CSSS508, Week 8

Strings

Chuck Lanfear

May 22, 2019

Updated: Mar 28, 2019

UW CS&SS

Data Today

We'll use data on food safety inspections in King County from data.kingcounty.gov.

Note these data are *quite large*. You will want to save them and load them from a *local directory*.

```r
library(tidyverse)
restaurants <-
  read_csv("https://clanfear.github.io/CSSS508/Lectures/Week8/restaurants.csv",
                          col_types = "ccccccccnncciccccicccciD")
```

```
glimpse(restaurants)
```

```
## Observations: 258,630
## Variables: 23
## $ Name                       <chr> "@ THE SHACK, LLC ", "10 MERCER RES...
## $ Program_Identifier         <chr> "SHACK COFFEE", "10 MERCER RESTAURA...
## $ Inspection_Date            <chr> NA, "01/24/2017", "01/24/2017", "01...
## $ Description                <chr> "Seating 0-12 - Risk Category I", "...
## $ Address                    <chr> "2920 SW AVALON WAY", "10 MERCER ST...
## $ City                       <chr> "Seattle", "Seattle", "Seattle", "S...
## $ Zip_Code                   <chr> "98126", "98109", "98109", "98109",...
## $ Phone                      <chr> "(206) 938-5665", NA, NA, NA, NA, N...
## $ Longitude                  <dbl> -122, -122, -122, -122, -122, -122,...
## $ Latitude                   <dbl> 47.6, 47.6, 47.6, 47.6, 47.6, 47.6,...
## $ Inspection_Business_Name   <chr> NA, "10 MERCER RESTAURANT", "10 MER...
## $ Inspection_Type            <chr> NA, "Routine Inspection/Field Revie...
## $ Inspection_Score           <int> NA, 10, 10, 10, 15, 15, 15, 0, 15, ...
## $ Inspection_Result          <chr> NA, "Unsatisfactory", "Unsatisfacto...
## $ Inspection_Closed_Business <chr> NA, "false", "false", "false", "fal...
## $ Violation_Type             <chr> NA, "blue", "blue", "red", "blue", ...
## $ Violation_Description      <chr> NA, "4300 - Non-food contact surfac...
## $ Violation_Points           <int> 0, 3, 2, 5, 5, 5, 5, 0, 5, 10, 25, ...
## $ Business_ID                <chr> "PR0048053", "PR0049572", "PR004957...
## $ Inspection_Serial_Num      <chr> NA, "DAHSIBSJT", "DAHSIBSJT", "DAHS...
## $ Violation_Record_ID        <chr> NA, "IV43WZVLN", "IVCQ1ZIV0", "IVRE...
## $ Grade                      <int> NA, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2...
## $ Date                       <date> NA, 2017-01-24, 2017-01-24, 2017-0...
```

Strings

A general programming term for a unit of character data is a **string**, which is defined as *a sequence of characters*. In R the terms "strings" and "character data" are mostly interchangeable.

In other languages, a string often also refers a *sequence* of numeric information, such as binary strings (e.g. "01110000 01101111 01101111 01110000"). We rarely use these in R.

Note that these are *sequences* of numbers rather than single numbers, and thus *strings*.

String Basics

`nchar()`

We've seen the `nchar()` function to get the number of characters in a string. How many characters are in the ZIP codes?

```
restaurants %>%
  mutate(ZIP_length = nchar(Zip_Code)) %>%
  count(ZIP_length)
```

```
## # A tibble: 2 x 2
##   ZIP_length      n
##        <int>  <int>
## 1          5 258629
## 2         10      1
```

substr()

You may be familiar with `substr()` from the voting homework. We can use it to pull out just the first 5 digits of the ZIP code.

```r
restaurants <- restaurants %>%
    mutate(ZIP_5 = substr(Zip_Code, 1, 5))
restaurants %>% distinct(ZIP_5) %>% head()
```

```
## # A tibble: 6 x 1
##   ZIP_5
##   <chr>
## 1 98126
## 2 98109
## 3 98101
## 4 98032
## 5 98102
## 6 98004
```

paste()

We can combine parts of strings together using the `paste()` function, e.g. to make a whole mailing address:

```r
restaurants <- restaurants %>%
    mutate(mailing_address = paste(Address, ", ", City,
                                   ", WA ", ZIP_5, sep = ""))
restaurants %>% distinct(mailing_address) %>% head()
```

```
## # A tibble: 6 x 1
##   mailing_address
##   <chr>
## 1 2920 SW AVALON WAY, Seattle, WA 98126
## 2 10 MERCER ST, Seattle, WA 98109
## 3 1001 FAIRVIEW AVE N Unit 1700A, SEATTLE, WA 98109
## 4 1225 1ST AVE, SEATTLE, WA 98101
## 5 18114 E VALLEY HWY, KENT, WA 98032
## 6 121 11TH AVE E, SEATTLE, WA 98102
```

`paste()` with `collapse=`

`readr` functions can take an argument like `col_types = "cciDcnncii"`.

You can build this yourself from existing data using `class()` to get the type of data, and `substr()`, and `paste()` with the `collapse = ""` argument to take a vector and make it a single character string:

```
# use the class function to get the type of data
column_types <- sapply(restaurants, class)
# extract 1st character
column_types_1 <- substr(column_types, 1, 1)
# use paste with collapse to get one string
(column_types_collapse <- paste(column_types_1, collapse = ""))
```

```
## [1] "ccccccccnncciccccicccciDcc"
```

Make a function out of this!

I typically read in data using `read_csv()` without `col_types()`. You could use the functions above together to get column types to feed back into `read_csv()`:

```
readr_input_types <- function(data) {
    column_types_1 <- substr(sapply(data, class), 1, 1)
    return(paste(column_types_1, collapse = ""))
}
readr_input_types(swiss)
```

```
## [1] "nniinn"
```

```
readr_input_types(restaurants)
```

```
## [1] "ccccccccnnccicccciccciDcc"
```

`paste0()`

`paste0()` is a shortcut for `paste()` without any separator.

```r
paste(1:5, letters[1:5]) # sep is a space by default
```

```
## [1] "1 a" "2 b" "3 c" "4 d" "5 e"
```

```r
paste(1:5, letters[1:5], sep ="")
```

```
## [1] "1a" "2b" "3c" "4d" "5e"
```

```r
paste0(1:5, letters[1:5])
```

```
## [1] "1a" "2b" "3c" "4d" "5e"
```

`paste()` Practice

`sep=` controls what happens when doing entry-wise squishing of vectors you give to `paste()`, while `collapse=` controls if/how they go from a vector to a single string.

Here are some examples; make sure you understand how each set of arguments produce their results:

```
paste(letters[1:5], collapse = "!")
paste(1:5, letters[1:5], sep = "+")
paste0(1:5, letters[1:5], collapse = "???")
paste(1:5, "Z", sep = "*")
paste(1:5, "Z", sep = "*", collapse = " ~ ")
```

```
## [1] "a!b!c!d!e"
## [1] "1+a" "2+b" "3+c" "4+d" "5+e"
## [1] "1a???2b???3c???4d???5e"
## [1] "1*Z" "2*Z" "3*Z" "4*Z" "5*Z"
## [1] "1*Z ~ 2*Z ~ 3*Z ~ 4*Z ~ 5*Z"
```

`stringr`

`stringr`

`stringr` is yet another R package from the Tidyverse (like `ggplot2`, `dplyr`, `tidyr`, `lubridate`, `readr`).

It provides functions that:

- Replace some basic string functions like `paste()` and `nchar()` in a way that's a bit less touchy with missing values or factors
- Remove whitespace or pad it out
- Perform tasks related to **pattern matching**: Detect, locate, extract, match, replace, split.
  - These functions use **regular expressions** to describe patterns
  - Base R and `stringi` versions for these exist but are harder to use

Conveniently, *most* `stringr` functions begin with "`str_`" to make RStudio auto-complete more useful.

```r
library(stringr)
```

`stringr` Equivalencies

- `str_sub()` is like `substr()` but also lets you put in negative values to count backwards from the end (-1 is the end, -3 is third from end):

```r
str_sub("Washington", 1, -3)
```

```
## [1] "Washingt"
```

- `str_c()` ("string combine") is just like `paste()` but where default is `sep = ""` (like `paste0()`)

```r
str_c(letters[1:5], 1:5)
```

```
## [1] "a1" "b2" "c3" "d4" "e5"
```

`stringr` Equivalencies

- `str_length()` is equivalent to `nchar()`:

```r
nchar("weasels")
```

```
## [1] 7
```

```r
str_length("weasels")
```

```
## [1] 7
```

Changing Cases

`str_to_upper()`, `str_to_lower()`, `str_to_title()` convert cases, which is often a good idea to do before searching for values:

```
head(unique(restaurants$City))
```

```
## [1] "Seattle"  "SEATTLE"  "KENT"     "BELLEVUE" "KENMORE"  "Issaquah"
```

```
restaurants <- restaurants %>%
    mutate_at(vars(Name, Address, City),
              funs(str_to_upper))
head(unique(restaurants$City))
```

```
## [1] "SEATTLE"  "KENT"     "BELLEVUE" "KENMORE"  "ISSAQUAH" "BURIEN"
```

`str_trim()` Whitespace

Extra leading or trailing whitespace is common in text data:

```r
head(unique(restaurants$Name), 4)
```

```
## [1] "@ THE SHACK, LLC "    "10 MERCER RESTAURANT" "100 LB CLAM"
## [4] "1000 SPIRITS"
```

Any character column is potentially affected. We can use the `str_trim()` function in `stringr` to clean them up all at once:

```r
# use mutate_if to trim all the character columns
restaurants <- restaurants %>%
    mutate_if(is.character, str_trim)
head(unique(restaurants$Name), 4)
```

```
## [1] "@ THE SHACK, LLC"     "10 MERCER RESTAURANT" "100 LB CLAM"
## [4] "1000 SPIRITS"
```

Regular Expressions and Pattern Matching

What are Regular Expressions?

Regular expressions or **regex**es are how we describe patterns we are looking for in text in a way that a computer can understand. We write an **expression**, apply it to a string input, and then can do things with **matches** we find.

- **Literal characters** are defined snippets to search for like `SEA` or `206`
- **Metacharacters** let us be flexible in describing patterns:
  - backslash `\`, caret `^`, dollar sign `$`, period `.`, pipe `|`, question mark `?`, asterisk `*`, plus sign `+`, parentheses `(` and `)`, square brackets `[` and `]`, curly braces `{` and `}`
  - To treat a metacharacter as a literal character, you must **escape** it with two preceding backslashs `\\`, e.g. to match `(206)` including the parentheses, you'd use `\\(206\\)` in your regex

`str_detect()`

I want to get inspections for coffee shops. I'll say a coffee shop is anything that has "COFFEE", "ESPRESSO", or "ROASTER" in the name. The `regex` for this is `COFFEE|ESPRESSO|ROASTER` because `|` is a metacharacter that means "OR". Use the `str_detect()` function, which returns `TRUE` if it finds what you're looking for and `FALSE` if it doesn't (similar to `grepl()`):

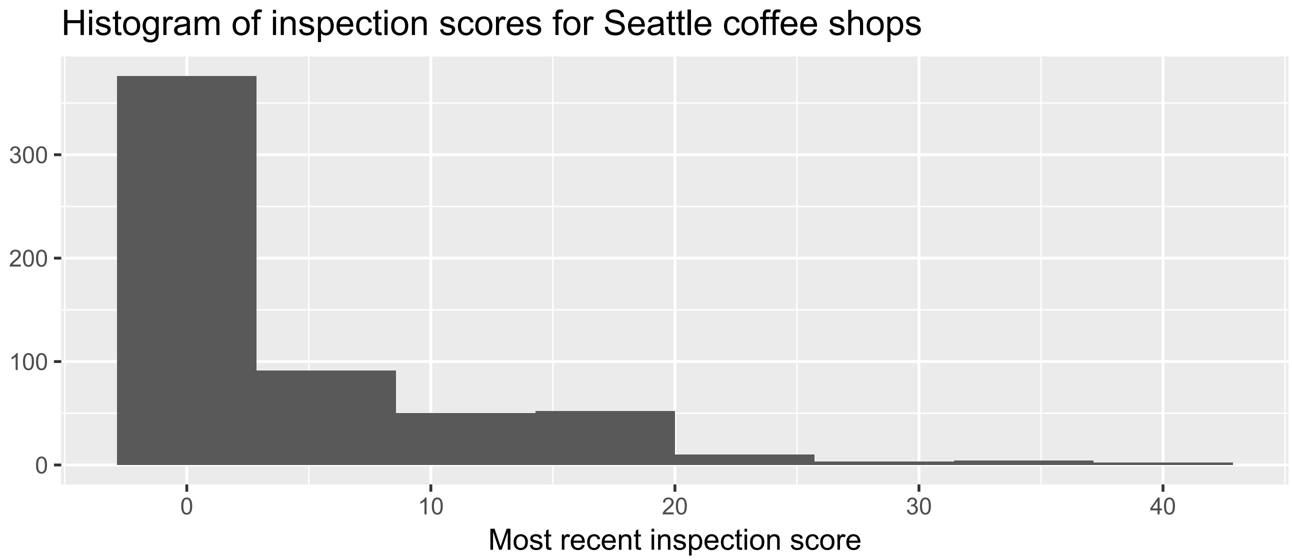
```r
coffee <- restaurants %>%
  filter(str_detect(Name, "COFFEE|ESPRESSO|ROASTER"))
coffee %>% distinct(Name) %>% head()
```

```
## # A tibble: 6 x 1
##   Name
##   <chr>
## 1 2 SISTERS ESPRESSO
## 2 701 COFFEE
## 3 909 COFFEE AND WINE
## 4 AJ'S ESPRESSO
## 5 ALKI HOMEFRONT SMOOTHIES & ESPRESSO
## 6 ALL CITY COFFEE
```

Will My Coffee Kill Me?

Let's take each unique business identifier, keep the most recent inspection score, and look at a histogram of scores:

```
coffee %>% select(Business_ID, Name, Inspection_Score, Date) %>%
       group_by(Business_ID) %>% filter(Date == max(Date)) %>%
       distinct(.keep_all=TRUE) %>% ggplot(aes(Inspection_Score)) +
    geom_histogram(bins=8) + xlab("Most recent inspection score") + ylab("") +
    ggtitle("Histogram of inspection scores for Seattle coffee shops")
```



`str_detect()`: Patterns

Let's look for phone numbers whose first three digits are "206" using `str_detect()`.

We will want it to work whether they have parentheses around the beginning or not, but NOT to match "206" occurring elsewhere:

```r
area_code_206_pattern <- "^\\(?206"
phone_test_examples <- c("2061234567", "(206)1234567",
                         "(206) 123-4567", "555-206-1234")
str_detect(phone_test_examples, area_code_206_pattern)
```

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

- `^` is a metacharacter meaning "look only at the *beginning* of the string"
- `\\(?` means look for a left parenthesis (`\\(`), but it's optional (`?`)
- `206` is the literal character to look for after the optional parenthesis

`str_view()`

`stringr` also has a function called `str_view()` that allows you to see in the viewer pane *exactly* what text is being selected with a regular expression.

```
str_view(phone_test_examples, area_code_206_pattern)
```

This will generate a small web page in the viewer pane (but not in Markdown docs).

Just be careful to not load an entire long vector / variable or it may crash RStudio as it tries to render a massive page!

How Many Rows Have Non-206 Numbers?

```r
restaurants %>%
  mutate(has_206_number =
           str_detect(Phone, area_code_206_pattern)) %>%
  group_by(has_206_number) %>% tally()
```

```
## # A tibble: 3 x 2
##   has_206_number      n
##   <lgl>           <int>
## 1 FALSE           66655
## 2 TRUE           109099
## 3 NA              82876
```

Extracting Patterns with `str_extract()`

Let's extract the directional part of Seattle of addresses: N, NW, SE, none, etc.

```
direction_pattern <- " (N|NW|NE|S|SW|SE|W|E)( |$)"
direction_examples <- c("2812 THORNDYKE AVE W", "512 NW 65TH ST",
                        "407 CEDAR ST", "15 NICKERSON ST")
str_extract(direction_examples, direction_pattern)
```

```
## [1] " W"   " NW " NA     NA
```

- The first space will match a space character, then
- `(N|NW|NE|S|SW|SE|W|E)` matches one of the directions in the group
- `( |$)` is a group saying either there is a space after, or it's the end of the address string (`$` means the end of the string)

Where are the Addresses?

```
restaurants %>%
  distinct(Address) %>%
  mutate(city_region =
           str_trim(str_extract(Address, direction_pattern))) %>%
  count(city_region) %>% arrange(desc(n))
```

```
## # A tibble: 9 x 2
##   city_region     n
##   <chr>       <int>
## 1 NE           2086
## 2 S            1764
## 3 <NA>         1745
## 4 N             879
## 5 SE            868
## 6 SW            705
## 7 E             538
## 8 NW            438
## 9 W             235
```

`str_replace()`: Replacing

Maybe we want to do a street-level analysis of inspections (e.g. compare The Ave to Pike Street). How can we remove building numbers?

```r
address_number_pattern <- "^[0-9]*-?[A-Z]? (1/2 )?"
address_number_test_examples <-
  c("2812 THORNDYKE AVE W", "1ST AVE", "10A 1ST AVE",
    "10-A 1ST AVE", "5201-B UNIVERSITY WAY NE",
    "7040 1/2 15TH AVE NW")
str_replace(address_number_test_examples,
            address_number_pattern, replacement = "")
```

```
## [1] "THORNDYKE AVE W"   "1ST AVE"           "1ST AVE"
## [4] "1ST AVE"           "UNIVERSITY WAY NE" "15TH AVE NW"
```

How Does the Building Number regex Work?

Let's break down `"^[0-9]*-?[A-Z]? (1/2 )?"`:

- `^[0-9]` means look for a digit between 0 and 9 (`[0-9]`) at the beginning (`^`)
- `*` means potentially match more digits after that
- `-?` means optionally (`?`) match a hyphen (`-`)
- `[A-Z]?` means optionally match (`?`) a letter (`[A-Z]`)
- Then we match a space (` `)
- `(1/2 )?` optionally matches a 1/2 followed by a space since this is apparently a thing with some address numbers

Removing the Street Numbers

```
restaurants <- restaurants %>%
  mutate(street_only = str_replace(Address, address_number_pattern,
                                   replacement = ""))
restaurants %>% distinct(street_only) %>% head(11)
```

```
## # A tibble: 11 x 1
##    street_only
##    <chr>
##  1 SW AVALON WAY
##  2 MERCER ST
##  3 FAIRVIEW AVE N UNIT 1700A
##  4 1ST AVE
##  5 E VALLEY HWY
##  6 11TH AVE E
##  7 112TH AVE NE #125
##  8 NE BOTHELL WAY
##  9 NW GILMAN BL C-08
## 10 NE 20TH ST STE 300
## 11 S ORCAS ST
```

How About Units/Suites Too?

Getting rid of unit/suite references is tricky, but a decent attempt would be to drop anything including and after "#", "STE", "SUITE", "SHOP", "UNIT":

```
address_unit_pattern <- " (#|STE|SUITE|SHOP|UNIT).*$"
address_unit_test_examples <-
  c("1ST AVE", "RAINIER AVE S #A", "FAUNTLEROY WAY SW STE 108",
    "4TH AVE #100C", "NW 54TH ST")
str_replace(address_unit_test_examples, address_unit_pattern,
            replacement = "")
```

```
## [1] "1ST AVE"           "RAINIER AVE S"     "FAUNTLEROY WAY SW"
## [4] "4TH AVE"           "NW 54TH ST"
```

How'd the Unit regex Work?

Breaking down `" (|#|STE|SUITE|SHOP|UNIT).*$"`:

- First we match a space
- `(#|STE|SUITE|SHOP|UNIT)` matches one of those words
- `.*$` matches *any* character (`.`) after those words, zero or more times (`*`), until the end of the string (`$`)

Removing Units/Suites

```
restaurants <- restaurants %>%
  mutate(street_only = str_trim(str_replace(street_only,
                             address_unit_pattern, replacement = "")))
restaurants %>% distinct(street_only) %>% head(11)
```

```
## # A tibble: 11 x 1
##    street_only
##    <chr>
##  1 SW AVALON WAY
##  2 MERCER ST
##  3 FAIRVIEW AVE N
##  4 1ST AVE
##  5 E VALLEY HWY
##  6 11TH AVE E
##  7 112TH AVE NE
##  8 NE BOTHELL WAY
##  9 NW GILMAN BL C-08
## 10 NE 20TH ST
## 11 S ORCAS ST
```

Where Does Danger Lurk?

Let's get one row per restaurant per date with the score, and see which streets the ones above 45 are on:

```r
restaurants %>%
  distinct(Business_ID, Date, Inspection_Score, street_only) %>%
  filter(Inspection_Score > 45) %>%
  count(street_only) %>%
  arrange(desc(n)) %>%
  head(n=5)
```

```
## # A tibble: 5 x 2
##   street_only           n
##   <chr>             <int>
## 1 UNIVERSITY WAY NE   108
## 2 S JACKSON ST        105
## 3 PACIFIC HWY S        90
## 4 NE 24TH ST           76
## 5 RAINIER AVE S        70
```

Splitting up Strings

You can split up strings using `tidyr::separate()`, seen in Week 5. Another option is `str_split()`, which will split strings based on a pattern separating parts and put these components in a list. `str_split_fixed()` will do that but with a matrix instead (and thus can't have varying numbers of separators):

```
head(str_split_fixed(restaurants$Violation_Description, " - ", n = 2))
```

```
##      [,1]  
## [1,] ""    
## [2,] "4300"
## [3,] "4800"
## [4,] "1200"
## [5,] "4100"
## [6,] "2120"
##      [,2]
## [1,] ""
## [2,] "Non-food contact surfaces maintained and clean"
## [3,] "Physical facilities properly installed,..."
## [4,] "Proper shellstock ID; wild mushroom ID;  parasite destruction procedures for fish"
## [5,] "Warewashing facilities properly installed,..."
## [6,] "Proper cold holding temperatures ( 42 degrees F to 45 degrees F)"
```

Other Useful `stringr` Functions

`str_pad(string, width, side, pad)`: Adds "padding" to any string to make it a given minimum width.

`str_subset(string, pattern)`: Returns all elements that contain matches of the pattern.

`str_which(string, pattern)`: Returns numeric indices of elements that match the pattern.

`str_replace_all(string, pattern, replacement)`: Performs multiple replacements simultaneously

Coming Up

Homework 6, Part 1 is due next week, and peer reviews due the week after.