COMP824 2023 Week 8 Special Types of Data

Department of Mathematical Sciences
Auckland University of Technology

Overview

Strings

Dates and Times

Factors

Reading

Chapter 14 - 16 Wickham and Grolemund (2020), R for Data Science https://r4ds.had.co.nz/

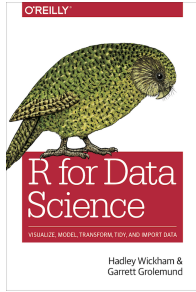
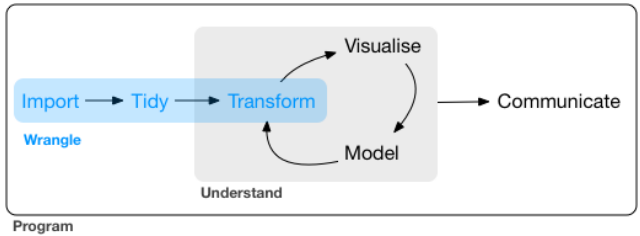


Figure 1: http://r4ds.had.co.nz/

The Process of Analytics



Learning objectives

- Understand key properties of special data types
- Use `stringr` to wrangle string variables
- Understand what regular expressions are and apply them when wrangling string variables
- Use `forcats` to wrangle factor variables
- Use `lubridate` to wrangle date and time variables

String Basics: Defining strings

```
string1 <- "This is a string"
string2 <- 'Include a "quote" inside a string with single quotes'
```

Special Characters: \

The \ character is used to “escape” the usual use of a character.

```
double_quote <- "\"" # or '"'
single_quote <- '\'' # or "'"
writeLines(double_quote)
```

```
"
```

```
writeLines(single_quote)
```

```
'
```

Special Characters: \

The \ character is also used for special commands

```
newline <- "This is the first line\n This is the second line"
writeLines(newline)
```

```
This is the first line
 This is the second line
```

Special Characters

Symbols/characters from other languages

```
(x <- "\u00b5" )
```

```
[1] "µ"
```

Combine multiple strings in a character vector

```
c("one", "two", "three")
```

```
[1] "one"   "two"   "three"
```

More info:

```
?'"'
```

`stringr` package

Figure 3: https://www.tidyverse.org/

```
library(stringr)
```

- Functions in the `stringr` package all start with `str_`.
- Examples: `str_length()`, `str_c()`, `str_sub()`
- Refer to Chapter 14 Wickham and Grolemund (2020) for further details.

stringr functions: str_length()

- `str_length()` - number of characters in a string

```
x <- c("STAT702", "R is the best", NA)
str_length(x)
```

```
[1]  7 13 NA
```

stringr functions: str_c()

str_c() - join strings (equivalent to paste())

- Join Strings

```
str_c("x", "y", "z")
str_c("x", "y", "z", sep = "*")
str_c("prefix-", c("a", "b", "c"), "-suffix")
```

```
[1] "xyz"
[1] "x*y*z"
[1] "prefix-a-suffix" "prefix-b-suffix" "prefix-c-suffix"
```

- Collapse a vector of strings

```
str_c(c("x", "y", "z"), collapse = ", ")
```

```
[1] "x, y, z"
```

stringr functions: str_sub()

- str_sub() - subset strings

```
x <- c("Apple", "BAnana", "Pear", "Eggplant")

#string, start_position, end_position
str_sub(x, 1, 3)  # from start
```

```
[1] "App" "BAn" "Pea" "Egg"
```

```
str_sub(x, -5, -1)  # from end
```

```
[1] "Apple" "Anana" "Pear"  "plant"
```

stringr functions: str_to_lower()

- str_to_lower() - change to lower case

```
str_to_lower(x)
```

```
[1] "apple"     "banana"    "pear"      "eggplant"
```

```
str_sub(x, 1, 1) <- str_to_lower(str_sub(x, 1, 1))
x
```

```
[1] "apple"     "bAnana"    "pear"      "eggplant"
```

stringr functions: str_sort()

- `str_sort()` - sort in alphabetical order (locale dependent)

```
str_sort(x, locale = "en") # English
```

```
[1] "apple"    "bAnana"   "eggplant" "pear"
```

```
str_sort(x, locale = "haw") # Hawaiian
```

```
[1] "apple"    "eggplant" "bAnana"   "pear"
```

stringr functions: str_view()

str_view(string, pattern)

```
x <- c("apple", "banana", "pear")
str_view(x, "an")
```

```
[2] | b<an><an>a
```

```
x <- c("apple", "banana", "pear")
str_view(x, "a")
```

```
[1] | <a>pple
[2] | b<a>n<a>n<a>
[3] | pe<a>r
```

Regular Expressions

Definition:

A regular expression, regex or regexp is a sequence of characters that define a search pattern.

Source: https://en.wikipedia.org/wiki/Regular_expression

Examples

- `.` wild card
- `[a-z]` all lowercase letters
- `\d` any digit

Regular Expressions - escape character

- Use \\ escape character to match special characters

```
str_view(c("abc", "a.c", "bef"), "a\\.c")
```

```
[2] | <a.c>
```

```
str_view(c("abc", "a.c", "bef"), "a[.]c")
```

```
[2] | <a.c>
```

```
x <- "a\\b"
writeLines(x)
```

```
a\b
```

```
str_view(x, "\\\\")
```

```
[1] | a<\>b
```

Anchors

- ^ - start of string
- $ - end of string

```
x <- c("apple", "banana", "pear")
str_view(x, "^a")  #starts with "a"
```

```
[1] | <a>pple
```

```
str_view(x, "a$")  #ends with "a"
```

```
[2] | banan<a>
```

Anchors - More examples

```
x <- c("apple pie", "apple", "apple cake")
str_view(x, "apple") # match string
```

```
[1] | <apple> pie
[2] | <apple>
[3] | <apple> cake
```

```
str_view(x, "^apple$") # match whole string
```

```
[2] | <apple>
```

Combining expressions

```
x
```

```
[1] "apple pie"  "apple"      "apple cake"
```

```
str_view(x, "apple|pie")  # apple or pie
```

```
[1] | <apple> <pie>
[2] | <apple>
[3] | <apple> cake
```

```
str_view(x, "pie$|cake$")  # ends with pie or cake
```

```
[1] | apple <pie>
[3] | apple <cake>
```

Combining expressions 2

```
x
```

```
[1] "apple pie"  "apple"      "apple cake"
```

```
str_view(x, "(pie|cake)$")  # ends with pie or cake
```

```
[1] | apple <pie>
[3] | apple <cake>
```

```
str_view(x, "[aeiou]")  # vowels
```

```
[1] | <a>ppl<e> p<i><e>
[2] | <a>ppl<e>
[3] | <a>ppl<e> c<a>k<e>
```

Combining expressions 3

```
x
```

```
[1] "apple pie"  "apple"      "apple cake"
```

```
str_view(x, "[^aeiou]")  # consonants
```

```
[1] | a<p><p><l>e< ><p>ie
[2] | a<p><p><l>e
[3] | a<p><p><l>e< ><c>a<k>e
```

```
str_view(c("grey", "gray", "white"), "gr(e|a)y")
```

```
[1] | <grey>
[2] | <gray>
```

Applications of Regular Expressions

- Get all R files in a directory

```
dir(pattern = "\\.R$")
```

```
[1] "mymake.R"
```

- List all files in a directory

```
list.files(path = "figs/", pattern = "\\.jpg$") %>%
  head(4)
```

```
[1] "archie.jpg"           "autlogo.jpg"
[3] "bigcar_smallcar.jpg" "bike_dog.jpg"
```

stringr functions: str_detect()

- Detect matches `str_detect()`

```
x <- c("apple", "banana", "pear")
str_detect(x, "e")
```

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

stringr functions: str_replace()

- Replace matches `str_replace()`

```
x <- c("apple", "pear", "banana")
str_replace(x, "[aeiou]", "-")
```

```
[1] "-pple"  "p-ar"   "b-nana"
```

```
str_replace_all(x, "[aeiou]", "-")
```

```
[1] "-ppl-"  "p--r"   "b-n-n-"
```

Example: Colours

```
colours <- c("red", "orange", "yellow", "green", "blue", "purple")
colour_match <- str_c(colours, collapse = "|")
colour_match
```

```
[1] "red|orange|yellow|green|blue|purple"
```

```
head(sentences, 4)
```

```
[1] "The birch canoe slid on the smooth planks."
[2] "Glue the sheet to the dark blue background."
[3] "It's easy to tell the depth of a well."
[4] "These days a chicken leg is a rare dish."
```

stringr functions: str_subset()

- Extract the entire element when a match occurs `str_subset()`

```
has_colour <- str_subset(sentences, colour_match)
head(has_colour, 4)
```

```
[1] "Glue the sheet to the dark blue background."
[2] "Two blue fish swam in the tank."
[3] "The colt reared and threw the tall rider."
[4] "The wide road shimmered in the hot sun."
```

Note the limitations of extracting using a string.

stringr functions: str_extract()

- Extract matches `str_extract()`

```
matches <- str_extract(has_colour, colour_match)
head(matches, 4)
```

```
[1] "blue" "blue" "red"  "red"
```

stringr functions: str_split()

```
sentences %>%
  head(3) %>%
  str_split(" ")
```

```
[[1]]
[1] "The"     "birch"   "canoe"   "slid"    "on"
[6] "the"     "smooth"  "planks."

[[2]]
[1] "Glue"        "the"         "sheet"       "to"
[5] "the"         "dark"        "blue"        "background."

[[3]]
[1] "It's"  "easy"  "to"    "tell"  "the"   "depth" "of"
[8] "a"     "well."
```

Advanced String Functions

For a more comprehensive set of tools for working with strings, try `stringi`.

```
library(stringi)
```

Dates and Times

Figure 4: https://www.tidyverse.org/

```
library(lubridate)
```

Refer to Chapter 16 Wickham and Grolemund (2020) for further details.

Describing Dates

- `m` month
- `d` day of month
- `M` minutes
- `y` year without century
- `Y` year with century

and more!

See `?parse_date_time` for a list.

Parsing dates `parse_date_time`

- Convert character string into a standard time format. Ignores “standard” separators.

```
x <- c("23-05-01", "23-05-02", "23/05/03")
parse_date_time(x, "ymd")
```

```
[1] "2023-05-01 UTC" "2023-05-02 UTC" "2023-05-03 UTC"
```

- Can add several formats

```
y <- c("Monday 1 May 2023", "2023/05/01 10:00")
parse_date_time(y, c("AdbY", "YmdHM"))
```

```
[1] "2023-05-01 00:00:00 UTC" "2023-05-01 10:00:00 UTC"
```

Parsing dates with `date()`: standard format (input)

```
(x <- tibble(
  date_original = c("2023-05-01", "2023-05-02", "2023-05-03"),
  temperature = c(18, 19, 17)
))
```

```
# A tibble: 3 x 2
  date_original temperature
  <chr>                <dbl>
1 2023-05-01              18
2 2023-05-02              19
3 2023-05-03              17
```

Parsing dates with `date()`: standard format (output)

```
x %>%
  mutate(
  date_new = date(date_original)
  )
```

```
# A tibble: 3 x 3
  date_original temperature date_new
  <chr>                 <dbl> <date>
1 2023-05-01               18 2023-05-01
2 2023-05-02               19 2023-05-02
3 2023-05-03               17 2023-05-03
```

Parsing dates with `date()`: inconsistent format 1

- Use `ymd` to parse

```
x <- tibble(
  date_original = c("23-05-01", "23-05-02", "23/05/03"),
  temperature = c(18, 19, 17) )

x %>%
  mutate(date_new = date(ymd(date_original))
         )
```

```
# A tibble: 3 x 3
  date_original temperature date_new
  <chr>                <dbl> <date>
1 23-05-01                18 2023-05-01
2 23-05-02                19 2023-05-02
3 23/05/03                17 2023-05-03
```

Parsing dates with `date()`: inconsistent format 2

```
x <- tibble(
  date_original = c("23-15-05", "23-16-05", "23/17/05"),
  temperature = c(18, 19, 17)
  )

x %>% mutate(
  date_new = date(ydm(date_original))
  )
```

```
# A tibble: 3 x 3
  date_original temperature date_new
  <chr>                <dbl> <date>
1 23-15-05                18 2023-05-15
2 23-16-05                19 2023-05-16
3 23/17/05                17 2023-05-17
```

ymd and ydm and others

- `ydm(x)` is equivalent to `parse_date_time(x, "ydm")`:
- Example:

```
x %>%
  mutate(date_new = date(ydm(date_original)))
```

```
# A tibble: 3 x 3
  date_original temperature date_new
  <chr>                <dbl> <date>
1 23-15-05                18 2023-05-15
2 23-16-05                19 2023-05-16
3 23/17/05                17 2023-05-17
```

```
x %>%
  mutate(date_new = date(parse_date_time(date_original, "ydm")))
```

- See `?ymd` for details of other functions.

Parsing dates with `date()`: multiple formats

```
x <- tibble(
  date_original = c("23-15-May", "23-16-May", "2023 May 17"),
  temperature = c(18, 19, 17))

x %>% mutate(
   date_new =  date(parse_date_time(date_original,
                                    c("ydb", "Ybd"))))
```

```
# A tibble: 3 x 3
  date_original temperature date_new
  <chr>                <dbl> <date>
1 23-15-May               18 2023-05-15
2 23-16-May               19 2023-05-16
3 2023 May 17             17 2023-05-17
```

Extracting parts of dates and times 1

- Extract date from a `dttm` variable

```
x <- tibble(
  date_original = c("10am 23-15-May", "11pm 23-16-May",
                    "2023 May 17 14:00"))
(xx <- x %>%  mutate(
      date_time_new =  parse_date_time(date_original,
                                       c("Ip ydb", "Ybd HM")),
      date_new = date(date_time_new)))
```

```
# A tibble: 3 x 3
  date_original     date_time_new       date_new
  <chr>             <dttm>              <date>
1 10am 23-15-May    2023-05-15 10:00:00 2023-05-15
2 11pm 23-16-May    2023-05-16 23:00:00 2023-05-16
3 2023 May 17 14:00 2023-05-17 14:00:00 2023-05-17
```

Extracting parts of dates and times 2

- Extract time from a `dttm` variable
- NB: `hms::as_hms` used here as there is an `hms` function in `lubridate`.

```
xx %>%
  mutate(
    time = hms::as_hms(date_time_new)
    )
```

```
# A tibble: 3 x 4
  date_original     date_time_new       date_new   time
  <chr>             <dttm>              <date>     <time>
1 10am 23-15-May    2023-05-15 10:00:00 2023-05-15 10:00
2 11pm 23-16-May    2023-05-16 23:00:00 2023-05-16 23:00
3 2023 May 17 14:00 2023-05-17 14:00:00 2023-05-17 14:00
```

Extracting parts of dates and times 3

- Extract day of week and month etc.

```
xx %>%
  select(-date_time_new) %>%
  mutate(
    month = month(date_new),
    wday = wday(date_new),
    wday_name = wday(date_new, label = TRUE)
   )
```

```
# A tibble: 3 x 5
  date_original     date_new   month  wday wday_name
  <chr>             <date>     <dbl> <dbl> <ord>
1 10am 23-15-May    2023-05-15     5     2 Mon
2 11pm 23-16-May    2023-05-16     5     3 Tue
3 2023 May 17 14:00 2023-05-17     5     4 Wed
```

Factors

Figure 5: https://www.tidyverse.org/

```
library(forcats)
```

Refer to Chapter 15 Wickham and Grolemund (2020) for further details.

Two reasons why are factors needed

- Sorting

```
x1 <- c("Dec", "Apr", "Jan", "Mar")
sort(x1)
```

```
[1] "Apr" "Dec" "Jan" "Mar"
```

- Typos

```
x2 <- c("Dec", "Apr", "Jam", "Mar")
```

Creating factors 1

```r
x1 <- c("Dec", "Apr", "Jan", "Mar")

month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
)

(y1 <- factor(x1, levels = month_levels))
```

```
[1] Dec Apr Jan Mar
12 Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct ... Dec
```

```r
sort(y1)
```

```
[1] Jan Mar Apr Dec
12 Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct ... Dec
```

Creating factors 2

- Values that don't match levels are NA

```
x2
```

```
[1] "Dec" "Apr" "Jam" "Mar"
```

```
factor(x2, levels = month_levels)
```

```
[1] Dec  Apr  <NA> Mar
12 Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct ... Dec
```

Creating factors with `forcats`

```
forcats::fct(x1, levels = month_levels)
```

```
[1] Dec Apr Jan Mar
12 Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct ... Dec
```

- Values that don't match levels give error

```
(y2 <- forcats::fct(x2, levels = month_levels))
```

Creating factors in tibbles 1

```
(x <- tibble(m = c("Dec", "Apr", "Jan", "Mar", "Mar") ) %>%
  arrange(m))
```

```
# A tibble: 5 x 1
  m
  <chr>
1 Apr
2 Dec
3 Jan
4 Mar
5 Mar
```

Creating factors in tibbles 2

```
(x <- x %>%
  mutate(m_fct = fct(m, levels = month_levels))%>%
  arrange(m_fct))
```

```
# A tibble: 5 x 2
  m     m_fct
  <chr> <fct>
1 Jan   Jan
2 Mar   Mar
3 Mar   Mar
4 Apr   Apr
5 Dec   Dec
```

Creating factors using `read_csv`

```
csv <- "
month,value
Jan,12
Feb,56
Mar,12"

df <- read_csv(csv,
         col_types = cols(month = col_factor(month_levels)))
df$month
```

```
[1] Jan Feb Mar
12 Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct ... Dec
```

Identifying factor levels

```
levels(x$m_fct)
```

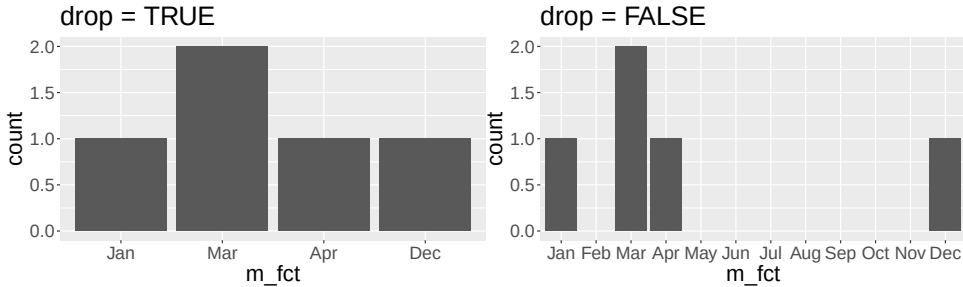
```
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep"
[10] "Oct" "Nov" "Dec"
```

```
x %>% count(m_fct)
```

```
# A tibble: 4 x 2
  m_fct     n
  <fct> <int>
1 Jan       1
2 Mar       2
3 Apr       1
4 Dec       1
```

Plots with factor levels

```
p1 <- ggplot(data = x) + geom_bar(aes(m_fct)) +
                         ggtitle("drop = TRUE")
p2 <- ggplot(data = x) + geom_bar(aes(m_fct)) +
                         scale_x_discrete(drop = FALSE) +
                         ggtitle("drop = FALSE")
cowplot::plot_grid(p1, p2)
```



Example: USA General Social Survey

- Contained in `forcats` package
- `forcats::gss_cat`

```
gss_cat
```

```
# A tibble: 21,483 x 9
   year marital         age race  rincome partyid relig denom
  <int> <fct>         <int> <fct> <fct>   <fct>   <fct> <fct>
1  2000 Never marri~    26 White $8000 ~ Ind,ne~ Prot~ Sout~
2  2000 Divorced        48 White $8000 ~ Not st~ Prot~ Bapt~
3  2000 Widowed         67 White Not ap~ Indepe~ Prot~ No d~
# ... with 21,480 more rows, and 1 more variable:
#   tvhours <int>
```

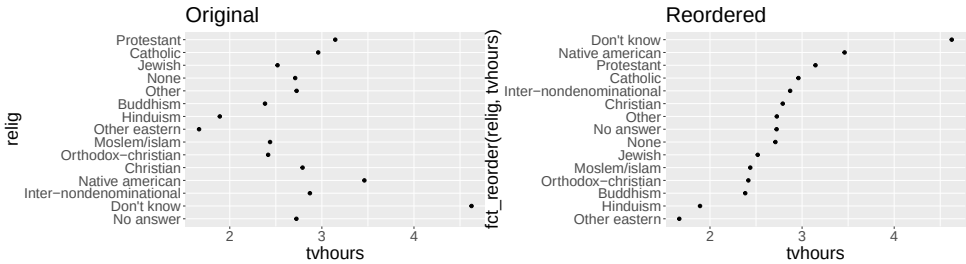
Example: Religion vs TV Hours

```r
(relig_summary <- gss_cat %>%
  group_by(relig) %>%
  summarise(
    age = mean(age, na.rm = TRUE),
    tvhours = mean(tvhours, na.rm = TRUE),
    n = n()
  ))
```

```
# A tibble: 15 x 4
  relig                      age tvhours     n
  <fct>                    <dbl>   <dbl> <int>
1 No answer                 49.5    2.72    93
2 Don't know                35.9    4.62    15
3 Inter-nondenominational   40.0    2.87   109
# ... with 12 more rows
```

Modifying Factor Order `fct_reorder`

```
p1 <- ggplot(relig_summary,
             aes(tvhours, relig)) + geom_point() +
      ggtitle("Original")
p2 <- ggplot(relig_summary,
             aes(tvhours, fct_reorder(relig, tvhours))) +
      geom_point() +  ggtitle("Reordered")
cowplot::plot_grid(p1, p2)
```



Example: Income Levels

```
levels(gss_cat$rincome)
```

```
 [1] "No answer"      "Don't know"     "Refused"
 [4] "$25000 or more" "$20000 - 24999" "$15000 - 19999"
 [7] "$10000 - 14999" "$8000 to 9999"  "$7000 to 7999"
[10] "$6000 to 6999"  "$5000 to 5999"  "$4000 to 4999"
[13] "$3000 to 3999"  "$1000 to 2999"  "Lt $1000"
[16] "Not applicable"
```

Example: Age vs Income

```
(rincome_summary <- gss_cat %>%
  group_by(rincome) %>%
  summarise(
    age = mean(age, na.rm = TRUE),
    tvhours = mean(tvhours, na.rm = TRUE),
    n = n()   ))
```

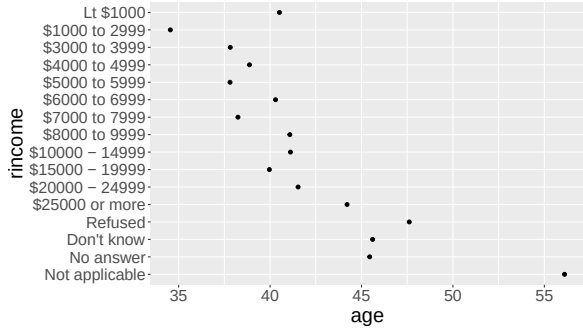
```
# A tibble: 16 x 4
  rincome       age tvhours     n
  <fct>       <dbl>   <dbl> <int>
1 No answer    45.5    2.90   183
2 Don't know   45.6    3.41   267
3 Refused      47.6    2.48   975
# ... with 13 more rows
```

- Don’t want to reorder income as category order is meaningful

Modifying Factor Order Selectively `fct_relevel`

```
rincome_summary <- rincome_summary %>%
  mutate(rincome = fct_relevel(rincome, "Not applicable"))

ggplot(rincome_summary, aes(age, rincome)) +
  geom_point()
```



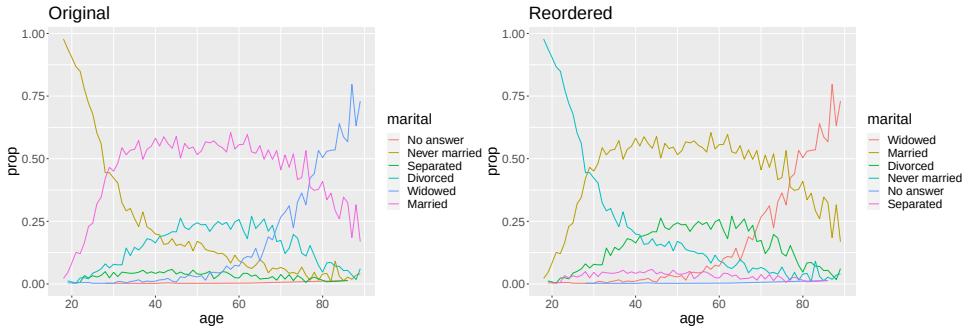
Example: Marital Status and Age

```
(by_age <- gss_cat %>%
  filter(!is.na(age)) %>%
  count(age, marital) %>%
  group_by(age) %>%
  mutate(prop = n / sum(n)))
```

```
# A tibble: 351 x 4
# Groups:   age [72]
    age marital           n   prop
  <int> <fct>         <int>  <dbl>
1    18 Never married    89 0.978
2    18 Married           2 0.0220
3    19 Never married   234 0.940
# ... with 348 more rows
```

Modifying Factor Order for a Line Plot

```
p1 <- ggplot(by_age, aes(age, prop, colour = marital)) +
      geom_line(na.rm = TRUE) + ggtitle("Original")
p2 <- ggplot(by_age, aes(age, prop,
                         colour = fct_reorder2(marital, age, prop))) +
      geom_line() + labs(colour = "marital") + ggtitle("Reordered")
cowplot::plot_grid(p1, p2)
```

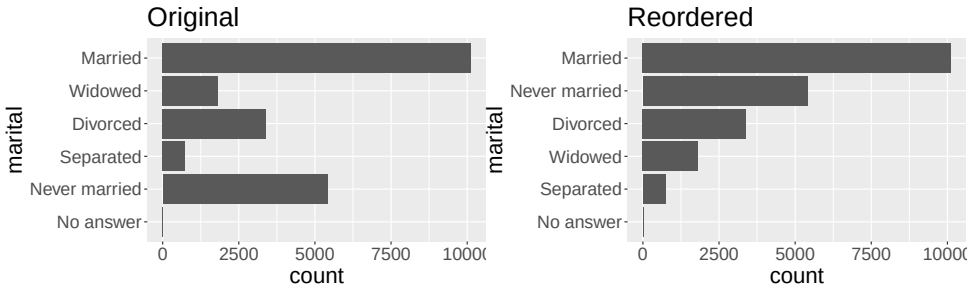


Modifying Factor Order for a Bar Plot

```
p1 <- gss_cat %>%
  mutate(marital = marital) %>%
  ggplot(aes(y=marital)) +
    geom_bar() +
  ggtitle("Original")

p2 <- gss_cat %>%
  mutate(marital = marital %>% fct_infreq() %>% fct_rev()) %>%
  ggplot(aes(y=marital)) +
    geom_bar() +
  ggtitle("Reordered")
```

Modifying Factor Order for a Bar Plot



Summary

Strings

Dates and Times

Factors

Learning objectives

- Understand key properties of special data types
- Use `stringr` to wrangle string variables
- Understand what regular expressions are and apply them when wrangling string variables
- Use `forcats` to wrangle factor variables
- Use `lubridate` to wrangle date and time variables

References

Wickham, Hadley, and Garrett Grolemund. 2020. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*.