AI for Medicine Course 3 Week 2 lecture notebook - Cleaning Text

For this notebook you'll be using the `re` module, which is part of Python's Standard Library and provides support for regular expressions (which you may know as `regexp`).

- If you aren't familiar with `regexp`, we recommend checking the documentation.

Regular expressions allow you to perform searches and replacements in strings based on patterns. Let's start by looking at some examples.

You'll be using the search method, which looks like this:

```
search(pattern, text)
```

It will output a match if one is found, or None otherwise.

For the following three examples, you'll try to match the pattern to the string "Pleural Effusion." Take note of these special characters:

- ^ denotes "starts with" followed by the pattern
- $ denotes "ends with" preceded by the pattern
- | denotes "or" followed by another pattern

Go ahead and import the `re` library, then run the following three cells. Can you see why the first two examples output a match, while the third one does not?

Import Library

In [1]:
```
import re
```

Examples of Search Patterns

In [2]:
```
# Search if string starts with 'Pl' or ends with 'ion'
m = re.search(pattern = "^Pl|ion$", string = "Pleural Effusion")

# return the matched string
if m:
    print(m.group(0))
else:
    print(None)
```

```
Pl
```

In [3]:
```
# Search if string starts with 'Sa' or ends in 'ion'
m = re.search(pattern = "^Sa|ion$", string = "Pleural Effusion")

# return the matched string
if m:
    print(m.group(0))
else:
    print(None)
```

```
ion
```

In [4]:
```
# Search if string starts with 'Eff'
m = re.search(pattern="^Eff", string="Pleural Effusion")

# return the matched string
if m:
    print(m.group(0))
else:
    print(None)
```

```
None
```

Notice that even though 'Eff' exists in the string, the string does not begin with 'Eff', so there is no match.

Now let's try a more advanced example. Your goal here is to match the pattern "a single letter of the alphabet" followed by a number.

Characters in a set [a-zA-Z]

[]

- [a-z] matches lowercase letters
- [A-Z] matches uppercase letters
- [a-zA-Z] matches lowercase and uppercase letters.

You can match any lowercase or uppercase single letter followed by '123' like this:

In [5]:
```
# Match a letter followed by 123
m = re.search(pattern ='[a-zA-Z]123', string = "99C123")
print(f"{m.group(0)}")
```

```
C123
```

Notice how the match includes the letter 'C' in C123.

'Lookbehind' assertion

If you want to match the single letter but not include it in the returned match, you can use the lookbehind assertion

(?<=...)

Here is the documentation for lookbehind:

> Matches if the current position in the string is preceded by a match for ... that ends at the current position. This is called a positive lookbehind assertion. (?<=abc)def will find a match in 'abcdef', since the lookbehind will back up 3 characters and check if the contained pattern matches. The contained pattern must only match strings of some fixed length, meaning that abc or a|b are allowed, but a* and a{3,4} are not. Note that

> patterns which start with positive lookbehind assertions will not match at the beginning of the string being searched; you will most likely want to use the search() function rather than the match() function:

Note that you'll need to put `?<=[a-zA-Z]` within parentheses, like this: `(?<=[a-zA-Z])`; otherwise you'll see an error message.

In [6]:
```
# Match a letter followed by 123, exclude the letter.
m = re.search(pattern = '(?<=[a-zA-Z])123', string = "99C12399")
print(f"{m.group(0)}")
```

```
123
```

Notice the difference here. The match is returned because '123' is preceded by a letter 'C', but the letter 'C' is not returned as part of the matched substring. You're 'looking back' but not including the lookback as part of the returned substring.

Match 123 followed by a letter

Similarly, you can match a letter followed by '123', including the letter.

In [7]:
```
# Match 123 followed by a letter
m = re.search(pattern = '123[a-zA-Z]', string = "99123C99")
print(f"{m.group(0)}")
```

```
123C
```

Notice that the letter 'C' is included in the returned match.

'Lookahead' assertion

Similarly, you can match '123' followed by a letter, but exclude the letter from the match.

- You can do this by using the lookahead assertion. (?=...)

Here is the documentation:

> Matches if ... matches next, but doesn't consume any of the string. This is called a lookahead assertion. For example, Isaac (?=Asimov) will match 'Isaac ' only if it's followed by 'Asimov'.

Similar to the lookbehind, you'll need to wrap `?=[a-zA-Z]` around parentheses, like this: `(?=[a-zA-Z])`, to avoid an error message.

In [8]:
```
# Match 123 followed by a letter, exclude the letter from returned match.
m = re.search(pattern = '123(?=[a-zA-Z])', string = "99123C99")
print(f"{m.group(0)}")
```

```
123
```

Notice that the returned match does not include the letter.

String Cleaning

Let's implement a `clean()` function. It should receive a sentence as input, clean it up and then return the clean version of it. "Cleaning" in this case refers to:

1. Convert to lowercase only
2. Change "and/or" to "or"
3. Change "/" to "or" when used to indicate equality between two words such as tomatos/tomatoes
4. Replace double periods ".." with single period "."
5. Insert the appropiate space after periods or commas
6. Convert multiple whitespaces to a single whitespace

Let's take this one step at a time, and pay attention to how the sentence changes along the way.

Here's the sample sentence:

In [9]:
```
# Choose a sentence to be cleaned
sentence = "     BIBASILAR OPACITIES,likely representing bilateral pleural effusions with ATELECTASIS   and/or PNEUMONIA/bron
```

Step 1: lowercase

Now, use the built-in `lower()` method to change all characters of a string to lowercase. Quick and easy!

In [10]:
```
# Convert to all lowercase letters
sentence = sentence.lower()
sentence
```

Out[10]:
```
'     bibasilar opacities,likely representing bilateral pleural effusions with atelectasis   and/or pneumonia/bronchopneumon
ia..'
```

Step 2: and/or -> or

The `re` module provides the `sub()` method, which substitutes patterns in a string with another string. Here you'll be looking for 'and/or' and replacing it with just 'or'.

In [11]:
```
sentence = re.sub('and/or', 'or', sentence)
sentence
```

Out[11]:
```
'     bibasilar opacities,likely representing bilateral pleural effusions with atelectasis   or pneumonia/bronchopneumoni
a..'
```

Step 3: / -> or

In [12]:
```
sentence = re.sub('(?<=[a-zA-Z])/(?=[a-zA-Z])', ' or ', sentence)
sentence
```

Out[12]:
```
'     bibasilar opacities,likely representing bilateral pleural effusions with atelectasis   or pneumonia or bronchopneumoni
a..'
```

Step 4: .. -> .

When finding a specific string and replacing, you can also use Python's built-in `replace()` method.

Otherwise, when matching special characters like `.`, use backslash to specify that you're looking for the actual period character `\.`

In [13]:
```
# Replace .. with . using re.sub (option 1)
tmp1 = re.sub("\.\.", ".", sentence)
```

```
print(tmp1)

# Replace .. with . using string.replace (option 2)
tmp2 = sentence.replace('..','.')
print(tmp2)
```

```
     bibasilar opacities,likely representing bilateral pleural effusions with atelectasis   or pneumonia or bronchopneumoni
a.
     bibasilar opacities,likely representing bilateral pleural effusions with atelectasis   or pneumonia or bronchopneumoni
a.
```

In [14]:
```
# Replace .. with . using string.replace
sentence = sentence.replace("..", ".")
sentence
```

Out[14]:
```
'     bibasilar opacities,likely representing bilateral pleural effusions with atelectasis   or pneumonia or bronchopneumoni
a.'
```

Step 5: add whitespace after punctuation

For step 5, let's use a built-in Python function, `translate()`.

- This will return a copy of your string, mapped to a translation table that you define. It's usually used alongside the `maketrans()` method, and you can read about both of them here.

In [23]:
```
# Define a dictionary to specify that ! is replaced by !!!
# and 's' is replaced by ''
translation_dict = {'!': '!!!',
                    'z': 's'
                   }
print(translation_dict)
# Create the translation table
translation_tbl = str.maketrans(translation_dict)
print(translation_tbl)
```

```
{'!': '!!!', 'z': 's'}
{33: '!!!', 122: 's'}
```

Note that each key in the dictionary should be a character of length 1 (the key can't be a word of length 2 or more).

In [24]:
```
# Choose a string to be translated
tmp_str = "colonization, realization, organization!"
print(tmp_str)

# Translate the string using the translation table
tmp_str2 = tmp_str.translate(translation_tbl)
print(tmp_str2)
```

```
colonization, realization, organization!
colonisation, realisation, organisation!!!
```

Notice how z is replaced by s, and ! is replaced by !!!

Add whitespace after punctuation.

- Now apply this to replace '.' with '. ', where the period is followed by a whitespace.
- Similarly, replace ',' with ', ', so that the comma is followed by a whitespace.

In [25]:
```
# Creat translation table using a dictionary comprehension
translation_dict = {key: f"{key} " for key in ".,"}

# View the translation dictionary
display(translation_dict)

# View the translation dictionary with some formatting for easier reading
# Use vertical bars to help see the whitespace more easily.
for key, val in translation_dict.items():
    print(f"key: |{key}| \tval:|{val}|")
```

```
{'.': '. ', ',': ', '}

key: |.|        val:|. |
key: |,|        val:|, |
```

In [26]:
```
# Create the translation table using the translation dictionary
punctuation_spacer = str.maketrans(translation_dict)

# Apply the translation table to add whitespace after punctuation
sentence = sentence.translate(punctuation_spacer)
sentence
```

Out[26]:
```
'worrisome nodule in the Right Upper  lobe. CANNOT exclude neoplasm. . '
```

Step 6: trim whitespace

Nice! For step 6, you can trim multiple whitespaces with Python's `join()` method. Sidenote: This can be also done using `regexp`.

In [27]:
```
# Split the string using whitespace as the delimiter
# This removes all whitespace between words
sentence_list = sentence.split()
sentence_list
```

Out[27]:
```
['worrisome',
 'nodule',
 'in',
 'the',
 'Right',
 'Upper',
 'lobe.',
 'CANNOT',
 'exclude',
 'neoplasm.',
 '.']
```

In [28]:
```
# Join the tokens with a single whitespace.
# This ensures that there is a single whitespace between words
sentence = ' '.join(sentence_list)
sentence
```

Out[28]:
```
'worrisome nodule in the Right Upper lobe. CANNOT exclude neoplasm. .'
```

The sentence is now cleaner and easier to work with!

Putting it all together

Now you can put all that together into a function and test this implementation on more sentences.

In [29]:
```
def clean(sentence):
    lower_sentence = sentence.lower()
    corrected_sentence = re.sub('and/or', 'or', lower_sentence)
    corrected_sentence = re.sub('(?<=[a-zA-Z])/(?=[a-zA-Z])', ' or ', corrected_sentence)
    clean_sentence = corrected_sentence.replace("..", ".")
    punctuation_spacer = str.maketrans({key: f"{key} " for key in ".,"})
    clean_sentence = clean_sentence.translate(punctuation_spacer)
    clean_sentence = ' '.join(clean_sentence.split())
    return clean_sentence

sentences = ["     BIBASILAR OPACITIES,likely representing bilateral pleural effusions with ATELECTASIS   and/or PNEUMONIA.."
             "Small left pleural effusion/decreased lung volumes bilaterally.left RetroCardiac Atelectasis.",
             "PA  and lateral views of the chest demonstrate   clear lungs,with NO focal air space opacity and/or pleural eff
             "worrisome nodule in the Right Upper  lobe.CANNOT exclude neoplasm.."]

for n, sentence in enumerate(sentences):
    print("\n##########################\n")
    print(f"Sentence number: {n+1}")
    print(f"Raw sentence: \n{sentence}")
    print(f"Cleaned sentence: \n{clean(sentence)}")
```

```
##########################

Sentence number: 1
Raw sentence:
     BIBASILAR OPACITIES,likely representing bilateral pleural effusions with ATELECTASIS   and/or PNEUMONIA..
Cleaned sentence:
bibasilar opacities, likely representing bilateral pleural effusions with atelectasis or pneumonia.

##########################

Sentence number: 2
Raw sentence:
Small left pleural effusion/decreased lung volumes bilaterally.left RetroCardiac Atelectasis.
Cleaned sentence:
small left pleural effusion or decreased lung volumes bilaterally. left retrocardiac atelectasis.

##########################

Sentence number: 3
Raw sentence:
PA  and lateral views of the chest demonstrate   clear lungs,with NO focal air space opacity and/or pleural effusion.
Cleaned sentence:
pa and lateral views of the chest demonstrate clear lungs, with no focal air space opacity or pleural effusion.

##########################

Sentence number: 4
Raw sentence:
worrisome nodule in the Right Upper  lobe.CANNOT exclude neoplasm..
Cleaned sentence:
worrisome nodule in the right upper lobe. cannot exclude neoplasm.
```

Congratulations

Congratulations on finishing this lecture notebook!** By now, you should have a better grasp of `regexp` along with some built-in Python methods for cleaning text. You'll be seeing the `clean()` function again in the upcoming graded assignment. Good luck and have fun!

In [ ]:
```
```