Lecture 4: Text Basics

Statistical Computing, 36-350

Monday September 21, 2015

Outline

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

Why characters?

A lot of data out there is in character form!

- Web pages can be scraped
- Email messages can be analyzed
- Survey responses can be processed

Even if you only care about numbers eventually, you may have to know how to extract them from text

The simplest distinction

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', 'n'
```

String: a sequence of characters bound together

```
"Lincoln"
```

(Note: R does not have a separate type for characters and strings)

```r
class("L")
```

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

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

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

Making strings

Use single or double quotes to construct a string; whatever you use as the starting choice has to match the end. Why do we prefer double quotes?

```r
"Lincoln"
```

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

```r
"Abraham Lincoln"
```

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

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

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

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

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

Use `nchar()` to get the length of a string:

```r
nchar("Lincoln")
```

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

```r
nchar("Abraham Lincoln's Hat")
```

```
## [1] 21
```

```r
nchar("As Lincoln never said, \"Four score and seven beers ago\"")
```

```
## [1] 55
```

Whitespace

The space `" "` is a character; so is the empty string `""`

Some characters are special, so we have "escape characters" to specify them in strings

- Quotes within strings: `\"`
- Tab: `\t`
- New line: `\n`

The character data type

One of the atomic data types, like `numeric` or `logical`

Can go into scalars, vectors, arrays, lists, or be the type of a column in a data frame

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

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

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

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

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

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

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

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

Character-valued variables

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

```r
president = "Lincoln"
nchar(president)
```

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

```r
presidents = c("Fillmore","Pierce","Buchanan","Davis","Johnson")
presidents[3]
```

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

```r
presidents[-(1:3)]
```

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

Substring operations

Substring: a smaller string from a bigger string, but still a string in its own right

A string itself is not a vector or a list, so we cannot use subscripts like `[[ ]]` or `[ ]` to extract substrings; we must use `substr()` instead

```
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"
```

Substrings with vectors

`substr()` vectorizes over all its arguments:

```
presidents
```

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

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

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

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

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

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

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

```
substr(presidents,7,7) # Can you explain?
```

```
## [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`, its second argument

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

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

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

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

Pattern is recycled over elements of the input vector:

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

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

Note that it outputs a `list` of character vectors—why is this a good format here?

Casting data types into strings

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

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

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

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

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

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

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

```r
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()`:

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

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

With 2 or more vector arguments, combines them with recycling:

```r
paste(presidents,41:45)
```

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

```r
paste(presidents,c("R","D"))# Not historically accurate!
```

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

```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 )"
```

We can change the separator between pasted-together items by setting the `sep` argument:

```r
paste(presidents, "(", 41:45, ")", sep="_")
```

```
## [1] "Fillmore_(_41_)" "Pierce_(_42_)"   "Buchanan_(_43_)" "Davis_(_44_)"
## [5] "Johnson_(_45_)"
```

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

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

(Exercise: what happens if you give `sep` a vector?)

Condensing multiple strings

We can produce one big string by setting the `collapse` argument:

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

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

(Note: the default value of `collapse` is `NULL`, i.e., it won't collapse anything)

A function for writing regression formulas

R has a standard syntax for models: outcome and predictors

```r
my.formula = function(dep,indeps,df) {
  rhs = paste(colnames(df)[indeps], collapse=" + ")
  return(paste(colnames(df)[dep], "~", rhs, collapse=""))
}
my.formula(2,c(3,5,7),df=state.x77)
```

```
## [1] "Income ~ Illiteracy + Murder + Frost"
```

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." . . .

Reading text from a file

```r
linc = readLines("http://www.stat.cmu.edu/~ryantibs/statcomp/lectures/lincoln.txt")
length(linc)
```

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

```r
head(linc)
```

```
## [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"
```

`linc` is a vector of strings, one element per line of text

A hint of next time: search

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

```r
grep("God", linc)
```

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

```r
grepl("God", linc)
```

```
##  [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
```

```r
linc[grep("God", linc)]
```

```
## [1] "God, and each invokes His aid against the other.  It may seem strange that any"
## [2] "men should dare to ask a just God's assistance in wringing their bread from the"
## [3] "offenses which, in the providence of God, must needs come, but which, having"
## [4] "attributes which the believers in a living God always ascribe to Him?  Fondly"
## [5] "pass away.  Yet, if God wills that it continue until all the wealth piled by"
## [6] "God gives us to see the right, let us strive on to finish the work we are in,"
```

Reconstituting

Make one long string, then split the words

```r
linc = paste(linc, collapse=" ")
linc.words = strsplit(linc, split=" ")[[1]]
length(linc.words)
```

```
## [1] 725
```

```r
head(linc.words)
```

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

Counting words

Tabulate how often each word appears, put in order:

```r
wc = table(linc.words)
wc = sort(wc,decreasing=TRUE)
head(wc,20)
```

```
## linc.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 `linc.words` (types); `wc` contains how often they appear (tokens)

A few undesirable results

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

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

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

Commas matter:

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

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

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

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

Capitalization matters:

```
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 pure strings

Summary

- Text is data, just like everything else
- `substr()` extracts and substitutes
- `strsplit()` splits a big string into smaller ones
- `paste()` bunches together smaller strings into a big one
- `table()` for counting how many tokens belong to each type

Next time: searching for text patterns using regular expressions