Biostats 597E

Week 6 - Introduction to Regular Expression in R

Regular Expression

Regular expression is a pattern that describes a set of string. We can use that to

- Check whether a string contains certain pattern, e.g. whether a string contains "statistics"
- Replace certain pattern in a string with new string, e.g. replace consecutive spaces with a single space
- Extract certain pattern from a string, e.g. extract social security number from a string
- And many more

The building block of a regular expression pattern is single character

R Functions - grep and grepl

```
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
     fixed = FALSE, useBytes = FALSE, invert = FALSE)
grepl(pattern, x, ignore.case = FALSE, perl = FALSE,
      fixed = FALSE, useBytes = FALSE)
```

grep returns a vector of the indicies of the elements of x that yielded a match. If `value=TRUE` then it returns a character vector containing the selected elements of x

grepl returns a logical vector (match or not)

```
x <- c("statistics", "epidemiology", "biostatistics")
grep("statistics", x)
grep("statistics", x, value = T)
grep("math", x)
grepl("statistics", x)
```

R Functions - sub and gsub

```
sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
    fixed = FALSE, useBytes = FALSE)
gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
     fixed = FALSE, useBytes = FALSE)
```

sub and **gsub** perform replacement of the first and all matches respectively.

```
x <- c("stat 101", "epidemiology", "biostat 597", "biostat-stat")
sub("stat", "statistics", x)
gsub("stat", "statistics", x)
gsub("STAT", "statistics", x, ignore.case = TRUE)
```

Metacharacters

Some characters have special meanings in regular expressions. We need to escape them in regular expression.

Metacharacters:

```
$ * + . ? [ ] ^ { } | ( ) \
```

They can escaped by preceeding double slash in pattern `'\\'`, e.g. using `\\$` for `$` and using `\\\\` for `\\`.

```
gsub("\\.", "_", "var.name")
gsub("\\\\", "/", "c:\\a.txt")
gsub("\\*", "X", "3 * 6")
```

Character Class

A character class is a list of characters enclosed between [and] which matches any single character in that list.

Example We want to match a pattern 'a' followed by 'x' or 'y' or 'z', that is we want to match 'ax', 'ay' or 'az'

```
grepl("a[xyz]", c("maax", "maay", "yaz"))
```

Metacharacters lose their special meanings in character class except ^ - \. No need to escape

```
gsub("[$()]", "#", "($$$)\\")
gsub("[\\$\\(\\)]", "#", "($$$)\\") # will add "\" to pattern
```

Predefined Character Classes

- **[:alpha:]** Alphabetic characters: [:lower:] and [:upper:]
- **[:alnum:]** Alphanumeric characters: [:alpha:] and [:digit:]
- **[:digit:]** Digits: 0 1 2 3 4 5 6 7 8 9
- **[:punct:]** Punctuation characters: ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~
- **[:space:]** Space characters
- more can be found in R `?regex`

Example: remove all punctuation characters from a string

```
gsub("[:punct:]", "", "<!--[if lt IE 7]>") # not work
gsub("[[:punct:]]", "", "<!--[if lt IE 7]>")
gsub("[:punct:]", "", "p u n c t :") # it matches characters of p u n c t :
```

Beginning and End of a String or Word

- ^ matches the beginning of a string (not first character)
- $ matches the end of a string (not last character)
- \< matches the beginning of a word (not first character)
- \> matches the end of a word (not last character)

```
gsub("^[[:space:]]", "", " boy girl ") # remove leading space character
gsub("[[:space:]]$", "", " boy girl ") # remove trailing space character
gsub("[[:space:]]\\<", "", " boy girl ") # remove space before a word
gsub("\\>[[:space:]]", "", " boy girl ") # remove space after a word
grepl("\\<b",  "boy girl ") # any word starting with b
grep("^Sepal", names(iris), value = TRUE) # names starting with Sepal
```

We will learn how to remove consecutive spaces.

Reptitions

A regular expression may be followed by one of several repetition quantifiers:

- **?**: The preceding item is optional and will be matched at most once.
- *****: The preceding item will be matched zero or more times.
- **+**: The preceding item will be matched one or more times.
- **{n}**: The preceding item is matched exactly n times.
- **{n,}**: The preceding item is matched n or more times.
- **{n,m}**: The preceding item is matched at least n times, but not more than m times.

Reptitions Example

```
gsub("^[[:space:]]+", "", "     hello") # remove all leading spaces
gsub("[[:space:]]+$", "", "hello world    ") # remove all trailing spaces
gsub("^0+", "", "00123") #remove all leading zeroes
# convert multiple consecutive spaces to one space
gsub(" +", " ", c("Joe   Smith", "Tom     K"))
```

Match ssn: 3 digits followed by an optional "-", then followed by 2 digits, then followed by optional "-", then followed by 4 digits

pattern:

```
p <- "[[:digit:]]{3}-?[[:digit:]]{2}-?[[:digit:]]{4}"


grepl(p, "123-45-1234")
grepl(p, "123456789")
grepl(p, "12-345-56789")
grepl(p, "12345-6789")
```

Example

Find all **.Rmd** files in a directory

```
all_files <- list.files("/Users/xgu/Desktop/Biostats597E",
  recursive = TRUE, full.names = TRUE)

Rmd_files <- grep("\\.Rmd$", all_files, value = TRUE)
```

Example

When R reads a table from text files or csv files, the variable names may contain punctuation characters like '.'. '.' is legal in R variable name, but may not be legal in other languages such as SAS.

We want to convert all punctuation characters in R's variable name to "_".

```
names(iris)
names(iris) <- gsub("[[:punct:]]", "_", names(iris))
```

Match Email Address

We assume we restrict the email only contains characters of alphabetic, number, '.', '_', '-'

The pattern should be one block followed by '@' followed by one block without '.' followed by '.' followed by another block without '.'.

```
p <- "[[:alnum:]_.-]+@[[:alnum:]_-]+\\.[[:alnum:]_-]+"
grepl(p, "tom@gmail.com")
grepl(p, "alice-b@google.com")
grepl(p, "tom@gmail")
grepl(p, "a1244_b@yahoo.com")
```

Note: This pattern is not a perfect one for email.

For More Details

Refer to `?regex` `?gsub` `?grep` for more details.

There are a lot of more about regular expression not discuessed here, but what we have discussed should be good to handle many situations.

Exercise

Extract most popular baby names from **babynames.html** file

```
babynames <- readLines("babynames.html")
```