Lab 7

Junyu Wang

October 13th, 2015

Regular Expressions

Preliminaries

First, load the data.

```
library("readr")

flags <-
read_csv("https://raw.githubusercontent.com/gastonstat/stat133/master/datasets/flags.csv")
```

We'll be working only with the first column (the `name` column), which contains the names of countries.

```
countries <- flags$name

head(countries)
```

```
## [1] "Afghanistan"    "Albania"        "Algeria"        "American-Samoa"
## [5] "Andorra"        "Angola"
```

You may have noticed that in the `countries` vector, words in a country's name are separated by `-`. Let's replace them with spaces, so that for example `American-Samoa` becomes `American Samoa`. (More on this later.)

```
# The syntax is gsub(pattern, replacement, x).
countries <- gsub("-", " ", countries)

# We also could have done chartr("-", " ", countries).

head(countries)
```

```
## [1] "Afghanistan"    "Albania"        "Algeria"        "American Samoa"
## [5] "Andorra"        "Angola"
```

grep

grep is the most basic regex function: it simply returns which elements of a vector match a particular pattern. For example, suppose we wanted to know which countries have the word `islands` in their name.

```
grep("islands", countries)
```

```
## integer(0)
```

It would appear that there are none. But that's not actually true.

```
countries[33]
```

```
## [1] "Cape Verde Islands"
```

So grep failed to match the pattern `islands` to the word `Islands`. There are a couple of ways to fix that.

```
grep("Islands", countries)
```

```
## [1]  33  34  40  42 107 152 177
```

```
# Or
grep("islands", countries, ignore.case = TRUE)
```

```
## [1]  33  34  40  42 107 152 177
```

The second way has the advantage of being able to match both `islands` and `Islands`, if necessary, as well as `IsLands`, etc.

You can also view the country names instead of their indices.

```
grep("islands", countries, ignore.case = TRUE, value = TRUE)
```

```
## [1] "Cape Verde Islands"  "Cayman Islands"      "Comorro Islands"
## [4] "Cook Islands"        "Maldive Islands"     "Soloman Islands"
## [7] "Turks Cocos Islands"
```

Your turn! Use grep to find which countries have more than one word in their name. (Hint: What character is present in that case?)

```
# Fill in.
grep(" ", countries, value = TRUE)
```

```
##  [1] "American Samoa"           "Antigua Barbuda"
##  [3] "British Virgin Isles"     "Cape Verde Islands"
##  [5] "Cayman Islands"           "Central African Republic"
##  [7] "Comorro Islands"          "Cook Islands"
##  [9] "Costa Rica"               "Dominican Republic"
## [11] "El Salvador"              "Equatorial Guinea"
## [13] "Falklands Malvinas"       "French Guiana"
## [15] "French Polynesia"         "Germany DDR"
## [17] "Germany FRG"              "Guinea Bissau"
## [19] "Hong Kong"                "Ivory Coast"
## [21] "Maldive Islands"          "Netherlands Antilles"
## [23] "New Zealand"              "North Korea"
## [25] "North Yemen"              "Papua New Guinea"
## [27] "Puerto Rico"              "San Marino"
## [29] "Sao Tome"                 "Saudi Arabia"
## [31] "Sierra Leone"             "Soloman Islands"
## [33] "South Africa"             "South Korea"
```

```
## [35] "South Yemen"              "Sri Lanka"
## [37] "St Helena"                "St Kitts Nevis"
## [39] "St Lucia"                 "St Vincent"
## [41] "Trinidad Tobago"          "Turks Cocos Islands"
## [43] "US Virgin Isles"          "Vatican City"
## [45] "Western Samoa"
```

grep Variants

There are several variants of grep suitable for other types of string manipulation.

grepl is identical to grep, but returns a corresponding logical vector instead of a numeric vector.

regexpr and gregexpr are useful for finding the particular location of a match. regexpr gives the index of the first match (if any) in each element of a vector, while gregexpr gives the indices of all matches and returns a list.

```
regexpr("an", head(countries), ignore.case = TRUE)
```

```
## [1]  5  4 -1  7  1  1
## attr(,"match.length")
## [1]  2  2 -1  2  2  2
## attr(,"useBytes")
## [1] TRUE
```

```
gregexpr("an", head(countries), ignore.case = TRUE)
```

```
## [[1]]
## [1]  5 10
## attr(,"match.length")
## [1] 2 2
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1] 4
## attr(,"match.length")
## [1] 2
## attr(,"useBytes")
## [1] TRUE
##
## [[3]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"useBytes")
## [1] TRUE
##
## [[4]]
## [1] 7
```

```
## attr(,"match.length")
## [1] 2
## attr(,"useBytes")
## [1] TRUE
##
## [[5]]
## [1] 1
## attr(,"match.length")
## [1] 2
## attr(,"useBytes")
## [1] TRUE
##
## [[6]]
## [1] 1
## attr(,"match.length")
## [1] 2
## attr(,"useBytes")
## [1] TRUE
```

sub and gsub are useful for replacing matches. sub replaces the first match (if any) in each element of a vector, while gsub replaces all matches.

```
sub("an", "BANANA", head(countries), ignore.case = TRUE)
```

```
## [1] "AfghBANANAistan"    "AlbBANANAia"        "Algeria"
## [4] "AmericBANANA Samoa" "BANANAdorra"        "BANANAgola"
```

```
gsub("an", "BANANA", head(countries), ignore.case = TRUE)
```

```
## [1] "AfghBANANAistBANANA" "AlbBANANAia"         "Algeria"
## [4] "AmericBANANA Samoa"  "BANANAdorra"         "BANANAgola"
```

Your turn! Replace all instances of `us` (or `US`, etc.) with `United States`, but only output the countries whose names were changed. (Hint: How can you find which countries contain `us`?)

```
# Fill in.
gsub("us", "United States", grep("us", countries, ignore.case = TRUE, value =
TRUE), ignore.case = TRUE)
```

```
## [1] "AUnited Statestralia"       "AUnited Statestria"
## [3] "CyprUnited States"          "MauritiUnited States"
## [5] "United States Virgin Isles" "United StatesA"
## [7] "United StatesSR"
```

Metacharacters

So far we only to know how to match literal strings, except for ignoring case. What if we wanted to know which countries' names start with `A`? The metacharacter `^` denotes the beginning of the string.

```
grep("^A", countries, value = TRUE)
```

```
##  [1] "Afghanistan"     "Albania"          "Algeria"
##  [4] "American Samoa"  "Andorra"          "Angola"
##  [7] "Anguilla"        "Antigua Barbuda" "Argentina"
## [10] "Argentine"       "Australia"        "Austria"
```

Similarly, the metacharacter `$` denotes the end of the string.

Your turn! Find which countries' names end in `stan`.

```
# Fill in.
grep("stan$", countries, value = TRUE)
```

```
## [1] "Afghanistan" "Pakistan"
```

You may have noticed that metacharacters are not treated literally. If you want to match a metacharacter, make sure to precede it by `\\`, as in `\\^'. If you want to match`, use `\\\\`.

```
# Switch dollar sign to letter s:
word <- "Bo$$"
gsub("$", "s", word) #wrong
```

```
## [1] "Bo$$s"
```

```
gsub("\\$", "s", word) #correct
```

```
## [1] "Boss"
```

Suppose you wanted to know which countries' names start with `A` and whose third letter is g, but you didn't care about the second letter? The metacharacter `.` represents any character.

```
grep("^A.g", countries, value = TRUE)
```

```
## [1] "Afghanistan" "Algeria"     "Angola"      "Anguilla"    "Argentina"
## [6] "Argentine"
```

The metacharacters `?`, `*`, and `+` modify the previous item: `?` matches zero or one times, `*` matches any number of times (including zero), and `+` matches at least once. So `.*` matches any character string, while `.+` matches any non-empty character string.

Your turn! Find which countries' names start with `S` and have an `l` in their name.

```
# Fill in.
grep("^S.*l", countries, value = TRUE)
```

```
## [1] "Senegal"         "Seychelles"      "Soloman Islands" "Somalia"
## [5] "St Helena"       "Swaziland"       "Switzerland"
```

Character Classes

What if you wanted to know which countries' names start with `A` or `Z`? The metacharacter `|` functions as a logical or. Left and right parentheses (`(` and `)`) are also metacharacters used for grouping.

```
grep("^(A|Z)", countries, value = TRUE)
```

```
##  [1] "Afghanistan"     "Albania"         "Algeria"
##  [4] "American Samoa"  "Andorra"         "Angola"
##  [7] "Anguilla"        "Antigua Barbuda" "Argentina"
## [10] "Argentine"       "Australia"       "Austria"
## [13] "Zaire"           "Zambia"          "Zimbabwe"
```

If you want to do this for, say, any vowel, you would have to write `(A|E|I|O|U)`, for example. Character classes simplify this: the character class for the preceding expression is just `[AEIOU]`. The previous code chunk can be equivalently written as follows:

```
grep("^[AZ]", countries, value = TRUE)
```

```
##  [1] "Afghanistan"     "Albania"         "Algeria"
##  [4] "American Samoa"  "Andorra"         "Angola"
##  [7] "Anguilla"        "Antigua Barbuda" "Argentina"
## [10] "Argentine"       "Australia"       "Austria"
## [13] "Zaire"           "Zambia"          "Zimbabwe"
```

Several character classes are predefined in R. For example, `[:digits:]` is equivalent to `0123456789`, and `[:lower:]` is equivalent to `abcdefghijklmnopqrstuvwxyz`. So the character class `[0123456789]` can be more easily written as `[[:digits:]]`, etc. See the regex documentation for R for a more extensive list of predefined character classes.

Your turn! Find which countries' names start with two or more vowels.

```
# Fill in.
grep("^[AEIOU]{2,}", countries, value = TRUE, ignore.case = TRUE)
```

```
## [1] "Australia" "Austria"   "UAE"
```

Finally, when inside of a character class, `^` denotes negation, so `[^AZ]` matches any character except `A` or `Z`.

Your turn! Find which countries' names start with a vowel and end in a consonant.

```
# Fill in.
grep("^[AEIOU].*[^AEIOU]$", countries, value = TRUE, ignore.case = TRUE)
```

```
##  [1] "Afghanistan"     "Ecuador"         "Egypt"
##  [4] "El Salvador"     "Iceland"         "Iran"
##  [7] "Iraq"            "Ireland"         "Israel"
## [10] "Italy"           "Ivory Coast"     "Oman"
## [13] "UK"              "Uruguay"         "US Virgin Isles"
## [16] "USSR"
```