ASDS2 Exercise 1.1 Regex

April 9, 2025

[1]:
```
# Importing relevant packages

import pandas as pd
import re
```

1 Advanced Social Data Science 2 (ASDS2) Exercises

1.1 Overview and regular expressions

1.1.1 1: Thinking about text as data

Go to Kaggle's database of text data sets here: https://www.kaggle.com/datasets?topic=nlpDatasets

1. Find an interesting data set. (Try searching the data sets or playing around with the sorting rule in the top right). It doesn't have to be social sciencey, just whatever looks interesting to you.
2. Describe the variables in the data. What's there in addition to the text itself, if anything?
3. What's a meaningful latent variable which might vary across the texts? (For example, 'sentiment' might plausibly vary across movie reviews).
4. Assume you could measure the latent variable from (3). How might that latent variable correlate with other properties of the units of the data? (These can be observed variables in the data, or other, unobserved properties).

I chose this data set https://www.kaggle.com/datasets/tharunmss/water-bottle-dataset-flipkart. The dataset contains customer reviews of water bottles scraped from an Indian e-Commerce platform.

There are 6 columns: - 1. index - unique identifier to each review entry - 2. product_name - name of water bottle - 3. overall_rating - average rating for the product - 4. rating - individual rating of user on a scale from 1 to 5 stars - 5. title - headline/summary of the review, capturing the main sentiment - 6. cleaned_review - complete text of customer's review (=text variable)

A meaningful latent variable could be the sentiment of the review. It would probably be reasonable to assume that sentiment and indvidual rating (and maybe overall rating) positively correlate. Maybe it also correlated with the length of the review, so that really negative or really positive sentiment result in longer reviews.

1.1.2 2: Importing text data

1. The file mach.csv, available at the course Absalon page, contains part of Machiavelli's The Prince, subdivided into 188 sections. Download it to your computer.
2. Import the file into Python using read_csv() from pandas.

(Tip: Check the content of the data frame using the .head()-function, to assess whether everything is tidy and ready to go).

[2]:
```
# load data
mach_df = pd.read_csv("mach.csv")

print(mach_df.shape)
mach_df.head()
```

```
(188, 2)
```

[2]:

	Unnamed: 0	text
0	Mach_1.txt.content	DEDICATORY LETTER Niccolo Machiavelli to His M…
1	Mach_10.txt.content	But considerable problems arise if territorie…
2	Mach_100.txt.content	them, saying that he could not fight well with…
3	Mach_101.txt.content	the Swiss, they are not confident of being abl…
4	Mach_102.txt.content	Empire; and all the vigour that was drained fr…

3. Using the search function from Python's re module (or a Pandas equivalent), find out in which section(s) the following terms appear:
   - lion
   - flatterers
   - ccmnot

[3]:
```
terms = ["lion", "flatterers", "ccmnot"]
sections = {}

for term in terms:

    indices = []

    for index, text in enumerate(mach_df["text"]):

        if re.search(term, text):
            indices.append(index)

    sections[term] = indices

print(sections)
```

```
{'lion': [26, 27, 44, 47, 97, 112, 139, 166], 'flatterers': [74, 75, 76],
'ccmnot': [53]}
```

4. Why might a nonsensical term like 'ccmnot' be in the corpus?

From investigating the section that contains 'ccmnot', it seems it is a spelling mistake and was supposed to say 'cannot'. Perhaps the Machiavelli text was made digital by scanning and this word was misread.

(Tip: Try printing the content of the text containing 'ccmnot'. Does it contain more text than the notebook displays by default? How could we change this?)

[4]:
```
term = "ccmnot"
sections = {}

full_text = []

for text in mach_df["text"]:

    if re.search(term, text):
        full_text.append(text)

print(full_text[0])

# alternative
print(mach_df.iloc[53]["text"])
```

```
 But let us return to our subject. I maintain that anyone who considers what I
have written will realise that either hatred or contempt led to the downfall of
the emperors I have discussed; he will recognise that some of them acted in one
way and others in the opposite way, and that one ruler in each group was
successful and the others ended badly. Because Pertinax and Alexander were new
rulers, it was useless and harmful for them to act like Marcus, who was an
hereditary ruler. Likewise, it was harmful for Caracalla, Commodus and Maximinus
to act like Severus, because they lacked the ability required to follow in his
footsteps. Therefore, a new ruler in a new principality ccmnot imitate the
conduct of Marcus, nor again is it necessary to imitate that of Severus. Rather,
he should imitate Severus in the courses of action that are necessary for
establishing himself in power, and imitate Marcus in those that are necessary
for maintaining power that is already established and secure, thus achieving
glory.
 But let us return to our subject. I maintain that anyone who considers what I
have written will realise that either hatred or contempt led to the downfall of
the emperors I have discussed; he will recognise that some of them acted in one
way and others in the opposite way, and that one ruler in each group was
successful and the others ended badly. Because Pertinax and Alexander were new
rulers, it was useless and harmful for them to act like Marcus, who was an
hereditary ruler. Likewise, it was harmful for Caracalla, Commodus and Maximinus
to act like Severus, because they lacked the ability required to follow in his
footsteps. Therefore, a new ruler in a new principality ccmnot imitate the
conduct of Marcus, nor again is it necessary to imitate that of Severus. Rather,
he should imitate Severus in the courses of action that are necessary for
establishing himself in power, and imitate Marcus in those that are necessary
for maintaining power that is already established and secure, thus achieving
glory.
```

1.1.3 3: Regular expressions

In this exercise, we're continuing with Python's re module. The following can be solved using one or more of these three functions in re: `search` `split` `sub`

Hint: Take a look at the documentation for Python's re module to find solutions, and test your regular expression on regextester.com or consult regex101.com

1. Define a function that can check if a string contains a certain set of characters (for this excercise a-z or A-Z or 0-9), and test your function on some strings to confirm that it works.

[5]:
```
# define function
def check_characters(pattern, strings):
    results = []

    for string in strings:
        result = bool(re.search(pattern, string))
        results.append(result)

    return results

# set strings
strings = ["Hello",
           "12345",
           "Hello123",
           "hello",
           "!@€$"]

# set pattern
pattern = r"[a-z]"

print(check_characters(pattern, strings))

# set different pattern
pattern = r"[A-Z0-9]"

print(check_characters(pattern, strings))
```

```
[True, False, True, True, False]
[True, True, True, False, False]
```

2. Define a function that can check if a string contains an *a* followed by **zero** or more *b*'s.

Examples:

"ac" is a match

"abc" is a match

"bbc" is not a match

[6]:
```
# define function
def check_characters(pattern, strings):
    results = []

    for string in strings:
```

```
        result = bool(re.search(pattern, string))
        results.append(result)

    return results

# set strings
strings = ["ac",
           "abc",
           "bbc"]

# set pattern
pattern = r"ab*"

check_characters(pattern, strings)
```

[6]: `[True, True, False]`

3. Define a function that can check if a string contains an *a* followed by **one** or more *b*'s.

(Now "ac" should no longer be a match!)

[7]:
```
# define function
def check_characters(pattern, strings):
    results = []

    for string in strings:
        result = bool(re.search(pattern, string))
        results.append(result)

    return results

# set strings
strings = ["ac",
           "abc",
           "bbc"]

# set pattern
pattern = r"ab+"

check_characters(pattern, strings)
```

[7]: `[False, True, False]`

4. Using the sample string 'The quick brown fox jumps over the lazy dog', search for the words 'fox', 'dog', 'horse'.

[8]:
```
# string
string = "The quick brown fox jumps over the lazy dog"
```

```
words = ["fox", "dog", "horse"]


results = []

for word in words:
    result = bool(re.search(word, string))
    results.append(result)

results
```

[8]: `[True, True, False]`

5. Define a string containing a sentence with the word 'Road' in it, and use the re.sub()-function to abbreviate 'Road' as 'Rd.'.

(For example: "The quick brown fox jumps over the lazy dog on Hampton Road" –> "The quick brown fox jumps over the lazy dog on Hampton Rd.")

[9]:
```
sentence = "All roads lead to Rome"

re.sub("road", "rd", sentence)
```

[9]: `'All rds lead to Rome'`

6. Define a string containing a sentence and perform very simple tokenization by splitting at all whitespaces.

(The result should be a list where each element in the list corresponds to a word from the sentence)

[10]:
```
sentence = "This is a sentence."

tokens = sentence.split()

print(tokens)
```

```
['This', 'is', 'a', 'sentence.']
```

7. Define a string containing a sentence and replace whitespaces with an underscore. After, reverse this by replacing underscores with a whitespace.

[11]:
```
sentence = "This is a sentence."

sentence = re.sub(" ", "_", sentence)
print(sentence)

sentence = re.sub("_", " ", sentence)
print(sentence)
```

```
This_is_a_sentence.
This is a sentence.
```

8. Define a string containing a sentence with a few cases of multiple spaces between words and remove all those cases.

[12]:
```
sentence = "This   is   a  sentence    with  multiple    spaces."

sentence = re.sub(r'\s+', ' ', sentence)

print(sentence)
```

```
This is a sentence with multiple spaces.
```