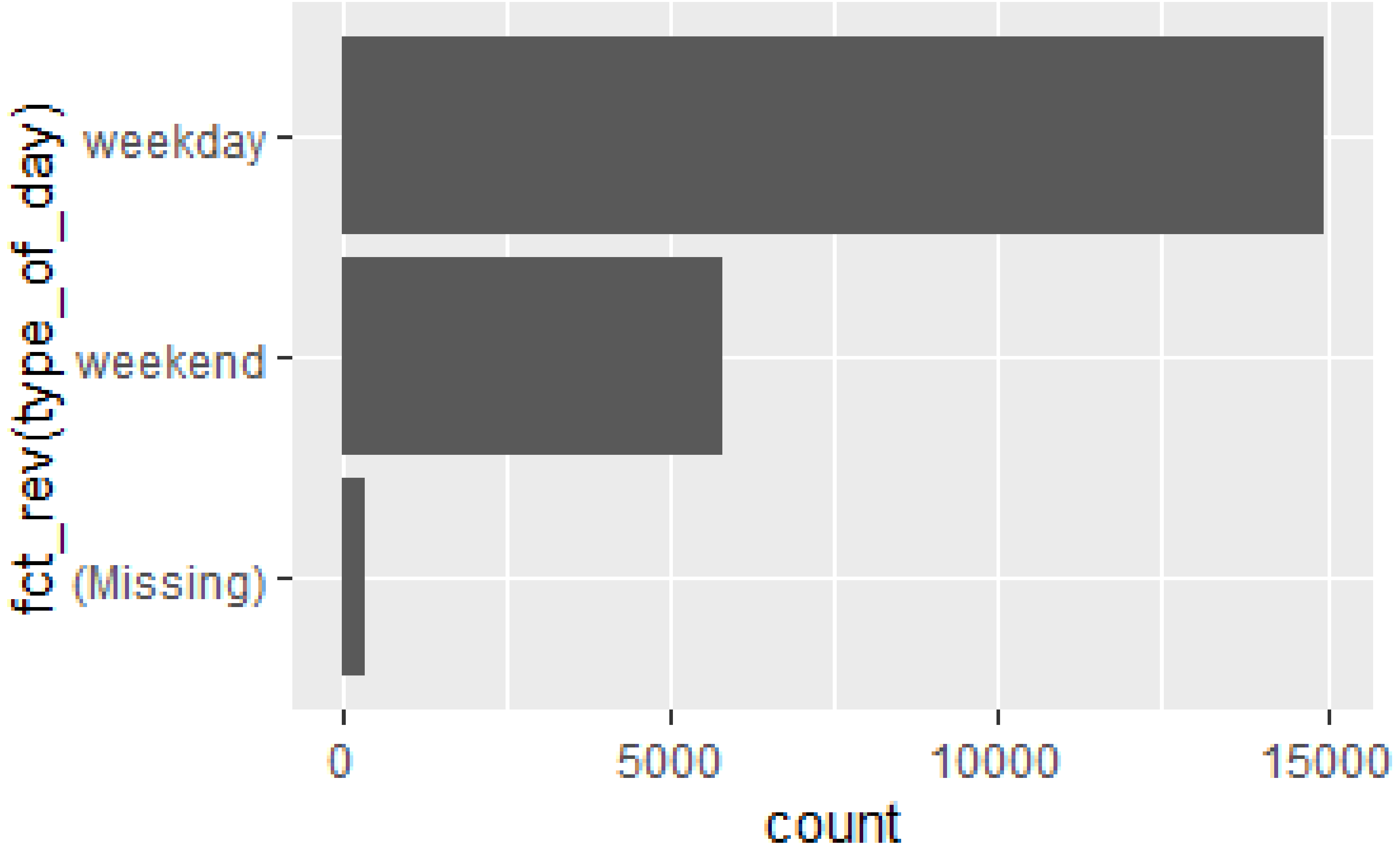
Place certain **forcats** functions inside **ggplot()** calls to temporarily reorder factors without permanently altering levels.

```
crime %>%
  distinct(DAYOFWEEK)

# A tibble: 8 x 1
  DAYOFWEEK
  <chr>
1 SATURDAY
2 THURSDAY
3 TUESDAY
4 WEDNESDAY
5 SUNDAY
6 FRIDAY
7 MONDAY
8 NA
```

fct_rev() reverses the order of factor levels

```
day %>%
  ggplot(aes(x = fct_rev(type_of_day))) +
   geom_bar() +
   coord_flip()
```



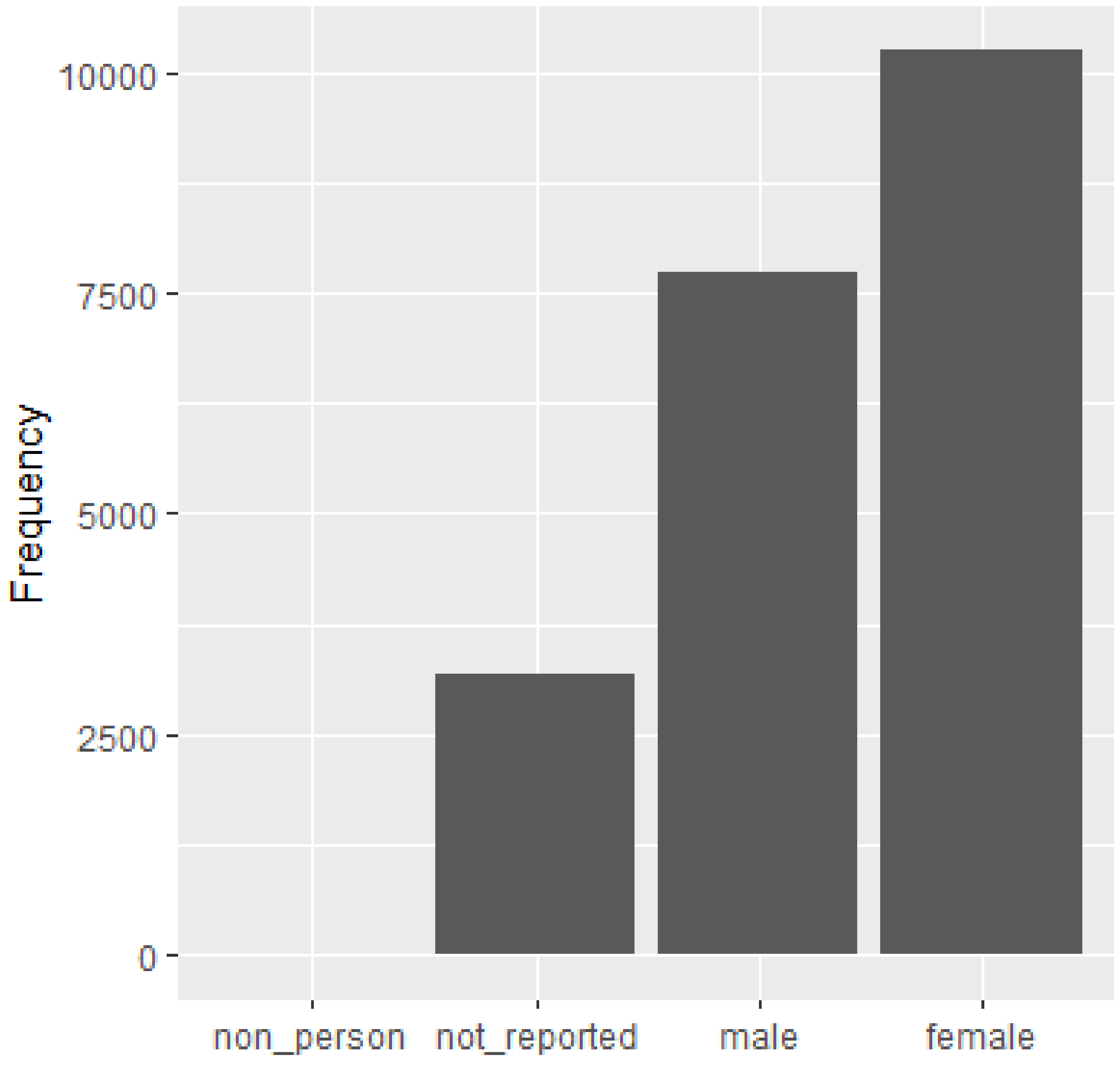
YOUR TURN!

Using our *crimes* data set and the VICTIM_GENDER variable, fill in the blanks (in the provided R script) to:

1. *Give missing values an explicit factor level so they appear in summaries and on plots.*
2. *Collapse factor levels into "female", "male", "non_person", and "not_reported".*
3. *Count the number of victim per reported gender.*
4. *Use fct_reorder() to make a plot (read documentation!).*

SOLUTION

```
crime %>%
 transmute(
  VICTIM_GENDER = fct_explicit_na(VICTIM_GENDER),
  VICTIM_GENDER = fct_collapse(
   VICTIM_GENDER,
   female = c("FEMALE", "F - FEMALE"),
   male = c("MALE", "M - MALE"),
   non_person = "NON-PERSON (BUSINESS",
   not_reported = c("(Missing)", "UNKNOWN")
   )
  ) %>%
 count(VICTIM_GENDER) %>%
 ggplot(aes(x = fct_reorder(VICTIM_GENDER, n),
        y = n)) +
  geom_col() +
  labs(x = NULL,
     y = "Frequency")
```



FOR MORE INFORMATION

https://forcats.tidyverse.org/

03/ DATES AND TIMES

lubridate FUNCTIONS AND COMMON TASKS

Sorry, but lubridate functions don't have a common prefix.

Common tasks we're covering

- Parse strings into dates/times
- Extract components of dates
- Adding/subtracting periods and durations
- ... and more (that we're not covering)

CREATING DATE/TIME VALUES AND VARIABLES

Parse strings into dates and times (letters dictate order) with functions like these:

- **ymd()**
- **dmy_h()**
- **ydm_hm()**
- **mdy_hms()**

… and many more functions!

lubridate handles many string formats!

```
# year, month, day
ymd("2019-08-20")
[1] "2019-08-20“

# some parsing functions allow unquoted numbers
ymd(20190820)
[1] "2019-08-20”

# day, month, year, hour
dmy_h("20/08/2019 14")
[1] "2019-08-20 14:00:00 UTC“

# year, day, month, hour, minute
ydm_hm("2019/20/08 07:20")
[1] "2019-08-20 07:20:00 UTC“

# month, day, year, hour, minute, second
mdy_hms("August 20, 2019 10:12:32")
[1] "2019-08-20 10:12:32 UTC"
```

EXTRACT COMPONENTS OF DATES

Boolean components

```
# check if datetime in am
am("2019-08-20 17:00:00")
[1] FALSE

# check for daylight savings time
dst(now())
[1] FALSE

# check for leap year (requires date input)
x <- as_date("2019-08-20")
leap_year(x)
[1] FALSE
```

Numeric components

```
# extract year
year("2019-08-20")
[1] 2019

# extract full weekday name
wday("2019-08-20", label = TRUE, abbr = TRUE)
[1] Tue
Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat

# extract hour
hour("2019-08-20 02:42")
[1] 2

# extract calendar year quarter
quarter("2019-08-20")
[1] 3
```

YOUR TURN!

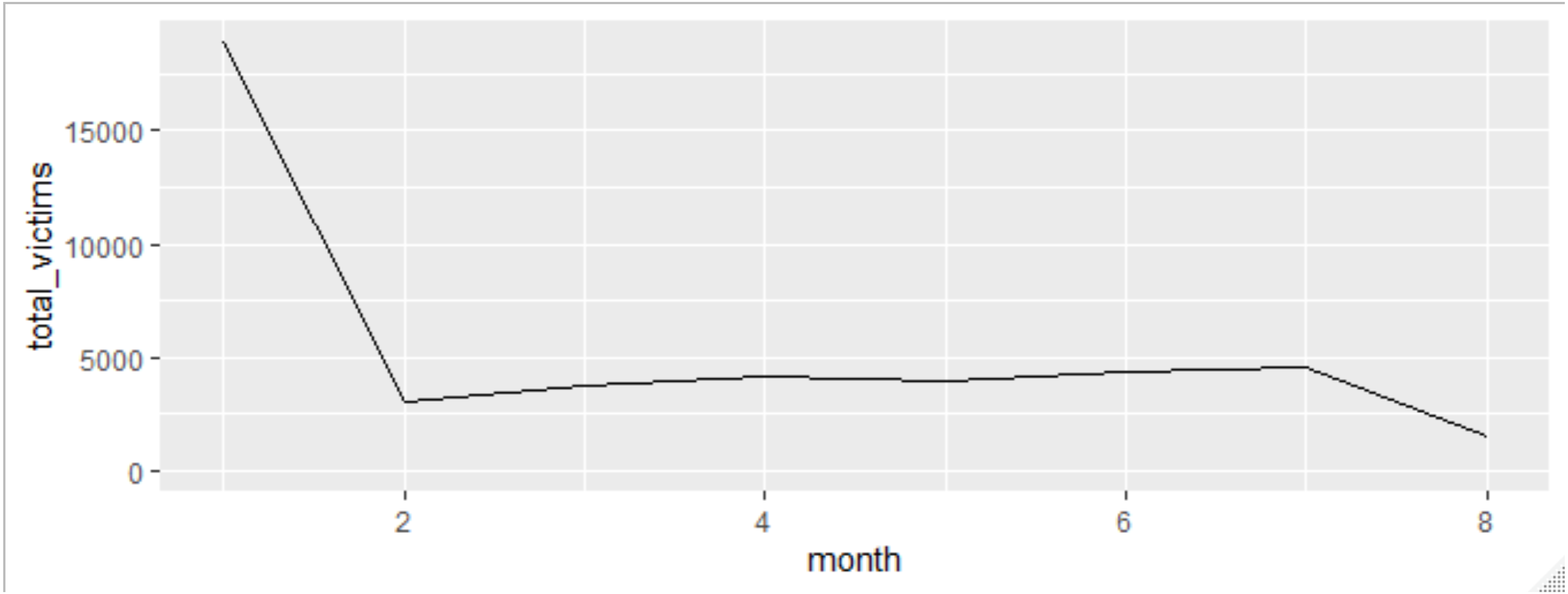
The Cincinnati Police Department has a question:
Do certain months have more victims than other months?

Using our *crimes* data set, fill in the blanks and asterisks (in the provided R script) and read the comments to answer this question.

SOLUTION

```
crime %>%
  # convert the DATE_REPORTED variable into
  # a datetime variable showing the month, day, year, hour, minute
  mutate(DATE_REPORTED = mdy_hm(DATE_REPORTED),
         # create a month variable by extracting the month
         # from the DATE_REPORTED variable
         month = month(DATE_REPORTED)) %>%
  # what should you group by?
  group_by(month) %>%
  # we need a total_victims statistic
  summarize(total_victims = sum(TOTALNUMBERVICTIMS, na.rm = TRUE)) %>%
  # create a line graph to show change over time
  ggplot(aes(x = month, y = total_victims)) +
    geom_line()
```

SOLUTION



DURATIONS

How old is Surge? R stores this calculation as a difftime object with the attribute naming the units.

```
# Thanks Wikipedia!
(surge_age <- today() - ymd(19970727))

Time difference of 8148 days
```

lubridate can store this information as a **duration** which always uses seconds, avoiding ambiguity with different time units.

```
as.duration(surge_age)

[1] "703987200s (~22.31 years)"
```

WORKING WITH DURATIONS

Function to create durations

(they all begin with *d*)

```
dseconds(20)
[1] "20s"

dminutes(c(11, 525600))
[1] "660s (~11 minutes)"
[2] "31536000s (~52.14 weeks)"

dweeks(1:4)
[1] "604800s (~1 weeks)"  "1209600s (~2 weeks)"
[3] "1814400s (~3 weeks)" "2419200s (~4 weeks)"
```

... and many more functions!

Add and multiply durations

```
3 * dhours(1)
[1] "10800s (~3 hours)"

dyears(2) + dweeks(3) + dhours(1)
[1] "64890000s (~2.06 years)"
```

Add and subtract durations involving days

```
today() - dyears(2)
[1] "2017-11-18"
```

WHERE DURATIONS FAIL US

Leap years

```
(five_somewhere <- ymd_hms("2016-01-01 17:00:00"))
[1] "2016-01-01 17:00:00 UTC"

five_somewhere + dyears(1)
[1] "2016-12-31 17:00:00 UTC"
```

Daylight saving time

```
(hashtag_fall <- ymd_hms("2019-11-02 15:00:00", tz = "America/New_York"))
[1] "2019-11-02 15:00:00 EDT"

hashtag_fall + ddays(1)
[1] "2019-11-03 14:00:00 EST"
```

PERIODS TO SAVE THE DAY

lubridate also uses periods—time spans that are not fixed lengths but work with “human” times

```
hashtag_fall
[1] "2019-11-02 15:00:00 EDT"

hashtag_fall + days(1)
[1] "2019-11-03 15:00:00 EST"
```

Examples of creating periods (no common prefix)

```
seconds(20)
[1] "20S"
minutes(c(11, 525600))
[1] "11M 0S"     "525600M 0S"
weeks(1:4)
[1] "7d 0H 0M 0S"  "14d 0H 0M 0S" "21d 0H 0M 0S" "28d 0H 0M 0S"
```

ADDING AND MULTIPLYING PERIODS

Add and multiply periods

```
4 * (years(2) + minutes(3))
[1] "8y 0m 0d 0H 12M 0S"

days(6) + minutes(600) + seconds(3)
[1] "6d 0H 600M 3S"
```

Add periods to dates

```
# leap year
five_somewhere + dyears(1)
[1] "2016-12-31 17:00:00 UTC"
five_somewhere + years(1)
[1] "2017-01-01 17:00:00 UTC"

# daylight saving time
hashtag_fall + ddays(1)
[1] "2019-11-03 14:00:00 EST"
hashtag_fall + days(1)
[1] "2019-11-03 15:00:00 EST"
```

FOR MORE INFORMATION

Other tasks with **lubridate**:

- Accounting for and changing time zones
- Determining if two time intervals overlap

https://lubridate.tidyverse.org/

FOR THE REST OF TODAY…

Spend the last 30-45 minutes of today’s class session working through the *Session 4 Midterm Project* .pdf file with your group members.