What are Regular Expressions

Abstract

Any data wrangling that involves text in which data is embedded could benefit from the capabilities that regular expressions provide. This paper introduces regular expressions and provides some examples of how we can write python code to perform regular expressions on given text.
Keywords:
Regular Expressions

Regular Expressions

The simplest explanation for Regular expressions is that it is used to manipulate alphanumeric data. For example if we ever used the "split("<*character to split by*>") function in pretty much any programming language, we've used regular expression. examples below show we use regular expression function of the string object itself:

In [34]:

```
sentence = 'The simplest explanation for Regular expressions is that it is used to man
ipulate alphanumeric data'


print(sentence.split())
```

```
['The', 'simplest', 'explanation', 'for', 'Regular', 'expressions', 'is', 'that', 'it'
, 'is', 'used', 'to', 'manipulate', 'alphanumeric', 'data']
```

Note that the sentence was split by the blank space between each word, and the result was returned as a list of words. In the above example, the split function can either be given a character to split by, or use its default separators. For example:

In [32]:

```
sentences = 'Sentence 1\nSentence2. Sentence 3\t sentence4'


print(sentences.split())
```

```
['Sentence', '1', 'Sentence2.', 'Sentence', '3', 'sentence4']
```

Note that the split function split by the " ", "\n", ".", and "\t" characters. Let's see if it works with '&'.

In [31]:

```
sentences = 'Sentence1&Sentence2&Sentence3&sentence4'
```

```
print(sentences.split())
```

```
['Sentence1&Sentence2&Sentence3&sentence4']
```

Nope!! The character '&' is not in the list of characters the split function was built to split by. Let's try by providing the split character explicitly:

In [30]:

```
sentences = 'Sentence1&Sentence2&Sentence3&sentence4'


print(sentences.split('&'))
```

```
['Sentence1', 'Sentence2', 'Sentence3', 'sentence4']
```

Another way to use regular expression is to use the **'re'** library in python (RegEx in C#, etc.). In python, there **'re'** library provides the following functions:

findall|

```
Returns all non-overlapping matching patters in a string as a list.
```

finditer

```
same as findall but returns an iterator rather than a list.
```

match

```
match a pattern at the start of a string
```

search

```
scan a string for the match in the pattern.
```

split

```
breaks up a string to peices that are separated by given pattern
```

sub, subn

```
replace all with **sub** and first n occurrences with **subn** of the given patters.
```

Regular expression functions are a quick and easy way to parse data, and in more complex data wrangling exercises, the regular expression pattens can be devised to extract more "hard to extract" data. With regular expression patterns, we can define exactly how to match patterns by creating them. For example, the patterns below can be used to identify their respective patterns in a given text.
non-alpha_pattern = "[^a-zA-Z0-9 -]"
white_space_pattern = "\s+"
and we can use these patterns in any of the functions above. For example, let's try replacing non-alphanumeric characters with blanks:

In [55]:

```
import re


non_alpha_pattern = '[^0-9a-zA-Z]+'
sentence = 'This%sentence* has &&too many!non-alphanumeric characters'
new_sentence = re.sub(non_alpha_pattern,' ',sentence)
print(new_sentence)
```

```
This sentence has too many non-alphanumeric characters
```

Let us remove extra blanks.

In [56]:

```
import re


white_space_pattern = "\s+"
sentence = "This    sentence   has too many           spaces between     words"
new_sentence = re.sub(white_space_pattern," ",sentence)
print(new_sentence)
```

```
This sentence has too many spaces between words
```

The patterns in a regular expression follow a syntax. For example, '\s+' means 1 or more **s**paces. '\d' matches any decimal digits and '\D' is any non-decimal digit. Notice that '\D+' would mean 1 or more non-decimal digit character. With these directives, we can write a plethora of patterns that can do anything we want.

Conclusion

This paper touched on the surface of regular expression capabilities. Its application in parsing data from text data is widespread among various disciplines and is a welcome tool to use in data wrangling phase of a data science project.

References

Regular Expression HOWTO
Python RegEx
Regular Expressions (Regex) Tutorial: How to Match Any Pattern of Text -
McKinney, Wes. Python for Data Analysis . O'Reilly Media. Kindle Edition.