String manipulation

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.

Packages

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

Lecture notes and exercises

`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 <- "CS3072 is my favorite class"
string1
```

```
## [1] "CS3072 is my favorite class"
```

You can combine character strings in a vector.

```
string2 <- c("CS3072", "Data Science", "Effat University")
string2
```

```
## [1] "CS3072"           "Data Science"     "Effat University"
```

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

```
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"
```

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

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

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(string5)
```

```
## \
```

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] "CS3072 is my favorite class"
```

```
str_length(string1)
```

```
## [1] 27
```

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

```r
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.

```r
str_c("CS 3072", "is", "my", "favorite", "class")
```

```
## [1] "CS 3072ismyfavoriteclass"
```

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

```r
str_c("CS 3072", "is", "my", "favorite", "class", sep = " ")
```

```
## [1] "CS 3072 is my favorite class"
```

str_to_lower() and str_to_upper()

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

```r
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.

```r
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 first letter of each state (you can ignore two word states).

```
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_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, pattern = "in")
```

We can match any character using .

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

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

Escape it using `\.` This is the regular expression.

But we represent regular expressions using strings and is also an escape symbol in strings.

Escape again!

To create the regular expression `\.`, use the string "`\\.`"

```
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.

```
nc_states
```

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

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

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

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

```
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.

```
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!*.

```
jeopardy <- read_csv("data/questions.csv")
```

```
## Rows: 40865 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): category, question, answer
## dbl (2): value, year
##
## 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.
```

- `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 %>% filter(str_detect(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(answerstrlength = str_length(answer)) %>% arrange(desc(answerstrlength)) %>% head(5)
```

```
## # A tibble: 5 x 6
##   category                        value question           answer  year answe~1
##   <chr>                           <dbl> <chr>              <chr>  <dbl>   <int>
## 1 L.A. STORY                        600 "They're 2 of the~ a mic~  2007      86
## 2 I FEEL LIKE SUCH AN IDIOM         600 "<a href=\"http:/~ hidin~  2009      82
## 3 HIDING ON THE INTERNET            600 "Flickr listed th~ Inter~  2009      79
## 4 ARE YOU READY FOR SOME FOOTBALL?  200 "One of the 3 cur~ (any ~  2007      77
## 5 MATH                             2000 "For the series  ~ to ta~  2007      74
## # ... with abbreviated variable name 1: answerstrlength
```

(3) What answer has the most digits? Answer : "1939 (or 1942)" and "1952 & 1956" has 8 digits which is the highest number of digits.

```
jeopardy %>% mutate(answerdigitnum = str_count(answer, "[0-9]")) %>% arrange(desc(answerdigitnum)) %>% h
```

```
## # A tibble: 5 x 6
##   category                    value question                answer  year answe~1
##   <chr>                       <dbl> <chr>                   <chr>  <dbl>   <int>
## 1 AFI'S 100 YEARS 100 MOVIES    800 "One of the 2 years, bo~ 1939 ~  2007       8
## 2 YEARS                         600 "Adlai Stevenson lost t~ 1952 ~  2009       8
## 3 '80s SONG LYRICS              800 "\"Jenny, I've got your~ 867-5~  2008       7
## 4 TV MATH                       800 "Col. Steve Austin's ti~ 6,000~  2007       7
## 5 THE SCREEN ACTORS GUILD       400 "Of 12,000, 20,000, or ~ 120,0~  2008       6
## # ... with abbreviated variable name 1: answerdigitnum
```

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

```
jeopardy %>% filter(str_detect(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 %>% filter(str_detect(question, "[1][8-9][0-9][0-9]"))
```

```
## # A tibble: 6,749 x 5
##    category                                value question         answer  year
```

```
##    <chr>                                    <dbl> <chr>            <chr>  <dbl>
##  1 AMERICAN AUTHORS                           800 "During the War~ Washi~  2009
##  2 MATHEM-ATTACK!                            1200 "(<a href=\"htt~ a mat~  2009
##  3 AMERICAN AUTHORS                          2000 "He reviewed fi~ Phili~  2009
##  4 AMERICAN AUTHORS                           200 "While he was i~ Hemin~  2007
##  5 AMERICAN AUTHORS                           400 "In 1884 she mo~ Willa~  2007
##  6 BEST PICTURE OSCAR-WINNERS IN OTHER WORDS  400 "1980: \"Regula~ Ordin~  2007
##  7 DOWN MEXICO WAY                            400 "In 1986 Mexico~ the W~  2007
##  8 BEST PICTURE OSCAR-WINNERS IN OTHER WORDS  800 "1932: \"Magnif~ Grand~  2007
##  9 BEST PICTURE OSCAR-WINNERS IN OTHER WORDS 1200 "1976: \"A Sing~ Rocky   2007
## 10 BEST PICTURE OSCAR-WINNERS IN OTHER WORDS 1600 "1954: \"Docksi~ On th~  2007
## # ... with 6,739 more rows
```

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

```
jeopardy %>% filter(str_starts(answer, "[AEIOUaeiou]{1,3}"))
```

```
## # A tibble: 8,529 x 5
##    category                value question                           answer  year
##    <chr>                   <dbl> <chr>                              <chr>  <dbl>
##  1 ANIMAL COLLECTIVE         200 "Synonym for dignity that's the t~ a pri~  2009
##  2 I'D RATHER BE SKIING      200 "If you're a beginner, you might ~ a bun~  2009
##  3 A STATE OF COLLEGE-NESS   400 "Antioch, Bowling Green, Kent Sta~ Ohio    2009
##  4 PARLEZ VOUS?              400 "Duck, duck, l'oie; (l'oie of cou~ a goo~  2009
##  5 A STATE OF COLLEGE-NESS   600 "DePaul, Wheaton, Northwestern"    Illin~  2009
##  6 ANIMAL COLLECTIVE         800 "It can be a pack of dogs, or a p~ a ken~  2009
##  7 I'D RATHER BE SKIING     1200 "In this type of race you have to~ a sla~  2009
##  8 PARLEZ VOUS?             1000 "\"Huitieme\" is French for this ~ eighth  2009
##  9 MATHEM-ATTACK!           1200 "(<a href=\"http://www.j-archive.~ a mat~  2009
## 10 WORD ORIGINS             1600 "From the Latin for \"much writin~ a pol~  2009
## # ... with 8,519 more rows
```

(7) Using a single code pipeline, return all answers that end with `ugh` but not `ough`.

```
jeopardy %>% filter(str_ends(answer, "ugh")) %>% filter(str_ends(answer, "ough", TRUE))
```

```
## # A tibble: 5 x 5
##   category                value question                           answer  year
##   <chr>                   <dbl> <chr>                              <chr>  <dbl>
## 1 COLIN POWELL             2000 "In 2009 this controversial radio ~ (Rush~  2009
## 2 OPERA                     400 "Poignant, given Pagliaccio's prof~ laugh   2007
## 3 LITERARY BROTHERS        1200 "Alec Waugh's first novel was \"Th~ Evely~  2007
## 4 FAMOUS COLLEGE DROPOUTS   400 "This conservative radio talk show~ Rush ~  2008
## 5 BIG FAN                   400 "Fans of this radio host: Dittohea~ Rush ~  2009
```

(8) Use a single code pipeline to create a new variable `prop_vowel` that is the proportion of all letters in each answer that are vowels. What is the highest? Lowest?

```
jeopardy %>%
  mutate(vowels = str_count(answer, "[AEIOUaeiou]"),
         letters = str_count(answer, "[[:alpha:]]"),
```

```
        prop_vowel = vowels / letters) %>%
  select(answer, vowels, letters, prop_vowel) %>%
  arrange(desc(prop_vowel)) %>%
  filter(letters > 5,
         !is.na(prop_vowel)) %>%
  slice(1:3, (n() - 2):n())
```

```
## # A tibble: 6 x 4
##   answer          vowels letters prop_vowel
##   <chr>            <int>   <int>      <dbl>
## 1 a queue              5       6      0.833
## 2 a lei & a lee        6       8      0.75
## 3 queue / cue          6       8      0.75
## 4 Lynyrd Skynyrd       0      13      0
## 5 Lynyrd Skynyrd       0      13      0
## 6 rhythms              0       7      0
```

Additional Resources

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