CSE 252- Data Science

Regular expressions and text mining

String manipulations and regex

Working with strings

- In R, a piece of text is represented as a sequence of characters (letters, numbers, and symbols).
- The data type R provides for storing sequences of characters is character. Formally, the mode of an object that holds character strings in R is "character".
- You express character strings by surrounding text within double quotes:

```r
x <- "This is R!"
x
```

```
## [1] "This is R!"
```

```r
class(x)
```

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

Working with strings

You can use single quotes as well

```
x <- 'This is R!'
x
```

```
## [1] "This is R!"
```

```
class(x)
```

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

Working with strings

When writing strings, you can insert single quotes in a string with double quotes, and vice versa:

```
x <- "Why call langauge 'python' ? "
x
```

```
## [1] "Why call langauge 'python' ? "
```

```
x <- 'Why call langauge "python" ? '
x
```

```
## [1] "Why call langauge \"python\" ? "
```

Working with strings

If you try to include single quotes within single quotes, or double quotes within double quotes, it will not work

```
x <- "Why call langauge "python" ? "
```

```
x <- 'Why call langauge 'python' ? '
```

But if by some reason, you desperately want to do it, use escaping character (we will talk about this later)

Working with strings

Use \ as an escape before quotation mark

```r
x <- 'Why call langauge \'python\' ? '
x
```

```
## [1] "Why call langauge 'python' ? "
```

```r
class(x)
```

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

```r
is.character(x)
```

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

Check if x is a character

Working with strings

- The function **paste()** is perhaps one of the most important functions that you can use to create and build strings.
- **paste()** takes one or more R objects, converts them to "character", and then it concatenates (pastes) them to form one or several character strings. Its usage has the following form:

```r
PI <- paste("Life of", pi)
PI
```

```
## [1] "Life of 3.14159265358979"
```

Life of Pi

Working with strings

- Element wise concatenation of two vectors
- By default **space** is the seperator

```r
x <- 1:10
a <- letters[1:10]
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
a
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```r
k <- paste(x,a)
k
```

```
##  [1] "1 a"  "2 b"  "3 c"  "4 d"  "5 e"  "6 f"  "7 g"  "8 h"  "9 i"  "10 j"
```

Working with strings

Define your own seperator

```
k <- paste(x,a, sep="-")
k
```

```
##  [1] "1-a"  "2-b"  "3-c"  "4-d"  "5-e"  "6-f"  "7-g"  "8-h"  "9-i"  "10-j"
```

```
length(k)
```

```
## [1] 10
```

Working with strings

The argument collapse is an optional string to indicate if you want all the terms to be collapsed into a single string

```r
k <- paste(x,a, sep="-", collapse = ',')
k
```

```
## [1] "1-a,2-b,3-c,4-d,5-e,6-f,7-g,8-h,9-i,10-j"
```

```r
length(k)
```

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

Working with strings

Basic string manipulations

Function	Description
`nchar()`	number of characters
`tolower()`	convert to lower case
`toupper()`	convert to upper case
`casefold()`	case folding
`chartr()`	character translation
`abbreviate()`	abbreviation
`substring()`	substrings of a character vector
`substr()`	substrings of a character vector

Working with strings

nchar() counts the number of characters in the string

```
x <- 'This is R!'
nchar(x)
```

```
## [1] 10
```

Working with strings

If you apply it to the character vector, you will get back a vector with number of characters for each element

Built in vector with month names

```
month.name
```

```
##  [1] "January"   "February"  "March"     "April"     "May"
##  [6] "June"      "July"      "August"    "September" "October"
## [11] "November"  "December"
```

```
nchar(month.name)
```

```
##  [1] 7 8 5 5 3 4 4 6 9 7 8 8
```

Working with strings

upper and lower cases

```r
toupper(x)
```

```
## [1] "THIS IS R!"
```

```r
tolower(x)
```

```
## [1] "this is r!"
```

```r
casefold(x, upper=T)
```

```
## [1] "THIS IS R!"
```

```r
casefold(x, upper=F)
```

```
## [1] "this is r!"
```

Working with strings

Abbreviations are handy especially when you need to use them as labels/legends for the plots

```r
summer <- read.csv('summer.csv', stringsAsFactors = F)
discipline <- unique(summer$Discipline)
discipline[1:10]
```

```
##  [1] "Swimming"         "Athletics"        "Cycling Road"
##  [4] "Cycling Track"    "Fencing"          "Artistic G."
##  [7] "Shooting"         "Tennis"           "Weightlifting"
## [10] "Wrestling Gre-R"
```

```r
disc_abbr <- abbreviate(discipline, minlength = 5)
disc_abbr[1:10]
```

```
##        Swimming       Athletics    Cycling Road   Cycling Track
##         "Swmmn"         "Athlt"         "CyclR"         "CyclT"
##         Fencing     Artistic G.        Shooting          Tennis
##         "Fncng"         "ArtG."         "Shtng"         "Tenns"
##   Weightlifting Wrestling Gre-R
##         "Wghtl"         "WGr-R"
```

Working with strings

One common operation when working with strings is the extraction and replacement of some characters. There a various ways in which characters can be replaced. If the replacement is based on the positions that characters occupy in the string, you can use the functions **substr()** and **substring()**

substr() extracts or replaces substrings in a character vector. Its usage has the following form:

```
x <- "ABCDEF"
substr(x, start=3,stop =5)
```

```
## [1] "CDE"
```

Working with strings

You can replace the portion of the string with other string

```r
y <- c("may", "the", "force", "be", "with", "you")
substr(y, 2, 2) <- "#"
y
```

```
## [1] "m#y"   "t#e"   "f#rce" "b#"    "w#th"  "y#u"
```

```r
y <- c("may", "the", "force", "be", "with", "you")
substr(y, 2, 3) <- ":)"
y
```

```
## [1] "m:)"   "t:)"   "f:)ce" "b:"    "w:)h"  "y:)"
```

[Star Wars]()

Working with strings

- **stringr**
- Powerful but easy to learn
- Built on stringi

- Concise and consistent

  - All functions start with str_

  - All functions take a vector of strings as the first argument

```
install.packages('stringr')
```

Working with strings

Works the same way as substr() but allows for negative subsetting as well

```r
library(stringr)
str_sub("ABCDEF", start=1, end=3)
```

```
## [1] "ABC"
```

Getting the last two characters

```r
str_sub("ABCDEF", start=-2, end=-1)
```

```
## [1] "EF"
```

Working with strings

str_detect() detects the presence or absence of a pattern and returns a logical vector

```
geners <- c("Action, Adventure, Comedy","Comedy",
"Comedy, Drama, Drama, Romance",
"Crime, Drama, History")
str_detect(string = geners,  pattern = "Drama")
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

str_subset() Keep strings matching a pattern

```
str_subset(string = geners,  pattern = "Drama")
```

```
## [1] "Comedy, Drama, Drama, Romance" "Crime, Drama, History"
```

Working with strings

str_count() Count the number of matches in a string.

```
str_count(string = geners, pattern = "Drama")
```

```
## [1] 0 0 2 1
```

Working with strings

How many romcom movies are there ?
Romcom is the abbreviation for Romantic Comedy

```r
movies <- read.csv('movies3.csv', stringsAsFactors = F)
movies$Comedy <- str_detect(string = movies$Genre,  pattern = "Comedy")
movies$Romance <- str_detect(string = movies$Genre,  pattern = "Romance")
table(Romance= movies$Romance, Comedy=movies$Comedy)
```

```
##         Comedy
## Romance FALSE TRUE
##   FALSE  1478  886
##   TRUE    202  319
```

Working with strings

str_split() Split up a string into pieces, the result is a list of vectors for each string

```r
str_split(geners, ",")
```

```
## [[1]]
## [1] "Action"      " Adventure" " Comedy"
##
## [[2]]
## [1] "Comedy"
##
## [[3]]
## [1] "Comedy"    " Drama"    " Drama"    " Romance"
##
## [[4]]
## [1] "Crime"     " Drama"    " History"
```

Working with strings

Use simplify=T to get back a character matrix

```r
gen_m <- str_split(geners, ",", simplify = T)
gen_m
```

```
##      [,1]     [,2]         [,3]        [,4]
## [1,] "Action" " Adventure" " Comedy"   ""
## [2,] "Comedy" ""           ""          ""
## [3,] "Comedy" " Drama"     " Drama"    " Romance"
## [4,] "Crime"  " Drama"     " History"  ""
```

Working with strings

str_replace() Replace matched patterns in a string.

```
str_replace(geners, pattern = ",", replacement = " &")
```

```
## [1] "Action & Adventure, Comedy"      "Comedy"
## [3] "Comedy & Drama, Drama, Romance" "Crime & Drama, History"
```

Only the first match is replaced

For all matches use str_replace_all

```
str_replace_all(geners, pattern = ",", replacement = " &")
```

```
## [1] "Action & Adventure & Comedy"      "Comedy"
## [3] "Comedy & Drama & Drama & Romance" "Crime & Drama & History"
```

Working with strings

- stre_extract() extracts the pattern from the text
- Return NA if the pattern is not matched

- **str_extract()** returns the first match, use **str_extract_all()** for all matches

```
str <-c("123abd", "ab567cd", "abc5.00")
str_extract(str, pattern="123")
```

```
## [1] "123" NA    NA
```

Regular Expressions

Regular expressions

A **regular expression**, **regex** is, in theoretical computer science and formal language theory, a sequence of characters that define a search pattern. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings, or for input validation.

A **regular expression** is a specific pattern in a set of strings. For instance, one could have the following pattern : 2 digits, 2 letters and 4 digits.

Regular expressions

The most basic type of regular expressions are the literal characters which are characters that match themselves.

However, not all characters match themselves. Any character that is not a literal character is a metacharacter. This type of characters have a special meaning and they allow you to transform literal characters in very powerful ways.

Metacharacters

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

If you want to match the metacharacter you need to use \ (escape) before the metacharacter

Regular expressions

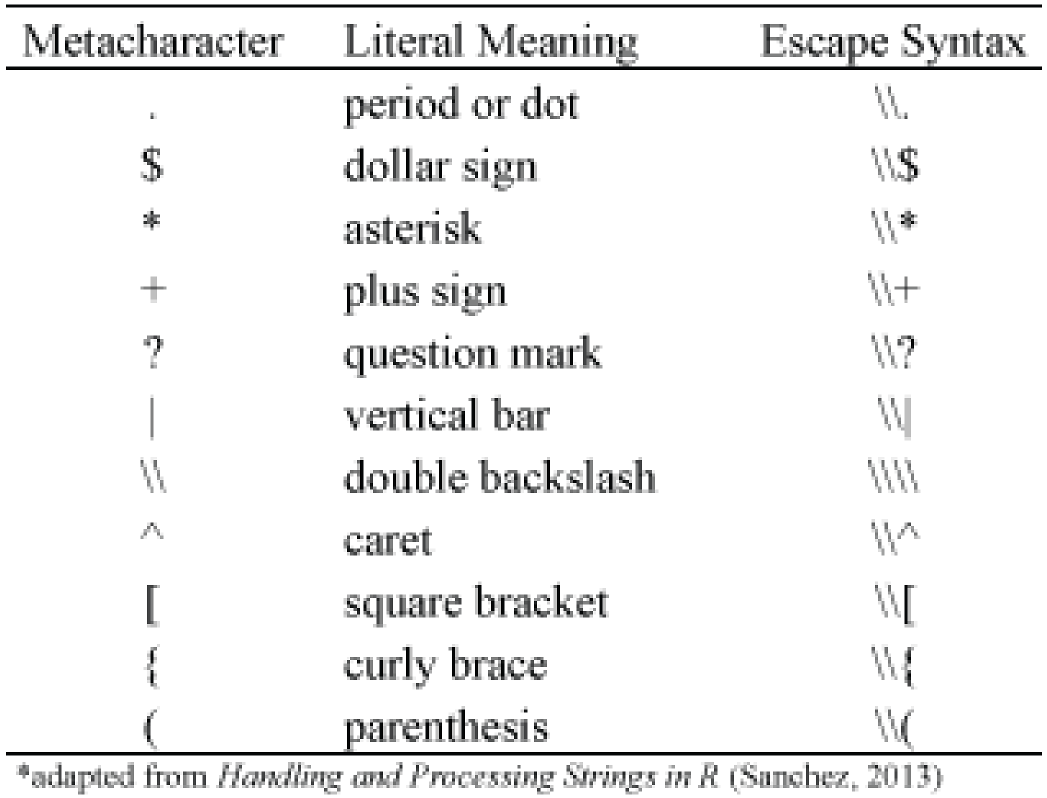
The first metacharacter you should learn about is the dot or period **"."**, better known as the wild metacharacter. This metacharacter is used to match ANY character except for a new line.

To actually match the dot character, what you need to do is escape the metacharacter. In most languages, the way to escape a metacharacter is by adding a backslash character in front of the metacharacter: "\.". When you use a backslash in front of a metacharacter you are “escaping” the character, this means that the character no longer has a special meaning, and it will match itself.

However, R is a bit different. Instead of using a backslash you have to use two backslashes: "5\\.00". This is because the backslash "\", which is another metacharacter, has a special meaning in R.

Regular expressions

Escape syntax for metacharacters



Metacharacter	Literal Meaning	Escape Syntax
.	period or dot	\\.
$	dollar sign	\\$
*	asterisk	*
+	plus sign	\\+
?	question mark	\\?
|	vertical bar	\\\|
\\	double backslash	\\\\
^	caret	\\^
[	square bracket	\\[
{	curly brace	\\{
(	parenthesis	\\(

*adapted from *Handling and Processing Strings in R* (Sanchez, 2013)

Regular expressions

All strings in the vector are matches

```r
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = ".")
```

```
## [1] "123abd"  "ab567cd" "abc5.00"
```

Use escape

```
str_subset(string = str,  pattern = "\\.")
```

```
## [1] "abc5.00"
```

Regular expressions

Other metacharacters:

^ indicates the beginning of the string
The following regex will match all the strings that start with “ab” characters

```
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = "^ab")
```

```
## [1] "ab567cd" "abc5.00"
```

- The $ sign indicates the end of the string
- The following regex will match all the strings ending with “cd”

```r
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = "cd$")
```

```
## [1] "ab567cd"
```

Regular expressions

Character ranges give you convenient shortcut based on the dash metacharacter "-" to indicate a range of characters. A character range consists of a character set with two characters separated by a dash or minus "-" sign.

For example: you want to match a string that has numbers from 1-4

```
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = "[1-4]")
```

```
## [1] "123abd"
```

Regular expressions

- The following regex will match all the strings containing lower letters from f to z
- As there are no such strings, the result is empty vector

```
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = "[f-z]")
```

```
## character(0)
```

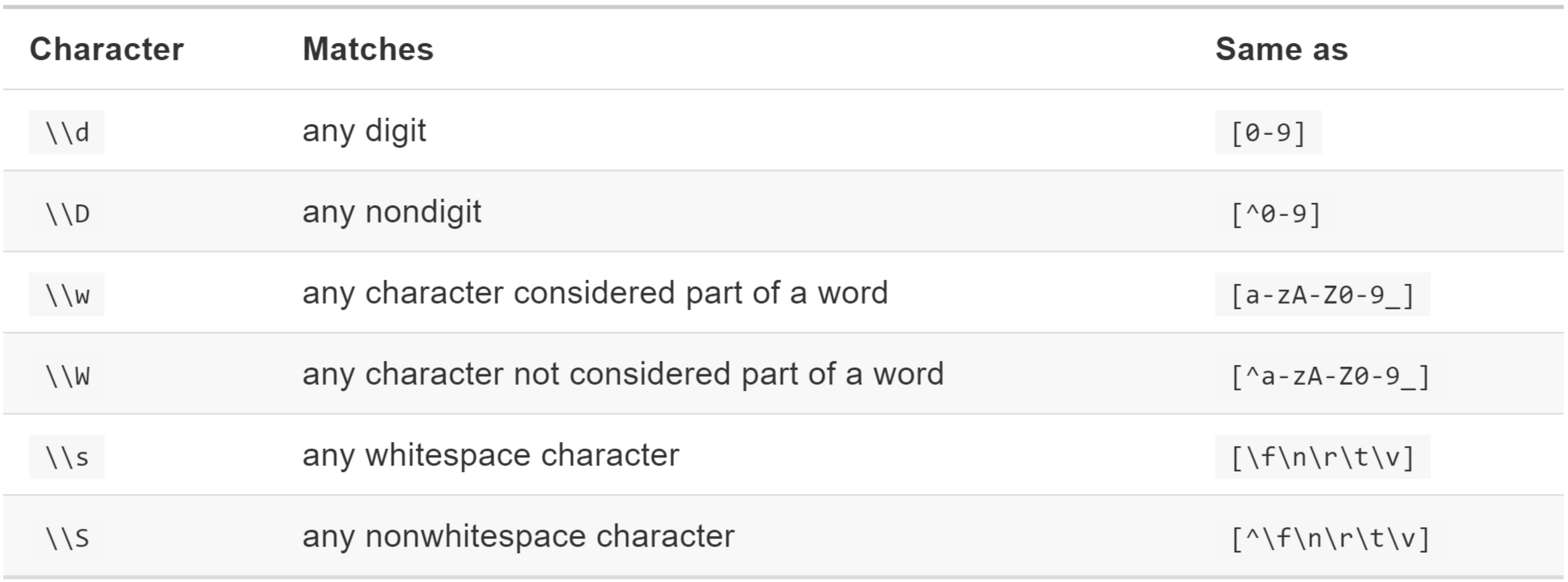
Regular expressions

Character classes

To match one of several characters in a specified set we can enclose the characters of concern with square brackets [ ]. In addition, to match any characters **not** in a specified character set we can include the caret ^ at the beginning of the set within the brackets. The following displays the general syntax for common character classes but these can be altered easily as shown in the examples that follow:

Regular expressions

Closely related with character sets and character ranges, regular expressions provide another useful construct called **character classes** which, as their name indicates, are used to match a certain class of characters. The most common character classes in most regex engines are:



Character	Matches	Same as
`\\d`	any digit	`[0-9]`
`\\D`	any nondigit	`[^0-9]`
`\\w`	any character considered part of a word	`[a-zA-Z0-9_]`
`\\W`	any character not considered part of a word	`[^a-zA-Z0-9_]`
`\\s`	any whitespace character	`[\f\n\r\t\v]`
`\\S`	any nonwhitespace character	`[^\f\n\r\t\v]`

Regular expressions

POSIX character classes

Class	Description	Same as
`[:alnum:]`	any letter or digit	`[a-zA-Z0-9]`
`[:alpha:]`	any letter	`[a-zA-Z]`
`[:digit:]`	any digit	`[0-9]`
`[:lower:]`	any lower case letter	`[a-z]`
`[:upper:]`	any upper case letter	`[A-Z]`
`[:space:]`	any whitespace inluding space	`[\f\n\r\t\v ]`
`[:punct:]`	any punctuation symbol	
`[:print:]`	any printable character	
`[:graph:]`	any printable character excluding space	
`[:xdigit:]`	any hexadecimal digit	`[a-fA-F0-9]`
`[:cntrl:]`	ASCII control characters	

Regular expressions

- The Awards column contains text information about awards won and nominations
- Look at the first 20 records. The text is standard for all the movies
- Can you create a column that will show how many wins and nominations each movies has ?

```r
movies$Awards[1:20]
```

```
##  [1] "4 wins & 8 nominations."
##  [2] "7 wins & 17 nominations."
##  [3] "1 win & 2 nominations."
##  [4] "9 wins & 28 nominations."
##  [5] "2 wins & 67 nominations."
##  [6] "1 nomination."
##  [7] "Won 1 Oscar. Another 87 wins & 171 nominations."
##  [8] "2 wins & 3 nominations."
##  [9] "2 wins & 4 nominations."
## [10] "Nominated for 1 Golden Globe. Another 5 wins & 7 nominations."
## [11] "1 win & 1 nomination."
## [12] "1 win & 3 nominations."
## [13] "Nominated for 1 Oscar. Another 1 win & 3 nominations."
## [14] "1 win & 2 nominations."
## [15] "Nominated for 2 Oscars. Another 11 wins & 5 nominations."
## [16] "Nominated for 1 Golden Globe. Another 3 wins & 31 nominations."
## [17] "2 wins & 2 nominations."
## [18] "2 wins & 2 nominations."
## [19] "1 win & 3 nominations."
## [20] "1 win & 5 nominations."
```

Regular expressions

- You can use str_replace_all, with character class [a-zA-Z] to delete all alphabetical letters from the string first. You replace the match with nothing
- We are left with numbers and punctuation marks
- Youy can use another regex to get rid off ounctuations

```r
movies$awards_num <- str_replace_all(movies$Awards, pattern="[a-zA-Z]",
                                 replacement = "")
movies$awards_num[1:20]
```

```
##  [1] "4  & 8 ."             "7  & 17 ."          "1  & 2 ."
##  [4] "9  & 28 ."            "2  & 67 ."          "1 ."
##  [7] " 1 .  87  & 171 ." "2  & 3 ."           "2  & 4 ."
## [10] "   1  .  5  & 7 ."  "1  & 1 ."           "1  & 3 ."
## [13] "   1 .  1  & 3 ."   "1  & 2 ."           "   2 .  11  & 5 ."
## [16] "   1  .  3  & 31 ." "2  & 2 ."           "2  & 2 ."
## [19] "1  & 3 ."             "1  & 5 ."
```

Regular expressions

- Or you can use pattern [^0-9] which says match everything that is not number (^ is the negation)
- Delete everything that is not a number

```r
movies$awards_num <- str_replace_all(movies$Awards, pattern="[^0-9]",
                                     replacement = " ")
movies$awards_num[1:20]
```

```
##  [1] "4        8             "
##  [2] "7        17             "
##  [3] "1       2             "
##  [4] "9        28             "
##  [5] "2        67             "
##  [6] "1            "
##  [7] "    1                87        171             "
##  [8] "2        3             "
##  [9] "2        4             "
## [10] "              1                       5        7             "
## [11] "1       1            "
## [12] "1       3             "
## [13] "              1                1       3             "
## [14] "1       2             "
## [15] "              2                 11        5             "
## [16] "              1                       3       31              "
## [17] "2        2             "
## [18] "2        2             "
## [19] "1       3             "
## [20] "1       5             "
```

Regular expressions

- We are left with lot of white space that need to be deleted
- The new vector need to have the numbers separated by a single space
- The regex we use here is \\s+, where \\s is the metacharacter for space and + matches at least 1 times.
- Next we need to trim white space

```
movies$awards_num <- str_replace_all(movies$awards_num, pattern="\\s+",
                                  replacement = " ")
movies$awards_num[1:20]
```

```
##  [1] "4 8 "        "7 17 "       "1 2 "        "9 28 "       "2 67 "
##  [6] "1 "          " 1 87 171 " "2 3 "        "2 4 "        " 1 5 7 "
## [11] "1 1 "        "1 3 "        " 1 1 3 "     "1 2 "        " 2 11 5 "
## [16] " 1 3 31 "    "2 2 "        "2 2 "        "1 3 "        "1 5 "
```

Regular expressions

- Trim white spaces
- After we will split the string using space as a pattern

```r
movies$awards_num <- trimws(movies$awards_num)
movies$awards_num[1:20]
```

```
##  [1] "4 8"        "7 17"       "1 2"        "9 28"       "2 67"       "1"
##  [7] "1 87 171" "2 3"        "2 4"        "1 5 7"      "1 1"        "1 3"
## [13] "1 1 3"      "1 2"        "2 11 5"     "1 3 31"     "2 2"        "2 2"
## [19] "1 3"        "1 5"
```

Regular expressions

- Result is a matrix but still character
- Use apply to make it numeric

```
x1 <- str_split(movies$awards_num, pattern = " ",simplify = T)
head(x1)
```

```
##      [,1] [,2] [,3]
## [1,] "4"  "8"  ""
## [2,] "7"  "17" ""
## [3,] "1"  "2"  ""
## [4,] "9"  "28" ""
## [5,] "2"  "67" ""
## [6,] "1"  ""   ""
```

Regular expressions

- We got numeric matrix using apply function
- Next we can use rowSums() to sum all wins and nominations

```
x1 <- apply(x1, 2, as.numeric)
head(x1)
```

```
##      [,1] [,2] [,3]
## [1,]    4    8   NA
## [2,]    7   17   NA
## [3,]    1    2   NA
## [4,]    9   28   NA
## [5,]    2   67   NA
## [6,]    1   NA   NA
```

Regular expressions

- As our data contains NA, use na.rm = T, which stands for remove.na
- Then assign x1 to the column awards_num in the dataframe movies

```
x1 <- rowSums(x1, na.rm=T)
x1[1:20]
```

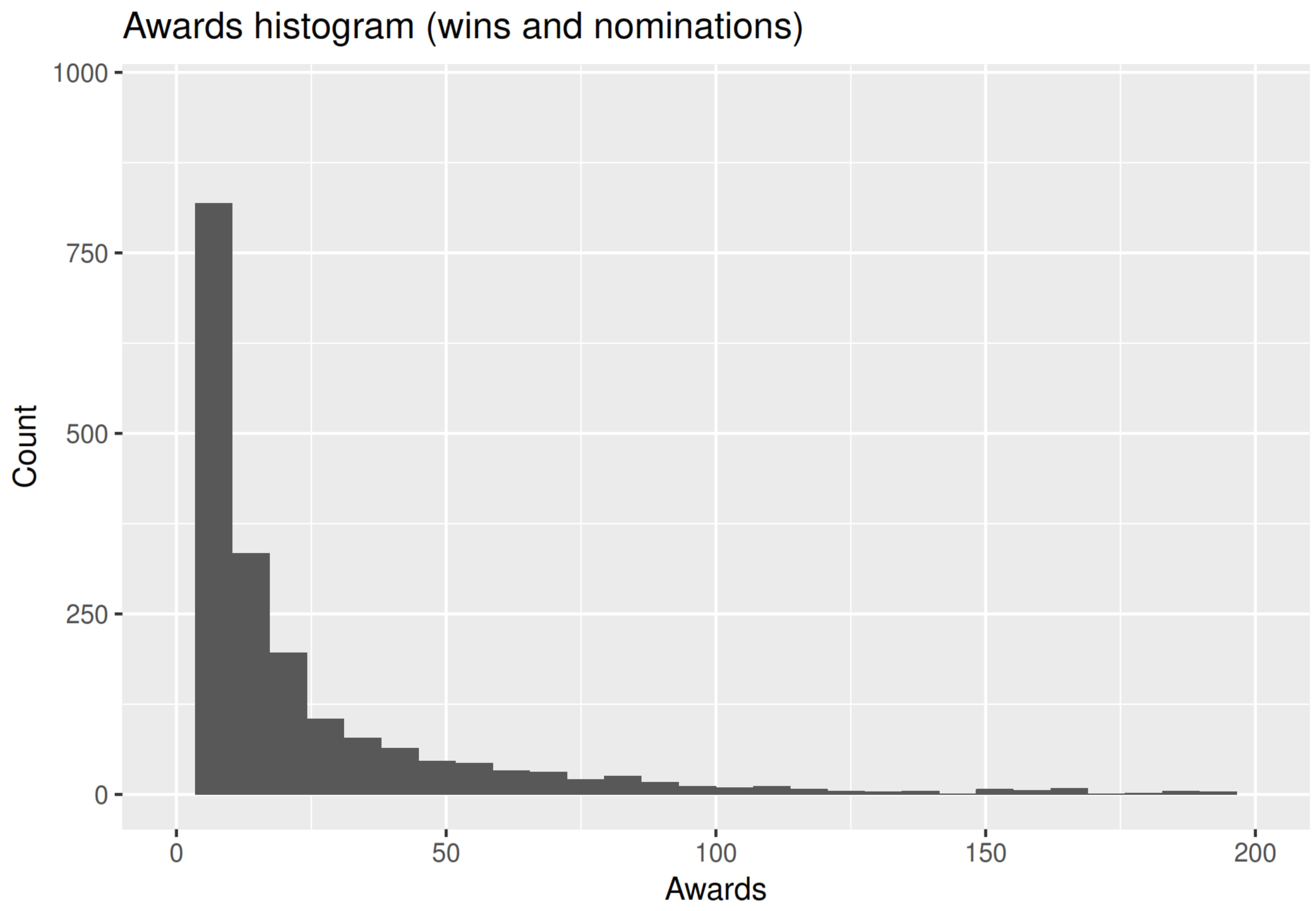
```
##  [1]  12  24   3  37  69   1 259   5   6  13   2   4   5   3  18  35   4
## [18]   4   4   6
```

```
movies$awards_num <- x1
```

Regular expressions

Create the histogram of awards

```
ggplot(movies, aes(x=awards_num))+geom_histogram()+
  xlim(c(0,200))+labs(title="Awards histogram (wins and nominations)",
                      x="Awards", y="Count")
```



Regular expressions

- Now lets try to count wins only
- Count all wins except for golden globe and Oscar

Possible strings

2 wins & 3 nominations.

1 win & 1 nomination.

Nominated for 1 Oscar. Another 1 win & 3 nominations.

Nominated for 2 Oscars. Another 11 wins & 5 nominations.

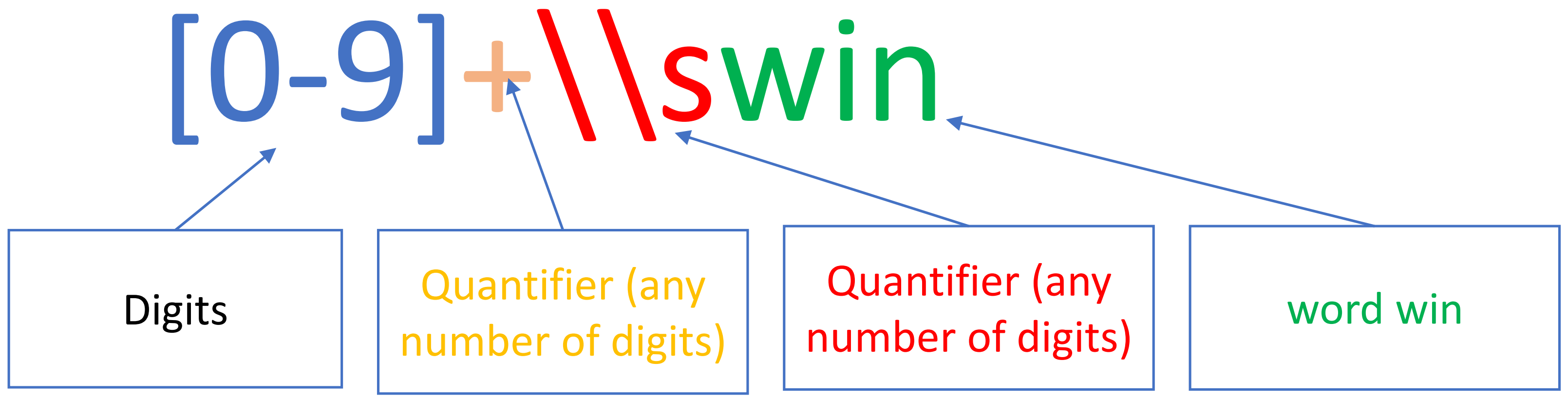
Regular expressions

Snapshot of the awards variable

```
##  [1] "4 wins & 8 nominations."
##  [2] "7 wins & 17 nominations."
##  [3] "1 win & 2 nominations."
##  [4] "9 wins & 28 nominations."
##  [5] "2 wins & 67 nominations."
##  [6] "1 nomination."
##  [7] "Won 1 Oscar. Another 87 wins & 171 nominations."
##  [8] "2 wins & 3 nominations."
##  [9] "2 wins & 4 nominations."
## [10] "Nominated for 1 Golden Globe. Another 5 wins & 7 nominations."
## [11] "1 win & 1 nomination."
## [12] "1 win & 3 nominations."
## [13] "Nominated for 1 Oscar. Another 1 win & 3 nominations."
## [14] "1 win & 2 nominations."
## [15] "Nominated for 2 Oscars. Another 11 wins & 5 nominations."
## [16] "Nominated for 1 Golden Globe. Another 3 wins & 31 nominations."
## [17] "2 wins & 2 nominations."
## [18] "2 wins & 2 nominations."
## [19] "1 win & 3 nominations."
```

Regular expressions

Lets look at the regex pattern


[0-9]+\\swin
Digits
Quantifier (any number of digits)
Quantifier (any number of digits)
word win

+ Quantifier: The preceding item will be matched 1 or more times

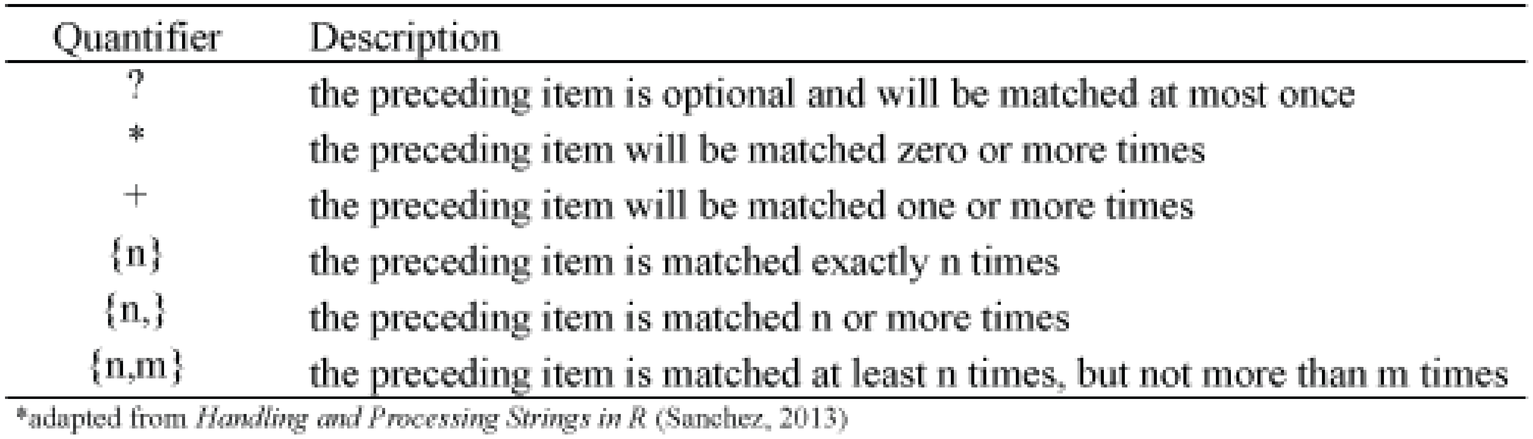
```
m1 <- str_extract_all(movies$Awards, pattern="[0-9]*\\swin", simplify=T)
m1[1:20]
```

```
##  [1] "4 win"  "7 win"  "1 win"  "9 win"  "2 win"  ""       "87 win"
##  [8] "2 win"  "2 win"  "5 win"  "1 win"  "1 win"  "1 win"  "1 win"
## [15] "11 win" "3 win"  "2 win"  "2 win"  "1 win"  "1 win"
```

Regular expressions

Quantifiers

When we want to match a **certain number** of characters that meet a certain criteria we can apply quantifiers to our pattern searches. The quantifiers we can use are:



Quantifier	Description
?	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

*adapted from *Handling and Processing Strings in R* (Sanchez, 2013)

Regular expressions

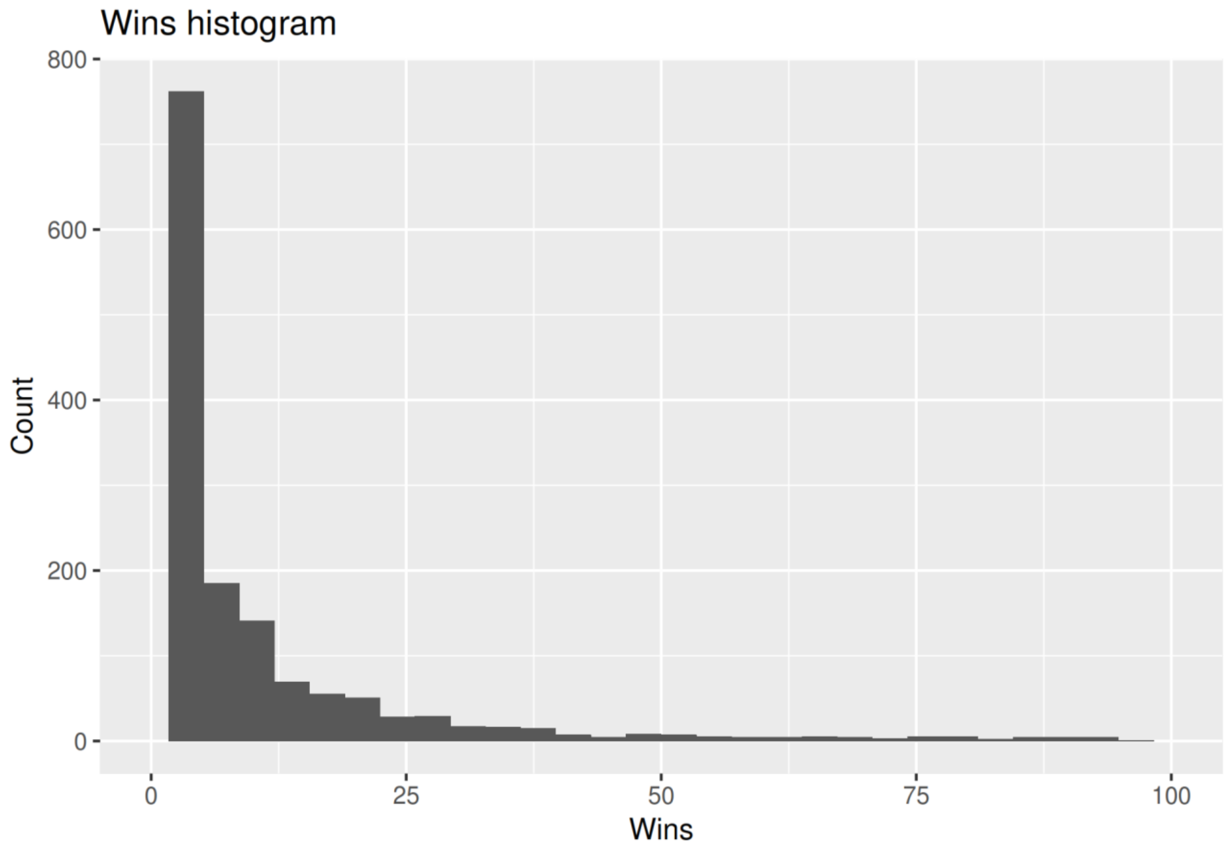
str_remove_all() is a wrapper for str_replace_all() where replacement=“”

```r
m1 <- str_remove_all(m1, pattern="[a-zA-Z]")
# No need to trim the white spaces
m1 <- as.numeric(m1)
```

Regular expressions

Add the variable to the movies data frame and create the histogram

```
movies$Wins <- m1
ggplot(movies, aes(x=Wins))+geom_histogram()+
labs(title="Wins histogram", x="Wins", y="Count")+
  xlim(c(0,100))
```



Regular expressions

Can you extract the number of the Oscar wins ?

Use dplyr to answer to the question: which movies won the biggest number of Oscars. Get the dataframe with the name of the movie and number of Oscars – Use **select()** to select the variables, **arrange()** and head()

Try different plots to visualize the new variable

Regular expressions: extracting the phone numbers

- Extract the phone numbers in one column and the names in another:
- Combine them in the dataframe

```
phones <- c("Anna 077-789663", "Hagopik 99656565",
          "Serozh2 099-65-6569 MALYAR")
```

These are random names and random numbers, don’t call them

MORE THAN AN EDUCATION - A COMMITMENT