Getting and Cleaning Data - Week 4

Carmelo Ramirez

07/11/2020

EDITING TEXT VARIABLES

tolower()

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

fileUrl <-  "https://data.baltimorecity.goc/views/dz54-2aru/rows.csv?accessType=DOWNLOAD"

download.file(fileUrl, destfile = "./data/cameras.csv", method = "curl")

cameraData <- read.csv("./data/cameras.csv")

names(cameraData)

## All column names to lower case
tolower(names(cameraData))
```

Fixing character vectors - *strsplit()*

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

```
## Split strings by '.'
splitNames = strsplit(names(cameraData), "\\.")
splitNames[[5]]
```

Fixing character vector - *sub()*

- *sub()* just replace the first instance of the value looked for

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

Fixing character vectors - *gsub()*

- Replace all value looked for in string

```
gsub("_", "", string)
```

Finding values - *gerep()*, *grepl()*

- *grep()* finds all indices where the item equals the value selected.
- *grepl()* returns a vector of TRUE and FALSE, according to the criteria selected

```
## Returns indices where value appears
grep("Alameda", cameraData$intersection)

## grep returns the values (not indices) where the value appear
grep("Alameda", cameraData$intersection, value = TRUE)

table(grepl("Alameda", cameraData$intersection))
```

More useful string functions

- *nchar()* returns number of characters in string

```
library(stringr)
nchar("Jeffrey Leek")
```

- *substr()*: returns a substring of a string, from a start to and end index

```
substr("Jeffery Leek", 1, 7)
```

- *paste()*: Combines two strings, separated by space (default) or by a character selected by *sep* parameter.

```
paste("Jeffery", "Leek")
```

- *paste0()*: Combines two string, without separation

```
paste0("Jeffery", "Leek")
```

- *str_trim()*: trim outside spaces in string.

```
str_trim("Jeff     ")
```

Important points about text in data sets

- Names of variables should be
  - All lower cases 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 insted of 0/1 and Male/Female versus 0/1 or M/F)

REGULAR EXPRESSIONS

- Regular Expressions can be thought of as a combination of literals and metacharacters
- Regular expressions have a rich set of metacharacters
- Simplest pattern consist only of literals; a match occurs if the sequence of literals occurs anywhere in the text being tested.

Metacharacters

- ^: represents the start of a line. For example, ^i think will match:
  - **i think** we all rule for participating
  - **i think** i have been outed
- *$*: represents the end of a line. For example, morning$ will match:
  - well they had something this **morning**
  - then had to catch a tram home in the **morning**
- *[ ]*: represents a list of characters that will be accepted at a given point in the match. For example, [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
- .: is used to refer to any character. For example: 9.11 will match:
  - its stupid the post **9-11** rules
  - Front Door **9:11**:46 AM

-/: It translate to "or", we can use it to combine two expressions, the subexpression being called alternatives. We can include any number of alternatives. For example flood|fire will match: - is **firewire** like usb on none macs? - the global **flood** makes sense within the context of the bible

- *?*: The question mark indicates that the indicated expression is optional. For example, [Gg]eorge([Ww].)? [Bb]ush will match:
  - I bet I can spell better than you and **george bush** combined

-* 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". For example, _(.*)_ will match: - anyone wanna chat? (24, m, germany) - () - *{}*: are referred to as interval quantifiers; they let us specify the minimum and maximum number of matches of an expression. For example, _[Bb]ush(+[^]+ +){1,5} debate

WORKING WITH DATES

date() function

```
d1 = date()

d1
```

```
## [1] "Mon Nov 09 09:30:02 2020"
```

```
class(d1)
```

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

###Sys.Date() function

```
d2 = Sys.Date()

d2
```

```
## [1] "2020-11-09"
```

```
class(d2)
```

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

Formatting dates

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

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

```
## [1] "Mon Nov 09"
```

Creating Dates

```
x = c("1jan1960", "2jan1960", "31mar1960")

z = as.Date(x, "%d%b%Y")

z
```

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

Lubridate Library

```
library(lubridate)
```

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

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
ymd("20140108")
```

```
## [1] "2014-01-08"
```

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

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

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

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

Dealing with times using Lubridate package:

```
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"
```