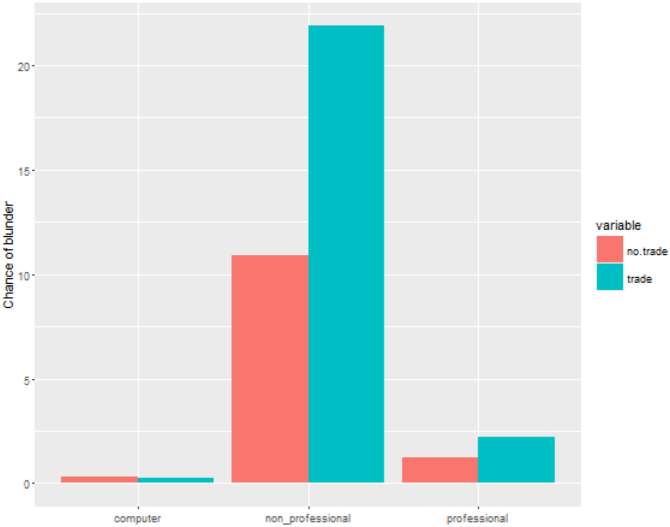
Character data and regular expressions

Yufeng Huang

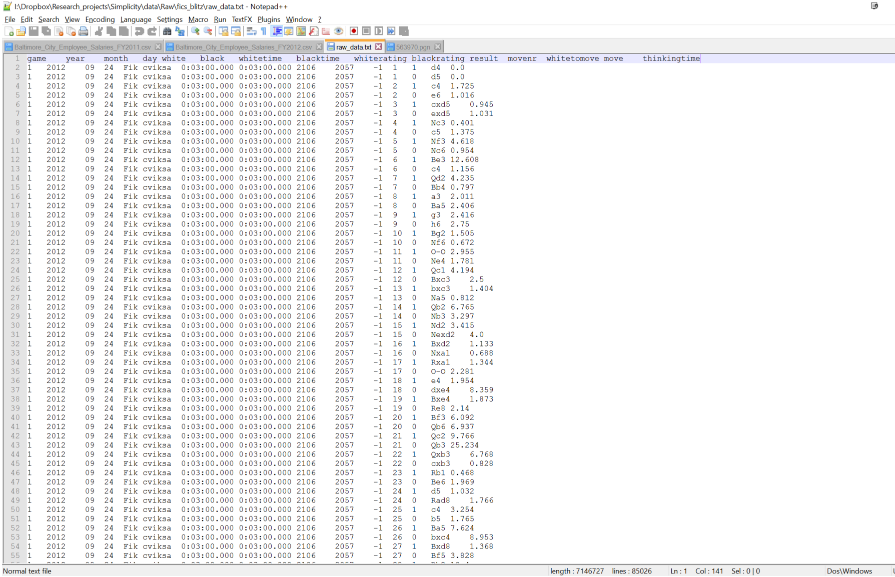
Associate Professor of Marketing, Simon Business School

August 8, 2022

My spare-time project: human players risk mistakes to achieve simplicity



Chess data: processed



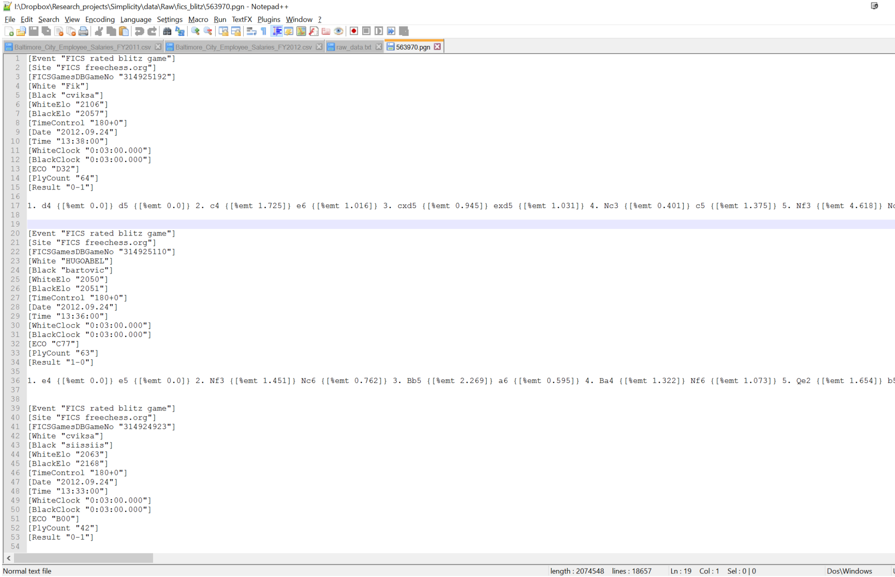
I:\Dropbox\Research_projects\Simplicity\data\Raw\fics_blitz\raw_data.txt - Notepad++

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
Baltimore_City_Employee_Salaries_FY2011.csv
Baltimore_City_Employee_Salaries_FY2012.csv
raw_data.txt
563970.pgn

```
game    year    month   day white   black   whitetime   blacktime   whiterating blackrating result  movenr  whitetomove move    thinkingtime
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  1   1   d4  0.0
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  1   0   d5  0.0
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  2   1   c4  1.725
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  2   0   e6  1.016
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  3   1   cxd5    0.945
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  3   0   exd5    1.031
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  4   1   Nc3 0.401
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  4   0   c5  1.375
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  5   1   Nf3 4.618
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  5   0   Nc6 0.954
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  6   1   Be3 12.608
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  6   0   c4  1.156
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  7   1   Qd2 4.235
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  7   0   Bb4 0.797
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  8   1   a3  2.011
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  8   0   Ba5 2.406
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  9   1   g3  2.416
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  9   0   h6  2.75
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  10  1   Bg2 1.505
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  10  0   Nf6 0.672
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  11  1   O-O 2.955
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  11  0   Ne4 1.781
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  12  1   Qc1 4.194
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  12  0   Bxc3    2.5
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  13  1   bxc3    1.404
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  13  0   Na5 0.812
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  14  1   Qb2 6.765
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  14  0   Nb3 3.297
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  15  1   Nd2 3.415
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  15  0   Nexd2   4.0
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  16  1   Bxd2    1.133
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  16  0   Nxa1    0.688
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  17  1   Rxa1    1.344
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  17  0   O-O 2.281
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  18  1   e4  1.954
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  18  0   dxe4    8.359
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  19  1   Bxe4    1.873
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  19  0   Re8 2.14
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  20  1   Bf3 6.092
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  20  0   Qb6 6.937
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  21  1   Qc2 9.766
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  21  0   Qb3 25.234
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  22  1   Qxb3    6.768
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  22  0   cxb3    0.828
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  23  1   Rb1 0.468
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  23  0   Be6 1.969
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  24  1   d5  1.032
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  24  0   Rad8    1.766
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  25  1   c4  3.254
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  25  0   b5  1.765
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  26  1   Ba5 7.624
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  26  0   bxc4    8.953
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  27  1   Bxd8    1.368
1   2012    09  24  Fik cviksa  0:03:00.000 0:03:00.000 2106    2057    -1  27  0   Bf5 3.828
```

Normal text file   length : 7146727   lines : 85026   Ln : 1   Col : 141   Sel : 0 | 0   Dos\Windows

Chess data: raw


I:\Dropbox\Research_projects\Simplicity\data\Raw\fics_blitz\563970.pgn - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
Baltimore_City_Employee_Salaries_FY2011.csv
Baltimore_City_Employee_Salaries_FY2012.csv
raw_data.txt
563970.pgn
Normal text file
length : 2074548 lines : 18657
Ln : 19 Col : 1 Sel : 0 | 0
Dos\Windows

```
[Event "FICS rated blitz game"]
[Site "FICS freechess.org"]
[FICSGamesDBGameNo "314925192"]
[White "Fik"]
[Black "cviksa"]
[WhiteElo "2106"]
[BlackElo "2057"]
[TimeControl "180+0"]
[Date "2012.09.24"]
[Time "13:38:00"]
[WhiteClock "0:03:00.000"]
[BlackClock "0:03:00.000"]
[ECO "D32"]
[PlyCount "64"]
[Result "0-1"]

1. d4 {[%emt 0.0]} d5 {[%emt 0.0]} 2. c4 {[%emt 1.725]} e6 {[%emt 1.016]} 3. cxd5 {[%emt 0.945]} exd5 {[%emt 1.031]} 4. Nc3 {[%emt 0.401]} c5 {[%emt 1.375]} 5. Nf3 {[%emt 4.618]} Nc


[Event "FICS rated blitz game"]
[Site "FICS freechess.org"]
[FICSGamesDBGameNo "314925110"]
[White "HUGOABEL"]
[Black "bartovic"]
[WhiteElo "2050"]
[BlackElo "2051"]
[TimeControl "180+0"]
[Date "2012.09.24"]
[Time "13:36:00"]
[WhiteClock "0:03:00.000"]
[BlackClock "0:03:00.000"]
[ECO "C77"]
[PlyCount "63"]
[Result "1-0"]

1. e4 {[%emt 0.0]} e5 {[%emt 0.0]} 2. Nf3 {[%emt 1.451]} Nc6 {[%emt 0.762]} 3. Bb5 {[%emt 2.269]} a6 {[%emt 0.595]} 4. Ba4 {[%emt 1.322]} Nf6 {[%emt 1.073]} 5. Qe2 {[%emt 1.654]} b5


[Event "FICS rated blitz game"]
[Site "FICS freechess.org"]
[FICSGamesDBGameNo "314924923"]
[White "cviksa"]
[Black "siissiis"]
[WhiteElo "2063"]
[BlackElo "2168"]
[TimeControl "180+0"]
[Date "2012.09.24"]
[Time "13:33:00"]
[WhiteClock "0:03:00.000"]
[BlackClock "0:03:00.000"]
[ECO "B00"]
[PlyCount "42"]
[Result "0-1"]

```

Review of data types

- Data types
    - logical
    - numeric
    - character
- What is coersion?
    - what happens if I coerce numeric, logical and character together into a vector?

Text data

- Text data are everywhere!
- String basics
  - sub-string (substr)
  - string split (strsplit)
- Regular expressions
  - the idea behind pattern matching
  - basic syntax
  - special characters

String basics

What we know: character data or strings

```
# we know that we can create a character vector
a <- c("the", "weather", "is", "good", "today")
a

## [1] "the"     "weather" "is"      "good"    "today"

# we know they can be compared as in a dictionary
a[1] < a[2]

## [1] TRUE

# we know that common data structure works with character data
identical(a, "the weather is good today")

## [1] FALSE

# we know data frame and lists can contain numeric and character data
df <- data.frame(id = c(1, 2, 3),
                position = c("Assistant Professor", "Associate Professor", "Full Professor"))
df

##   id            position
## 1  1 Assistant Professor
## 2  2 Associate Professor
## 3  3      Full Professor
```

paste() will paste things into strings

```
# paste() will combine objects together into a string
paste("the life of", pi, sep = " ")     # space is the default separator

## [1] "the life of 3.14159265358979"

# can use some separator other than the default " "
paste("to", "be", "or", "not", "to", "be", sep = "-")

## [1] "to-be-or-not-to-be"

# recycling rule applies if two things are different length
paste("X", 1:5, sep = ".")

## [1] "X.1" "X.2" "X.3" "X.4" "X.5"
```

Example: use paste() to rename column names

```r
# reshape a data frame
library(reshape)
long_dat <- data.frame(id = c(1, 1, 2, 2),
        year = c(2019, 2020, 2020, 2021),
        inc = c(100, 110, 110, 120))

wide_dat <- cast(data = long_dat,
        formula = id ~ year,
        value = "inc")

# What does wide_dat look like?
wide_dat
```

```
##   id 2019 2020 2021
## 1  1  100  110   NA
## 2  2   NA  110  120
```

Example: use paste() to rename column names

```
# examine the column name
colnames(wide_dat)

## [1] "id"   "2019" "2020" "2021"

# rename column name
colnames(wide_dat)[2:4] <- paste("inc", colnames(wide_dat), sep = "_")[2:4]

# wide_dat again
wide_dat

##   id inc_2019 inc_2020 inc_2021
## 1  1      100      110       NA
## 2  2       NA      110      120
```

Some useful string functions

Counting number of characters (vs elements)

```r
# How many characters are counted by nchar
nchar(c("How", "many", "characters?"))
```

```
## [1]  3  4 11
```

```r
# How many elements are counted by length
length(c("How", "many", "characters?"))
```

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

Conversion between upper and lower cases

```
# convert to lower cases
tolower(c("This iS", "sUPeR FuN"))

## [1] "this is"   "super fun"

# convert to upper cases
toupper(c("This iS", "sUPeR FuN"))

## [1] "THIS IS"   "SUPER FUN"

# can use the 'Hmisc' library
#    to convert initials to upper
library(Hmisc)
capitalize(tolower(c("This iS", "sUPeR FuN")))

## [1] "This is"   "Super fun"
```

Conversion between upper and lower cases

```
# cat() will beautify the string in the console display

random_string <- "Today's weather is very nice,
let's not do our homework!!"

random_string

## [1] "Today's weather is very nice, \nlet's not do our homework!!"

cat(random_string)

## Today's weather is very nice,
## let's not do our homework!!
```

Two very useful string functions: substr() and strsplit()

Finding part of a string (and do something with it)

- Often we want part of a string
- And very often this part is very well-defined
- We cover 3 cases of this; for example:
  - substr(): if we want the second to fourth character in a string
  - strsplit(): if we want the part of the string after the space
  - grep() and other regular expression functions: if we want to find a specific pattern in a string and/or do something with it

Substring: substr()

```
# find sub-string from start position to stop position
substr("abcdef", start = 1, stop = 3)

## [1] "abc"

# can substr a vector
substr(c("abc", "def"), start = 1, stop = 2)

## [1] "ab" "de"

# substring() is similar but can take vector arguments for start and stop
substring("abcdef", first = 1:6, last = 1:6)

## [1] "a" "b" "c" "d" "e" "f"

# substring() can also omit "last"
substring("abcdef", first = 1:3)        # default is to stop at end of string

## [1] "abcdef" "bcdef"  "cdef"
```

```
# Your turn:
substring("abcdef", first = 1:6, last = 5:6)
#    as usual: good for understanding but don't try this at home
```

Substring: substr()

```
# for example, extract area code
phone_number <- "(585) 345 7890"
substr(phone_number, start = 2, stop = 4)

## [1] "585"

# replace area code with a californian number
substr(phone_number, start = 2, stop = 4) <- as.character(424)  # simply "424"
phone_number

## [1] "(424) 345 7890"
```

```
# compare with this example: what do we get?
phonenr <- c("(585) 123 4567", "424 876 5432")
substr(phoenr, start = 2, stop = 4)
```

Split a string

```
# a sentense
sentense <- "this is a string"

# split it by space
split.sentense <- strsplit(sentense, " ")
split.sentense

## [[1]]
## [1] "this"   "is"     "a"      "string"

# note that the result is a list
#    unlist it to create a vector
unlist(split.sentense)

## [1] "this"   "is"     "a"      "string"
```

strsplit()

- The information we want is structured by patterned separators
- Can use strsplit() to split the string, by given patterns
- Results are returned in **a list**

Split a vector of strings

```
# phone numbers
numbers <- c("585-234-5678", "424-123-3452", "810-259-1234")

# split it by '-'
split.numbers <- strsplit(numbers, "-")
split.numbers

## [[1]]
## [1] "585"  "234"  "5678"
##
## [[2]]
## [1] "424"  "123"  "3452"
##
## [[3]]
## [1] "810"  "259"  "1234"

# how would you get area code?
area.code <- character(3)       # vector of length 3 with empty characters
area.code[1] <- split.numbers[[1]][1]
area.code[2] <- split.numbers[[2]][1]
area.code[3] <- split.numbers[[3]][1]
area.code       # remark: should use a "for-loop," will get to that next week

## [1] "585" "424" "810"
```

Split can generate “uneven” results

```
# names, note that structures are different
names <- c("Adam Smith", "George W. Bush")

# split it by space
split.names <- strsplit(names, " ")
split.names     # elements with different length

## [[1]]
## [1] "Adam"  "Smith"
##
## [[2]]
## [1] "George" "W."     "Bush"

# first name's easy to get; how about last name?
last.name <- character(2)
last.name[1] <- tail(split.names[[1]], n = 1)
last.name[2] <- tail(split.names[[2]], n = 1)
# tail(..., n = 1) finds the last element

last.name

## [1] "Smith" "Bush"
```

Be careful about the patterns you specify

```
# names, note that space could mean different things
names <- c("Paul B. Ellickson", "Oleksandr 'Alex' Shcherbakov",
    "Jean Francois Houde", "Xavi Vidal Berastein")

# split it by space
split.names <- strsplit(names, " ")
split.names     # will not recognize first, middle and last name

## [[1]]
## [1] "Paul"      "B."        "Ellickson"
##
## [[2]]
## [1] "Oleksandr"   "'Alex'"      "Shcherbakov"
##
## [[3]]
## [1] "Jean"     "Francois" "Houde"
##
## [[4]]
## [1] "Xavi"      "Vidal"     "Berastein"

names.correct <- c("Paul B. Ellickson", "Oleksandr('Alex') Shcherbakov",
    "Jean-Francois Houde", "Xavi Vidal-Berastein")
# then do the split from here...
```

Regular expression basics

Barack Obama’s 2008 “Yes, we can” speech (Nov 5, 2008)

"It was the call of workers who organised, women who reached for the ballot, a president who chose the moon as our new frontier, and a king who took us to the mountaintop and pointed the way to the promised land: Yes, we can, to justice and equality. Yes, we can, to opportunity and prosperity. Yes, we can heal this nation. Yes, we can repair this world. Yes, we can.”

How many “can” in the paragraph?

```r
# in Obama's famous "yes, we can" speech (Nov 5, 2008)
speech.char <- "Yes, we can, to justice and equality. Yes, we can, to opportunity and prosperity.
Yes, we can heal this nation. Yes, we can repair this world. Yes, we can."

# first decompose it into a vector
speech.vec <- unlist(strsplit(speech.char, " "))        # sep by space
speech.vec

##  [1] "Yes,"         "we"           "can,"         "to"           "justice"
##  [6] "and"          "equality."    "Yes,"         "we"           "can,"
## [11] "to"           "opportunity"  "and"          "prosperity."  "\nYes,"
## [16] "we"           "can"          "heal"         "this"         "nation."
## [21] "Yes,"         "we"           "can"          "repair"       "this"
## [26] "world."       "Yes,"         "we"           "can."

# note: \n refers to "new line" or a press of enter, here as a special character

# where are the "can"s?
match <- grep("can", speech.vec)
#    Note: match pattern and returns location (index) in the vector
match

## [1]  3 10 17 23 29

# how many?
length(match)

## [1] 5
```

Regular expressions

- **Confirm** a pattern in a string
  - grep()
  - grepl()
- **Locate** where a pattern is in a string
  - regexpr()
  - gregexpr()
- **Extract** matched patterns
  - grep()
- **Replace** the pattern with another string
  - sub()
  - gsub()

Confirm a pattern

```
# recall the speech (as a scalar) and the splitted (vector) version
speech.char <- "Yes, we can, to justice and equality. Yes, we can, to opportunity and prosperity.
Yes, we can heal this nation. Yes, we can repair this world. Yes, we can."

speech.vec <- unlist(strsplit(speech.char, " "))

# confirm we find "can"
match <- grep("can", speech.vec)

grep("can", speech.char)        # indices (only 1 element)

## [1] 1

grep("can", speech.vec)  # indices

## [1]  3 10 17 23 29

grepl("can", speech.vec)        # logical, found (T) or not found (F)

##  [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [25] FALSE FALSE FALSE FALSE  TRUE
```

Note that == would not work

```
# strict equality would not work
which(speech.vec == "can")

## [1] 17 23

# and this is because some of the
#    matched elements are "can." and "can,"
```

Confirm a general pattern

```
# now some "can"s are capitalized to reflect emphasis
speech.char <- "Yes, we caN, to justice and equality. Yes, we can, to opportunity and prosperity.
Yes, we cAN heal this nation. Yes, we Can repair this world. Yes, we can."

speech.vec <- unlist(strsplit(speech.char, " "))

# confirm we find "can" (but not all of them)
grep("can", speech.vec)

## [1] 10 29

#  did not find everything because we specifically queried lower cases

# have to match a general pattern
grep("[Cc][Aa][Nn]", speech.vec)

## [1]  3 10 17 23 29

#    Note: every '[]' indicates 'either or' here
```

Extract a match

```
# recall speech.vec
speech.vec <- unlist(strsplit(speech.char, " "))

# what are the elements that match "can"?
grep("[Cc][Aa][Nn]", speech.vec, value = TRUE)

## [1] "caN," "can," "cAN"  "Can"  "can."
```

Locate a pattern

```
# recall the speech (as a scalar) and the splitted (vector) version
speech.char <- "Yes, we can, to justice and equality. Yes, we can, to opportunity and prosperity.
Yes, we can heal this nation. Yes, we can repair this world. Yes, we can."

# indices as elements of the vector (here scalar)
grep("can", speech.char)

## [1] 1

# FIRST APPEARED location as in part of a string
regexpr("can", speech.char)

## [1] 9
## attr(,"match.length")
## [1] 3
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE

# location of EVERY APPEARANCE (can be used on a character vector)
gregexpr("can", speech.char)

## [[1]]
## [1]   9  47  92 122 153
## attr(,"match.length")
## [1] 3 3 3 3 3
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

Substitute a pattern to something else

```
# recall the version with some capitalized CANs
speech.char <- "Yes, we caN, to justice and equality. Yes, we can, to opportunity and prosperity.
Yes, we cAN heal this nation. Yes, we Can repair this world. Yes, we can."

# replace the FIRST match
sub.1 <- sub("[Cc][Aa][Nn]", "can", speech.char)
cat(sub.1)

## Yes, we can, to justice and equality. Yes, we can, to opportunity and prosperity.
## Yes, we cAN heal this nation. Yes, we Can repair this world. Yes, we can.

#    Note: original string has a new line "\n" but cat() prints the new line

# replace ALL match
sub.2 <- gsub("[Cc][Aa][Nn]", "can", speech.char)
cat(sub.2)

## Yes, we can, to justice and equality. Yes, we can, to opportunity and prosperity.
## Yes, we can heal this nation. Yes, we can repair this world. Yes, we can.
```

Objects in pattern matching

	Meaning
\\n	new line
\\d	any digit
\\D	any non-digit
\\s	space
\\b	word boundary
[a-z]	any lower case letter
[A-Z]	any upper case letter
[0-9]	any digit
.	any one character
*	pattern repeated zero or more times
+	pattern repeated one or more times
$	represents end of the string
^	represents the beginning of the string
{2}	repeated exactly twice
...	

Special characters

Character	Meaning	Refer to as symbol?
.	any one character	\\.
$	end of the string	\\$
+	repetition at *least* once	\\+
?	repetition at *most* once	\\?
[	grouping single character	\\[
|	grouping groups of character	\\\|
\	backslash used here	\\\\
	...	

Special characters: example

```
# split a Windows local folder
fileloc <- "C:\\Program Files\\R\\R-3.3.1\\bin\\"
cat(fileloc)

## C:\Program Files\R\R-3.3.1\bin\

# note that we use double back slash to
#    "escape" from backslash as a special symbol

# what if we only want to record R\R-3.3.1\bin?
split.fileloc <- strsplit(fileloc, "\\\\")
split.fileloc

## [[1]]
## [1] "C:"            "Program Files" "R"             "R-3.3.1"
## [5] "bin"

# basically, a slash is an "escape" so '\\' means
#    we literally want the symbol \
#    However, in fileloc itself \ is stored as \\
#    So in the end it turned out to be '\\\\'
```

What's going on?

- What's going on?
  - '\' represents an "escape" from a special character
  - '\\' means escape this character so it prints out the symbol '\'
  - so in variable fileloc, each symbol '\' is stored as '\\'
- Now in pattern matching
  - we want to match '\' twice in fileloc
  - but we need to specify each slash by '\\'
  - so in the end four slashes

Your turn: general patterns

```r
# example 1
string1 <- "+-3-2+1"    # I want 321

gsub("\\D+", "", string1)

# compare alternatives:
gsub("\\D", "", string1)

sub("\\D+", "", string1)

sub("\\D", "", string1)
```

Your turn: general patterns

```
# example 1
string1 <- "+-3-2+1"    # I want 321

gsub("\\D+", "", string1)

# compare alternatives:
gsub("\\D", "", string1)

sub("\\D+", "", string1)

sub("\\D", "", string1)
```

```
## [1] "321"
## [1] "321"
## [1] "3-2+1"
## [1] "-3-2+1"
```

Real-ish example 1: locating general phone number patterns

```
# example
sentense <- "My phone number is (585)-234-5678.
    His phone number is (426)-811-1234.
    And the office hotline is (888)-888-8888"

# what are the phone numbers?
split.sentense <- unlist(strsplit(sentense, " "))
phone.numbers <- grep("\\([0-9]{3}\\)\\-[0-9]{3}\\-[0-9]{4}",
        split.sentense, value = TRUE)
phone.numbers

## [1] "(585)-234-5678." "(426)-811-1234." "(888)-888-8888"

# now what if we don't want non-numeric?
as.numeric(phone.numbers)       # this won't do...

                             ## Warning:  NAs introduced by coercion

## [1] NA NA NA

# correct way
phone.clean <- gsub("\\D*", "", phone.numbers)  # any non-digit, repeated any times
phone.clean

## [1] "5852345678" "4268111234" "8888888888"

#    can now use as.numeric
```

Real example 2: which Airbnb listings provide Wifi?

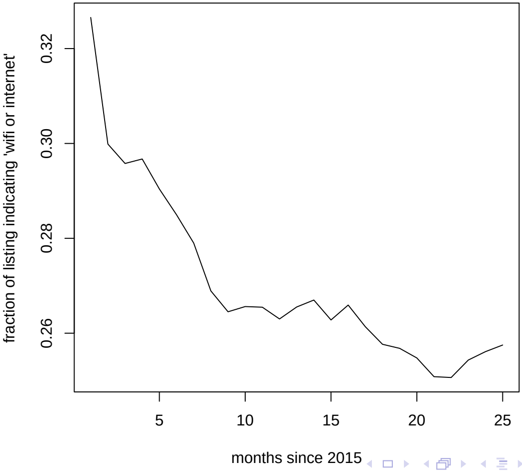
```
# remember we had a sizable airbnb dataset in Week 1
library(data.table)

                                                  ##
                                  ## Attaching package:  'data.table'

                       ## The following object is masked from 'package:reshape':
                                                  ##
                                             ##     melt

listings <- fread('listings.csv')

# check header
names(listings)

##  [1] "V1"                             "id"
##  [3] "listing_url"                    "scrape_id"
##  [5] "last_scraped"                   "name"
##  [7] "summary"                        "space"
##  [9] "description"                    "experiences_offered"
## [11] "neighborhood_overview"          "notes"
## [13] "transit"                        "thumbnail_url"
## [15] "medium_url"                     "picture_url"
## [17] "xl_picture_url"                 "host_id"
## [19] "host_url"                       "host_name"
## [21] "host_since"                     "host_location"
## [23] "host_about"                     "host_response_time"
## [25] "host_response_rate"             "host_acceptance_rate"
## [27] "host_is_superhost"              "host_thumbnail_url"
## [29] "host_picture_url"               "host_neighbourhood"
## [31] "host_listings_count"            "host_total_listings_count"
## [33] "host_verifications"             "host_has_profile_pic"
## [35] "host_identity_verified"         "street"
## [37] "neighbourhood"                  "neighbourhood_cleansed"
## [39] "neighbourhood_group_cleansed"   "city"
## [41] "state"                          "zipcode"
## [43] "market"                         "country_code"
```

```
# check first obs of description (truncated in console)
listings$description[5:10]

## [1] "Designer's down town loft, constructed by an architect, more than once published in architectural
## [2] "A breath away from  Zappeion and the National Royal Garden  just a few seconds away . There are su
## [3] "Spacious first floor 2 bedroom apartment perfectly located for sightseeing, transport and shopping
## [4] "Old neoclassic architecture two level store. One floor (level) is for rent. Pedestrian area !"
## [5] "The house is located in one of the oldest and picturesque neighbourghoods of the center of Athens,
## [6] "the.flat |  A high-ceilinged penthouse of a 50s residential building in the centre of Athens, righ
```

```
# Question: which listing provides wifi?
#    What fraction provides Wifi?
#    Any growth of Wifi over time?

# Can grep() to match "wifi"
listings$is_wifi <- grepl("wifi", listings$description)
table(listings$is_wifi) # can't be that small... what to do?

##
##  FALSE   TRUE
## 110277   6200
```

```r
# What should we match on?
listings$is_wifi <- grepl("wifi|wi-fi|wireless|internet",
        tolower(listings$description))
#    to be fair, you can argue "internet" can be wired

table(listings$is_wifi) # much higher now

##
## FALSE  TRUE
## 85593 30884
```

```r
# Over time? Take a (data.table) aggregate
#    WHY?!
frac_wifi <- listings[, .(wifi = mean(is_wifi)), date_code]
plot(frac_wifi$date_code, frac_wifi$wifi,
        type = 'l', ylab = "fraction of listing indicating 'wifi or internet'",
        xlab = "months since 2015")
```



So far

- Strings have patterns and we can match them to do things
    - find part of it that is between xth and yth characters
        - e.g. the part between 2nd and 4th character of “together” is “oge”
    - split strings into a list using a certain pattern
        - e.g. “together” can be split by “e” and the result is?
    - pattern match a string using the regular expressions

Date and time

Date and time

- Date are often recorded in very “nasty” ways...

```r
# for example
date1 <- c("1999aug3", "2000jan25", "2001sep16")
date2 <- c("01-01-2001", "03-04-2002", "25-03-2003")
date3 <- c("990101", "000202", "010325")
```

- Time could be added to make it worse

```r
# for example
time1 <- c("2005-09-18 08:15:01 PDT", "2006-08-25 09:20:01 PDT")
time2 <- c("2005Oct21 18:47", "2011Dec25 06:47")
```

- Of course can process this by regular expressions (but this is cumbersome)
    - many statistics software provide date and time functions to deal with this

Date

```
# use as.Date to convert to date
date1 <- c("1999aug3", "2000jan25", "2001sep16")
class(date1)    # it's a character vector

## [1] "character"

# transform it to date
Date1 <- as.Date(date1, format = "%Y%b%d")      # can omit 'format = '

# nicely displayed
Date1

## [1] "1999-08-03" "2000-01-25" "2001-09-16"

# now it's date
class(Date1)

## [1] "Date"

# can take difference (plus will not make sense)
Date1[2] - Date1[1]

## Time difference of 175 days
```

Date: taking differences

```
# more generally, can take differences with a particular unit
difftime(Date1[2], Date1[1], units = "days")

## Time difference of 175 days

difftime(Date1[2], Date1[1], units = "weeks")

## Time difference of 25 weeks
```

Date and time format codes

symbol	object	example
%d	day (in number)	14
%a	weekday	Mon
%m	month (in number)	2
%b	month (in abbrev.)	feb
%B	month (in full)	February
%y	year (2 digit)	01
%Y	year (4 digit)	2001
%H	hour (24 hr)	23
%I	hour (12 hr)	11
%p	AM/PM	pm
%M	minute	54
%S	second	01

Transform and format dates

```r
# recall date2
date2 <- c("01-01-2001", "03-04-2002", "25-03-2003")
Date2 <- as.Date(date2, "%d-%m-%Y")
# NOTE: I have to specify the separator '-'
Date2

## [1] "2001-01-01" "2002-04-03" "2003-03-25"

# format Date2 into year and month
year <- format(Date2, "%Y")
year

## [1] "2001" "2002" "2003"

month <- format(Date2, "%m")
month

## [1] "01" "04" "03"

# or simply use the month function
month(Date2)

## [1] 1 4 3
```

Your turn

```r
# recall date3
date3 <- c("990101", "000202", "010325")
Date3 <-


# format Date3 into full month name and weekday
monthname <-

weekday <-
```

Time data: POSIXct

```
# recall time1
time1 <- c("2005-09-18 08:15:01 PDT", "2006-08-25 09:20:01 PDT")

# convert it into POSIXct class (google)
Time1_ct <- as.POSIXct(time1, tz = "US/Pacific")
#    Note: time zone is a nasty animal; usually we don't
#          need to deal with it if time is all local;
#          i.e. we don't need to convert time between
#          time zones within the same data set

Time1_ct        # displayed nicely

## [1] "2005-09-18 08:15:01 PDT" "2006-08-25 09:20:01 PDT"

# in fact, POSIXct data are stored as #seconds since a baseline
unclass(Time1_ct)

## [1] 1127056501 1156522801
## attr(,"tzone")
## [1] "US/Pacific"
```

Your turn

```
# recall time2
time2 <- c("2005Oct21 18:47", "2011Dec25 06:47")

# can't work on this directly
as.POSIXct(time2)

## Error in as.POSIXlt.character(x, tz, ...):  character string is not in
a standard unambiguous format
```

```
# so you need to specify a format on this...

Time2_ct <- as.POSIXct(time2, format = "%Y%b%d %H:%M")
Time2_ct

## [1] "2005-10-21 18:47:00 EDT" "2011-12-25 06:47:00 EST"
```

Time data: POSIXlt

```
# recall time1
time1 <- c("2005-09-18 08:15:01 PDT", "2006-08-25 09:20:01 PDT")

# POSIXlt is a different storage format
Time1_lt <- as.POSIXlt(time1, tz = "US/Pacific")

# in fact, POSIXlt is stored as a list
#    The list nature of POSIXlt allows flexible operations on time
unclass(Time1_lt)

## $sec
## [1] 1 1
##
## $min
## [1] 15 20
##
## $hour
## [1] 8 9
##
## $mday
## [1] 18 25
##
## $mon
## [1] 8 7
##
## $year
## [1] 105 106
##
## $wday
## [1] 0 5
##
## $yday
## [1] 260 236
##
## $isdst
## [1] 1 1
```

Conclusion

- Strings have patterns and those patterns allow us to do many things
- Extract a **fixed** part of a string: substr()
- Regular expressions:
  - extract a matched pattern: grep()
  - replace a matched pattern: gsub()
  - split a string based on a certain pattern: strsplit()
- Date-related functions