Week 4

Contents

1 Week 4

1.1 Editing Text Variables

1.1.1 Fixing character vectors - tolower(), toupper()

```
if(!file.exists("./data")){dir.create("./data")}
```

```
cam <- read.csv("data/cameras.csv")
head(cam)
```

address	direction	street	crossStreet	intersection	Location.1
S CATON AVE & BENSON AVE	N/B	Caton Ave	Benson Ave	Caton Ave & Benson Ave	(39.2693779962, -76.6688185297)
S CATON AVE & BENSON AVE	S/B	Caton Ave	Benson Ave	Caton Ave & Benson Ave	(39.2693157898, -76.6689698176)
WILKENS AVE & PINE HEIGHTS AVE	E/B	Wilkens Ave	Pine Heights	Wilkens Ave & Pine Heights	(39.2720252302, -76.676960806)
THE ALAMEDA & E 33RD ST	S/B	The Alameda	33rd St	The Alameda & 33rd St	(39.3285013141, -76.5953545714)
E 33RD ST & THE ALAMEDA	E/B	E 33rd	The Alameda	E 33rd & The Alameda	(39.3283410623, -76.5953594625)
ERDMAN AVE & N MACON ST	E/B	Erdman	Macon St	Erdman & Macon St	(39.3068045671, -76.5593167803)

```
str(cam)
```

```
## 'data.frame':    80 obs. of  6 variables:
##  $ address     : chr  "S CATON AVE & BENSON AVE" "S CATON AVE & BENSON AVE" "WILKENS AVE & PINE HEIG
##  $ direction   : chr  "N/B" "S/B" "E/B" "S/B" ...
##  $ street      : chr  "Caton Ave" "Caton Ave" "Wilkens Ave" "The Alameda" ...
##  $ crossStreet : chr  "Benson Ave" "Benson Ave" "Pine Heights" "33rd St" ...
##  $ intersection: chr  "Caton Ave & Benson Ave" "Caton Ave & Benson Ave" "Wilkens Ave & Pine Heights"
##  $ Location.1  : chr  "(39.2693779962, -76.6688185297)" "(39.2693157898, -76.6689698176)" "(39.27202
```

```
names(cam)
```

```
## [1] "address"      "direction"    "street"       "crossStreet"  "intersection"
## [6] "Location.1"
```

```
tolower(names(cam))
```

```
## [1] "address"      "direction"    "street"       "crossstreet"  "intersection"
## [6] "location.1"
```

1.1.2 Fixing character vectors - strsplit()

- Good for automatically splitting variable names
- Important parameters: *x*, *split*

```r
splitnames <- strsplit(names(cam),"\\.")
splitnames
```

```
## [[1]]
## [1] "address"
##
## [[2]]
## [1] "direction"
##
## [[3]]
## [1] "street"
##
## [[4]]
## [1] "crossStreet"
##
## [[5]]
## [1] "intersection"
##
## [[6]]
## [1] "Location" "1"
```

In R, strsplit is a function that splits the elements of a character vector (names(cam) in this case) into substrings according to the matches to substring separator ("\." in this case).

"\." is a regular expression that matches a period (.). The double backslash (\) is necessary because the backslash itself is an escape character in R strings, so to represent a literal backslash you need to use two backslashes. A period is a special character in regular expressions that matches any character, so to represent a literal period in a regular expression you need to precede it with a backslash.

So in the last one, the name "location.1" become "location" and "1"

1.1.3 Quick aside - lists

```r
myList <-
  list(letters = c("A", "b", "c"),
       numbers = 1:3,
       matrix(1:25, ncol = 5))
head(myList)
```

```
## $letters
## [1] "A" "b" "c"
##
## $numbers
## [1] 1 2 3
##
## [[3]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

```r
myList[1]
```

```
## $letters
## [1] "A" "b" "c"
```

```
myList$letters
```

```
## [1] "A" "b" "c"
```

```
myList[[1]]
```

```
## [1] "A" "b" "c"
```

1.1.4 Fixing character vectors - sapply()

- Applies a function to each element in a vector or list
- Important parameters: *X*,*FUN*

```
splitnames[6]
```

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

```
splitnames[[6]][1]
```

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

```
firstElement <- function(x){x[1]}

sapply(splitnames, firstElement)
```

```
## [1] "address"      "direction"    "street"       "crossStreet"  "intersection"
## [6] "Location"
```

1.1.5 Peer review experiment data

```
reviews <- read.csv("./data/reviews.csv"); solutions <- read.csv("./data/solutions.csv")
head(reviews,2)
```

id	solution_id	reviewer_id	start	stop	time_left	accept
1	3	27	1304095698	1304095758	1754	1
2	4	22	1304095188	1304095206	2306	1

```
head(solutions)
```

id	problem_id	subject_id	start	stop	time_left	answer
1	156	29	1304095119	1304095169	2343	B
2	269	25	1304095119	1304095183	2329	C
3	34	22	1304095127	1304095146	2366	C
4	19	23	1304095127	1304095150	2362	D
5	605	26	1304095127	1304095167	2345	A
6	384	27	1304095131	1304095270	2242	C

```
names(reviews)
```

```
## [1] "id"          "solution_id" "reviewer_id" "start"       "stop"
## [6] "time_left"   "accept"
```

```r
names(solutions)
```

```
## [1] "id"         "problem_id" "subject_id" "start"      "stop"
## [6] "time_left"  "answer"
```

```r
str(reviews)
```

```
## 'data.frame':    199 obs. of  7 variables:
##  $ id         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ solution_id: int  3 4 5 1 10 2 9 8 7 11 ...
##  $ reviewer_id: int  27 22 28 26 29 29 25 23 29 26 ...
##  $ start      : int  1304095698 1304095188 1304095276 1304095267 1304095456 1304095471 1304095343 NA
##  $ stop       : int  1304095758 1304095206 1304095320 1304095423 1304095469 1304095513 1304095382 NA
##  $ time_left  : int  1754 2306 2192 2089 2043 1999 2130 NA 1899 2024 ...
##  $ accept     : int  1 1 1 1 1 1 1 NA 1 1 ...
```

```r
str(solutions)
```

```
## 'data.frame':    205 obs. of  7 variables:
##  $ id        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ problem_id: int  156 269 34 19 605 384 538 312 327 194 ...
##  $ subject_id: int  29 25 22 23 26 27 28 24 22 23 ...
##  $ start     : int  1304095119 1304095119 1304095127 1304095127 1304095127 1304095131 1304095133 1304
##  $ stop      : int  1304095169 1304095183 1304095146 1304095150 1304095167 1304095270 1304095201 1304
##  $ time_left : int  2343 2329 2366 2362 2345 2242 2311 2314 2328 2337 ...
##  $ answer    : chr  "B" "C" "C" "D" ...
```

1.1.6 Fixing character vectors - sub()

Important parameters: *pattern*, *replacement*, *x*

```r
sub("_","",names(reviews))
```

```
## [1] "id"         "solutionid" "reviewerid" "start"      "stop"
## [6] "timeleft"   "accept"
```

The sub function in R is used to replace the first match of a specific pattern in a string. In this case, sub(“*”,” ”,names(reviews)) is used to remove the first underscore (*””) from the names of the reviews data frame.

1.1.7 Fixing character vectors - gsub()

```r
testName <- "this_is_a_test"
sub("_","",testName)
```

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

```r
gsub("_","",testName)
```

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

1.1.8 Finding values - grep(),grepl()

grep: This function returns the indices of the elements in the input vector which match the pattern. For example, grep(“a”, c(“apple”, “banana”, “cherry”)) will return 1 2 because “apple” (the 1st element) and “banana” (the 2nd element) both contain the letter “a”.

grepl: This function returns a logical vector (with TRUE or FALSE values) of the same length as the input, indicating whether each element matches the pattern.

```r
grep("Alameda", cam$intersection)
```

```
## [1]  4  5 36
```

```r
table(grepl("Alameda",cam$intersection))
```

```
##
## FALSE  TRUE
##    77     3
```

```r
cam2 <- cam[!grepl("Alameda",cam$intersection),]
```

1.1.9 More on grep()

```r
grep("Alameda", cam$intersection, value = TRUE)
```

```
## [1] "The Alameda  & 33rd St"   "E 33rd  & The Alameda"
## [3] "Harford \n & The Alameda"
```

```r
grep("JeffStreet",cam$intersection)
```

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

The grep function in R searches for specific patterns in a vector or a factor and returns the indices where the pattern is found. If no match is found, it returns integer(0).

```r
length(grep("JeffStreet",cam$intersection))
```

```
## [1] 0
```

1.1.10 Important points about text in data sets

- Names of variables should be
  - All lower case when possible
  - Descriptive (Diagnosis versus Dx)
  - Not duplicated
  - Not have underscores or dots or white spaces
- Variables with character values
  - Should usually be made into factor variables (depends on application)
  - Should be descriptive (use TRUE/FALSE instead of 0/1 and Male/Female versus 0/1 or M/F)

1.2 Regular Expressions

- Regular expressions can be thought of as a combination of literals and *metacharacters*
- To draw an analogy with natural language, think of literal text forming the words of this language, and the metacharacters defining its grammar
- Regular expressions have a rich set of metacharacters

1.2.1 Literals

Simplest pattern consists only of literals. The literal “nuclear” would match to the following lines:

```
Ooh. I just learned that to keep myself alive after a
nuclear blast! All I have to do is milk some rats
then drink the milk. Aweosme. :}

Laozi says nuclear weapons are mas macho
```

```
Chaos in a country that has nuclear weapons -- not good.

my nephew is trying to teach me nuclear physics, or
possibly just trying to show me how smart he is
so I'll be proud of him [which I am].

lol if you ever say "nuclear" people immediately think
DEATH by radiation LOL
```

1.2.2 Regular Expressions

- Simplest pattern consists only of literals; a match occurs if the sequence of literals occurs anywhere in the text being tested
- What if we only want the word “Obama”? or sentences that end in the word “Clinton”, or “clinton” or “clinto”? We need a way to express
- whitespace word boundaries
- sets of literals
- the beginning and end of a line
- alternatives (“war” or “peace”) Metacharacters to the rescue!

1.2.3 Metacharacters `^`, `$`

Some metacharacters represent the start of a line

`` `^``i think ``

will match the lines

```
i think we all rule for participating
i think i have been outed
i think this will be quite fun actually
i think i need to go to work
i think i first saw zombo in 1999.
```

`$` represents the end of a line

`morning$`

will match the lines

```
well they had something this morning
then had to catch a tram home in the morning
dog obedience school in the morning
and yes happy birthday i forgot to say it earlier this morning
I walked in the rain this morning
good morning
```

1.2.4 Character Classes with `[]`

We can list a set of characters we will accept at a given point in the match

`[Bb][Uu][Ss][Hh]`

will match the lines

```
The democrats are playing, "Name the worst thing about Bush!"
I smelled the desert creosote bush, brownies, BBQ chicken
```

```
BBQ and bushwalking at Molonglo Gorge
Bush TOLD you that North Korea is part of the Axis of Evil
I'm listening to Bush - Hurricane (Album Version)
```

```
^[Ii] am
```

will match

```
i am so angry at my boyfriend i can't even bear to
look at him

i am boycotting the apple store

I am twittering from iPhone

I am a very vengeful person when you ruin my sweetheart.

I am so over this. I need food. Mmmm bacon...
```

Similarly, you can specify a range of letters [a-z] or [a-zA-Z]; notice that the order doesn't matter

```
^[0-9][a-zA-Z]
```

will match the lines:

```
7th inning stretch
2nd half soon to begin. OSU did just win something
3am - cant sleep - too hot still.. :(
5ft 7 sent from heaven
1st sign of starvagtion
```

When used at the beginning of a character class, the "^" is also a metacharacter and indicates matching characters NOT in the indicated class

```
[^?.]$
```

will match the lines

```
i like basketballs
6 and 9
dont worry... we all die anyway!
Not in Baghdad
helicopter under water? hmmm
```

1.2.5 More Metacharacters .

"." is used to refer to any character. So

```
9.11
```

will match the lines

```
its stupid the post 9-11 rules
if any 1 of us did 9/11 we would have been caught in days.
NetBios: scanning ip 203.169.114.66
Front Door 9:11:46 AM
Sings: 0118999881999119725...3 !
```

1.2.6 More Metacharacters: |

This does not mean "pipe" in the context of regular expressions; instead it translates to "or"; we can use it to combine two expressions, the subexpressions being called alternatives

```
flood|fire
```

will match the lines

```
is firewire like usb on none macs?
the global flood makes sense within the context of the bible
yeah ive had the fire on tonight
... and the floods, hurricanes, killer heatwaves, rednecks, gun nuts, etc.
```

We can include any number of alternatives...

```
flood|earthquake|hurricane|coldfire
```

will match the lines

```
Not a whole lot of hurricanes in the Arctic.
We do have earthquakes nearly every day somewhere in our State
hurricanes swirl in the other direction
coldfire is STRAIGHT!
'cause we keep getting earthquakes
```

The alternatives can be real expressions and not just literals

```
^[Gg]ood|[Bb]ad
```

will match the lines

```
good to hear some good knews from someone here
Good afternoon fellow american infidels!
good on you-what do you drive?
Katie... guess they had bad experiences...
my middle name is trouble, Miss Bad News
```

1.2.7 More Metacharacters: (and)

Subexpressions are often contained in parentheses to constrain the alternatives

```
^([Gg]ood|[Bb]ad)
```

will match the lines

```
bad habbit
bad coordination today
good, becuase there is nothing worse than a man in kinky underwear
Badcop, its because people want to use drugs
Good Monday Holiday
Good riddance to Limey
```

1.2.8 More Metacharacters: ?

The question mark indicates that the indicated expression is optional

```
[Gg]eorge( [Ww]\.)? [Bb]ush
```

will match the lines

```
i bet i can spell better than you and george bush combined
BBC reported that President George W. Bush claimed God told him to invade I
a bird in the hand is worth two george bushes
```

1.2.9 One thing to note...

In the following

```
[Gg]eorge( [Ww]\.)? [Bb]ush
```

we wanted to match a "." as a literal period; to do that, we had to "escape" the metacharacter, preceding it with a backslash In general, we have to do this for any metacharacter we want to include in our match

1.2.10 More metacharacters: * and +

The * and + signs are metacharacters used to indicate repetition; * means "any number, including none, of the item" and + means "at least one of the item"

```
(.*)
```

will match the lines

```
anyone wanna chat? (24, m, germany)
hello, 20.m here... ( east area + drives + webcam )
(he means older men)
()
```

```
[0-9]+ (.*)[0-9]+
```

will match the lines

```
working as MP here 720 MP battallion, 42nd birgade
so say 2 or 3 years at colleage and 4 at uni makes us 23 when and if we fin
it went down on several occasions for like, 3 or 4 *days*
Mmmm its time 4 me 2 go 2 bed
```

1.2.11 More metacharacters: { and }

{ and } are referred to as interval quantifiers; the let us specify the minimum and maximum number of matches of an expression

```
[Bb]ush( +[^ ]+ +){1,5} debate
```

will match the lines

```
Bush has historically won all major debates he's done.
in my view, Bush doesn't need these debates..
bush doesn't need the debates? maybe you are right
That's what Bush supporters are doing about the debate.
Felix, I don't disagree that Bush was poorly prepared for the debate.
indeed, but still, Bush should have taken the debate more seriously.
Keep repeating that Bush smirked and scowled during the debate
```

1.2.12 More metacharacters: and

- m,n means at least m but not more than n matches
- m means exactly m matches
- m, means at least m matches

1.2.13 More metacharacters: (and) revisited

- In most implementations of regular expressions, the parentheses not only limit the scope of alternatives divided by a "|", but also can be used to "remember" text matched by the subexpression enclosed
- We refer to the matched text with \1, \2, etc.

1.2.14 More metacharacters: (and) revisited

So the expression

```
+([a-zA-Z]+) +\1 +
```

will match the lines

```
time for bed, night night twitter!
blah blah blah blah
my tattoo is so so itchy today
i was standing all all alone against the world outside...
hi anybody anybody at home
estudiando css css css css.... que desastritooooo
```

1.2.15 More metacharacters: (and) revisited

The * is "greedy" so it always matches the *longest* possible string that satisfies the regular expression. So

```
^s(.*)s
```

matches

```
sitting at starbucks
setting up mysql and rails
studying stuff for the exams
spaghetti with marshmallows
stop fighting with crackers
sore shoulders, stupid ergonomics
```

1.2.16 More metacharacters: (and) revisited

The greediness of * can be turned off with the ?, as in

```
^s(.*?)s$
```

1.2.17 Summary

- Regular expressions are used in many different languages; not unique to R.
- Regular expressions are composed of literals and metacharacters that represent sets or classes of characters/words
- Text processing via regular expressions is a very powerful way to extract data from "unfriendly" sources (not all data comes as a CSV file)
- Used with the functions `grep`,`grepl`,`sub`,`gsub` and others that involve searching for text strings (Thanks to Mark Hansen for some material in this lecture.)

1.3 Working with Dates

1.3.1 Starting simple

```
d1 <- date()
d1
```

```
## [1] "Tue Jun 27 22:47:50 2023"
```

```
class(d1)
```

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

1.3.2 Date class

```r
d2 <- Sys.Date()
d2
```

```
## [1] "2023-06-27"
```

```r
class(d2)
```

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

1.3.3 Formatting dates

`%d` = day as number (0-31), `%a` = abbreviated weekday,`%A` = unabbreviated weekday, `%m` = month (00-12), `%b` = abbreviated month, `%B` = unabbrevidated month, `%y` = 2 digit year, `%Y` = four digit year

```r
format(d2, "%a %b %d")
```

```
## [1] "Tue Jun 27"
```

1.3.4 Creating dates

```r
x = c("1jan1960", "2jan1960", "31mar1960", "30jul1960"); z = as.Date(x, "%d%b%Y")
z
```

```
## [1] "1960-01-01" "1960-01-02" "1960-03-31" "1960-07-30"
```

```r
z[1] - z[2]
```

```
## Time difference of -1 days
```

```r
as.numeric(z[1]-z[2])
```

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

1.3.5 Converting to Julian

```r
weekdays(d2)
```

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

```r
months(d2)
```

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

```r
julian(d2)
```

```
## [1] 19535
## attr(,"origin")
## [1] "1970-01-01"
```

1.3.6 Lubridate

```r
library(lubridate); ymd("20140108")
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
## [1] "2014-01-08"
```

```
mdy("08/04/2013")
```

```
## [1] "2013-08-04"
```

```
dmy("03-04-2013")
```

```
## [1] "2013-04-03"
```

1.3.7 Dealing with times

```
ymd_hms("2011-08-03 10:15:03")
```

```
## [1] "2011-08-03 10:15:03 UTC"
```

```
ymd_hms("2011-08-03 10:15:03",tz="Pacific/Auckland")
```

```
## [1] "2011-08-03 10:15:03 NZST"
```

```
?Sys.timezone
```

1.3.8 Some functions have slightly different syntax

```
x = dmy(c("1jan2013", "2jan2013", "31mar2013", "30jul2013"))
wday(x[1])
```

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

```
wday(x[1],label=TRUE)
```

```
## [1] Tue
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

1.3.9 Notes and further resources

- More information in this nice lubridate tutorial http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/
- The lubridate vignette is the same content http://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html
- Ultimately you want your dates and times as class "Date" or the classes "POSIXct", "POSIXlt". For more information type `?POSIXlt`

1.4 Data Resources

1.4.1 Open Government Sites

- United Nations http://data.un.org/
- U.S. http://www.data.gov/
  - List of cities/states with open data
- United Kingdom http://data.gov.uk/
- France http://www.data.gouv.fr/
- Ghana http://data.gov.gh/
- Australia http://data.gov.au/
- Germany https://www.govdata.de/
- Hong Kong http://www.gov.hk/en/theme/psi/datasets/
- Japan http://www.data.go.jp/
- Many more http://www.data.gov/opendatasites

1.4.2 Gapminder

http://www.gapminder.org/

1.4.3 Survey data from the United States

http://www.asdfree.com/

1.4.4 Infochimps Marketplace

http://www.infochimps.com/marketplace

1.4.5 Kaggle

http://www.kaggle.com/

1.4.6 Collections by data scientists

- Hilary Mason http://bitly.com/bundles/hmason/1
- Peter Skomoroch https://delicious.com/pskomoroch/dataset
- Jeff Hammerbacher http://www.quora.com/Jeff-Hammerbacher/Introduction-to-Data-Science-Data-Sets
- Gregory Piatetsky-Shapiro http://www.kdnuggets.com/gps.html
- http://blog.mortardata.com/post/67652898761/6-dataset-lists-curated-by-data-scientists

---

1.4.7 More specialized collections

- Stanford Large Network Data
- UCI Machine Learning
- KDD Nugets Datasets
- CMU Statlib
- Gene expression omnibus
- ArXiv Data
- Public Data Sets on Amazon Web Services

1.4.8 Some API's with R interfaces

- twitter and twitteR package
- figshare and rfigshare
- PLoS and rplos
- rOpenSci
- Facebook and RFacebook
- Google maps and RGoogleMaps