Reading a Web Log File

Strategy for Reading Data via Regular Expressions

Duncan Temple Lang

STA141B Spring 2023

We have a file eeyore.log that contains log messages from a Web server about requests it received. Each line provides information about a single request for a document. There are 8548 lines in the file.

Each line is of the form

```
114.188.183.88 - - [01/Nov/2015:03:41:50 -0800] "GET /stat141/Code/Session1.txt HTTP/1.1" \
404 223 "https://www.google.co.jp/" \
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/5
```

(Note that I have split this across 3 lines for formatting reasons. It is a single line in the file.)

Each line contains the following features/elements

- IP address of the machine requesting the document - 114.188.183.88
- optionl remote logname - `-`
- an optional login, here given as `-`, i.e., not provided
- date-time stamp - 01/Nov/2015:03:41:50 -0800
- a quoted triple
  - HTTP operation/command - GET
  - path to the document being requestion /stat141/Code/Session1.txt
  - protocol and version HTTP/1.1
- operation status - 404 for Not Found
- number of bytes returned - 223
- referral URL
- user browser information from which the request was made

We start by reading the lines from the file into a character vector in R:

```
ll = readLines('eeyore.log')
```

Our strategy is to incrementally build a regular expression that matches each entire line and uses capture groups to match the different features/elements lised above. We can then use gregexpr and the capture information to extract those elements for each line and construct a data.frame with those variables.

To probably simplify the process, we'll extract entire elements such as the quoted triple and then in a second step extract its sub-elements. We could do this all in one step, but the regular expression will get slightly more complex, but not much more so. So, in fact, we'll probably do the simpler case first and then enhance the regular expression to do it all in one go, having "banked" the simpler version.

We could write the entire regular expression in one go, but it is often better to incrementally grow it, checking at each step that it matches all of the lines. If we wrote the full regular expression and it didn't match, we would have a hard time determining why. So the incremental construction helps identify what didn't match as we know which part we added most recently that caused it to stop matching.

So we will start by capturing the IP address and the two - characters corresponding to the login names. We'll use grepl() to see what elements of `ll` the regular expression matched and did not match. We'll move to gregexpr() later when we want to see what parts matched and where the subpatterns matched within each string. But for now we are focused on matching each entire line or not:

```
w = grepl("^[0-9.]+ - -", ll)
```

Next we check all matched:

```
table(w)
```

```
## w
## FALSE  TRUE
##   161  8387
```

161 of the ~ 8500 didn't. Let's examine some of these and see why the regular expression didn't match those:

```
head(ll[!w])
```

These start wit

```
"67.166.147.49 - kcolson [01......."
```

So the second - could actually be a regular login name, not always a -. So let's change the regular expression to allow for a - OR a sequence of one or more lower case letters:

```
w = grepl("^[0-9.]+ - (-|[a-z]+)", ll)
table(w)
```

```
## w
## TRUE
## 8548
```

This matches all of them.

We could now move on to also matching the date-time stamp. However, we'll see that we still have a slight problem with matching the first login. Quick summary: we should add the trailing space after the second login to our regular expression to make certain that we matched the entire login. If it contained a number or an underscore _, our regexp would match but not all of the login word.

```
w = grepl("^[0-9.]+ - (-|[a-z]+) ", ll)
table(w)
```

```
## w
## FALSE  TRUE
##    35  8513
```

We didn't match 35 of the ~ 8500 lines Again, we examine these and see

```
"108.213.78.140 - rickyt1 [02
```

So we need to allow for numbers in the login:

```
w = grepl("^[0-9.]+ - (-|[a-z0-9]+) ", ll)
table(w)
```

```
## w
## TRUE
## 8548
```

This matches all of our lines.

If we had moved on to matching the date-time stamp before checking we matched the entire login text, we would be debugging 2 things - the different login name AND the complex date-time stamp text. So try to complete on one element entirely before moving onto the next.

The date-time stamp is enclosed with a `[ ]` pair. These are the same characters we use to identify a character class in a regular expression. So we have to escape each of these to have them be treated as literal values. We try

```
w = grepl("^[0-9.]+ - (-|[a-z]+) \\[^]+\\]", ll)
table(w)
```

```
## w
## FALSE
##  8548
```

This doesn't match any line. So time to debug.

What we are trying to do here is say

- find the literal `[`
- then one or more characters that is not `]`
- followed by the literal `]` character

This is a common idiom - start character, one or more of anything but the end character, the end character. We use this a lot for getting text in quotes - start quote, anything but that quote, that quote to end the sequence.

What we wrote above is just wrong as a regular expression but we didn't get an error. It was not actually wrong, but it certainly wasn't what we meant. We can try to use `perl = TRUE` to see if it raises an error,

```
w = grepl("^[0-9.]+ - (-|[a-z]+) \\[^]+\\]", ll)
```

but it doesn't either.

It would be easier to read if we could put spaces between the elements but then these would part of the text to match. But let's do this as a thought experiment. We have

```
 \\[  ^]+ \\]
```

The middle part is along the lines of what we want but incorrect. We want

```
 \\[  [^]]+ \\]
```

That is, the middle part should be a character set/class so enclosed within `[ ]` and what we want within the set is "any character except]" We indicate this with `^]` so we should have `[^]]` We may need to escape the `]` within the `[ ]`

```
w = grepl("^[0-9.]+ - (-|[a-z]+) \\[[^\\]]+\\]", ll)
table(w)
```

```
## w
## FALSE
##  8548
```

We should add the 0-9 for the login part, but that is not what is causing not match at all. We'll also use the `perl = TRUE`

```
w = grepl("^[0-9.]+ - (-|[a-z0-9]+) \\[[^\\]]+\\]", ll, perl = TRUE)
table(w)
```

```
## w
## TRUE
## 8548
```

This matches all the lines. We could also not escape the `]` within the character set markers, i.e., and use either `perl = TRUE` or FALSE:

```
w = grepl("^[0-9.]+ - (-|[a-z0-9]+) \\[[^]]+\\]", ll)
table(w)
```

```
## w
## TRUE
## 8548
```

This is slightly confusing. While it is more typing, it is probably a good idea to use `perl = TRUE` throughout.

We now move onto the next element which is the

`"GET /stat141/Code/Session1.txt HTTP/1.1"`

We could read the GET and the file and the HTTP and the 1.1. However, as we mentioned earlier, we are going to get the entire item in quotes and post-process it later to extract the elements.

So we match

- “,
- one or more characters that are not ”
- a ”

We do this with `"[^"]+"`. We add this to the end of our regular expression. We also enclose the part within the “ ” pair via () to capture the group. So we have

```
w = grepl('^([0-9.]+) - (-|[a-z0-9]+) \\[([^]]+)\\] "([^"]+)" ', ll)
table(w)
```

```
## w
## TRUE
## 8548
```

Note that we also add the following space based on what we learned from only matching part of the login.

The next two fields are the status and the number of bytes and should be integers.

```
w = grepl('^([0-9.]+) - (-|[a-z0-9]+) \\[([^]]+)\\] "([^"]+)" ([0-9]+) ([0-9]+)', ll)
table(w)
```

```
## w
## FALSE  TRUE
##  2537  6011
```

So again, let’s look at what didn’t match.

```
head(ll[!w])
```

```
## [1] "68.180.229.48 - - [01/Nov/2015:05:17:05 -0800] \"GET /stat242/data/covtype.info HTTP/1.1\" 304 -
## [2] "68.180.229.48 - - [01/Nov/2015:05:19:19 -0800] \"GET /stat242/Maak_d.R HTTP/1.1\" 304 - \"-\" \"
## [3] "68.180.229.48 - - [01/Nov/2015:06:18:53 -0800] \"GET /stat242/Lectures/Lec16 HTTP/1.1\" 304 - \"
## [4] "137.208.57.141 - - [01/Nov/2015:06:37:16 -0800] \"HEAD /cgi-bin/testForm1.pl HTTP/1.1\" 404 - \"
## [5] "98.238.243.161 - - [01/Nov/2015:08:05:33 -0800] \"GET /stat141/Class.css HTTP/1.1\" 304 - \"http
## [6] "98.238.243.161 - - [01/Nov/2015:08:05:45 -0800] \"GET /stat141/Homeworks.html HTTP/1.1\" 304 - "
```

We see that the “number of bytes” term may be `-`, so we add that as a possibility

```
w = grepl('^([0-9.]+) - (-|[a-z0-9]+) \\[([^]]+)\\] "([^"]+)" ([0-9]+) (-|[0-9]+)', ll)
table(w)
```

```
## w
## TRUE
## 8548
```

Note that it is either - or a sequence of digits. We don’t add the - to the character set of digits which would match -123 or 1-23.

Now we match all lines again. We capture the last two terms which are both enclosed/surrounded by quotes, like the fifth term (the operation, file and protocol).

```
w = grepl('^([0-9.]+) - (-|[a-z0-9]+) \\[([^]]+)\\] "([^"]+)" ([0-9]+) (-|[0-9]+) "([^"]+)" "([^"]+)"',
table(w)
```

```
## w
## FALSE  TRUE
##     1  8547
```

This matches all but one line:

```
ll[!w]
```

```
## [1] "41.220.68.249 - - [03/Nov/2015:02:41:58 -0800] \"GET /stat141/Lectures/Day1.pdf HTTP/1.1\" 200
```

After some thinking, we find the issue is that the referer term is empty, i.e., "". Our (sub)pattern is "([^"]+)" which requires one or more characters between the "". We can change this to zero-or-more by replacing the + with *:

```
w = grepl('^([0-9.]+) - (-|[a-z0-9]+) \\[([^]]+)\\] "([^"]+)" ([0-9]+) (-|[0-9]+) "([^"]*)" "([^"]+)"',
table(w)
```

```
## w
## TRUE
## 8548
```

Now we seem to match all lines. While we probably have the correct matches, this is not guaranteed yet. We may match the entire line, but not the correct sub-patterns to the correct content in all lines. So now we turn to extracting the matches for the subpatterns using gregexpr().

```
m = gregexpr('^([0-9.]+) - (-|[a-z0-9]+) \\[([^]]+)\\] "([^"]+)" ([0-9]+) (-|[0-9]+) "([^"]*)" "([^"]+)
```

We check how many matches there are for each line

```
table(sapply(m, length))
```

```
##
##    1
## 8548
```

So all have length 1. The value is the starting position for each match and it should be 1, the start of the string. Let's confirm this

```
table(sapply(m, `[`, 1) == 1)
```

```
##
## TRUE
## 8548
```

Let's also verify that the length of the entire match is the number of characters in each string/line:

```
mlen = sapply(m, function(x) attr(x, "match.length"))
table(mlen)
```

```
## mlen
##  77  83  88  89  90  91  92  95  96  97  99 100 101 102 103 104 111 112 113 115 117 118 119 121 122
##   1   3   1   3   4   1   2   4   2   1  10   4   8   5  28   2   2   1   1   4   7   1   3   1   1
## 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
##   2   3   1   6  20  15  16  11  11  25  19  21  75  29  36  61  40  16  16  38 101  49  94  85  38
## 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
##  13  20  20  10  14  20  14  10  68   9  58  12  10  10  13  12  21  50  42  47  40  42  28  23  21
## 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229
##  40  44  48  41  22  25  33  31  20  36  25  40  25  41  33  30  31  31  22  29  19  30  30  45  59
## 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260
## 125 150 140 126 117 143 184 150 111 147 146 129 112 161 140 115 100  99 110  75  65  75 102 115 126
## 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291
```

```
##  91  57  85 106  90  72  78  58  61  26  39  39  42  28  57  39  26  28  19  17  15  25  17  30  18
## 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 320 322 323 324
##  10  10   8   8   6   6  13  28  11   3   3   1   1   4   4  30   4   1   4   2   1   1   1   2   2
## 400 407 410 414 416 417 422 424 473
##   2   1   1   1   2   1   1   1   1
```

```
table(mlen == nchar(ll))
```

```
##
## TRUE
## 8548
```

Now let's look at the subpatterns and where they start and end within each line and extract their values.

Looking at the first element of `m`, we see

```
m[[1]]
```

```
## [1] 1
## attr(,"match.length")
## [1] 236
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
## attr(,"capture.start")
##
## [1,] 1 18 21 50 91 95 100 128
## attr(,"capture.length")
##
## [1,] 14 1 26 39 3 3 25 108
## attr(,"capture.names")
## [1] "" "" "" "" "" "" "" ""
```

Note the attributes capture.start and capture.length. These give us the start of each capture group matched in the string and the number of characters matched for that capture group match.

We can use substring to get these, e.g.,

```
s = attr(m[[1]], "capture.start")
substring(ll[1], s, s + attr(m[[1]], "capture.length"))
```

```
## [1] "114.188.183.88 "
## [2] "- "
## [3] "01/Nov/2015:03:41:50 -0800]"
## [4] "GET /stat141/Code/Session1.txt HTTP/1.1\""
## [5] "404 "
## [6] "223 "
## [7] "https://www.google.co.jp/\""
## [8] "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 S
```

Where is the remote login field? We didn't capture that field as it is `-` in all fields.

We can loop over each pair of `ll` and `m` and extract the substrings for the capture groups:

```
caps = mapply(function(str, match) {
                s = attr(match, "capture.start")
                substring(str, s, s + attr(match, "capture.length"))
              }, ll, m)
```

This returns a matrix with dimensions 8 rows and 8548 columns. We can transpose it and convert it to a data.frame

```
caps2 = as.data.frame(t(caps))
```

We wrote a function getCaptures() to do this:

```
source("getCaptures.R")
getCaptures
```

```
## function (pat, x, matches = gregexpr(pat, x, perl = TRUE, ...),
##     asDataFrame = TRUE, row.names = x, dropNoName = TRUE, ...,
##     SIMPLIFY = asDataFrame)
## {
##     if (length(matches) == 0)
##         return(list())
##     if (is.null(attr(matches[[1]], "capture.start")))
##         stop("no capture group information. Does the regular expression contain capture groups?")
##     ans = mapply(getCapture, x, matches, MoreArgs = list(asDataFrame = FALSE),
##         SIMPLIFY = SIMPLIFY)
##     if (asDataFrame) {
##         ans = as.data.frame.matrix(t(ans), row.names = seq_along(x),
##             make.names = FALSE)
##         rownames(ans) = row.names
##         vars = attr(matches[[1]], "capture.names")
##         if (length(vars) && !all(vars == "")) {
##             names(ans) = vars
##             if (any(w <- (vars == "")))
##                 ans = ans[!w]
##         }
##         ans
##     }
##     else ans
## }
```

```
rx = '^([0-9.]+) - (-|[a-z0-9]+) \\[([^]]+)\\] "([^"]+)" ([0-9]+) (-|[0-9]+) "([^"]*)" "([^"]+)"'
caps = getCaptures(rx, ll, row.names = NULL)
```

We check the dimensions

```
dim(caps)
```

```
## [1] 8548    8
```

and that the number of rows is the same as the number of lines in our log file:

```
length(ll)
```

```
## [1] 8548
```

```
stopifnot(nrow(caps) == length(ll))
```

What are the column names?

```
names(caps)
```

```
## [1] "V1" "V2" "V3" "V4" "V5" "V6" "V7" "V8"
```

Let's change our regular expression to name each capture group

```
rx = '^(?P<ip>[0-9.]+) - (?P<who>-|[a-z0-9]+) \\[(?P<time>[^]]+)\\] "(?P<operation>[^"]+)" (?P<status>[
caps = getCaptures(rx, ll, row.names = NULL)
```

Now, getCaptures() puts these names on the columns of the resulting data.frame:

```
names(caps)
```

```
## [1] "ip"        "who"       "time"      "operation" "status"    "bytes"     "referer"   "useragent"
```

Now let's check the values. Are all the values in the bytes column numbers?

```
table(grepl("^[0-9]+$", caps$bytes ))
```

```
##
## FALSE  TRUE
##  2537  6011
```

That number should be familiar. It corresponds to the number of lines that have - as the value for bytes. But let's verify this:

```
w2 = grepl("^[0-9]+$", caps$bytes )
length(unique(caps$bytes[!w2]))
```

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

```
unique(caps$bytes[!w2])
```

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

So we can convert this column to integer values and this will convert the - values to NAs which is fine.

```
caps$bytes = as.integer(caps$bytes)
```

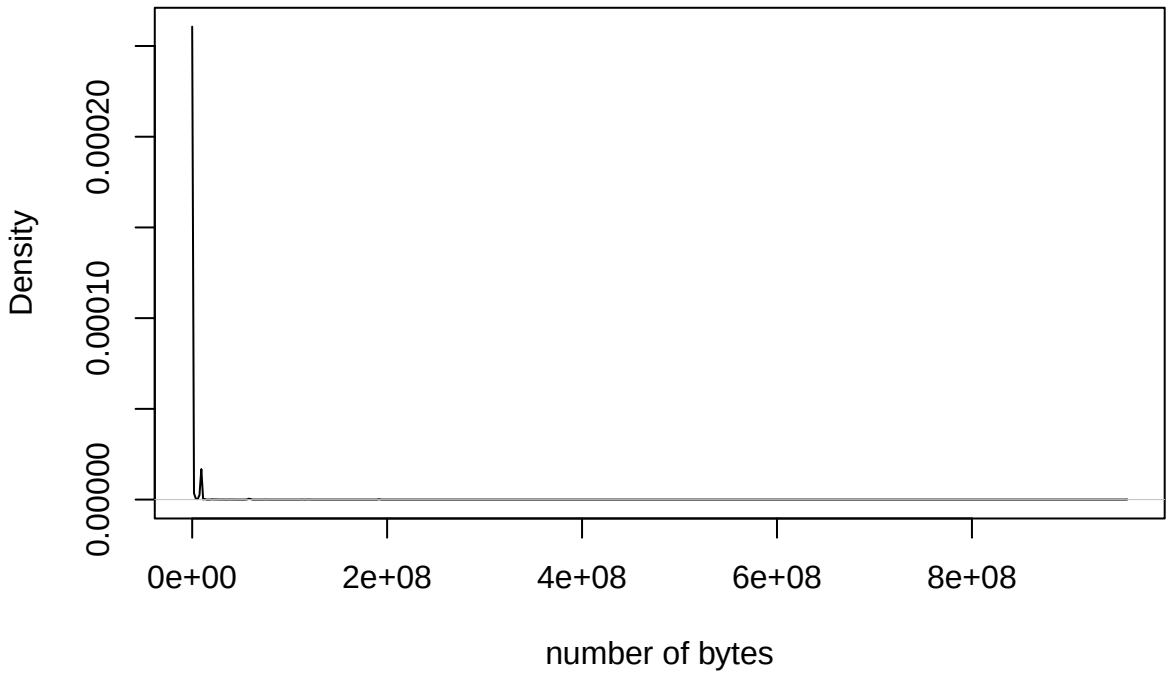
```
## Warning: NAs introduced by coercion
```

```
summary(caps$bytes)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
##         1       230      3719   1258359     12022 959380863      2537
```

The values look reasonable, but a) a density plot would be more informative, and b) the minimum of 1 is curious, and likewise the maximum value. So we'll explore these:

```
plot(density(caps$bytes[!is.na(caps$bytes)]), xlab = "number of bytes", main = "")
```



We don't see much as the range of the horizontal scale is so large due to very large maximum. Let's examine this

```
caps[which.max(caps$bytes),]
```

```
##                   ip who                          time                                                operation sta
## 2256 49.140.187.241   - 02/Nov/2015:00:13:27 -0800 GET /stat242/Data/trip_data_1.csv.zip HTTP/1.1
##                                                                                                  useragent
## 2256 Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
```

That is a large file, but legitimately large.

What about the minimum

```
caps[which.min(caps$bytes),]
```

```
##                ip who                         time                                      operation status by
## 1772 50.185.43.37   - 01/Nov/2015:20:09:44 -0800 GET /stat141/Hws/assignment3.html HTTP/1.1    206
##                                                                                        referer
## 1772 http://eeyore.ucdavis.edu/stat141/Homeworks.html
##
## 1772 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46
```

This doesn't look right - an assignment document with only 1 byte. What does the status value 206 mean? Looking at documentation, we see this is `206 Partial Content` and "is used when the Range header is sent from the client to request only part of a resource." So it too is legitimate.

Date-time stamp

Now let's look at the time column:

```
head(caps$time)
```

```
## [1] "01/Nov/2015:03:41:50 -0800" "01/Nov/2015:03:41:50 -0800" "01/Nov/2015:03:42:10 -0800" "01/Nov/20
## [5] "01/Nov/2015:03:42:13 -0800" "01/Nov/2015:03:42:13 -0800"
```

We can convert this to a POSIXct vector:

```
caps$timestamp = as.POSIXct(strptime(caps$time, "%d/%b/%Y:%H:%M:%S %T"))
table(is.na(caps$timestamp))
```

```
##
## TRUE
## 8548
```

It is always important to check if we introduced NAs during this type of conversion. Ours are all NAs. So we got the formatting string wrong. Let's remove the %T:

```
caps$timestamp = as.POSIXct(strptime(caps$time, "%d/%b/%Y:%H:%M:%S"))
table(is.na(caps$timestamp))
```

```
##
## FALSE
##  8548
```

Now we have not NAs, but let's look at the first few values along with the original `time` strings:

```
head(caps[, c("time", "timestamp")])
```

```
##                          time           timestamp
## 1 01/Nov/2015:03:41:50 -0800 2015-11-01 03:41:50
## 2 01/Nov/2015:03:41:50 -0800 2015-11-01 03:41:50
## 3 01/Nov/2015:03:42:10 -0800 2015-11-01 03:42:10
## 4 01/Nov/2015:03:42:12 -0800 2015-11-01 03:42:12
## 5 01/Nov/2015:03:42:13 -0800 2015-11-01 03:42:13
## 6 01/Nov/2015:03:42:13 -0800 2015-11-01 03:42:13
```

These look correct.

```
length(unique(caps$ip))
```

```
## [1] 707
```

```
table(caps$status)
```

```
##
##  200  206  301  302  304  403  404
## 4792   68   74   62 2291   12 1249
```

Separating the Operation, File and Protocol Version

```
head(caps$operation)
```

```
## [1] "GET /stat141/Code/Session1.txt HTTP/1.1" "GET /favicon.ico HTTP/1.1"
## [3] "GET /stat141/Code/ HTTP/1.1"             "GET /stat141/ HTTP/1.1"
## [5] "GET /stat141/Class.css HTTP/1.1"         "GET /stat141/OmegaTech.css HTTP/1.1"
```

Are all the protocol versions 1.1

```
w = grepl("HTTP/1.1$", caps$operation)
table(w)
```

```
## w
## FALSE  TRUE
##   334  8214
```

Some are 1.0, so we check

```
w = grepl("HTTP/1.[01]$", caps$operation)
table(w)
```

```
## w
## TRUE
## 8548
```

We now can use capture groups on these strings to extract the HTTP action/operation, the file path and the version. Again, we'll use named capture groups:

```
w = grepl("^(?P<action>[A-Z]+) (?P<file>.*) HTTP/1.(?P<version>[10])$", caps$operation, perl = TRUE)
table(w)
```

```
## w
## TRUE
## 8548
```

And now we can call getCaptures

```
ops = getCaptures("^(?P<action>[A-Z]+) (?P<file>.*) HTTP/1.(?P<version>[10])$", caps$operation, row.nam
dim(ops)
```

```
## [1] 8548    3
```

```
names(ops)
```

```
## [1] "action"  "file"    "version"
```

This looks fine, so we append this data.frame with the entire `caps` data.frame:

```
caps2 = cbind(caps, ops)
```

What are the Login Names

```
sort(table(caps2$who), decreasing = TRUE)
```

```
##
##        - yuecongw dchenucd    fangh   luming rayljazz  rickyt1 kamirira btenberg  kcolson ladyapus l
##     8387       16       13       13       13       13       13       12       11       11       11
##   ejxiao asterial  boyaliu  rskapul    xiliu
##        5        2        2        2        2
```

UserAgent, Web browser, Operating System

How any different user agents are there?

```
length(unique(caps2$useragent))
```

```
## [1] 238
```

We'll take a look at a few of these:

```
head(unique(caps2$useragent))
```

```
## [1] "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 S
## [2] "Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 S
## [3] "Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)"
## [4] "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)"
## [5] "Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)"
```

```
## [6] "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5) AppleWebKit/536.30.1 (KHTML, like Gecko) Version/
```

How many start with "Mozilla/5.0"?

```
table(grepl("^Mozilla/5.0", caps2$useragent))
```

```
##
## FALSE  TRUE
##   916  7632
```

Or with "Mozilla"?

```
w = grepl("^Mozilla", caps2$useragent)
table(w)
```

```
## w
## FALSE  TRUE
##   728  7820
```

What about the ones that don't start with "Mozilla"

```
head(caps2$useragent[!w])
```

```
## [1] "-"
## [2] "-"
## [3] "Safari/10601.2.7.2 CFNetwork/720.5.7 Darwin/14.5.0 (x86_64)"
## [4] "Safari/10601.2.7.2 CFNetwork/720.5.7 Darwin/14.5.0 (x86_64)"
## [5] "Safari/10601.2.7.2 CFNetwork/720.5.7 Darwin/14.5.0 (x86_64)"
## [6] "Safari/10601.2.7.2 CFNetwork/720.5.7 Darwin/14.5.0 (x86_64)"
```

So some blank ones ("-") and others starting with Safari, the Mac browser.

So let's look at the ones that rest, i.e., that don't start with Safari or Mozilla:

```
w = grepl("^(Safari|Mozilla)", caps2$useragent)
table(w)
```

```
## w
## FALSE  TRUE
##   538  8010
```

```
head(caps2$useragent[!w])
```

```
## [1] "-"                                                    "-"
## [3] "R (3.1.3 x86_64-apple-darwin10.8.0 x86_64 darwin10.8.0)" "R (3.1.3 x86_64-apple-darwin10.8.0 x8
## [5] "R (3.1.2 x86_64-w64-mingw32 x86_64 mingw32)"          "Google favicon"
```

So we have "R", "Google favicon"

```
tt = sort(table(caps2$useragent[!w]), decreasing = TRUE)
tt[ tt > 5]
```

```
##
##                         R (3.2.2 x86_64-apple-darwin13.4.0 x86_64 darwin13.4.0)
##                                                                                77
##                                                                    Google favicon
##                                                                                53
##                                  R (3.0.2 x86_64-pc-linux-gnu x86_64 linux-gnu)
##                                                                                51
##                         R (3.0.3 x86_64-apple-darwin10.8.0 x86_64 darwin10.8.0)
##                                                                                41
##                         R (3.1.2 x86_64-apple-darwin13.4.0 x86_64 darwin13.4.0)
```

```
##                                                                          39
##                         R (3.1.1 x86_64-apple-darwin10.8.0 x86_64 darwin10.8.0)
##                                                                          36
##                              MobileSafari/601.1 CFNetwork/758.1.6 Darwin/15.0.0
##                                                                          34
##                                    R (3.2.2 x86_64-w64-mingw32 x86_64 mingw32)
##                                                                          33
##                                                                           -
##                                                                          19
##                             MobileSafari/600.1.4 CFNetwork/711.5.6 Darwin/14.0.0
##                                                                          16
##  com.apple.WebKit.WebContent/10600.8.9 CFNetwork/720.5.7 Darwin/14.5.0 (x86_64)
##                                                                          14
##                                                                      Nessus
##                                                                          14
##                                    R (3.1.2 x86_64-w64-mingw32 x86_64 mingw32)
##                                                                          10
##                         R (3.1.3 x86_64-apple-darwin10.8.0 x86_64 darwin10.8.0)
##                                                                           9
##                         R (3.2.1 x86_64-apple-darwin13.4.0 x86_64 darwin13.4.0)
##                                                                           9
## com.apple.WebKit.WebContent/10600.6.3 CFNetwork/720.3.13 Darwin/14.3.0 (x86_64)
##                                                                           8
##       facebookexternalhit/1.1 (+http://www.facebook.com/externalhit_uatext.php)
##                                                                           7
##                                         R (3.1.3 i386-w64-mingw32 i386 mingw32)
##                                                                           7
## com.apple.WebKit.WebContent/10600.7.12 CFNetwork/720.4.4 Darwin/14.4.0 (x86_64)
##                                                                           6
##                         R (3.2.1 x86_64-apple-darwin10.8.0 x86_64 darwin10.8.0)
##                                                                           6
```

How many start with R

```
table(grepl("^R \\(", caps2$useragent))
```

```
##
## FALSE  TRUE
##  8214   334
```

Let's look at the requests/lines where the user agent is "Google favicon"

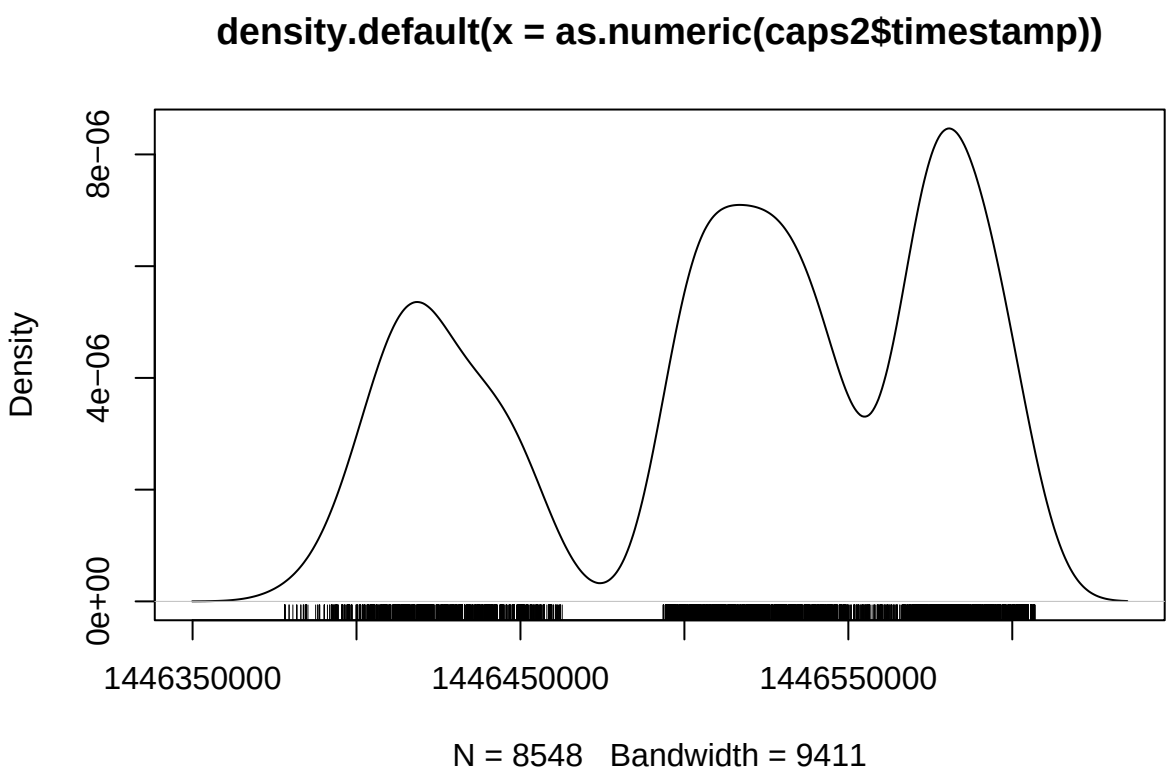
```
idx = grep("Google favicon", ll)
head(ll[idx])
```

```
## [1] "66.249.84.180 - - [01/Nov/2015:08:47:57 -0800] \"GET / HTTP/1.1\" 200 - \"-\" \"Google favicon\
## [2] "66.249.84.182 - - [01/Nov/2015:08:47:57 -0800] \"GET /favicon.ico HTTP/1.1\" 404 209 \"-\" \"Goo
## [3] "66.249.84.180 - - [01/Nov/2015:10:42:55 -0800] \"GET /stat141/ HTTP/1.1\" 200 4176 \"-\" \"Googl
## [4] "66.249.84.180 - - [01/Nov/2015:10:42:55 -0800] \"GET /favicon.ico HTTP/1.1\" 404 209 \"-\" \"Goo
## [5] "66.249.84.181 - - [01/Nov/2015:11:48:07 -0800] \"GET /favicon.ico HTTP/1.1\" 404 209 \"-\" \"Goo
## [6] "66.249.84.181 - - [01/Nov/2015:14:11:44 -0800] \"GET / HTTP/1.1\" 200 - \"-\" \"Google favicon\
```

We can extract the operating system (OS) from the useragent field and see which OSes are most common.

When did the Request Occur?

```
plot(density(as.numeric(caps2$timestamp)))
rug(as.numeric(caps2$timestamp))
```



HTTP Operations

What are the HTTP operations

```
table(caps2$action)
```

```
##
##    GET HBESPY   HEAD   POST WFZWXO
##   8457      1     84      5      1
```

HTTP Version and Useragent

```
table(caps2$version)
```

```
##
##    0    1
##  334 8214
```

```
table(caps$useragent, caps2$version)
```

Referer Pages/URLs

```
length(unique(caps2$referer))
```

```
## [1] 97
```

```
dsort(table(caps2$referer))
```

Spaces in the File Paths

Do any of the file paths contain spaces?

```
grep(" ", caps2$file)
```

```
## integer(0)
```