03 - String Processing Comprehension Check

Gabriele Mineo - Harvard Data Science Professional

String Parsing

Question 1

Which of the following is NOT an application of string parsing?

- Removing unwanted characters from text.
- Extracting numeric values from text.
- Formatting numbers and characters so they can easily be displayed in deliverables like papers and presentations. [X]
- Splitting strings into multiple values.

Defining Strings: Single and Double Quotes and How to Escape

Question 1

Which of the following commands would not give you an error in R?

- `cat(" LeBron James is 6'8\" ")` [X]
- `cat(' LeBron James is 6'8" ')`
- `cat(` LeBron James is 6'8" `)`
- `cat(" LeBron James is 6\'8" ")`

stringr Package

Question 1

Which of the following are advantages of the stringr package over string processing functions in base R? Select all that apply.

- Base R functions are rarely used for string processing by data scientists so it's not worth learning them.
- Functions in stringr all start with "str_", which makes them easy to look up using autocomplete. [X]
- Stringr functions work better with pipes. [X]
- The order of arguments is more consistent in stringr functions than in base R. [X]

Case Study 1: US Murders Data

Question 1

You have a dataframe of monthly sales and profits in R

```
> head(dat)
# A tibble: 5 x 3
  Month    Sales    Profit
  <chr>    <chr>    <chr>
    January     $128,568    $16,234
    February    $109,523    $12,876
    March       $115,468    $17,920
```

```
April       $122,274    $15,825
May         $117,921    $15,437
```

Which of the following commands could convert the sales and profits columns to numeric? Select all that apply.

- `dat %>% mutate_at(2:3, parse_number)` [X]
- `dat %>% mutate_at(2:3, as.numeric)`
- `dat %>% mutate_all(parse_number)`
- `dat %>% mutate_at(2:3, funs(str_replace_all(., c("\\$|,"), "")))` `%>%mutate_at(2:3, as.numeric)` [X]

Case Study 2: Reported Heights

Question 1

In the video, we use the function not_inches to identify heights that were incorrectly entered

```
not_inches <- function(x, smallest = 50, tallest = 84) {
  inches <- suppressWarnings(as.numeric(x))
  ind <- is.na(inches) | inches < smallest | inches > tallest
  ind
}
```

In this function, what TWO types of values are identified as not being correctly formatted in inches?

- Values that specifically contain apostrophes (‘), periods (.) or quotations (“).
- Values that result in NA’s when converted to numeric [X]
- Values less than 50 inches or greater than 84 inches [X]
- Values that are stored as a character class, because most are already classed as numeric.

Question 2

Which of the following arguments, when passed to the function `not_inches`, would return the vector `c(FALSE)`?

- `c(175)`
- `c("5’8\"")`
- `c(70)` [X]
- `c(85) (the height of Shaquille O'Neal in inches)`

Question 3

Our function `not_inches` returns the object `ind`. Which answer correctly describes ind?

- `ind` is a logical vector of TRUE and FALSE, equal in length to the vector `x` (in the arguments list). TRUE indicates that a height entry is incorrectly formatted. [X]
- `ind`is a logical vector of TRUE and FALSE, equal in length to the vector `x`(in the arguments list). TRUE indicates that a height entry is correctly formatted.
- `ind` is a data frame like our `reported_heights` table but with an extra column of TRUE or FALSE. TRUE indicates that a height entry is incorrectly formatted.
- `ind` is a numeric vector equal to `reported_heights$heights` but with incorrectly formatted heights replaced with NAs.

Regex

Question 1

Given the following code

```
> s
[1] "70"       "5 ft"     "4'11"     ""         "."        "Six feet"
```

What `pattern` vector yields the following result?

```
str_view_all(s, pattern)
70
5 ft
4'11
.
Six feet
```

- `pattern <- "\\d|ft"` [X]
- `pattern <- "\d|ft"`
- `pattern <- "\\d\\d|ft"`
- `pattern <- "\\d|feet"`

Character Classes, Anchors, and Qualifiers

Question 1

You enter the following set of commands into your R console. What is your printed result?

```
> animals <- c("cat", "puppy", "Moose", "MONKEY")
> pattern <- "[a-z]"
> str_detect(animals, pattern)
```

- TRUE
- TRUE TRUE TRUE TRUE
- TRUE TRUE TRUE FALSE [X]
- TRUE TRUE FALSE FALSE

Question 2

You enter the following set of commands into your R console. What is your printed result?

```
> animals <- c("cat", "puppy", "Moose", "MONKEY")
> pattern <- "[A-Z]$"
> str_detect(animals, pattern)
```

- FALSE FALSE FALSE FALSE
- FALSE FALSE TRUE TRUE
- FALSE FALSE FALSE TRUE [X]
- TRUE TRUE TRUE FALSE

Question 3

You enter the following set of commands into your R console. What is your printed result?

```
> animals <- c("cat", "puppy", "Moose", "MONKEY")
> pattern <- "[a-z]{4,5}"
> str_detect(animals, pattern)
```

- FALSE TRUE TRUE FALSE [X]
- TRUE TRUE FALSE FALSE
- FALSE FALSE FALSE TRUE
- TRUE TRUE TRUE FALSE

Search and Replace with Regex

Question 1

Given the following code

animals <- c(“moose”, “monkey”, “meerkat”, “mountain lion”) Which TWO “pattern” vectors would yield the following result?

str_detect(animals, pattern) [1] TRUE TRUE TRUE TRUE

- `pattern <- "mo*"` [X]
- `pattern <- "mo?"` [X]
- `pattern <- "mo+"`
- `pattern <- "moo*"`

Question 2

You are working on some data from different universities. You have the following vector

```
> schools
[1] "U. Kentucky"                "Univ New Hampshire"          "Univ. of Massachusetts"      "Universi
[5] "U California"               "California State University"
```

You want to clean this data to match the full names of each university

```
> final
[1] "University of Kentucky"      "University of New Hampshire" "University of Massachusetts" "Universi
[5] "University of California"    "California State University"
```

What of the following commands could accomplish this?

```
schools %>%
  str_replace("Univ\\.?|U\\.?", "University ") %>%
  str_replace("^University of |^University ", "University of ")
```

```
schools %>%
  str_replace("^Univ\\.?\\s|^U\\.?\\s", "University ") %>%
  str_replace("^University of |^University ", "University of ") [X]
```

```
schools %>%
  str_replace("^Univ\\.\\s|^U\\.\\s", "University") %>%
  str_replace("^University of |^University ", "University of ")
```

```
schools %>%
  str_replace("^Univ\\.?\\s|^U\\.?\\s", "University") %>%
  str_replace("University ", "University of ")
```

Groups with Regex

Question 1

Rather than using the pattern_with_groups vector from the video, you accidentally write in the following code

```
problems <- c("5.3", "5,5", "6 1", "5 .11", "5, 12")
pattern_with_groups <- "^([4-7])[,\\.](\\d*)$"
str_replace(problems, pattern_with_groups, "\\1'\\2")
```

What is your result?

- [1] “5’3” “5’5” “6 1” “5 .11” “5, 12” [X]
- [1] “5.3” “5,5” “6 1” “5 .11” “5, 12”
- [1] “5’3” “5’5” “6’1” “5 .11” “5, 12”
- [1] “5’3” “5’5” “6’1” “5’11” “5’12”

Question 2

You notice your mistake and correct your pattern regex to the following

```
problems <- c("5.3", "5,5", "6 1", "5 .11", "5, 12")
pattern_with_groups <- "^([4-7])[,\\.\\s](\\d*)$"
str_replace(problems, pattern_with_groups, "\\1'\\2")
```

What is your result?

- [1] “5’3” “5’5” “6 1” “5 .11” “5, 12”
- [1] “5.3” “5,5” “6 1” “5 .11” “5, 12”
- [1] “5’3” “5’5” “6’1” “5 .11” “5, 12” [X]
- [1] “5’3” “5’5” “6’1” “5’11” “5’12”

Testing and Improving

Question 1

In our example, we use the following code to detect height entries that do not match our pattern of x’y”.

```
converted <- problems %>%
  str_replace("feet|foot|ft", "'") %>%
  str_replace("inches|in|''|\"", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")

pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
index <- str_detect(converted, pattern)
converted[!index]
```

Which answer best describes the differences between the regex string we use as an argument in `str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")`

And the regex string in `pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"`?

- The regex used in str_replace looks for either a comma, period or space between the feet and inches digits, while the pattern regex just looks for an apostrophe; the regex in str_replace allows for one or more digits to be entered as inches, while the pattern regex only allows for one or two digits.

- The regex used in str_replace allows for additional spaces between the feet and inches digits, but the pattern regex does not.
- The regex used in str_replace looks for either a comma, period or space between the feet and inches digits, while the pattern regex just looks for an apostrophe; the regex in str_replace allows none or more digits to be entered as inches, while the pattern regex only allows for the number 1 or 2 to be used.
- The regex used in str_replace looks for either a comma, period or space between the feet and inches digits, while the pattern regex just looks for an apostrophe; the regex in str_replace allows for none or more digits to be entered as inches, while the pattern regex only allows for one or two digits. [X]

Question 2

You notice a few entries that are not being properly converted using your str_replace and str_detect code

```
yes <- c("5 feet 7inches", "5 7")
no <- c("5ft 9 inches", "5 ft 9 inches")
s <- c(yes, no)

converted <- s %>%
  str_replace("feet|foot|ft", "'") %>%
  str_replace("inches|in|''|\"", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")

pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
str_detect(converted, pattern)
[1]  TRUE FALSE FALSE
```

It seems like the problem may be due to spaces around the words feet|foot|ft and inches|in. What is another way you could fix this problem?

```
converted <- s %>%
  str_replace("\\s*(feet|foot|ft)\\s*", "'") %>%
  str_replace("\\s*(inches|in|''|\")\\s*", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2") [X]
```

```
converted <- s %>%
  str_replace("\\s+feet|foot|ft\\s+", "'") %>%
  str_replace("\\s+inches|in|''|\"\\s+", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")
```

```
converted <- s %>%
  str_replace("\\s*|feet|foot|ft", "'") %>%
  str_replace("\\s*|inches|in|''|\"", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")
```

```
converted <- s %>%
  str_replace_all("\\s", "") %>%
  str_replace("\\s|feet|foot|ft", "'") %>%
  str_replace("\\s|inches|in|''|\"", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")
```

Using Groups and Quantifiers

Question 1

```
s <- c("5'10", "6'1\"", "5'8inches", "5'7.5")
tab <- data.frame(x = s)
```

If you use the extract code from our video, the decimal point is dropped. What modification of the code would allow you to put the decimals in a third column called "decimal"?

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"), regex = "(\\d)'(\\d{1,2})(\\.)?"
```

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"), regex = "(\\d)'(\\d{1,2})(\\.\\d+)"
```

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"), regex = "(\\d)'(\\d{1,2})\\.\\d+?"
```

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"), regex = "(\\d)'(\\d{1,2})(\\.\\d+)?
```

String Splitting

Question 1

You have the following table

```
>schedule
day     staff
Monday      Mandy, Chris and Laura
Tuesday     Steve, Ruth and Frank
```

You want to turn this into a more useful data frame.

Which two commands would properly split the text in the "staff" column into each individual name? Select ALL that apply.

- `str_split(schedule$staff, ",|and")`
- `str_split(schedule$staff, ", | and ")` [X]
- `str_split(schedule$staff, ",\\s|\\sand\\s")` [X]
- `str_split(schedule$staff, "\\s?(,|and)\\s?")`

Question 2

You have the following table

```
> schedule
      day           staff
Monday  Mandy, Chris and Laura
Tuesday     Steve, Ruth and Frank
```

What code would successfully turn your "Schedule" table into the following tidy table

```
< tidy
  day     staff
  <chr>   <chr>
Monday  Mandy
Monday  Chris
Monday  Laura
Tuesday Steve
Tuesday Ruth
```

```
Tuesday Frank
```

```
tidy <- schedule %>%
  mutate(staff = str_split(staff, ", | and ")) %>%
  unnest()
```

```
tidy <- separate(schedule, staff, into = c("s1","s2","s3"), sep = ",") %>%
  gather(key = s, value = staff, s1:s3) [X]
```

```
tidy <- schedule %>%
  mutate(staff = str_split(staff, ", | and ", simplify = TRUE)) %>%   unnest()
```

Recoding

Question 1

Using the gapminder data, you want to recode countries longer than 12 letters in the region "Middle Africa" to their abbreviations in a new column, "country_short". Which code would accomplish this?

```
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(recode(country,
                          "Central African Republic" = "CAR",
                          "Congo, Dem. Rep." = "DRC",
                          "Equatorial Guinea" = "Eq. Guinea"))
```

```
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(country_short = recode(country,
                          c("Central African Republic", "Congo, Dem. Rep.", "Equatorial Guinea"),
                          c("CAR", "DRC", "Eq. Guinea")))
```

```
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(country = recode(country,
                          "Central African Republic" = "CAR",
                          "Congo, Dem. Rep." = "DRC",
                          "Equatorial Guinea" = "Eq. Guinea"))
```

```
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(country_short = recode(country,
                          "Central African Republic" = "CAR",
                          "Congo, Dem. Rep." = "DRC",
                          "Equatorial Guinea" = "Eq. Guinea"))
```