CSSS508, Lecture 8

Working with Text Data

Michael Pearce

(based on slides from Chuck Lanfear)

May 17, 2022

UW CS&SS

Topics

Last time, we learned about,

1. Aside: Visualizing the Goal
2. Building blocks of functions
3. Simple functions
4. Using functions with `apply()`

Today, we will cover,

1. Basics of Strings
2. Strings in Base R
3. Strings in `stringr` (tidyverse)

1. Basics of Strings

Basics of Strings

- A general programming term for a unit of character data is a **string**
  - Strings are a *sequence of characters*
  - In R, "strings" and "character data" are mostly interchangeable.
  - Some languages have more precise distinctions, but we won't worry about that here!
- We can create strings by surrounding text, numbers, spaces, or symbols with quotes!
  - Examples: `"Hello! My name is Michael"` or `"%*$#01234"`

Basics of Strings

R can treat strings in funny ways!

```r
"01" == "1"
```

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

```r
"01" == 1
```

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

```r
"1" == 1
```

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

Reminder: We can check **data types** using the `class()` function!

```r
c(class("1"),class(1))
```

```
## [1] "character" "numeric"
```

2. Strings in Base R

- `nchar()`
- `substr()`
- `paste()`

Data: King County Restaurant Inspections!

Today we'll study real data on **food safety inspections in King County**, collected from data.kingcounty.gov.

Note these data are *fairly large*. The following code can be used to download the data directly from my Github page:

```
load(url("https://pearce790.github.io/CSSS508/Lectures/Lecture8/resta
```

Quick Examination of the Data

```
names(restaurants)
```

```
##  [1] "Name"                       "Program_Identifier"
##  [3] "Inspection_Date"            "Description"
##  [5] "Address"                    "City"
##  [7] "Zip_Code"                   "Phone"
##  [9] "Longitude"                  "Latitude"
## [11] "Inspection_Business_Name"   "Inspection_Type"
## [13] "Inspection_Score"           "Inspection_Result"
## [15] "Inspection_Closed_Business" "Violation_Type"
## [17] "Violation_Description"      "Violation_Points"
## [19] "Business_ID"                "Inspection_Serial_Num"
## [21] "Violation_Record_ID"        "Grade"
## [23] "Date"
```

```
dim(restaurants)
```

```
## [1] 258630     23
```

Quick Examination of the Data

Good Questions to Ask:

- What does each row represent?
- Is the data in long or wide format?
- What are the key variables?
- How are the data stored? (*data type*)

`nchar()`

The `nchar()` function calculates the *number of characters* in a given string.

- `length()` doesn't work with strings!!
- Why not?

```
nchar("Mike Pearce")
```

```
## [1] 11
```

In our `restaurants` data, let's see how many characters are in each zip code:

```
length_zip <- nchar(restaurants$Zip_Code)
table(length_zip)
```

```
## length_zip
##      5     10
## 258629      1
```

substr()

The substr() function allows us to extract characters from a string.

For example, we can extract the third through fifth elements of a string as follows:

```
substr("98126",3,5)
```

```
## [1] "126"
```

`substr()`

Let's extract the first five chararacters from each zip code in the restaurants data, and add it to our dataset.

```r
library(dplyr)
restaurants$ZIP_5 <- substr(restaurants$Zip_Code,1,5)
restaurants %>% distinct(ZIP_5) %>% head()
```

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

`paste()`

We combine strings together using `paste()`. By default, it puts a space between different strings.

For example, we can combine `"Michael"` and `"Pearce"` as follows:

```r
paste("Michael","Pearce")
```

```
## [1] "Michael Pearce"
```

More complex `paste()` commands

There are two additional common arguments to use with `paste()`:

1. `sep=` controls what separates vectors, entry-wise
2. `collapse=` controls if/how multiple outputs are collapsed into a single string.

```
paste("CSSS","508",sep= "_")
```

```
## [1] "CSSS_508"
```

```
paste(c("CSSS","STAT"),"508",sep= "_")
```

```
## [1] "CSSS_508" "STAT_508"
```

```
paste(c("CSSS","STAT"),"508",sep= "_",collapse=" , ")
```

```
## [1] "CSSS_508 , STAT_508"
```

When do we get one string as output vs. two?

`paste()`

Let's use `paste()` to create complete mailing addresses for each restaurant:

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

```
## # A tibble: 6 × 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
```

3. Strings in `stringr`

- `str_length()`
- `str_sub()`
- `str_c()`
- `str_to_upper()`, `str_to_lower()`, and `str_to_title()`
- `str_trim()`
- `str_detect()`
- `str_replace()`

`stringr`

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

It provides TONS of functions for working with strings:

- Some are equivalent/better versions of Base R functions
- Some can do *fancier* tricks with strings

Most `stringr` functions begin with "`str_`" to make RStudio auto-complete more useful.

We'll cover the basics today, but know there's much more out there!

```r
library(stringr)
```

Equivalencies: `str_length()`

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

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

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

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

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

Equivalencies: `str_sub()`

`str_sub()` is like `substr()`:

```r
str_sub("Washington", 2,4)
```

```
## [1] "ash"
```

`str_sub()` 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", 4, -3)
```

```
## [1] "hingt"
```

Equivalencies: `str_c()`

`str_c()` ("string combine") is just like `paste()` but where the default is `sep = ""` (no space!)

```r
str_c(c("CSSS","STAT"),508)
```

```
## [1] "CSSS508" "STAT508"
```

```r
str_c(c("CSSS","STAT"),508,sep=" ")
```

```
## [1] "CSSS 508" "STAT 508"
```

```r
str_c(c("CSSS","STAT"),508,sep = " ",collapse = ", ")
```

```
## [1] "CSSS 508, STAT 508"
```

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:

```
unique_cities <- unique(restaurants$City)
unique_cities %>% head()
```

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

```
str_to_upper(unique_cities) %>% head()
```

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

```
str_to_lower(unique_cities) %>% head()
```

```
## [1] "seattle"  "seattle"  "kent"     "bellevue" "kenmore"  "issaquah"
```

```
str_to_title(unique_cities) %>% head()
```

```
## [1] "Seattle"  "Seattle"  "Kent"     "Bellevue" "Kenmore"  "Issaquah"
```

Whitespace: `str_trim()`

Extra leading or trailing whitespace is common in text data:

```
unique_names <- unique(restaurants$Name)
unique_names %>% head(3)
```

```
## [1] "@ THE SHACK, LLC "    "10 MERCER RESTAURANT"
## [3] "100 LB CLAM"
```

We can remove the whitespace using `str_trim()`:

```
str_trim(unique_names) %>% head(3)
```

```
## [1] "@ THE SHACK, LLC"     "10 MERCER RESTAURANT"
## [3] "100 LB CLAM"
```

Patterns!

It's common to want to see if a string satisfies a certain *pattern*.

We did this with numeric values earlier in this course!

```
cars %>% filter(speed < 5 | speed > 24)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3    25   85
```

```
cars %>% filter(dist > 2 & dist <= 10)
```

```
##   speed dist
## 1     4   10
## 2     7    4
## 3     9   10
```

Patterns: `str_detect()`

We can do similar pattern-checking using `str_detect()`:

```
str_detect(string,pattern)
```

- `string` is the character string (or vector of strings) we want to examine
- `pattern` is the pattern that we're checking for inside `string`
- Output: TRUE/FALSE vector indicating if pattern was found

```
str_detect(string = c("Hello","my name","is Michael"),
           pattern = "m")
```

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

```
str_detect(string = c("Hello","my name","is Michael"),
           pattern = "M")
```

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

Results are case-sensitive!!

Patterns: `str_detect()`

Let's see which phone numbers are in the 206 area code:

```r
unique_phones <- unique(restaurants$Phone)
unique_phones %>% tail(4)
```

```
## [1] "(360) 698-0417" "(206) 525-7747" "(206) 390-9205"
## [4] "(425) 557-4474"
```

```r
str_detect(unique_phones,"206") %>% tail(4)
```

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

Replacement: `str_replace()`

What about if you want to replace a string with something else? Use `str_replace()`!

This function works very similarly to `str_detect()`, but with one extra argument:

```
str_replace(string, pattern, replacement)
```

- `replacement` is what `pattern` is substituted for.

```
str_replace(string="Hi, I'm Michael",
            pattern="Hi",replacement="Hello")
```

```
## [1] "Hello, I'm Michael"
```

Replacement: `str_replace()`

In the `Date` variable, let's replace each dash ("-") with an underscore ("_")

```
dates <- restaurants$Date
dates %>% tail(3)
```

```
## [1] "2017-03-21" "2017-03-21" "2016-10-10"
```

```
str_replace(dates,"-","_") %>% tail(3)
```

```
## [1] "2017_03-21" "2017_03-21" "2016_10-10"
```

Wait, what?

Replacement: `str_replace_all()`

`str_replace()` only changes the **first** instance of a pattern in each string!

If we want to replace **all** patterns, use `str_replace_all()`

```
dates <- restaurants$Date
dates %>% tail(3)
```

```
## [1] "2017-03-21" "2017-03-21" "2016-10-10"
```

```
str_replace_all(dates,"-","_") %>% tail(3)
```

```
## [1] "2017_03_21" "2017_03_21" "2016_10_10"
```

Quick Summary

We've seen lots of functions today!

Don't try to memorize them! Instead, use this page as a reference.

- Character Length: `nchar` and `str_length`
- Subsetting: `substr` and `str_sub`
- Combining: `paste` and `str_c`
- Case Changes: `str_to_upper()`, `str_to_lower()`, and `str_to_title()`
- Removing Whitespace: `str_trim`
- Pattern Detection/Replacement: `str_detect()` and `str_replace()`

Activity 1: Base R Functions

The variable `Inspection_Date` is in the format "MM/DD/YYYY". In this question, we'll change the format using functions for strings.

1. How long is each character string in this variable?
2. Use `substr()` to extract the month of each entry and save it to an object called "months"
3. Use `substr()` to extract the year of each entry and save it to an object called "years"
4. Use `paste()` to combine each month and year, separated by an underscore (`_`). Save this as a new variable in the data called "Inspection_Date_Formatted"

Activity: My Answers

The variable `Inspection_Date` is in the format "MM/DD/YYYY". In this question, we'll change the format using functions for strings.

1.How long is each character string in this variable?

```
table(nchar(restaurants$Inspection_Date))
```

```
##
##     10
## 258000
```

2.Use `substr()` to extract the month of each entry and save it to an object called "months"

3.Use `substr()` to extract the year of each entry and save it to an object called "years"

```
months <- substr(restaurants$Inspection_Date,1,2)
years <- substr(restaurants$Inspection_Date,7,10)
```

Activity: My Answers

4.Use `paste()` to combine each month and year, separated by an underscore (`_`). Save this as a new variable in the data called "Inspection_Date_Formatted"

```
restaurants$Inspection_Date_Formatted <-
  paste(months,years,sep="_")
restaurants %>%
  select(Name,Inspection_Date,Inspection_Date_Formatted) %>%
  head(5)
```

```
## # A tibble: 5 × 3
##   Name                   Inspection_Date Inspection_Date_Formatted
##   <chr>                  <chr>           <chr>
## 1 "@ THE SHACK, LLC "    <NA>            NA_NA
## 2 "10 MERCER RESTAURANT" 01/24/2017      01_2017
## 3 "10 MERCER RESTAURANT" 01/24/2017      01_2017
## 4 "10 MERCER RESTAURANT" 01/24/2017      01_2017
## 5 "10 MERCER RESTAURANT" 10/10/2016      10_2016
```

Activity 2: HW 8

Let's examine the coffee shops of King County!

1. Filter your data to only include rows in which the `Name` includes the word "coffee" (in any case!)

2. Create a new variable in your data which includes the length of the business name, after removing beginning/trailing whitespace.

3. Create a new variable in your data for the inspection year, *using a `stringr` function!*

4. Create side-by-side boxplots for the length of business name vs. year.

5. Calculate the maximum `Inspection_Score` by business and year.

6. Create a line plot of maximum score ("MaxScore") over time ("Year"), by business ("Name"). That is, you should have a single line for each business. (Don't try to label them, as there are far too many!)

Activity: My Solutions

1. Filter your data to only include rows in which the `Name` includes the word "coffee" (in any case!)

```
coffee <- restaurants
coffee$Name <- str_to_lower(coffee$Name)
coffee <- coffee %>% filter(str_detect(Name,"coffee"))
```

Activity: My Solutions

2.Create a new variable in your data which includes the length of the business name, after removing beginning/trailing whitespace.

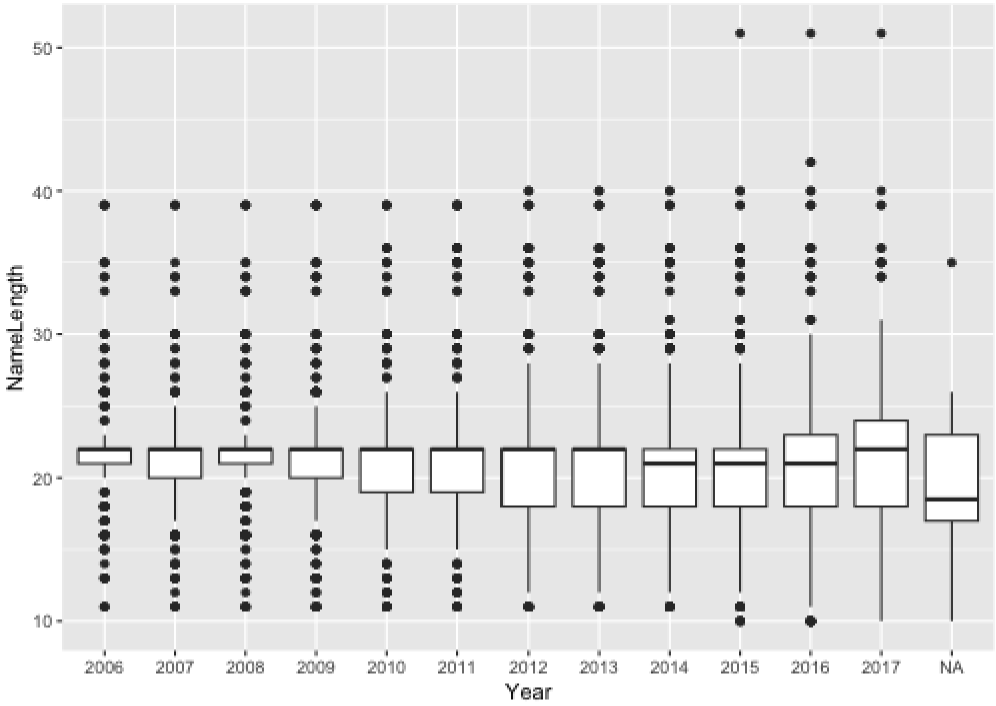
3. Create a new variable in your data for the inspection year.

```
coffee$NameLength <- str_length(str_trim(coffee$Name))
coffee$Year <- str_sub(coffee$Inspection_Date,-4,-1)
```

Activity: My Solutions

4. Create side-by-side boxplots for the length of business name vs. year.

```
library(ggplot2)
ggplot(coffee,aes(Year,NameLength))+geom_boxplot()
```



Activity: My Solutions

5. Calculate the maximum `Inspection_Score` by business and year.

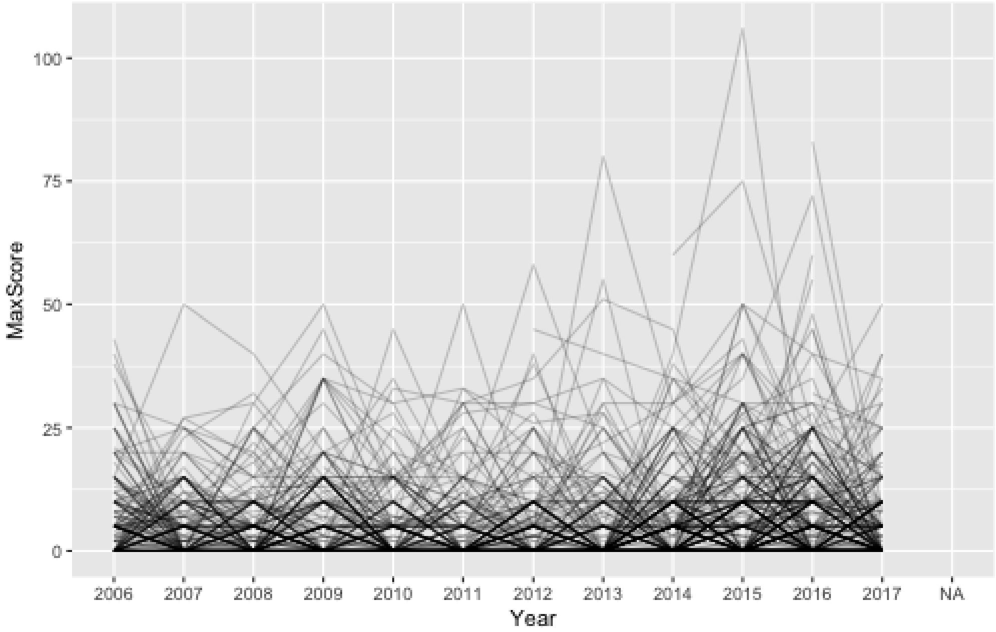
```r
coffee_summary <- coffee %>% group_by(Name,Year) %>%
  summarize(MaxScore=max(Inspection_Score))
```

```
## `summarise()` has grouped output by 'Name'. You can override using
## the `.groups` argument.
```

Activity: My Solutions

6. Create a line plot of maximum score ("MaxScore") over time ("Year"), by business ("Name"). That is, you should have a single line for each business. (Don't try to label them, as there are far too many!)

```
ggplot(coffee_summary,aes(Year,MaxScore,group=Name))+
  geom_line(alpha=.2)
```


MaxScore
NA
Year