String Manipulation

Dav King

4/12/22

Main Ideas

- Working with string data is essential for a number of data science tasks, including data cleaning, data preparation, and text analysis.
- The `stringr` package in `R` (part of the `tidyverse`) contains useful tools for working with character strings.

Coming Up

- HW due on Thursday at 11:59 PM
- Lab 2 in groups on Thursday

Lecture Notes and Exercises

In addition to the `tidyverse`, we will use the `stringr` package.

```
library(tidyverse)
library(stringr)
```

`stringr` provides tools to work with character strings. Functions in `stringr` have consistent, memorable names.

- All begin with `str_` (`str_count()`, `str_detect()`, `str_trim()`, etc).
- All take a vector of strings as their first arguments.
- We only have time to explore the basics. I encourage you to explore on your own using the **additional resources** below.

Preliminaries

Character strings in `R` are defined by double quotation marks. These can include numbers, letters, punctation, whitespace, etc.

```
string1 <- "STA 199 is my favorite class"
string1
```

```
## [1] "STA 199 is my favorite class"
```

You can combine character strings in a vector.

```
string2 <- c("STA 199", "Data Science", "Duke")
string2
```

```
## [1] "STA 199"      "Data Science" "Duke"
```

Question: What if we want to include a quotation in a string? Why doesn't the code below work?

You interrupt the parentheses of the string.

```
string3 <- "I said "Hello" to my class"
```

To include a double quote in a string **escape it** using a backslash. Try it now in the code chunk below and name your string `string4`.

```
string4 <- "I said \"Hello\" to my class"
string4
```

```
## [1] "I said \"Hello\" to my class"
```

If you want to include an actual backslash, **escape it** as shown below. This may seem tedious but it will be important later.

```
string5 <- "\\"
string5
```

```
## [1] "\\"
```

The function `writeLines()` shows the content of the strings not including escapes. Try it for `string1`, `string2`, `string3`, `string4`, and `string5` in the code chunk below.

```
writeLines(c(string1, string2, string4, string5))
```

```
## STA 199 is my favorite class
## STA 199
## Data Science
## Duke
## I said "Hello" to my class
## \
```

U.S. States

To demonstrate the basic functions from `stringr` we will use a vector of all 50 U.S. states.

```
states <- c("alabama", "alaska", "arizona", "arkansas", "california",
            "colorado", "connecticut", "delaware", "florida", "georgia",
            "hawaii", "idaho", "illinois", "indiana", "iowa", "kansas",
            "kentucky", "louisiana", "maine", "maryland", "massachusetts",
            "michigan", "minnesota", "mississippi", "missouri", "montana",
            "nebraska", "nevada", "new hampshire", "new jersey",
            "new mexico", "new york", "north carolina", "north dakota", "ohio",
            "oklahoma", "oregon", "pennsylvania", "rhode island",
            "south carolina", "south dakota", "tennessee", "texas", "utah",
            "vermont", "virginia", "washington", "west virginia", "wisconsin",
            "wyoming")
```

str_length() Given a string, return the number of characters.

```
string1
```

```
## [1] "STA 199 is my favorite class"
```

```
str_length(string1)
```

```
## [1] 28
```

Given a vector of strings, return the number of characters in each string.

```
str_length(states)
```

```
##  [1]  7  6  7  8 10  8 11  8  7  7  6  5  8  7  4  6  8  9  5  8 13  8  9 11  8
## [26]  7  8  6 13 10 10  8 14 12  4  8  6 12 12 14 12  9  5  4  7  8 10 13  9  7
```

str_c() Combine two (or more) strings.

```
str_c("STA 199", "is", "my", "favorite", "class")
```

```
## [1] "STA 199ismyfavoriteclass"
```

Use `sep` to specify how the strings are separated.

```
str_c("STA 199", "is", "my", "favorite", "class", sep = " ")
```

```
## [1] "STA 199 is my favorite class"
```

str_to_lower() and str_to_upper()

Convert the case of a string from lower to upper or vice versa.

```
str_to_upper(states)
```

```
##  [1] "ALABAMA"        "ALASKA"         "ARIZONA"        "ARKANSAS"
##  [5] "CALIFORNIA"     "COLORADO"       "CONNECTICUT"    "DELAWARE"
##  [9] "FLORIDA"        "GEORGIA"        "HAWAII"         "IDAHO"
## [13] "ILLINOIS"       "INDIANA"        "IOWA"           "KANSAS"
## [17] "KENTUCKY"       "LOUISIANA"      "MAINE"          "MARYLAND"
## [21] "MASSACHUSETTS"  "MICHIGAN"       "MINNESOTA"      "MISSISSIPPI"
## [25] "MISSOURI"       "MONTANA"        "NEBRASKA"       "NEVADA"
## [29] "NEW HAMPSHIRE"  "NEW JERSEY"     "NEW MEXICO"     "NEW YORK"
## [33] "NORTH CAROLINA" "NORTH DAKOTA"   "OHIO"           "OKLAHOMA"
## [37] "OREGON"         "PENNSYLVANIA"   "RHODE ISLAND"   "SOUTH CAROLINA"
## [41] "SOUTH DAKOTA"   "TENNESSEE"      "TEXAS"          "UTAH"
## [45] "VERMONT"        "VIRGINIA"       "WASHINGTON"     "WEST VIRGINIA"
## [49] "WISCONSIN"      "WYOMING"
```

str_sub()

Extract parts of a string from `start` to `end`, inclusive.

```
str_sub(states, 1, 4)
```

```
##  [1] "alab" "alas" "ariz" "arka" "cali" "colo" "conn" "dela" "flor" "geor"
## [11] "hawa" "idah" "illi" "indi" "iowa" "kans" "kent" "loui" "main" "mary"
## [21] "mass" "mich" "minn" "miss" "miss" "mont" "nebr" "neva" "new " "new "
## [31] "new " "new " "nort" "nort" "ohio" "okla" "oreg" "penn" "rhod" "sout"
## [41] "sout" "tenn" "texa" "utah" "verm" "virg" "wash" "west" "wisc" "wyom"
```

```
str_sub(states, -4, -1)
```

```
##  [1] "bama" "aska" "zona" "nsas" "rnia" "rado" "icut" "ware" "rida" "rgia"
## [11] "waii" "daho" "nois" "iana" "iowa" "nsas" "ucky" "iana" "aine" "land"
## [21] "etts" "igan" "sota" "ippi" "ouri" "tana" "aska" "vada" "hire" "rsey"
## [31] "xico" "york" "lina" "kota" "ohio" "homa" "egon" "ania" "land" "lina"
## [41] "kota" "ssee" "exas" "utah" "mont" "inia" "gton" "inia" "nsin" "ming"
```

Practice: Combine `str_sub()` and `str_to_upper()` to capitalize each state.

```
str_sub(states, 1, 1) <- str_to_upper(str_sub(states, 1, 1))
states
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
## [13] "Illinois"       "Indiana"        "Iowa"           "Kansas"
## [17] "Kentucky"       "Louisiana"      "Maine"          "Maryland"
## [21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"
## [25] "Missouri"       "Montana"        "Nebraska"       "Nevada"
## [29] "New hampshire"  "New jersey"     "New mexico"     "New york"
## [33] "North carolina" "North dakota"   "Ohio"           "Oklahoma"
## [37] "Oregon"         "Pennsylvania"   "Rhode island"   "South carolina"
## [41] "South dakota"   "Tennessee"      "Texas"          "Utah"
## [45] "Vermont"        "Virginia"       "Washington"     "West virginia"
## [49] "Wisconsin"      "Wyoming"
```

```
str_to_upper(states)
```

```
##  [1] "ALABAMA"        "ALASKA"         "ARIZONA"        "ARKANSAS"
##  [5] "CALIFORNIA"     "COLORADO"       "CONNECTICUT"    "DELAWARE"
##  [9] "FLORIDA"        "GEORGIA"        "HAWAII"         "IDAHO"
## [13] "ILLINOIS"       "INDIANA"        "IOWA"           "KANSAS"
## [17] "KENTUCKY"       "LOUISIANA"      "MAINE"          "MARYLAND"
## [21] "MASSACHUSETTS"  "MICHIGAN"       "MINNESOTA"      "MISSISSIPPI"
## [25] "MISSOURI"       "MONTANA"        "NEBRASKA"       "NEVADA"
## [29] "NEW HAMPSHIRE"  "NEW JERSEY"     "NEW MEXICO"     "NEW YORK"
## [33] "NORTH CAROLINA" "NORTH DAKOTA"   "OHIO"           "OKLAHOMA"
## [37] "OREGON"         "PENNSYLVANIA"   "RHODE ISLAND"   "SOUTH CAROLINA"
## [41] "SOUTH DAKOTA"   "TENNESSEE"      "TEXAS"          "UTAH"
## [45] "VERMONT"        "VIRGINIA"       "WASHINGTON"     "WEST VIRGINIA"
## [49] "WISCONSIN"      "WYOMING"
```

str_sort() Sort a string. Below we sort in decreasing alphabetical order.

```
str_sort(states, decreasing = TRUE)
```

```
##  [1] "Wyoming"        "Wisconsin"      "West virginia"  "Washington"
##  [5] "Virginia"       "Vermont"        "Utah"           "Texas"
##  [9] "Tennessee"      "South dakota"   "South carolina" "Rhode island"
## [13] "Pennsylvania"   "Oregon"         "Oklahoma"       "Ohio"
## [17] "North dakota"   "North carolina" "New york"       "New mexico"
## [21] "New jersey"     "New hampshire"  "Nevada"         "Nebraska"
## [25] "Montana"        "Missouri"       "Mississippi"    "Minnesota"
## [29] "Michigan"       "Massachusetts"  "Maryland"       "Maine"
## [33] "Louisiana"      "Kentucky"       "Kansas"         "Iowa"
## [37] "Indiana"        "Illinois"       "Idaho"          "Hawaii"
## [41] "Georgia"        "Florida"        "Delaware"       "Connecticut"
## [45] "Colorado"       "California"     "Arkansas"       "Arizona"
## [49] "Alaska"         "Alabama"
```

Regular Expressions

A **regular expression** is a sequence of characters that allows you to describe string patterns. We use them to search for patterns.

Examples of usage include the following data science tasks:

- extract a phone number from text data
- determine if an email address is valid
- determine if a password has some specified number of letters, characters, numbers, etc
- count the number of times "statistics" occurs in a corpus of text

To demonstrate regular expressions, we will use a vector of the states bordering North Carolina.

```
nc_states <- c("North Carolina", "South Carolina", "Virginia", "Tennessee",
               "Georgia")
```

Basic Match We can match exactly using a **basic match**.

```
str_view_all(nc_states, "in")
```

We can match any character using .

```
str_view_all(nc_states, ".a")
```

Question: What if we want to match a period .?

Escape it using \.

Another example using escapes:

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

Anchors Match the start of a string using ^.

```
str_view(nc_states, "^G")
```

Match the end of a string using $.

```
str_view(nc_states, "a$")
```

`str_detect()` Determine if a character vector matches a pattern.

```
nc_states
```

```
## [1] "North Carolina" "South Carolina" "Virginia"       "Tennessee"
## [5] "Georgia"
```

```
str_detect(nc_states, "a")
```

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

```


nc_states
```

`str_subset()`

```
## [1] "North Carolina" "South Carolina" "Virginia"       "Tennessee"
## [5] "Georgia"
```

```
str_subset(nc_states, "e$")
```

```
## [1] "Tennessee"
```

`str_count()` Determine how many matches there are in a string.

```r
nc_states
```

```
## [1] "North Carolina" "South Carolina" "Virginia"       "Tennessee"
## [5] "Georgia"
```

```r
str_count(nc_states, "a")
```

```
## [1] 2 2 1 0 1
```

`str_replace()` and `str_replace_all()` Replace matches with new strings.

```r
str_replace(nc_states, "a", "-")
```

```
## [1] "North C-rolina" "South C-rolina" "Virgini-"       "Tennessee"
## [5] "Georgi-"
```

Use `str_replace_all()` to replace all matches with new strings.

```r
str_replace_all(nc_states, "a", "-")
```

```
## [1] "North C-rolin-" "South C-rolin-" "Virgini-"       "Tennessee"
## [5] "Georgi-"
```

Many Matches The regular expressions below match more than one character.

- Match any digit using `\d` or [[:digit:]]
- Match any whitespace using `\s` or [[:space:]]
- Match f, g, or h using [fgh]
- Match anything but f, g, or h using [^fgh]
- Match lower-case letters using [a-z] or [[:lower:]]
- Match upper-case letters using [A-Z] or [[:upper:]]
- Match alphabetic characters using [A-z] or [[:alpha:]]

Remember these are regular expressions! To match digits you’ll need to escape the , so use “\d”, not “".

Practice

To practice manipulating strings we will use question and answer data from two recent seasons (2008 - 2009) of the television game show *Jeopardy!*.

```r
jeopardy <- read_csv("questions.csv")
```

- `category`: category of question
- `value`: value of question in dollars
- `question`: text of question
- `answer`: text of question answer
- `year`: year episode aired

```
glimpse(jeopardy)
```

```
## Rows: 40,865
## Columns: 5
## $ category <chr> "OLD FOLKS IN THEIR 30s", "MOVIES & TV", "A STATE OF COLLEGE-~
## $ value    <dbl> 200, 200, 200, 200, 200, 200, 400, 400, 400, 400, 400, 400, 6~
## $ question <chr> "goop.com is a lifestyles website from this Oscar-winning act~
## $ answer   <chr> "Gwyneth Paltrow", "Jay Leno", "Texas", "a pride", "a bunny h~
## $ year     <dbl> 2009, 2009, 2009, 2009, 2009, 2009, 2009, 2009, 2009, 2009, 2~
```

(1) Use a single code pipeline and a function from `stringr` to return all rows where the answer **contains** the word "Durham"

```
jeopardy[str_detect(jeopardy$answer, "Durham"),]
```

```
## # A tibble: 3 x 5
##   category        value question                                   answer  year
##   <chr>           <dbl> <chr>                                      <chr>  <dbl>
## 1 BULL             2000 "\"Bull City\", this place's nickname, is ~ Durham  2009
## 2 BASEBRAWL        1000 "In 1995 10 players were ejected for a bra~ the D~  2009
## 3 MOVIES BY QUOTE   800 "Crash: \"Man, that ball got out of here i~ Bull ~  2009
```

(2) Use a single code pipeline and `stringr` to find the length of all of the answers, sort by decreasing length, and return the five longest answers.

```
jeopardy %>%
  mutate(ans_length = str_length(answer)) %>%
  arrange(desc(ans_length)) %>%
  slice(1:5) %>%
  select(answer, ans_length)
```

```
## # A tibble: 5 x 2
##   answer                                                            ans_length
##   <chr>                                                                  <int>
## 1 a microphone & the masks of comedy & tragedy (a TV set, a movie ca~         86
## 2 hiding your light under a bushel (keeping your light underneath a ~         82
## 3 International Talk Like a Pirate Day (National Talk Like a Pirate ~         79
## 4 (any of) the (St. Louis) Rams, the Oakland Raiders, or the San Die~         77
## 5 to take the number that's between 3 and 5 (averaging the 2 middle ~         74
```

(3) What answer has the most digits?

```
jeopardy %>%
  mutate(digits = str_count(answer, "[0-9]")) %>%
  arrange(desc(digits)) %>%
  slice(1) %>%
  select(answer)
```

```
## # A tibble: 1 x 1
##   answer
##   <chr>
## 1 1939 (or 1942)
```

(4) Return all rows where the category has a period.

```
jeopardy[str_detect(jeopardy$category, "\\."),]
```

```
## # A tibble: 1,249 x 5
##    category           value question                                answer  year
##    <chr>              <dbl> <chr>                                   <chr>  <dbl>
##  1 I LOVE L.A. KERS     400 "Kobe called it \"idiotic criticism\" ~ Shaqu~  2009
##  2 I LOVE L.A. KERS     800 "A wizard at passing the ball, this La~ Magic~  2009
##  3 I LOVE L.A. KERS    1200 "This Laker giant was nicknamed \"The ~ Wilt ~  2009
##  4 I LOVE L.A. KERS    1600 "This Hall-of-Fame guard & former Lake~ Jerry~  2009
##  5 I LOVE L.A. KERS    2000 "This flashy Lakers forward was nickna~ James~  2009
##  6 IT’S AN L.A. THING   200 "Wanna live in this city, 90210? in Ju~ Bever~  2009
##  7 IT’S AN L.A. THING   400 "Originally the letters in this landma~ the H~  2009
##  8 IT’S AN L.A. THING   600 "Good times are Bruin in this district~ Westw~  2009
##  9 IT’S AN L.A. THING   800 "You can hit the Comedy Store, House o~ Sunse~  2009
## 10 IT’S AN L.A. THING  1000 "Originally called \"Nuestro Pueblo\" ~ the W~  2009
## # ... with 1,239 more rows
```

(5) Using a single code pipeline, return all rows where the question contains a (numeric) year between 1800 and 1999

```
jeopardy %>%
  mutate(year = str_detect(jeopardy$question, paste(c("18\\d\\d", "19\\d\\d"),
                                                    collapse = "|"))) %>%
  filter(year == T)
```

```
## # A tibble: 6,749 x 5
##    category                               value question         answer year
##    <chr>                                  <dbl> <chr>            <chr>  <lgl>
##  1 AMERICAN AUTHORS                         800 "During the War~ Washi~ TRUE
##  2 MATHEM-ATTACK!                          1200 "(<a href=\"htt~ a mat~ TRUE
##  3 AMERICAN AUTHORS                        2000 "He reviewed fi~ Phili~ TRUE
##  4 AMERICAN AUTHORS                         200 "While he was i~ Hemin~ TRUE
##  5 AMERICAN AUTHORS                         400 "In 1884 she mo~ Willa~ TRUE
##  6 BEST PICTURE OSCAR-WINNERS IN OTHER WORDS   400 "1980: \"Regula~ Ordin~ TRUE
##  7 DOWN MEXICO WAY                          400 "In 1986 Mexico~ the W~ TRUE
##  8 BEST PICTURE OSCAR-WINNERS IN OTHER WORDS   800 "1932: \"Magnif~ Grand~ TRUE
##  9 BEST PICTURE OSCAR-WINNERS IN OTHER WORDS  1200 "1976: \"A Sing~ Rocky  TRUE
## 10 BEST PICTURE OSCAR-WINNERS IN OTHER WORDS  1600 "1954: \"Docksi~ On th~ TRUE
## # ... with 6,739 more rows
```

(6) Using a single code pipeline, return all rows with answers that begin with three vowels.

```
jeopardy %>%
  mutate(vowel = str_detect(jeopardy$answer, "^[aeiou]{3}")) %>%
  filter(vowel == T)
```

```
## # A tibble: 1 x 6
##   category          value question                                answer  year vowel
##   <chr>             <dbl> <chr>                                   <chr>  <dbl> <lgl>
## 1 LET’S GET SAUCED   1600 "\"The butter of Provence\", this s~ aioli   2008 TRUE
```

Additional Resources

- `stringr` website
- `stringr` cheat sheet
- Regular Expressions cheat sheet
- R for Data Science: Strings