REGEX is that thing that scares everyone almost all the time. Hence, finding some alternative is always very helpful and peaceful too. Here’s a nice R package thst helps us do REGEX without knowing REGEX.

**REGEX**

This is the REGEX pattern to test the validity of a URL:

^(http)(s)?(\:\/\/)(www\.)?([^\ ]\*)$

A typical regular expression contains — Characters ( http ) and Meta Characters ([]). The combination of these two form a meaningful regular expression for a particular task.  
So, What’s the problem?

Remembering the way in which characters and meta-characters are combined to create a meaningful regex is itself a tedious task which sometimes becomes a bigger task than the actual problem of NLP which is the larger goal.

**Installation**

is available on RVerbalExpressions Github so you can use devtools or remotes to install it from Github.

# install.packages("devtools")

devtools::install\_github("VerbalExpressions/RVerbalExpressions")

**Pseudo-Problem**

Let’s create a pseudo-problem that we’d like to solve with regex through which we can understand this package to programmatically create regex.

A simpler one perhaps, We’ve got multiple text like and we’d like to extract the names from it. Here’s our input and output look like:

strings = c('123Problem233','233Solution434','223Tim Apple444')

Problem, Solution, Time Apple

Once we solve this, we’ll move forward with slightly complicated problems.

**Pseudo-Code**

Before we code, it’s always good to write-out a pseudo-code on a napkin or even a paper if you’ve got. That is, We want to extract names (which is composition of alphabets) except numbers (which is digits). We build a regex for one-line and then we iterate it for all the elements in our vector.

**Loading**

Like any other R package, we can load RVerbalExpressions with library() function.

library(RVerbalExpressions)

**Constructing the Expression**

**Extract Strings**

Like many other modern-day R packages, RVerbalExpressions support %>% pipe operator for better simplicity and readability of the code. But for this problem of extracting strings that are present between the numbers, we can simply use one function that is rx\_alpha() to say that we need alphabets from the given string.

expr = rx\_alpha()

stringr::str\_extract\_all(strings,expr)

[[1]]

[1] "P" "r" "o" "b" "l" "e" "m"

[[2]]

[1] "S" "o" "l" "u" "t" "i" "o" "n"

[[3]]

[1] "T" "i" "m" "A" "p" "p" "l" "e"\*

**Extract Numbers**

Similar to the text that we extracted, Extracting Numbers again is very English as we’ve to use the function rx\_digit() to say that we need numbers from the given text.

expr = rx\_digit()

stringr::str\_extract\_all(strings,expr)

[[1]]

[1] "1" "2" "3" "2" "3" "3"

[[2]]

[1] "2" "3" "3" "4" "3" "4"

[[3]]

[1] "2" "2" "3" "4" "4" "4"

**Another Constructor to extract the name as a word**

Here, we can use the function rx\_word() to match it as word (rather than letters).

expr = rx\_alpha() %>% rx\_word() %>% rx\_alpha()

stringr::str\_extract\_all(strings,expr)

[[1]]

[1] "Problem"

[[2]]

[1] "Solution"

[[3]]

[1] "Tim" "Apple"

**Expression**

What if we want to use the expression somewhere else or simply we need the regex expression. It’s simple because the expression is what we’ve constructed and printing what we constructed would reveal the relevant regex pattern.

expr

"[A-z]\\w+[A-z]"

**Summary**

Thus, we managed to build a regex pattern without knowing regex. Simply put, we programmatically generated a regex pattern using R (that doesn’t require the high-level knowledge of regex patterns) and accomplished a tiny task that we took up to demonstrate the potential.