Introduction to Strings/Text

STA 325, Supplemental Material

Rebecca C. Steorts

Agenda

- Introduction to strings and string operations
- Extracting and manipulating string objects
- Introduction to general search

Why Characters?

Most data we deal with is in character form!

- web pages can be scraped
- emails can be analyzed for network properties
- survey responses must be processed and compared

Even if you only care about numbers, it helps to be able to extract them from text and manipulate them easily.

Characters versus Strings

- ***Character***: a symbol in a written language, specifically what you can enter at a keyboard: letters, numerals, punctuation, space, newlines, etc.

```
'L', 'i', 'n', 'c', 'o', 'l'
```

- ***String***: a sequence of characters bound together

```
Lincoln
```

Characters/Strings

R does not have a separate type for characters and strings

```
mode("L")
```

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

```
mode("Lincoln")
```

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

```
class("Lincoln")
```

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

Making Strings

- To construct a string use single or double quotations.
- Use `nchar()` to get the length of a single string.

Constructing Strings

```
"Lincoln"
```

```
## [1] "Lincoln"
```

```
"Abraham Lincoln"
```

```
## [1] "Abraham Lincoln"
```

```
"Abraham Lincoln's Hat"
```

```
## [1] "Abraham Lincoln's Hat"
```

```
"As Lincoln never said, \"Four score and seven beers ago\""
```

```
## [1] "As Lincoln never said, \"Four score and seven beers ago\""
```

Whitespace

- The space, `" "` is a character
- Multiple spaces `"  "` and the empty string `""` are also characters
- Some characters are special, so we have "escape characters" to specify them in strings:

1. quotes within strings: `\"`
2. tab: `\t`
3. new line `\n` and carriage return `\r` (use the former rather than the latter when possible)

The character data type

- The character data type is one of the atomic data types, like `numeric` or `logical`
- The character data type can go into scalars, vectors, arrays, lists, or can be the type of a column in a data frame.

Constructing Strings

```
length("Abraham Lincoln's beard")
```

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

```
length(c("Abraham", "Lincoln's", "beard"))
```

```
## [1] 3
```

```
nchar("Abraham Lincoln's beard")
```

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

```
nchar(c("Abraham", "Lincoln's", "beard"))
```

```
## [1] 7 9 5
```

Character-Valued Variables

They work just like others, e.g., with vectors:

```
president <- "Lincoln" # variable assignment
nchar(president)  # not 9
```

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

```
presidents <-
  c("Fillmore","Pierce",
    "Buchanan","Davis","Johnson")
presidents[3] # shows element 3
```

```
## [1] "Buchanan"
```

```
presidents[-(1:3)] # removes elements 1 through 3
```

```
## [1] "Davis"   "Johnson"
```

Displaying Characters

- `print()`: directly prints to the console
- `cat()` writes the string directly to the console (removes the quotations)
- `message()`: Is for debugging purposes (read about more on your own from the R help files).

Displaying Characters

```
print("Abraham Lincoln")
```

```
## [1] "Abraham Lincoln"
```

```
cat("Abraham Lincoln")
```

```
## Abraham Lincoln
```

```
cat(presidents)
```

```
## Fillmore Pierce Buchanan Davis Johnson
```

```
message(presidents)
```

```
## FillmorePierceBuchananDavisJohnson
```

Substring Operations

Substring: This is a subset of a larger string. It's also a string in it's own right.

A string is not a vector or a list, so we ***cannot*** use subscripts like `[[ ]]` or `[ ]` to extract substrings.

To work with substring, we use `substr()` instead.

We take the string and break it into a small string (substring) beginning with the start character and ending with the stop character.

Substring Operations

```
phrase <- "Christmas Bonus"
substr (phrase, start=8, stop=12)
```

```
## [1] "as Bo"
```

We can also use `substr` to replace elements:

```
substr(phrase, 13, 13) <- "g"
phrase
```

```
## [1] "Christmas Bogus"
```

Does white space count as a character? Yes!

```
substr(phrase, 10, 10) <- "Z"
phrase
```

```
## [1] "ChristmasZBogus"
```

substr() for String Vectors

`substr()` vectorizes over all its arguments.

Let's recall our variable that we created earlier of a few presidents.

```
presidents
```

```
## [1] "Fillmore" "Pierce"   "Buchanan" "Davis"    "Johnson"
```

substr() for String Vectors

```
presidents
```

```
## [1] "Fillmore" "Pierce"   "Buchanan" "Davis"    "Johnson"
```

```
substr(presidents,1,2)   # First two characters
```

```
## [1] "Fi" "Pi" "Bu" "Da" "Jo"
```

substr() for String Vectors

```
presidents
```

```
## [1] "Fillmore" "Pierce"   "Buchanan" "Davis"    "Johnson"
```

```
nchar(presidents)
```

```
## [1] 8 6 8 5 7
```

```
# Last two characters
substr(presidents,nchar(presidents)-1,nchar(presidents))
```

```
## [1] "re" "ce" "an" "is" "on"
```

substr() for String Vectors

```
presidents
```

```
## [1] "Fillmore" "Pierce"   "Buchanan" "Davis"    "Johnson"
```

```
# No such substrings so return the null string
substr(presidents,20,21)
```

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

substr() for String Vectors

```
presidents
```

```
## [1] "Fillmore" "Pierce"   "Buchanan" "Davis"    "Johnson"
```

```
# Explain!
substr(presidents,7,7)
```

```
## [1] "r" ""  "a" ""  "n"
```

Dividing Strings into Vectors

`strsplit()` divides a string according to key characters by splitting each element of the character vector `x` at appearances of the pattern `split`.

The output is a list of character vectors.

Dividing Strings into Vectors

```
scarborough.fair <- "parsley, sage, rosemary, thyme"
strsplit (scarborough.fair, ",")
```

```
## [[1]]
## [1] "parsley"   " sage"     " rosemary" " thyme"
```

```
strsplit (scarborough.fair, ", ")
```

```
## [[1]]
## [1] "parsley"  "sage"     "rosemary" "thyme"
```

Dividing Strings into Vectors

Pattern is recycled over elements of the input vector:

```
strsplit(c(scarborough.fair,
           "Garfunkel, Oates", "Clement, McKenzie"), ", ")
```

```
## [[1]]
## [1] "parsley"  "sage"     "rosemary" "thyme"
##
## [[2]]
## [1] "Garfunkel" "Oates"
##
## [[3]]
## [1] "Clement"  "McKenzie"
```

Combining Vectors into Strings

Converting one variable type to another is called ***casting***:

```
as.character(7.2)                 # Obvious
```

```
## [1] "7.2"
```

```
as.character(7.2e12)              # Obvious
```

```
## [1] "7.2e+12"
```

```
as.character(c(7.2,7.2e12))   # Obvious
```

```
## [1] "7.2"     "7.2e+12"
```

```
as.character(7.2e5)               # Not quite so obvious
```

```
## [1] "720000"
```

Building strings from multiple parts

The `paste()` function is very flexible!

With one vector argument, works like `as.character()`:

```
paste(41:45)
```

```
## [1] "41" "42" "43" "44" "45"
```

Building strings from multiple parts

- We can build strings from multiple parts easily.
- For example, with 2 or more vector arguments, we can combine them with recycling:

```
paste(presidents,41:45) # None are historically accurate!
```

```
## [1] "Fillmore 41" "Pierce 42"   "Buchanan 43" "Davis 44"    "Johnson 45"
```

```
paste(presidents,c("R","D"))
```

```
## [1] "Fillmore R" "Pierce D"   "Buchanan R" "Davis D"    "Johnson R"
```

```
paste(presidents,"(",c("R","D"),41:45,")")
```

```
## [1] "Fillmore ( R 41 )" "Pierce ( D 42 )"   "Buchanan ( R 43 )"
## [4] "Davis ( D 44 )"    "Johnson ( R 45 )"
```

A More Complicated Example of Recycling

Exercise: Convince yourself of why this works as it does

```
rep(1:11,times=rep(2,11))
```

```
##  [1]  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10 11 11
```

```
paste(c("HW","Lab"),rep(1:11,times=rep(2,11)))
```

```
##  [1] "HW 1"   "Lab 1"  "HW 2"   "Lab 2"  "HW 3"   "Lab 3"  "HW 4"
##  [8] "Lab 4"  "HW 5"   "Lab 5"  "HW 6"   "Lab 6"  "HW 7"   "Lab 7"
## [15] "HW 8"   "Lab 8"  "HW 9"   "Lab 9"  "HW 10"  "Lab 10" "HW 11"
## [22] "Lab 11"
```

Condensing Multiple Strings

Default value of `collapse` is `NULL` (meaning the function doesn't use it)

Let's produce one large string and utilize `collapse`

```
paste(presidents, " (", 41:45, ")", sep="", collapse="; ")
```

```
## [1] "Fillmore (41); Pierce (42); Buchanan (43); Davis (44); Johnson (45)"
```

Text of Some Importance

If we shall suppose that American slavery is one of those offenses which, in the providence of God, must needs come, but which, having continued through His appointed time, He now wills to remove, and that He gives to both North and South this terrible war as the woe due to those by whom the offense came, shall we discern therein any departure from those divine attributes which the believers in a living God always ascribe to Him? Fondly do we hope, fervently do we pray, that this mighty scourge of war may speedily pass away. Yet, if God wills that it continue until all the wealth piled by the bondsman's two hundred and fifty years of unrequited toil shall be sunk, and until every drop of blood drawn with the lash shall be paid by another drawn with the sword, as was said three thousand years ago, so still it must be said "the judgments of the Lord are true and righteous altogether."

- Abraham Lincoln, Second Inaugural Address, 1865

Lincoln's Second Inaugural Address (1865)

The full text can be found in `lincoln.txt`

Let's start by looking at a summary of this textual data

```
al2 <- readLines("lincoln.txt")
length(al2)
```

```
## [1] 58
```

```
head(al2)
```

```
## [1] "Fellow-Countrymen:"
## [2] ""
## [3] "At this second appearing to take the oath of the Presidential office there is"
## [4] "less occasion for an extended address than there was at the first.  Then a"
## [5] "statement somewhat in detail of a course to be pursued seemed fitting and"
## [6] "proper.  Now, at the expiration of four years, during which public declarations"
```

al2 is a vector, one element per line of text

A Hint Of The Future: Search

Narrowing down entries: use `grep()` to find which strings have a matching search term

1. grep: returns the index value of each matched pattern
2. grepl: returns a logical output for each indices in the original vector with "TRUE" representing matched patterns

grep versus grepl

```
grep("God", al2)
```

```
## [1] 34 35 41 45 47 54
```

```
grepl("God", al2)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [34]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [45]  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [56] FALSE FALSE FALSE
```

```
al2[grep("God", al2)] # Returns the strings of matches
```

```
## [1] "God, and each invokes His aid against the other.  It may seem strange t
## [2] "men should dare to ask a just God's assistance in wringing their bread
## [3] "offenses which, in the providence of God, must needs come, but which, h
## [4] "attributes which the believers in a living God always ascribe to Him?
## [5] "pass away.  Yet, if God wills that it continue until all the wealth pil
## [6] "God gives us to see the right, let us strive on to finish the work we a
```

Task 1: Breaking words apart

Task 1: How would one make a long string using the text and then split all the words?

Task 2: Then how would we count how often each word appears and put these in order?

Task 3: What do we find that is unexpected and expected from this simple exercise?

```
al2 <- paste(al2, collapse=" ")
al2.words <- strsplit(al2, split=" ")[[1]]
head(al2.words)
```

```
## [1] "Fellow-Countrymen:" ""                   "At"
## [4] "this"               "second"             "appearing"
```

Task 2: Counting Words with table()

Tabulate how often each word appears, put in order:

```
wc <- table(al2.words)
wc <- sort(wc,decreasing=TRUE)
head(wc,20)
```

```
## al2.words
##   the    to           and    of  that   for    be    in    it     a  this
##    54    26    25    24    22    11     9     8     8     8     7     7
##   war which   all    by    we  with    as   but
##     7     7     6     6     6     6     5     5
```

`names(wc)` gives all the distinct words in `al2.words` (***types***); `wc` counts how often they appear (***tokens***)

Task 3: Unexpected

The null string is the third-most-common word:

```
names(wc)[3]
```

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

```
wc["years"]
```

```
## years
##     3
```

```
wc["years,"]
```

```
## years,
##      1
```

Task 3: Unexpected

Capitalization:

```
wc["that"]
```

```
## that
##   11
```

```
wc["That"]
```

```
## That
##    1
```

All of this can be fixed if we learn how to work with text patterns and not just constants.

Summary

- Text is data, just like everything else
- `substr()` extracts and substitutes
- `strsplit()` turns strings into vectors
- `paste()` turns vectors into strings
- `table()` for counting how many tokens belong to each type

Next time: searching for text patterns using regular expressions