Master Basic R Regular Expression in 15 minutes

Allen Tsung-Tse Chen

October 30, 2017

Title: Master Basic R Regular Expression in 15 minutes

Description: This post introduces the basics of using regular expression in R.

Introduction:

In simple terms, regular expression is a special formated text string used for finding, describing and matching patterns in a string of text. The following bullet points highlight some of the important uses:

1. Replace different words that represent the same meaning with the same word. For example, We might want to replace “UC Berkeley”, “UCBerkeley”, “UCB”, “CAL”, “Cal”, “Golden Bears” with “UCB”
2. Find the nine digits phone number in a string of text that have different formats. For example, we want to find “1234567890” from “123-456-7890”, “123.456.7890” or “(123)-456-7890”.
3. Check if a date follows a specific format. For example, we want the date format to be “mm-dd-yyyy”, and we would return false if the strings are “mm-dd-yy” or “mm/dd/yyyy”.
4. Convert a comma-delimited string into a space-delimited string For example, we might want to change “fish & tacos, is, delicious” into “fish tacos is delicious”.

Along with the above applications, regular expressions can basically perform all types of text search and text replace operations, and it can be seen as a broader version of string operations. Sometimes, one has to be extra careful because string operations might be a better choice to parse or manipulate if one is given a much cleaner or standardize data set.

Therefore, to summarize the motivation of this post, one will have a general idea and knowledge on how regular expression and various functions associated with it work after reading through this post. Most programming languages use regular expression, so this post will also help one’s future learning of other languages. Lastly, the abbreviation for regular expression is regex.

Data and libraries:

In the future examples, we will be using two data sets.

nba_data is taken from class assignments. This data contains the statistics for 441 NBA players

food_data is taken from Kaggle data set. This data contains the ingredients for 10000 (processed) food with its brand, categories, manufacturer, name, and others.

The data set can be downloaded from https://www.kaggle.com/datafiniti/food-ingredient-lists/data

```
library(ggplot2)
library(dplyr)
```

```
nba_data <- read.csv("../data/nba2017-player-statistics.csv", stringsAsFactors = FALSE)
nba_data$Team <- as.factor(nba_data$Team)
head(nba_data)
```

```
##              Player Team Position Experience   Salary Rank Age GP GS  MIN
## 1        Al Horford  BOS        C          9 26540100    4  30 68 68 2193
## 2      Amir Johnson  BOS       PF         11 12000000    6  29 80 77 1608
## 3     Avery Bradley  BOS       SG          6  8269663    5  26 55 55 1835
## 4 Demetrius Jackson  BOS       PG          R  1450000   15  22  5  0   17
## 5       Gerald Green  BOS       SF          9  1410598   11  31 47  0  538
## 6     Isaiah Thomas  BOS       PG          5  6587132    1  27 76 76 2569
##   FGM  FGA Points3 Points3_atts Points2 Points2_atts FTM FTA OREB DREB AST
## 1 379  801      86          242     293          559 108 135   95  369 337
## 2 213  370      27           66     186          304  67 100  117  248 140
## 3 359  775     108          277     251          498  68  93   65  269 121
## 4   3    4       1            1       2            3   3   6    2    2   3
## 5  95  232      39          111      56          121  33  41   17   68  33
## 6 682 1473     245          646     437          827 590 649   43  162 449
##   STL BLK  TO
## 1  52  87 116
## 2  52  62  77
## 3  68  11  88
## 4   0   0   0
## 5   9   7  25
## 6  70  13 210
```

```
food_data <- read.csv("../data/ingredients.csv", stringsAsFactors = FALSE)
names(food_data)[3] <- "ingredients"
# Bad format due to long text strings, but please look for column names.
# The column names are "brand", "categories", "ingredients", "manufacturer", "name".
head(food_data)
```

```
##           brand                          categories
## 1 Simon Fischer Grocery & Gourmet Food,Food,Grocery
## 2     McCormick Grocery & Gourmet Food,Food,Grocery
## 3    Jolly Time      Grocery & Gourmet Food,Grocery
## 4         Ziyad      Grocery & Gourmet Food,grocery
## 5   Fla-Vor-Ice      Grocery & Gourmet Food,grocery
## 6          Hero          Food,Other Grocery,Grocery
##
ingredients
## 1
Dried Prunes,Water,Corn Syrup,Sugar,Pectin.
## 2 Salt,Sugar,Molasses (Refinery Syrup, Molasses, Caramel Color),Spices (Including Black Pepper),Garlic Onion,Ta
pioca Maltodextrin,Bacon Fat and Cooked Bacon (Cured with Water, Salt, Sodium Erythorbate, Sodium Nitrate),Silicon
Dioxide (To Make Free Flowing),Autolyzed Yeast,Sunflower Oil,Corn Maltodextrin,Vinegar,Extractives of Paprika,and
Natural Flavor (Including Smoke)
## 3
Salt, Yellow 5 Lake, Tricalcium Phosphate And Artificial Butter Flavor
## 4
Mechanically hulled seasame seeds.Allergy Information: Packed in a facility that processes wheat, flour, peanuts a
nd tree nuts.,Mechanically hulled seasame seeds.Allergy Information: Packed in a facility that processes wheat,flo
ur,peanuts and tree nuts.
## 5
FALSE
## 6
Red Raspberries,Sugar,Glucose Syrup,Citric Acid,Pectin. Contains: Wheat.
##          manufacturer
## 1   Sokol And Company
## 2 McCormick & Co, Inc
## 3             Reese's
## 4               Ziyad
## 5         Fla-Vor-Ice
## 6          HERO, INC.
##                                                                      name
## 1                                    Simon Fischer Fruit Bttr Prune Lekvar
## 2 McCORMICK GRILL MATES MOLASSES BACON SEASONING 1 x 77g JAR AMERICAN IMPORT
## 3                                                        Jolly Time Popcorn
## 4                                                  Ziyad Tahini Sesame Sauce
## 5                                                Fla-Vor-Ice Plus Giant Pops
## 6                             Hero Fruit Sprd Blk Currant-12 Oz -pack of 8
```

Inspiration:

The inspiration of this post comes from Professor Sanchez's book on Handling Strings in R. I recommend everyone who wants a solid foundation in Regex should read his Regex section in his book first. A short example will be shown below.

Professor Sanchez's book "HandlingStrings in R": http://www.gastonsanchez.com/r4strings/regex1.html

```
# importing here to show emphasis
library("stringr")
categories5 <- slice(food_data, 1:6)
#str_view(categories5$categories, 'Grocery')
```

The result is hidden because the widget is not supported. In this example, we used the package "stringr" to call on the function str_view to give a visual representation of the pattern that is matched. We can see "Grocery" can be found in categories of all six food.

Learn Regex and useful functions in Base R

The most useful function in Regex is the grep() function. This function detects patterns within a string, and returns a character vector containing the selected elemtns of the input vector.

```
strawberry_food <- grep(pattern = "Strawberry", x = food_data$ingredients, value = TRUE)
strawberry_food[1]
```

```
## [1] "Sugar,Corn Syrup,Modified Food Starch,Contains 2% or Less of the Following: Peach Puree Concentrate,Bluebe
rry Puree,Strawberry Puree,Lemon Puree,Pear Juice Concentrate,Tangerine Juice Concentrate,Watermelon Juice Concent
rate,Apple Juice Concentrate,Cherry Juice Concentrate,Pomegranate Juice Concentrate,Coconut,Citric Acid,Phosphoric
Acid,Ascorbic Acid,Sodium Lactate,Sodium Citrate,Natural and Artificial Flavors,Color Added,Red 40 Lake,Blue 1 & 2
Lake,Yellow 5 & 6 Lake,Yellow 5 & 6,Red 40,Blue 1,Tapioca Dextrin,Beeswax,Carnauba Wax,Confectioner's Glaze,Salt,C
affeine."
```

```
length(strawberry_food)
```

```
## [1] 131
```

```
strawberry_food1 <- grep(pattern = "Strawberry", x = food_data$ingredients, value = FALSE)
head(strawberry_food1)
```

```
## [1]  47  49 105 208 212 308
```

```
food_data$name[47]
```

```
## [1] "Jelly Belly Jelly Bean"
```

From this example, we can see the regex function grep allows us to find “Strawberry” quickly in a huge list of ingredients. Unfortunately, Jelly Belly Jelly Bean doesn’t contain real Strawberry. If the parameter value is TRUE, then it returns a character vector. If False, it returns indices where pattern is found.

Next, let’s look at pattern locating functions!

```
freq <- regexpr("Water",food_data$ingredients)
freq[1:3]
```

```
## [1]  14 168  -1
```

```
food_data$ingredients[1]
```

```
## [1] "Dried Prunes,Water,Corn Syrup,Sugar,Pectin."
```

regexpr() searches for matches to “Water”, and returns the starting position of the found pattern. If the pattern is not found, then -1 is returned.

There is also sub() function, which replaces the first match, and gsub replaces all matches

```
food_data$ingredients[47] #original ingredient
```

```
## [1] "Sugar,Corn Syrup,Modified Food Starch,Contains 2% or Less of the Following: Peach Puree Concentrate,Bluebe
rry Puree,Strawberry Puree,Lemon Puree,Pear Juice Concentrate,Tangerine Juice Concentrate,Watermelon Juice Concent
rate,Apple Juice Concentrate,Cherry Juice Concentrate,Pomegranate Juice Concentrate,Coconut,Citric Acid,Phosphoric
Acid,Ascorbic Acid,Sodium Lactate,Sodium Citrate,Natural and Artificial Flavors,Color Added,Red 40 Lake,Blue 1 & 2
Lake,Yellow 5 & 6 Lake,Yellow 5 & 6,Red 40,Blue 1,Tapioca Dextrin,Beeswax,Carnauba Wax,Confectioner's Glaze,Salt,C
affeine."
```

```
banana <- sub("Strawberry", "Banana", food_data$ingredients)
banana[47]  #Strawberry is replaced by Banana
```

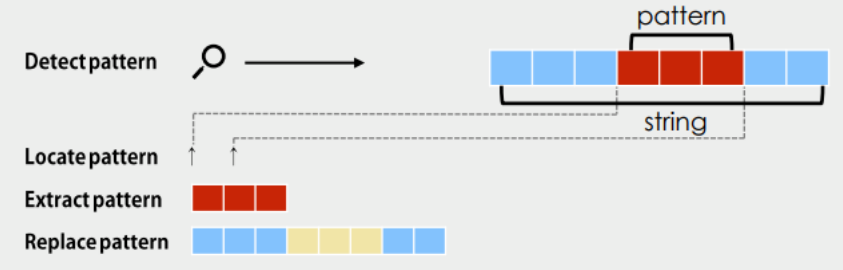
```
## [1] "Sugar,Corn Syrup,Modified Food Starch,Contains 2% or Less of the Following: Peach Puree Concentrate,Bluebe
rry Puree,Banana Puree,Lemon Puree,Pear Juice Concentrate,Tangerine Juice Concentrate,Watermelon Juice Concentrate
,Apple Juice Concentrate,Cherry Juice Concentrate,Pomegranate Juice Concentrate,Coconut,Citric Acid,Phosphoric Aci
d,Ascorbic Acid,Sodium Lactate,Sodium Citrate,Natural and Artificial Flavors,Color Added,Red 40 Lake,Blue 1 & 2 La
ke,Yellow 5 & 6 Lake,Yellow 5 & 6,Red 40,Blue 1,Tapioca Dextrin,Beeswax,Carnauba Wax,Confectioner's Glaze,Salt,Caf
feine."
```

```
sb <- gsub("Strawberry", "Secret_Recipe", food_data$ingredients)
sb[47] #first occurence
```

```
## [1] "Sugar,Corn Syrup,Modified Food Starch,Contains 2% or Less of the Following: Peach Puree Concentrate,Bluebe
rry Puree,Secret_Recipe Puree,Lemon Puree,Pear Juice Concentrate,Tangerine Juice Concentrate,Watermelon Juice Conc
entrate,Apple Juice Concentrate,Cherry Juice Concentrate,Pomegranate Juice Concentrate,Coconut,Citric Acid,Phospho
ric Acid,Ascorbic Acid,Sodium Lactate,Sodium Citrate,Natural and Artificial Flavors,Color Added,Red 40 Lake,Blue 1
& 2 Lake,Yellow 5 & 6 Lake,Yellow 5 & 6,Red 40,Blue 1,Tapioca Dextrin,Beeswax,Carnauba Wax,Confectioner's Glaze,Sa
lt,Caffeine."
```

```
sb[105] #third occurence
```

```
## [1] "Sugar,Corn Syrup,Modified Food Starch,Contains 2 or Less of the Following: Peach Puree Concentrate,Raspber
ry Puree,Secret_Recipe Juice Concentrate,Apple Juice Concentrate,Cherry Juice Concentrate,Watermelon Juice Concent
rate,Lime Juice Concentrate,Blueberry Puree,Lemon Puree,Orange Puree,Citric Acid,Fumaric Acid,Lactic Acid,Sodium C
itrate,Sodium Lactate,Ascorbic Acid,Tapioca Dextrin,Chocolate Liquor,Cocoa Butter,Soy Lecithin (an Emulsifier),Coc
oa Powder,Natural and Artificial Flavors,Color Added,Red 40 Lake,Red 40,Yellow 5 6 Lake,Yellow 5 6,Blue 1 2 Lake,B
lue 1,Beeswax,Carnauba Wax,Confectioner's Glaze,Salt."
```



Here is a good picture of what we just talked about.

Pattern locating functions return the start indices and length of the found pattern. Pattern extracting functions can return the found string, the whole text or indices of the row. Pattern replacing functions simply replace the found pattern with specified strings.

So far We had only looked at complete patterns. What if we just want basketball player whose name starts with 'K'? What if I want "Blackberry" or "Strawberry" or any fruit that ends with "berry"?

We use metacharacters. We use them to describe our desired patterns

- `.` matches any character except the new line (also called Wild Character)
- `|` represents Or, which means both works
- `[...]` lists permitted characters (i.e. The character can be any listed within the bracket)
- `[a-z]` specifies character ranges
- `[^...]` lists excluded characters
- `(...)` groups a pattern together.
- `*` matches characters at least 0 times
- `+` matches characters at least 1 time
- `?` matches at most 1 time
- `{n}` matches exactly n times.
- `{n,}` matches at least n times
- `{,n}` matches at most n times
- `{n, m}` matches between n and m times
- `\n` matches a new line
- `\t` matches a tab
- `\\d` matches a digit (i.e. [0-9])
- `\\D` matches non-digits (i.e. [^0-9])
- `[[:alnum:]]` or `[A-z0-9]` matches Alphaumeric characters
- `\\w` matches Word characters
- `[[:blank:]]` matches Space and tab
- `[[:punct:]]` matches Punctuation characters
- … and many others!

Suppose we want all the players' first names that start with 'k'.

```
grep("^K", nba_data$Player, value = TRUE)
```

```
##  [1] "Kelly Olynyk"             "Kay Felder"
##  [3] "Kevin Love"               "Kyle Korver"
##  [5] "Kyrie Irving"             "Kyle Lowry"
##  [7] "Kent Bazemore"            "Kris Humphries"
##  [9] "Khris Middleton"          "Kevin Seraphin"
## [11] "Kentavious Caldwell-Pope" "Kemba Walker"
## [13] "Kristaps Porzingis"       "Kyle O'Quinn"
## [15] "K.J. McDaniels"           "Kevin Durant"
## [17] "Kevon Looney"             "Klay Thompson"
## [19] "Kawhi Leonard"            "Kyle Anderson"
## [21] "Kyle Wiltjer"             "Kyle Singler"
## [23] "Kenneth Faried"           "Kosta Koufos"
## [25] "Karl-Anthony Towns"       "Kris Dunn"
```

Suppose we want players whose initial is A.C.

```
grep("^A.+[[:blank:]]C", nba_data$Player, value = TRUE)
```

```
## [1] "Allen Crabbe"
```

Suppose we want players whose name includes a "Book"

```
grep("B.*o{2,}.*k", nba_data$Player, value = TRUE, ignore.case = TRUE)
```

```
## [1] "Aaron Brooks"      "Brook Lopez"       "Trevor Booker"
## [4] "Russell Westbrook" "Devin Booker"
```

Those are A+ Athletes with high intellectual curiosity.

What about "love"?

```
grep("l.*o.*v.*e", nba_data$Player, value = TRUE, ignore.case = TRUE)
```

```
## [1] "Kevin Love"  "Kyle Korver"
```

They are on the same team!

Let's look at some complex examples.

What beverage should I pick If I want "Strawberry","Mango","Peach", "Banana", and "Apple" in it?

```
drinks <- grep("(?=.*Strawberry)(?=.*Banana)(?=.*Apple)(?=.*Mango)(?=.*Peach)", food_data$ingredients, value = FAL
SE, ignore.case = TRUE, perl = TRUE)
food_data$name[drinks[1]]  #Let's get the first occurence
```

```
## [1] "Fruit Juice Drink"
```

Of course, Fruit Juice Drink..

We can use `(?=)` to lookarounds, and perl is set to true because Perl-compatible regexps is used. This allows us to match strings that contain all five fruits in any order.

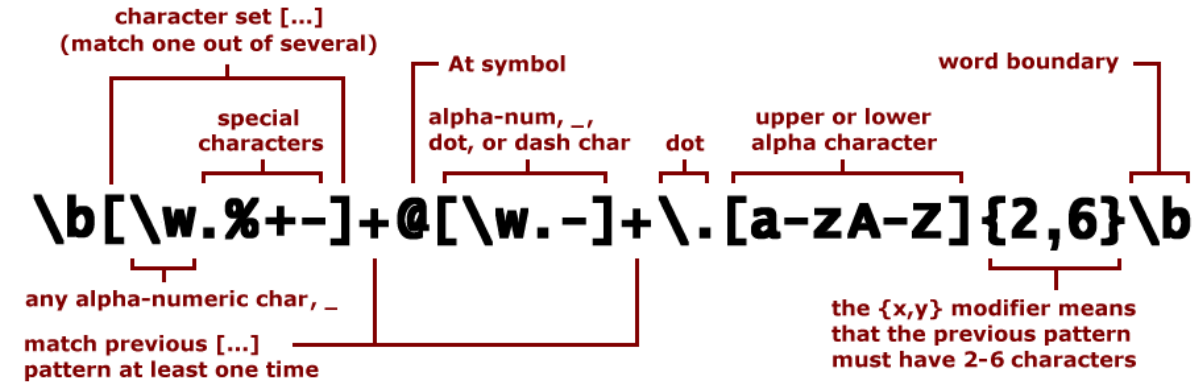
What if tonight I want to eat a dish that has freshly peeled “Shrimp”, chopped green “onion”, freshly grated Parmesan “cheese”, and olive “oil”?

```
dish <- grep("(?=.*shrimp)(?=.*onion)(?=.*cheese)(?=.*oil)", food_data$ingredients, value = FALSE, ignore.case = T
RUE, perl = TRUE)
food_data$name[dish]
```

```
## [1] "Gorton's Garlic Butter Shrimp Scampi, 10.5 oz."
```

Hmm.. Close enough.

Lastly, I want to show a real world example on email parsing.



A real world example of email parsing

Photo Credit: twiki.org This picture gives an example of how a regular expression that recognizes email pattern can be lengthy and complex, and as with many other languages, there are many ways to write a email regex.

As the patterns get more complex, regex might get more complex and tricky too. Therefore, it is always a good idea to keep practicing. There might be struggles in the beginning, but through practice you will become more comfortable with it!

Summary

The take-home message of this post would be Regular Expression is useful when one is dealing with a data set that involves string or texts. One can write such generalized patterns to select and extract desired pieces of data.

References and resources

- http://www.gastonsanchez.com/r4strings/regex1.html

This is an awesome resource written by Professor Sanchez.

- https://www.kaggle.com/datafiniti/food-ingredient-lists/data

There are a lot of data set that involves texts and strings in Kaggle.

- https://www.r-bloggers.com/regular-expressions-in-r-vs-rstudio/

This R Blog post talked about functions strsplit() and gsub().

- https://www.rstudio.com/wp-content/uploads/2016/09/RegExCheatsheet.pdf

This cheat sheet is wonderful that it touches a little bit of everything about Regular Expression.

- http://stat545.com/block022_regular-expression.html

This gives a more brief lesson on how to use regular expression.

- https://regexr.com/

This is a great place where everyone can practice regex!

- http://stringr.tidyverse.org/articles/regular-expressions.html

This goes over a lot of special cases. It is very comprehensive.

- http://biostat.mc.vanderbilt.edu/wiki/pub/Main/SvetlanaEdenRFiles/regExprTalk.pdf

Another great resource!

Thank you for reading my post!