Regular_Expressions_in_R

May 22, 2021

REGULAR EXPRESSIONS IN R

In this notebook, we will study some simple Regular Expression terms and apply them with R functions.

0.0.1 Table of contents

1 Loading in Data

Let's load in a small list of emails to perform some data analysis and take a look at it.

```
[ ]: email_df <- read.csv("https://ibm.box.com/shared/static/
      ↪cbim8daa5vjf5rf4rlz11330lvqbu7rk.csv")
     email_df
```

So our simple dataset contains a list of names and a list of their corresponding emails. Let's say we want to simply count the frequency of email domains. But several problems arise before we can even attempt this. If we attempt to simply count the email column, we won't end up with what we want since every email is unique. And if we split the string at the '@', we still won't have what we want since even emails with the same domains might have different regional extensions. So how can we easily extract the necessary data in a quick and easy way?

2 Regular Expressions

Regular Expressions are generic expressions that are used to match patterns in strings and text. A way we can exemplify this is with a simple expression that can match with an email string. But before we write it, let's look at how an email is structured:

$[test@testing.com](mailto://test@testing.com?cm_m mc = Email_Newsletter -_ Developer_Ed$

So, an email is composed by a string followed by an '@' symbol followed by another string. In R regular expressions, we can express this as:

$. + @.+$

Where:

- The '.' symbol matches with any character.
- The '+' symbol repeats the previous symbol one or more times. So, '.+' will match with any string.
- The '@' symbol only matches with the '@' character.

Now, for our problem, which is extracting the domain from an email excluding the regional url code, we need an expression that specifically matches with what we want:

@. +

.

Where the '\.' symbol specifically matches with the '.' character.

3 Regular Expressions in R

Now let's look at some R functions that work with R functions.

The grep function below takes in a Regular Expression and a list of strings to search through and returns the positions of where they appear in the list.

```
[ ]: grep("@.+",  c("test@testing.com" , "not an email", "test2@testing.com"))
```

Grep also has an extra parameter called 'value' that changes the output to display the strings instead of the list positions.

```
[ ]: grep("@.+",  c("test@testing.com", "not an email", "test2@testing.com"),␣
      ↪value=TRUE)
```

The next function, 'gsub', is a substitution function. It takes in a Regular Expression, the string you want to swap in with the matches and a list of strings you want to perform the swap with. The code cell below updates valid emails with a new domain:

```
[ ]: gsub("@.+", "@newdomain.com", c("test@testing.com", "not an email",␣
      ↪"test2@testing.com"))
```

The functions below, 'regexpr' and 'regmatches', work in conjunction to extract the matches found by a regular expression specified in 'regexpr'.

```
[ ]: matches <- regexpr("@.*", c("test@testing.com", "not an email", "test2@testing.
      ↪com"))
     regmatches(c("test@testing.com", "not an email", "test2@testing.com"), matches)
```

This function is actually perfect for our problem since we simply need to extract the specific information we want. So let's use it with the Regular Expression we defined above and store the extracted strings in a new column in our dataframe.

```
[ ]: matches <- regexpr("@.*\\.", email_df[,'Email'])
     email_df[,'Domain'] = regmatches(email_df[,'Email'], matches)
```

And this is the resulting dataframe:

[ ]:
```
email_df
```

Now we can finally construct the frequency table for the domains in our dataframe!

[ ]:
```
table(email_df[,'Domain'])
```

Scaling R with big data As you learn more about R, if you are interested in exploring platforms that can help you run analyses at scale, you might want to sign up for a free account on IBM Watson Studio, which allows you to run analyses in R with two Spark executors for free.