CT5102: Programming for Data Analytics

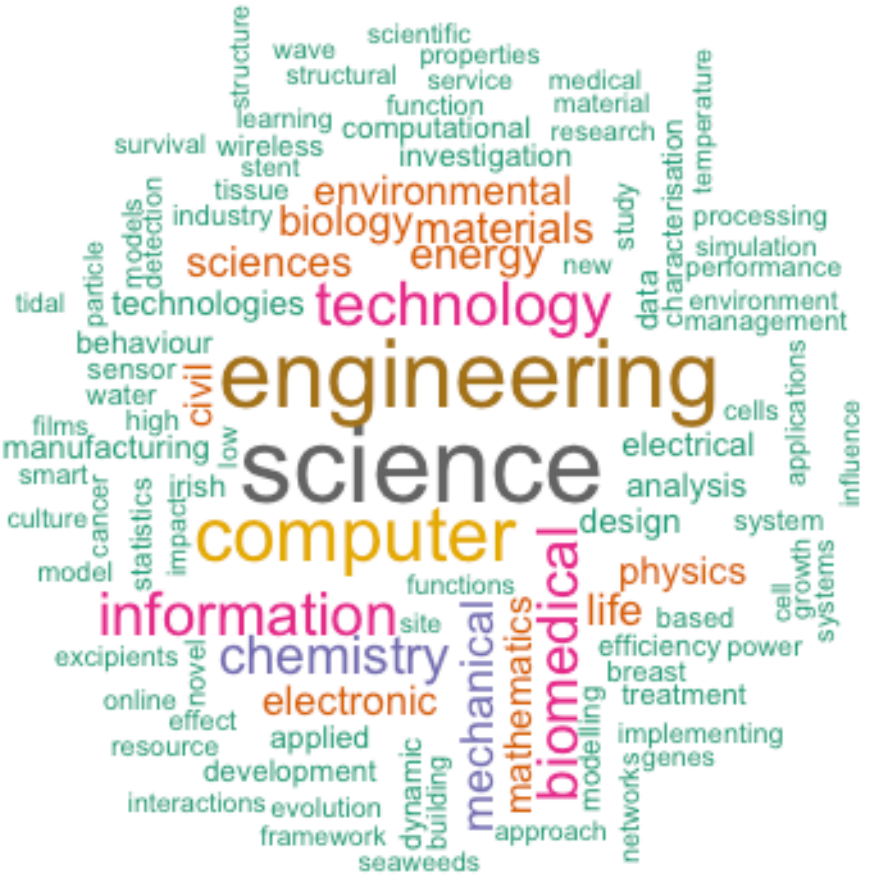
Week 6:
String Manipulation in R

https://github.com/JimDuggan/CT5102

Dr. Jim Duggan,
Information Technology,
School of Engineering & Informatics

Overview

- Character strings are important in data science
- Text mining, birth dates etc
- Coverage
  - String functions
  - Regex pattern
  - stringr package
- Ref: Matloff(2009)



grep(pattern, x)

Searches for a specified substring pattern in a vector x of strings

```
> r<-c("River Liffey","River Lee","Corrib")
> r
[1] "River Liffey" "River Lee"    "Corrib"
> i<-grep("River",r)
> i
[1] 1 2
> r[i]
[1] "River Liffey" "River Lee"
```

grepl(pattern, x)

Searches for a specified substring pattern in a vector x of strings, and returns a boolean vector

```
> r<-c("River Liffey","River Lee","Corrib")
> r
[1] "River Liffey" "River Lee"    "Corrib"
> b<-grepl("River",r)
> b
[1]  TRUE  TRUE FALSE
> r[b]
[1] "River Liffey" "River Lee"
```

nchar(x)

Finds the length of string x.

```
> nchar(r)
[1] 12  9  6
> s<-"Hello World"
> nchar(s)
[1] 11
> r
[1] "River Liffey" "River Lee"    "Corrib"
> nchar(r)
[1] 12  9  6
```

paste(...)

Concatenates several strings, returning the result in one long string.

```
> paste("CT5102", "Programming for Data Analytics", sep="-")
[1] "CT5102-Programming for Data Analytics"
```

sprintf(…)

Assembles a string from the parts in a formatted manner.

```
> id<-"12345678"
> grade<-78
> sprintf("Student %s received a grade of %d",id,grade)
[1] "Student 12345678 received a grade of 78"
```

substr(x, start, stop)

- Returns the substring in the given character position range start:stop in the given string x

```
> str<-"National University of Ireland Galway"
> substring(str,10,19)
[1] "University"
```

strsplit(x, split)

Splits a string x into an R list of substrings based on another string split in x

```
> numbers<-c("+353-76-111111","+44-112282772")
> strsplit(numbers,"-")
[[1]]
[1] "+353"    "76"      "111111"

[[2]]
[1] "+44"        "112282772"
```

sub(pattern, replacement, x)
gsub(pattern, replacement, x)

- Replace the first occurrence of a pattern with sub or replace all occurrences with gsub.
  - **pattern** – A pattern to search for, which is assumed to be a regular expression. Use an additional argument fixed=TRUE to look for a pattern without using regular expressions.
  - **replacement** – A character string to replace the occurrence (or occurrences for gsub) of pattern.
  - **x** – A character vector to search for pattern. Each element will be searched separately.

```
> sub("l","*","Hello")
[1] "He*lo"
> gsub("l","*","Hello")
[1] "He**o"
```

```
> sub("ll","*","Hello Bill")
[1] "He*o Bill"
> gsub("ll","*","Hello Bill")
[1] "He*o Bi*"
```

http://rfunction.com/archives/2354

Challenge 6.1

- Find the location of the maximum length string in a vector
- Find the location of a minimum length string in a vector
- Extract the name of the River's from the following vector. If the word "River" is not present, ignore the entry.

```
> r<-c("River Liffey","River Lee","Corrib")
> r
[1] "River Liffey" "River Lee"    "Corrib"
```

Regular Expressions

- Regular expressions essentially form a programming language for software that matches patterns in data (Chambers 2008)
- They have sets of rules that allow them to be parsed and interpreted
- R functions such as grep support the use of regular expressions

http://www.regular-expressions.info/rlanguage.html

http://regexr.com

Basic Patterns

- Single characters that match themselves. If a character is one of the special characters, it must be preceded by a backslash
- Bracket expressions, which match any of the characters between the brackets []
- Ranges can be specified [A-Z]
- The character . matches any character
- Anchors. “^” and “$” for the start and end of a string

Quantifiers

Symbol	Meaning
*	Matching zero or more repeats
+	Matching one or more repeats
?	Matching zero or one repeats
{n}	Match exactly n occurrences
{1,n}	Match from 1 to n occurrences

Examples

```
> c<-c("Galway","Mayo","Roscommon","Sligo","Leitrim")
> grep("[oi]",c)
[1] 2 3 4 5
> grep("Ga",c)
[1] 1
> grep("go",c)
[1] 4
> grep("[S-Z]",c)
[1] 4
```

```
> t<-c("AA001BB","AAAAA","AA1AA","BBB111BBB","CCC11C11")
> grep("[0-9]{3}",t)
[1] 1 4
> grep("[0-9]{2,3}",t)
[1] 1 4 5
> grep("[0-9]*",t)
[1] 1 2 3 4 5
> grep("B{3}",t)
[1] 4
> grep("B{3}[0-9]{3}",t)
[1] 4
> grep("B{3}|C{3}[0-9]{2,3}",t)
[1] 4 5
>
> grep("[[:digit:]]{3}",t)
[1] 1 4
> grep("[[:alpha:]]{5}",t)
[1] 2
```

Challenge 6.2

- Assume a phone number has the following correct formats
  - 08N-DDDDDDD
  - 08N.DDDDDDD

- Where N = {5,6,7} and D = {0,1,2,…,9}
- Write a function that validates a phone number, or a vector of phone numbers

Solution

```
vp<-function(v){
  # Need to escape . with \\
  p<-"^08[5-7](-|\\.)[[:digit:]]{7}$"
  b<-grepl(p,v)
  list(input=v,output=b)
}
```

Testing the function

```
test<-c("085*1234567","085.1234567","085-1234567",
        "084-1234567","085-234567","091.1234567")

result<-vp(test)

expected<-c(F,T,T,F,F,F)

passed<-all(expected == result$output)

if(passed)
  cat("All tests passed\n") else
  cat("Test suite did not pass...")
```

Output

```
> result<-vp(test)
> result
$input
[1] "085*1234567" "085.1234567" "085-1234567" "084-1234567" "085-234567"
"091.1234567"

$output
[1] FALSE  TRUE  TRUE FALSE FALSE FALSE
```

Additional Rules

- By default repetition is greedy, so the maximal possible number of repeats is used. This can be changed to 'minimal' by appending ? to the quantifier. (There are further quantifiers that allow approximate matching: see the TRE documentation.)
- Regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating the substrings that match the concatenated subexpressions.
- Two regular expressions may be joined by the infix operator |; the resulting regular expression matches any string matching either subexpression. For example, abba|cde matches either the string abba or the string cde. Note that alternation does not work inside character classes, where | has its literal meaning.
- Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole subexpression may be enclosed in parentheses to override these precedence rules.

Character Classes

- Important for making regular expressions independent of locale, and should be used instead of ranges in bracket expressions whenever possible (Chambers 2008)
- Letters need not be restricted to 52 characters [a-zA-Z]
- Examples of predefined character classes:

  [:alpha:] and [:alnum:]

IP Address (nnn.nnn.nnn.nnn)

```
> pattern<-"^([[:digit:]]{1,3}\\.){3}[[:digit:]]{1,3}$"
> pattern
[1] "^([[:digit:]]{1,3}\\.){3}[[:digit:]]{1,3}$"
> grepl(pattern,"1.1.1.1")
[1] TRUE
> grepl(pattern,"1.1.1.1.1")
[1] FALSE
> grepl(pattern,"1.1111.1.1.1")
[1] FALSE
> grepl(pattern,"111.1.1.000")
[1] TRUE
> grepl(pattern,"111.1*1.000")
[1] FALSE
```

List of character classes

Character Class	Description
[:alnum:]	Alphanumeric characters: [:alpha:] and [:digit:].
[:alpha:]	Alphabetic characters: [:lower:] and [:upper:].
[:blank:]	Blank characters: space and tab, and possibly other locale-dependent characters such as non-breaking space.
[:cntrl:]	Control characters. In ASCII, these characters have octal codes 000 through 037, and 177 (DEL). In another character set, these are the equivalent characters, if any.
[:graph:]	Graphical characters: [:alnum:] and [:punct:].
[:digit:]	Digits: 0 1 2 3 4 5 6 7 8 9

List of character classes

Character Class	Description
[:lower:]	Lower-case letters in the current locale.
[:print:]	Printable characters: [:alnum:], [:punct:] and space.
[:punct:]	Punctuation characters: ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~.
[:space:]	Space characters: tab, newline, vertical tab, form feed, carriage return, space and possibly other locale-dependent characters.
[:upper:]	Upper-case letters in the current locale.
[:xdigit:]	Hexadecimal digits:

Challenge 6.3

- The definition of a name in the R version of the S language is any string consisting of letters, numbers, and the two characters "." and "_", with the first character being a letter or "."
- Write a pattern expression using character classes
- Wrap this in a function

Pattern

```
> pattern<-"^[.[:alpha:]][._[:alnum:]]*$"
> pattern
[1] "^[.[:alpha:]][._[:alnum:]]*$"
> grepl(pattern,"test")
[1] TRUE
> grepl(pattern,".")
[1] TRUE
> grepl(pattern,".test123-")
[1] FALSE
> grepl(pattern,".test123_")
[1] TRUE
```

stringr package (Wickham 2010)

- Processes factors and characters in the same way
- Gives functions consistent names and arguments
- Produces outputs that can be easily used as inputs
- Two families of functions:
  - Basic string operations
  - Pattern matching that use regular expressions to detect, locate, match, replace, extract and split strings

Basic String Operations

- **str_c()**, equivalent to paste, but uses the empty string “” as the default separator
- **str_length()**, equivalent to nchar, preserves NA’s and converts factors to characters
- **str_sub()**, equivalent to substr
- **str_dup()**, duplicates characters within a string
- **str_trim()**, removes leading and trailing whitespaces
- **str_pad()**, pads a string with extra whitespace on the left, right or both sides.

Pattern Matching

Function	Description
str_detect()	Detects the presence or absence of a pattern and returns a logical vector. Based on **grepl**.
str_locate()	Locates the first position of a pattern and returns a numeric matric with columns start and end. str_locate_all locates all matches, returning a list of numeric matrices. Based on regexpr and gregexpr
str_extract()	Extracts text corresponding to the first match, returning a character vector. str_extract_all() returns a list of character vectors
str_match()	Extracts capture groups formed from the first match. str_match_all() extracts capture groups from all matches
str_replace()	Replaces the first matched pattern and returns a character vector. str_replace_all() replaces all matches
str_split_fixed()	Splits the string into a fixed number of pieces based on a pattern and returns a character matrix.

Example

```
> strings <- c(" 219 733 8965", "329-293-8753 ", "banana", "595 794
7569", "387 287 6718", "apple", "233.398.9187 ", "482 952 3315","239
923 8115","842 566 4692", "Work: 579-499-7527", "$1000", "Home: 543.
355.3679")
> strings
 [1] " 219 733 8965"      "329-293-8753 "      "banana"
"595 794 7569"       "387 287 6718"
 [6] "apple"              "233.398.9187 "      "482 952 3315"
"239 923 8115"       "842 566 4692"
[11] "Work: 579-499-7527" "$1000"              "Home: 543.355.3679"
> phone <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
> phone
[1] "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
```

str_detect()

```
> str_detect(strings, phone)
 [1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TR
UE FALSE  TRUE
> strings[str_detect(strings, phone)]
 [1] " 219 733 8965"      "329-293-8753 "      "595 794 7569"
"387 287 6718"       "233.398.9187 "
 [6] "482 952 3315"       "239 923 8115"       "842 566 4692"
"Work: 579-499-7527" "Home: 543.355.3679"
```

str_replace()

```
> str_replace(strings, phone, "XXX-XXX-XXXX")
 [1] " XXX-XXX-XXXX"       "XXX-XXX-XXXX "       "banana"
 [4] "XXX-XXX-XXXX"        "XXX-XXX-XXXX"        "apple"
 [7] "XXX-XXX-XXXX "       "XXX-XXX-XXXX"        "XXX-XXX-XXXX"
[10] "XXX-XXX-XXXX"        "Work: XXX-XXX-XXXX" "$1000"
[13] "Home: XXX-XXX-XXXX"
```

str_match()

```
> str_match(strings,phone)
      [,1]             [,2]  [,3]  [,4]
 [1,] "219 733 8965"   "219" "733" "8965"
 [2,] "329-293-8753"   "329" "293" "8753"
 [3,] NA               NA    NA    NA
 [4,] "595 794 7569"   "595" "794" "7569"
 [5,] "387 287 6718"   "387" "287" "6718"
 [6,] NA               NA    NA    NA
 [7,] "233.398.9187"   "233" "398" "9187"
 [8,] "482 952 3315"   "482" "952" "3315"
 [9,] "239 923 8115"   "239" "923" "8115"
[10,] "842 566 4692"   "842" "566" "4692"
[11,] "579-499-7527"   "579" "499" "7527"
[12,] NA               NA    NA    NA
[13,] "543.355.3679"   "543" "355" "3679"
```

Challenge 6.4

- Consider the following vector of phone numbers in **list1**. Perform the following transformations using the stringr package, only using the **list1** variable.
  - Convert the numbers to international format, which involves dropping the leading 0
  - For example, 087 → +353-87
  - Make all 085 numbers anonymous

```
> plist<-sample(c("087","086","085"),100,replace=T)
> head(plist)
[1] "087" "086" "087" "085" "087" "087"
> nums<-sample(1111111:9999999,100)
> head(nums)
[1] 6881792 8487119 2189813 5184062 5005676 9399138
> list1<-str_c(plist,nums,sep="-")
> head(list1)
[1] "087-6881792" "086-8487119" "087-2189813" "085-5184062"
[5] "087-5005676" "087-9399138"
```

```
> plist<-sample(c("087","086","085"),100,replace=T)
> plist[1:5]
[1] "087" "087" "085" "086" "086"
> nums<-sample(1111111:9999999,100)
> nums[1:5]
[1] 8062084 3040982 2962993 5158081 2801572
> list1<-str_c(plist,nums,sep="-")
> list1[1:5]
[1] "087-8062084" "087-3040982" "085-2962993" "086-5158081" "086-2801572"
>
> new<-str_c("+353-",
+            str_sub(list1, start=2, end=3),
+            str_sub(list1,start=4,end=length(plist)))
> new[1:5]
[1] "+353-87-8062084" "+353-87-3040982" "+353-85-2962993" "+353-86-5158081" "+353-86-28
01572"
>
>
> anon<-str_replace_all(new,"\\+353(-)85(-)[[:digit:]]{7}","+XXX-XX-XXXXXXX")
> anon[1:5]
[1] "+353-87-8062084" "+353-87-3040982" "+XXX-XX-XXXXXXX" "+353-86-5158081" "+353-86-28
01572"
>
```

Assignment 6

```
Hours
PT Contact
Hours
ST1100 Engineering Statistics (Approved) 5 0.00 0.00
MA385 Numerical Analysis I (Approved) 8 5 3.50 0.00
CT561 Systems Modelling and Simulation (Awaiting Processing) 8 5 2.00 0.00
```

The goal is to write string matching routines to extract information on course codes, descriptions and the semester they are taken in. The only input that can be used is that the value subject code types are: ("ST", "MA", "CT", "EE", "DER"). A matching course code has anything from 3 to four digits after the subject letters have been identified.

Course Code	Description	Semester
ST1100	Engineering Statistics	One
MA385	Numerical Analysis I	One
CT561	Systems Modelling and Simulation	One
CT475	Machine Learning & Data Mining	One