Regular Expression

https://regex101.com/r/f42lPM/1

References

- https://www.guru99.com/python-regular-expressions-complete-tutorial.html
- https://docs.python.org/3.4/library/re.html
- https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html#zz-1.9
- https://www.debuggex.com/cheatsheet/regex/python
- https://blog.finxter.com/python-regex-and-operator-tutorial-video/
- https://www.oreilly.com/library/view/regular-expressions-cookbook/9781449327453/ch04s04.html
- https://www.webfx.com/tools/emoji-cheat-sheet/
- https://www.kaggle.com/raenish/cheatsheet-text-helper-functions

Libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import re
import emoji

#Count vectorizer for N grams
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

# Nltk for tekenize and stopwords
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

General Principles

Major RE functions

re.findall - Module is used to search for "all" occurrences that match a given pattern.
re.sub - Substitute the matched RE patter with given text
re.match - The match function is used to match the RE pattern to string with optional flags
re.search - This method takes a regular expression pattern and a string and searches for that pattern with the string.

```
mystring = "preposterous"

# Gives you the index number of the start of the match
mystring.find("rous")
// 8

# If it doesn't find the matching string, it returns None
re.search("rous", mystring)
```

```
f = open("../input/alice-in-wonderland.txt")
alice_lines = f.readlines() # Read all the lines
alice_lines = [l.rstrip() for l in alice_lines] # strip whitespace in each line and then add it in list (list comprehension)
f.close()

for line in alice_lines:
        if re.search("Hatter", line): print( line ) # Look through each line and if a line has word "Hatter" in it, print that line
```

```
# Find word Hatter but also match not just "H"atter but also "h"atter

for line in alice_lines:
    if re.search("[Hh]atter", line): print(line)
```

[aeiuo] matches any lowercase vowel.
[1234567890_abc] matches a digit or an underscore or a or b or c
[....] always matches a single character, and you are specifying all the possibilities of what it could be.

[A-Z] matches any uppercase letter
[a-z] matches any lowercase letter
[0-9] matches any digit
You can combine them, for example in [A-Za-z0-9]

Caret(^) acts as a negation if used within a bracket
[^aeiou] matches any character that is not a lowercase vowel (what does that encompass?)
[^A-Za-z] matches anything but a letter
But it can be used as anchors as well
Anchors don't match any characters, they mark special places in a string: at the beginning and end of the string, and at the boundaries of words!
Caret ("^"), when not used within a bracket, matches words at the beginning of a string. So "^123" will only match strings that begin with "123".
\d matches a single digit, equivalent to [0-9]
\D matches a single character that is not a digit, equivalent to [^0-9]
\s matches a whitespace, equivalent to [\t\n\r\f\v]
\S matches a non-whitespace
\w matches an alphanumeric character, equivalent o [A-Za-z0-9_]
\W matches a non-alphanumeric character

period (.) matches any single character (e.g. letter digit punctuation whitespace etc.)
You need backslash ( ) if you want to match a literal period

plus (+) matchces characters one or more times
star (*) matches characters zero or more times
verticle line (|) in paranthesis is equivalent to "or" when matching words. For example, mov(es|ing|e|ed) matches "moves", "moving", "move", and "moved".
"?" means optionality. For instance, sings? will match "sing" as well as "sings".

Specific Case Examples (frequently used)

```
## any words that contain some b_ _ _ ed form
for line in alice_lines:
    if re.search("b\w\w\wed", line): print( line )
```

// she passed; it was **labelled** 'ORANGE MARMALADE', but to her great
begged the Mouse to tell them something more.
Queen jumped up and **bawled** out, "He's murdering the time! Off with his
'They were **obliged** to have him with them,' the Mock Turtle said: 'no
(she was **obliged** to say 'creatures,' you see, because some of them were
obliged to write with one finger for the rest of the day; and this was

```
## 3 sequences of numbers
for line in alice_lines:

    if re.search("[0-9][0-9][0-9]", line): print(line)
```

```
## Lines with words with at least 7 characters
for line in alice_lines:
    if re.search(".......+", line): print(line)
```

```
## look for lines in Alice in Wonderland that start with "The"
for line in alice_lines:
    if re.search("^The", line): print(line)
```

Find URL

```
def find_url(string):
    text = re.findall('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',string)
    return "".join(text) # converting return value from list to string
```

```
sentence="I love spending time at https://www.kaggle.com/"
find_url(sentence)
// 'https://www.kaggle.com/'

df['url']=df['text'].apply(lambda x:find_url(x))
```

Emoticons

```
sentence="I love ⚽ very much 😁"
find_emoji(sentence)

# Emoji cheat sheet - https://www.webfx.com/tools/emoji-cheat-sheet/
# Uniceode for all emoji : https://unicode.org/emoji/charts/full-emoji-list.html
```

```
df['emoji']=df['text'].apply(lambda x: find_emoji(x))
```

Remove emoji from text

```
def remove_emoji(text):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

```
sentence="Its all about \U0001F600 face"
print(sentence)
remove_emoji(sentence)
```

```
//Its all about 😀 face
'Its all about  face'

df['text']=df['text'].apply(lambda x: remove_emoji(x))
```

Email

```
def find_email(text):
    line = re.findall(r'[\w\.-]+@[\w\.-]+',str(text))
    return ",".join(line)

sentence="My gmail is abc99@gmail.com"
find_email(sentence)
//'abc99@gmail.com'
```

```
df['email']=df['text'].apply(lambda x: find_email(x))
```

Hashtag

```
sentence="#Corona is trending now in the world"
find_hash(sentence)
```

```
df['hash']=df['text'].apply(lambda x: find_hash(x))
```

Mention on SNS (@)

```
def find_at(text):
    line=re.findall(r'(?<=@)\w+',text)
    return " ".join(line)
```

```
sentence="@David,can you help me out"
find_at(sentence)
// “David”
```

```
df['at_mention']=df['text'].apply(lambda x: find_at(x))
```

Number

```
def find_number(text):
    line=re.findall(r'[0-9]+',text)
    return " ".join(line)
```

```
sentence="2833047 people are affected by corona now"
find_number(sentence)
//'2833047'
```

```
df['number']=df['text'].apply(lambda x: find_number(x))
```

Year

```
def find_year(text):
    line=re.findall(r"\b(19[40][0-9]|20[0-1][0-9]|2020)\b",text)
    return line
```

```
sentence="India got independence on 1947."
find_year(sentence)
//’1947’
```

```
df['year']=df['text'].apply(lambda x: find_year(x))
```

Non Alphanumeric

```
def find_nonalp(text):
    line = re.findall("[^A-Za-z0-9 ]",text)
    return line
```

```
sentence="'Twitter has lots of @ and # in posts.(general tweet)"
find_nonalp(sentence)
// ['@', '#', '.', '(', ')']
```

```
df['non_alp']=df['text'].apply(lambda x: find_nonalp(x))
```

Punctuations

```
def find_punct(text):
    line = re.findall(r'[!"\$%&\'()*+,\-.\/:;=#@?\[\\\]^_`{|}~]*', text)
    string="".join(line)
    return list(string)
```

```
example="Corona virus have kiled #24506 confirmed cases now.#Corona is un(tolerable)"
print(find_punct(example))
// ['#', '.', '#', '(', ')']
```

```
df['punctuation']=df['text'].apply(lambda x : find_punct(x))
```

Stopwords

```
def stop_word_fn(text):
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(text)
    non_stop_words = [w for w in word_tokens if not w in stop_words]
    stop_words= [w for w in word_tokens if w in stop_words]
    return stop_words
```

```
example_sent = "This is a sample sentence, showing off the stop words filtration."
stop_word_fn(example_sent)
// ['is', 'a', 'off', 'the']
```

```
df['stop_words']=df['text'].apply(lambda x : stop_word_fn(x))
```

Repetitive Character

```
def rep(text):
    grp = text.group(0)
    if len(grp) > 1:
        return grp[0:1] # can change the value here on repetition (If you want to change match repetitive characters to n numbers,chage the return line in the rep function to grp[0:n])

def unique_char(rep,sentence):
    convert = re.sub(r'(\w)\1+', rep, sentence)
    return convert
```

```
sentence="heyyy this is loong textttt sooon"
unique_char(rep,sentence)
//'hey this is long text son'
```

```
df['unique_char']=df['text'].apply(lambda x : unique_char(rep,x))
```

Dollar

```
def find_dollar(text):
    line=re.findall(r'\$\d+(?:\.\d+)?',text)
    return " ".join(line)

# \$ - dollar sign followed by
# \d+ one or more digits
# (?:\.\d+)? - decimal which is optional
```

```
sentence="this shirt costs $20.56"
find_dollar(sentence)
// '$20.56'
```

```
df['dollar']=df['text'].apply(lambda x : find_dollar(x))
```

Date (*mm/dd/yyyy format)*

```
def find_dates(text):
        line=re.findall(r'\b(1[0-2]|0[1-9])/(3[01]|[12][0-9]|0[1-9])/([0-9]{4})\b',text)
        return line
```

```
sentence="Todays date is 04/28/2020 for format mm/dd/yyyy, not 28/04/2020"
find_dates(sentence)
// [('04', '28', '2020')]
```

```
df['dates']=df['text'].apply(lambda x : find_dates(x))
```

Only Words

```
def only_words(text):
        line=re.findall(r'\b[^\d\W]+\b', text)
        return " ".join(line)
```

```
sentence="the world population has grown from 1650 million to 6000 million"
only_words(sentence)
```

```
'the world population has grown from million to million'
```

```
df['only_words']=df['text'].apply(lambda x : only_words(x))
```

Only Numbers

```
def only_numbers(text):
        line=re.findall(r'\b\d+\b', text)
        return " ".join(line)

sentence="the world population has grown from 1650 million to 6000 million"
only_numbers(sentence)
// '1650 6000'
```

```
df['only_num']=df['text'].apply(lambda x : only_numbers(x))
```

Is the Key word in the sentence? (without using IN)

```
def search_string(text,key):
        return bool(re.search(r"+key+", text))
```

```
sentence="Happy Mothers day to all Moms"
search_string(sentence,'day')
```

```
// True
```

```
df['search_day']=df['text'].apply(lambda x : search_string(x,'day'))
```

OR

```
my_string_rows = df[df['text'].str.contains("good")]
my_string_rows[['text']].sample(3)
```

If a string has multiple sentences in it and want to find all the sentences within that string that contains a certain key word

```
def pick_only_key_sentence(text,keyword):
    line=re.findall(r'([^.]*'+keyword+'[^.]*)', text)
    return line

sentence="People are fighting with covid these days.Economy has fallen down.How will we survice covid"
pick_only_key_sentence(sentence,'covid')
// ['People are fighting with covid these days', 'How will we survice covid']
```

```
df['pick_senence']=df['text'].apply(lambda x : pick_only_key_sentence(x,'covid'))
```

Caps Words (words that start with capital letter)

Extract words starting with capital letter. Some words like names,place or universal object are usually mentioned in a text starting with CAPS.

```
def find_capital(text):
    line=re.findall(r'\b[A-Z]\w+', text)
    return line

sentence="World is affected by corona crisis.No one other than God can save us from it"
find_capital(sentence)
// ['World', 'No', 'God']
```

Remove Tags

```
def remove_tag(string):
    text=re.sub('<.*?>','',string)
    return text

sentence="Markdown sentences can use <br> for breaks and <i></i> for italics"
remove_tag(sentence)
// 'Markdown sentences can use  for breaks and  for italics'
```

IP Address

IP address from text

```
def ip_add(string):
    text=re.findall('\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}',string)
    return text

sentence="An example of ip address is 125.16.100.1"
ip_add(sentence)
// ['125.16.100.1']
```

Extract Mac Address from Text

```
def mac_add(string):
    text=re.findall('(?:[0-9a-fA-F]:?){12}',string)
    return text
#https://stackoverflow.com/questions/26891833/python-regex-extract-mac-addresses-from-string/26892371
```

```
sentence="MAC ADDRESSES of this laptop - 00:24:17:b1:cc:cc .Other details will be mentioned"
mac_add(sentence)
// ['00:24:17:b1:cc:cc']
```