Week 7: Character Strings

Ellen Bledsoe

2024-02-29

Working with Strings

Working with character strings (text data) can be particularly challenging in R. Thankfully, we have the `stringr` to help!

All functions in the `stringr` package start with `str_`. There are *many* helpful functions in the `stringr` package. We'll only cover a handful here, but if you're looking to accomplish something with a string and aren't sure how to approach it, the `stringr` package is a good place to start.

`stringr` Functions

We'll cover a number of helpful `stringr` functions in this lesson:

- `str_length()` - count the number of characters in the string
- `str_count()` - count the number of times a pattern matches within a string
- `str_detect()` - determine if pattern is found within string
- `str_subset()` - return subset of strings that match the pattern
- `str_extract()` - return portion of each string that matches the pattern
- `str_remove()` - remove portion of the string that matches the pattern
- `str_replace()` - replace portion of string that matches the pattern with something else

Let's load the `tidyverse` and get started.

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

We will use some real data from my postdoc lab that I was tasked with cleaning.

It's a long story, but I ended up in an aquatic biogeochemistry lab for my postdoc. The lab had 4+ years of data from dugouts, which are small, human-made water resevoirs that are common across the prairie states and provinces.

Unfortunately, the data were collected by many different people without a standard data entry protocol, so there was extensive data cleaning that needed to happen to bring all of the datasets together across the years.

Many of the tools we are covering in this class (especially the last few weeks with joins, pivots, and strings) were integral to getting those datasets wrangled. Here is just one example of the type of dataset I was working with.

```
url <- "https://raw.githubusercontent.com/bleds22e/FAST_lab_training/master/merging_masters/data/carbon_

carbon <- read_csv(url) %>% rename(SampleID = `Sample ID`)
```

```
## Rows: 48 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): Sample ID
## dbl (2): TIC (PPM as mg/L C), TOC (PPM as mg/L C)
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

```
carbon
```

```
## # A tibble: 48 x 3
##    SampleID        ‘TIC (PPM as mg/L C)‘ ‘TOC (PPM as mg/L C)‘
##    <chr>                           <dbl>                 <dbl>
##  1 AMC3/JULY13                      51.7                  33.3
##  2 BMC3/AUG13                       58.3                  37.2
##  3 BMC2/AUG13                       89.7                  47.6
##  4 E56A-DEEP/AUG17                 259.                   57.2
##  5 ACB2/JUNE22                      50.0                  22.0
##  6 E14A/AUG17                      107.                   56.8
##  7 CFH/AUG6                         22.9                  18.8
##  8 ALA2/JULY22                     127.                  100.
##  9 AMC1/JUNE26                      48.9                  31.8
## 10 CLH/AUG4                         43.4                  32.1
## # i 38 more rows
```

Strings in Vectors

To start working with strings, let's start with a vector instead of a dataframe.

Let's pull out the first column (the `Sample ID` column) from the `carbon` dataframe.

```
sampleID <- carbon$SampleID
sampleID
```

```
##  [1] "AMC3/JULY13"     "BMC3/AUG13"      "BMC2/AUG13"      "E56A-DEEP/AUG17"
##  [5] "ACB2/JUNE22"     "E14A/AUG17"      "CFH/AUG6"        "ALA2/JULY22"
##  [9] "AMC1/JUNE26"     "CLH/AUG4"        "A14B/JUNE5"      "BLS/JULY7"
## [13] "ALH/JUNE4"       "ALA1/JULY2"      "A56C/JUNE2"      "B14B/JUNE29"
## [17] "AFH/MAY29"       "AMC2/JULY13"     "BLA1/AUG12"      "BT1/AUG10"
## [21] "C14B/JULY30"     "BLH/JULY7"       "CLS/AUG4"        "ACB1/JUNE22"
## [25] "B56A/JUNE30"     "B14A/JUNE29"     "ALHM2/JUNE23"    "AT1/JULY14"
## [29] "ALHM1/JUNE23"    "A56A/JUNE2"      "ALS/JUNE4"       "A14A/JUNE5"
## [33] "BFH/JULY1"       "BLHM1/AUG11"     "E56A/AUG17"      "BLB1/AUG7"
## [37] "D14A/JULY30"     "B56C/JULY29"     "BMC1/AUG13"      "D56A/JULY29"
## [41] "E14A-DEEP/AUG17" "AT2/JULY14"      "BT2/AUG10"       "BLHM2/AUG11"
## [45] "C14A/JULY16"     "BLA2/AUG12"      "BCB2/AUG7"       "C56A/JULY15"
```

Let's practice with each of our functions.

`str_length(string)`: count the number of characters in the string

```r
str_length(sampleID)
```

```
##  [1] 11 10 10 15 11 10  8 11 11  8 10  9  9 10 10 11  9 11 10  9 11  9  8 11 11
## [26] 11 12 10 12 10  9 10  9 11 10  9 11 11 10 11 15 10  9 11 11 10  9 11
```

`str_count(string, pattern)`: count the number of times a pattern matches within a string

```r
str_count(sampleID, "DEEP")
```

```
##  [1] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0 1 0 0 0 0 0 0 0
```

`str_detect(string, pattern)`: determine if pattern is found within string

```r
str_detect(sampleID, "DEEP")
```

```
##  [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [37] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

`str_subset(string, pattern)`: return subset of strings that match the pattern

```r
str_subset(sampleID, "DEEP")
```

```
## [1] "E56A-DEEP/AUG17" "E14A-DEEP/AUG17"
```

`str_extract()`: return portion of each string that matches the pattern

```r
# 3 characters after the /
str_extract(sampleID, "/(...)")
```

```
##  [1] "/JUL" "/AUG" "/AUG" "/AUG" "/JUN" "/AUG" "/AUG" "/JUL" "/JUN" "/AUG"
## [11] "/JUN" "/JUL" "/JUN" "/JUL" "/JUN" "/JUN" "/MAY" "/JUL" "/AUG" "/AUG"
## [21] "/JUL" "/JUL" "/AUG" "/JUN" "/JUN" "/JUN" "/JUN" "/JUL" "/JUN" "/JUN"
## [31] "/JUN" "/JUN" "/JUL" "/AUG" "/AUG" "/AUG" "/JUL" "/JUL" "/AUG" "/JUL"
## [41] "/AUG" "/JUL" "/AUG" "/AUG" "/JUL" "/AUG" "/AUG" "/JUL"
```

```
months <- str_extract(sampleID, "/(...)")
```

`str_remove()` - remove portion of the string that matches the pattern

```
str_remove(months, "/")
```

```
##  [1] "JUL" "AUG" "AUG" "AUG" "JUN" "AUG" "AUG" "JUL" "JUN" "AUG" "JUN" "JUL"
## [13] "JUN" "JUL" "JUN" "JUN" "MAY" "JUL" "AUG" "AUG" "JUL" "JUL" "AUG" "JUN"
## [25] "JUN" "JUN" "JUN" "JUL" "JUN" "JUN" "JUN" "JUN" "JUL" "AUG" "AUG" "AUG"
## [37] "JUL" "JUL" "AUG" "JUL" "AUG" "JUL" "AUG" "AUG" "JUL" "AUG" "AUG" "JUL"
```

`str_replace()`: replace portion of string that matches the pattern with something else

```
# we can also replace values (with something new or with a blank)
str_replace(months, "/", "")
```

```
##  [1] "JUL" "AUG" "AUG" "AUG" "JUN" "AUG" "AUG" "JUL" "JUN" "AUG" "JUN" "JUL"
## [13] "JUN" "JUL" "JUN" "JUN" "MAY" "JUL" "AUG" "AUG" "JUL" "JUL" "AUG" "JUN"
## [25] "JUN" "JUN" "JUN" "JUL" "JUN" "JUN" "JUN" "JUN" "JUL" "AUG" "AUG" "AUG"
## [37] "JUL" "JUL" "AUG" "JUL" "AUG" "JUL" "AUG" "AUG" "JUL" "AUG" "AUG" "JUL"
```

Using stringr in Data Frames

Like we saw with the `lubridate` functions, you will often want to use `stringr` functions within other `tidyverse` functions, such as `filter` or `mutate`.

Let's run through a few examples using the full `carbon` dataframe.

Perhaps we want only samples that were collected in August. We can use the `str_detect` function to set our condition. This works because the output of `str_detect` is a logical vector, as with other conditional statements or `is.na()`.

```
carbon %>%
  filter(str_detect(SampleID, "AUG"))
```

```
## # A tibble: 18 x 3
##    SampleID       ‘TIC (PPM as mg/L C)‘ ‘TOC (PPM as mg/L C)‘
##    <chr>                          <dbl>                 <dbl>
##  1 BMC3/AUG13                      58.3                  37.2
##  2 BMC2/AUG13                      89.7                  47.6
##  3 E56A-DEEP/AUG17                259.                   57.2
##  4 E14A/AUG17                     107.                   56.8
##  5 CFH/AUG6                        22.9                  18.8
##  6 CLH/AUG4                        43.4                  32.1
##  7 BLA1/AUG12                      54.5                  71.1
##  8 BT1/AUG10                       71.0                  29.9
##  9 CLS/AUG4                        36.4                  21.6
## 10 BLHM1/AUG11                     43.9                  25.3
## 11 E56A/AUG17                      60.6                  29.7
## 12 BLB1/AUG7                       29.4                  18.4
## 13 BMC1/AUG13                      49.8                  39.2
## 14 E14A-DEEP/AUG17                  111.                      55.9
## 15 BT2/AUG10                         34.9                     15.6
## 16 BLHM2/AUG11                       71.8                     37.8
## 17 BLA2/AUG12                        98.1                    105.
## 18 BCB2/AUG7                         53.8                     24.9
```

Perhaps we want to filter for sample names over a certain length (for some reason). We can do that as well, though we need to structure our condition a little differently this time, because the output of `str_count` is not a logical vector.

```
carbon %>%
  filter(str_count(SampleID) > 13)
```

```
## # A tibble: 2 x 3
##   SampleID        `TIC (PPM as mg/L C)` `TOC (PPM as mg/L C)`
##   <chr>                           <dbl>                 <dbl>
## 1 E56A-DEEP/AUG17                  259.                  57.2
## 2 E14A-DEEP/AUG17                  111.                  55.9
```

Alternatively, perhaps we want to create a column with the month the sample was collected. We can use the `str_extract` and `str_replace` columns in a mutate function.

```
carbon %>%
  mutate(Month = str_extract(SampleID, "/(...)"),
         Month = str_remove(Month, "/"))
```

```
## # A tibble: 48 x 4
##    SampleID        `TIC (PPM as mg/L C)` `TOC (PPM as mg/L C)` Month
##    <chr>                           <dbl>                 <dbl> <chr>
##  1 AMC3/JULY13                      51.7                  33.3 JUL
##  2 BMC3/AUG13                       58.3                  37.2 AUG
##  3 BMC2/AUG13                       89.7                  47.6 AUG
##  4 E56A-DEEP/AUG17                 259.                   57.2 AUG
##  5 ACB2/JUNE22                      50.0                  22.0 JUN
##  6 E14A/AUG17                      107.                   56.8 AUG
##  7 CFH/AUG6                         22.9                  18.8 AUG
##  8 ALA2/JULY22                     127.                  100.  JUL
##  9 AMC1/JUNE26                      48.9                  31.8 JUN
## 10 CLH/AUG4                         43.4                  32.1 AUG
## # i 38 more rows
```

Alternatively, we could create a column for whether or not the sample was taken from below the surface ("DEEP").

```
carbon %>%
  mutate(Deep = str_extract(SampleID, "DEEP"))
```

```
## # A tibble: 48 x 4
##    SampleID        `TIC (PPM as mg/L C)` `TOC (PPM as mg/L C)` Deep
##    <chr>                           <dbl>                 <dbl> <chr>
##  1 AMC3/JULY13                      51.7                  33.3 <NA>
##  2 BMC3/AUG13                       58.3                  37.2 <NA>
##  3 BMC2/AUG13                       89.7                  47.6 <NA>
##  4 E56A-DEEP/AUG17                 259.                   57.2 DEEP
##  5 ACB2/JUNE22                      50.0                  22.0 <NA>
##  6 E14A/AUG17                      107.                   56.8 <NA>
##  7 CFH/AUG6                         22.9                  18.8 <NA>
##  8 ALA2/JULY22                     127.                  100.  <NA>
##  9 AMC1/JUNE26                      48.9                  31.8 <NA>
## 10 CLH/AUG4                         43.4                  32.1 <NA>
## # i 38 more rows
```

Regular Expressions

While being able to match specific strings is helpful, often we have more complicated requirements, such as counting all the numbers from a string, removing the first 3 characters of a string, or extracting all of the values after a certain symbol.

When we need to perform more complicated tasks using strings, we can turn to something called "regular expressions," or "regex" for short. Regular expressions uses characters and special symbols to define certain search patterns in concise ways.

I'm not going to go deep into "regex," but you should know that they exist in case you need to use them in the future.

As one example, let's say I wanted to pull out all of the characters after the `/` in the `sampleID` vector, since they represent dates. We could use the regular expression `"(?<=/).*"` to do so.

```
str_extract(sampleID, "(?<=/).*")
```

```
##  [1] "JULY13" "AUG13"  "AUG13"  "AUG17"  "JUNE22" "AUG17"  "AUG6"   "JULY22"
##  [9] "JUNE26" "AUG4"   "JUNE5"  "JULY7"  "JUNE4"  "JULY2"  "JUNE2"  "JUNE29"
## [17] "MAY29"  "JULY13" "AUG12"  "AUG10"  "JULY30" "JULY7"  "AUG4"   "JUNE22"
## [25] "JUNE30" "JUNE29" "JUNE23" "JULY14" "JUNE23" "JUNE2"  "JUNE4"  "JUNE5"
## [33] "JULY1"  "AUG11"  "AUG17"  "AUG7"   "JULY30" "JULY29" "AUG13"  "JULY29"
## [41] "AUG17"  "JULY14" "AUG10"  "AUG11"  "JULY16" "AUG12"  "AUG7"   "JULY15"
```

Helpful Resources

While memorizing regular expressions is wildly daunting, there are thankfully numerous resources that we can use to help us out.

Here is a website where you can build and test regex.

Honestly, though, I use ChatGPT to build my regex!

To build the `"(?<=/).*"` expression from above, I asked ChatGPT to "use regex and `str_extract` to extract everything after a `/`, not including the `/`", and it produced exactly what I needed.