

## THỰC HÀNH BUỔI 3

(Data Collection and Preprocessing)

**Bài 1.** Sinh viên tham khảo đường link bên dưới và thực hành theo các phần trong website hướng dẫn với các hàm, ghi nhận lại kết quả:

Link: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

---

```
soup.title  
  
soup.title.name  
  
soup.title.string  
  
soup.title.parent.name  
  
soup.p  
  
soup.p['class']  
  
soup.a  
  
soup.find_all('a')  
  
soup.find(id="link3")  
  
print(soup.get_text())
```

---

**Bài 2:** Thực hiện crawler dữ liệu từ website <https://www.ayush.nz/> với 2 loại *technology*, *video-games*.

### Gợi ý:

#### 1. Import thư viện

```
import requests  
from bs4 import BeautifulSoup  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import os
```

#### 2. Khởi tạo 2 đường dẫn (link cho 2 loại):

```
seed_urls = ['https://notes.ayushsharma.in/technology',  
             'https://www.ayush.nz/video-games']
```

3. Xây dựng hàm **build\_dataset(seed\_urls)** để lấy dữ liệu với các thuộc tính dữ liệu là: **'title', 'excerpt', 'pub\_date', 'category'**. Sử dụng **DataFrame** để đưa dữ liệu về dạng bảng dữ liệu.

```
def build_dataset(seed_urls):
    news_data = []
    for url in seed_urls:
        news_category = url.split('/')[ -1]
        data = requests.get(url)
        soup = BeautifulSoup(data.text, 'html.parser')

        (#Sinh viên thực hiện code tiếp vào đây)

    return df
```

4. Thực hiện lưu file với định dạng mở rộng **.csv** có tên file **dataset.csv**

### Bài 3: Text processing (sử dụng tập dữ liệu ở bài 2)

Dữ liệu hỗ trợ: Sinh viên copy past vào file project sau đó thực thi (google colab) hoặc tạo file .py nếu sử dụng các công cụ khác hỗ trợ lập trình ( pycharm....)

```
CONTRACTION_MAP = {
    "ain't": "is not",
    "aren't": "are not",
    "can't": "cannot",
    "can't've": "cannot have",
    "'cause": "because",
    "could've": "could have",
    "couldn't": "could not",
    "couldn't've": "could not have",
    "didn't": "did not",
    "doesn't": "does not",
    "don't": "do not",
    "hadn't": "had not",
    "hadn't've": "had not have",
    "hasn't": "has not",
    "haven't": "have not",
    "he'd": "he would",
    "he'd've": "he would have",
    "he'll": "he will",
    "he'll've": "he he will have",
    "he's": "he is",
    "how'd": "how did",
    "how'd'y": "how do you",
    "how'll": "how will",
    "how's": "how is",
    "I'd": "I would",
    "I'd've": "I would have",
    "I'll": "I will",
```

"I'll've": "I will have",  
"I'm": "I am",  
"I've": "I have",  
"i'd": "i would",  
"i'd've": "i would have",  
"i'll": "i will",  
"i'll've": "i will have",  
"i'm": "i am",  
"i've": "i have",  
"isn't": "is not",  
"it'd": "it would",  
"it'd've": "it would have",  
"it'll": "it will",  
"it'll've": "it will have",  
"it's": "it is",  
"let's": "let us",  
"ma'am": "madam",  
"mayn't": "may not",  
"might've": "might have",  
"mightn't": "might not",  
"mightn't've": "might not have",  
"must've": "must have",  
"mustn't": "must not",  
"mustn't've": "must not have",  
"needn't": "need not",  
"needn't've": "need not have",  
"o'clock": "of the clock",  
"oughtn't": "ought not",  
"oughtn't've": "ought not have",  
"shan't": "shall not",  
"sha'n't": "shall not",  
"shan't've": "shall not have",  
"she'd": "she would",  
"she'd've": "she would have",  
"she'll": "she will",  
"she'll've": "she will have",  
"she's": "she is",  
"should've": "should have",  
"shouldn't": "should not",  
"shouldn't've": "should not have",  
"so've": "so have",  
"so's": "so as",  
"that'd": "that would",  
"that'd've": "that would have",  
"that's": "that is",  
"there'd": "there would",  
"there'd've": "there would have",  
"there's": "there is",  
"they'd": "they would",  
"they'd've": "they would have",  
"they'll": "they will",  
"they'll've": "they will have",

```
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what will",
"what'll've": "what will have",
"what're": "what are",
"what's": "what is",
"what've": "what have",
"when's": "when is",
"when've": "when have",
"where'd": "where did",
"where's": "where is",
"where've": "where have",
"who'll": "who will",
"who'll've": "who will have",
"who's": "who is",
"who've": "who have",
"why's": "why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you would",
"you'd've": "you would have",
"you'll": "you will",
"you'll've": "you will have",
"you're": "you are",
"you've": "you have"
}
```

## 1) Text Wrangling and Pre-processing

```
import spacy
import pandas as pd
import numpy as np
import nltk
nltk.download('stopwords')
from nltk.tokenize.toktok import ToktokTokenizer
import re
from bs4 import BeautifulSoup
import unicodedata

tokenizer = ToktokTokenizer()
stopword_list = nltk.corpus.stopwords.words('english')
stopword_list.remove('no')
stopword_list.remove('not')
```

## 2) Remove HTML tags

```
def strip_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    stripped_text = soup.get_text()
    return stripped_text
```

## 3) Loại bỏ ký tự có dấu

```
def remove_accented_chars(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text

remove_accented_chars('Sómě Áccěntěd těxt')
```

## 4) Expand contractions

```
def expand_contractions(text, contraction_mapping=CONTRACTION_MAP):

    contractions_pattern = re.compile('{{}}'.format('|'.join(contraction_mapping.keys()))),
    flags=re.IGNORECASE|re.DOTALL)

    def expand_match(contraction):
        match = contraction.group(0)
        first_char = match[0]
        expanded_contraction = contraction_mapping.get(match)\
            if contraction_mapping.get(match)\
            else contraction_mapping.get(match.lower())
        expanded_contraction = first_char+expanded_contraction[1:]
        return expanded_contraction

    expanded_text = contractions_pattern.sub(expand_match, text)
    expanded_text = re.sub("'", "", expanded_text)
    return expanded_text

expand_contractions("Y'all can't expand contractions I'd think")
```

## 5) Remove special characters

```
def remove_special_characters(text, remove_digits=False):
    pattern = r'^a-zA-Z0-9\s' if not remove_digits else r'^a-zA-Z\s'
    text = re.sub(pattern, '', text)
    return text

remove_special_characters("Well this was fun! What do you think? 123#@!",
                          remove_digits=True)
```

## 6) Text lemmatization

```
import spacy
nlp = spacy.load("en_core_web_sm")
def lemmatize_text(text):
    text = nlp(text)
    text = ' '.join([word.lemma_ if word.lemma_ != '-PRON-' else word.text for word in text])
    return text

lemmatize_text("My system keeps crashing! his crashed yesterday, ours crashes daily")
```

## 7) Text stemming

```
def simple_stemmer(text):
    ps = nltk.porter.PorterStemmer()
    text = ' '.join([ps.stem(word) for word in text.split()])
    return text

simple_stemmer("My system keeps crashing his crashed yesterday, ours crashes daily")
```

## 8) Remove stopwords

```
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopwords_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopwords_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text

remove_stopwords("The, and, if are stopwords, computer is not")
```

## 9) Building a text normalizer

```

def normalize_corpus(corpus, html_stripping=True, contraction_expansion=True,
                    accented_char_removal=True, text_lower_case=True,
                    text_lemmatization=True, special_char_removal=True,
                    stopword_removal=True, remove_digits=True):

    normalized_corpus = []
    # normalize each document in the corpus
    for doc in corpus:
        # strip HTML
        if html_stripping:
            doc = strip_html_tags(doc)
        # remove accented characters
        if accented_char_removal:
            doc = remove_accented_chars(doc)
        # expand contractions
        if contraction_expansion:
            doc = expand_contractions(doc)
        # lowercase the text
        if text_lower_case:
            doc = doc.lower()
        # remove extra newlines
        doc = re.sub(r'[\r|\n|\r\n|]+', ' ', doc)

        # lemmatize text
        if text_lemmatization:
            doc = lemmatize_text(doc)
        # remove special characters and/or digits
        if special_char_removal:
            # insert spaces between special characters to isolate them
            special_char_pattern = re.compile(r'([{.(-)!}])')
            doc = special_char_pattern.sub(" \\1 ", doc)
            doc = remove_special_characters(doc, remove_digits=remove_digits)
        # remove extra whitespace
        doc = re.sub(' +', ' ', doc)
        # remove stopwords
        if stopword_removal:
            doc = remove_stopwords(doc, is_lower_case=text_lower_case)

        normalized_corpus.append(doc)

    return normalized_corpus

```

## 10) Pre-process and normalize data news

```

dt['full_text'] = dt["title"].map(str) + ' . ' + dt["excerpt"]
dt['full_text']

```

```

dt['clean_text'] = normalize_corpus(dt['full_text'])
norm_corpus = list(dt['clean_text'])
dt.iloc[1][['full_text', 'clean_text']].to_dict()

```

#### 11) Save the news data

```
dt.to_csv('news.csv', index=False, encoding='utf-8')
```

#### 12) Sinh viên nghiên cứu thực hành Bag of Words Model, TF-IDF