

▼ Coursework 2: Wikipedia articles

This is the second coursework of ECS7023P Programming for AI and Data Science, which counts 65% towards the final grade of the module. The coursework is graded out of 100 marks.

Deadline: Thursday 2nd November, 2023 - 11.59pm

Marking criteria: While the most important marking criterion will be for the code to achieve the expected objective and output, marks will also be given for partial or close solutions, whereas marks can be deducted for code that is overly complex, inefficient, difficult to understand and/or to maintain.

Use of packages: In addition to built-in python functions and elements that we have seen in the lectures (see lecture notes), the only additional packages allowed for this exercise include *string* and *json*.

How to submit: You will submit a completed Jupyter notebook file with your solutions, as well as a PDF version of the Jupyter notebook which includes the outputs of your code (either two separate files or a single zip file that contains both the .ipynb and .pdf files). Your submission will include the python code that produces the required answers. Answers produced through means other than python code will not be deemed acceptable.

Note: This is an individual coursework, the solutions you submit need to be your own and developed on your own.

▼ Dataset

For this exercise, you are given a dataset that contains a sample of Wikipedia articles, with their content reduced to only text (that is with images and other non-textual content removed).

The dataset is contained in a file called 'wiki-articles.json', where each line contains an entry with a Wikipedia article: an ID, a URL, title and body text.

▼ Note

Note: For this coursework, you should remove all the punctuation prior to processing the text. To do this, use the following code, where *text* is the variable where you want to remove the punctuation:

```
import string

translator = str.maketrans('', '', string.punctuation)
text = text.translate(translator)
```

▼ Exercises

1. Ignoring the case, what is the most frequently occurring word across all articles in the dataset? Multiple occurrences of the same word within an article should only be counted once. **(5 marks)**

```
%%time

import json
import string

# Function created to load the articles
def load_wiki_articles(file_path):
    articles = []
    with open(file_path, 'r') as json_file:
        for line in json_file:
            entry = json.loads(line)
            articles.append(entry)
    return articles

# Function created to preprocess the text
def preprocess_text(text):
    translator = str.maketrans('', '', string.punctuation)
    text_no_punctuation = text.translate(translator)
    text_lowercase = text_no_punctuation.lower()
    return text_lowercase
```

```
# Load articles
wiki_articles = load_wiki_articles('wiki-articles.json')

# Count word occurrences
word_count = {}
for article in wiki_articles:
    text = article['text']
    preprocessed_text = preprocess_text(text)
    words = preprocessed_text.split()
    set_words = set(words)
    for word in set_words:
        word_count[word] = word_count.get(word, 0) + 1

# Find the most common word
most_common_word = max(word_count, key=word_count.get)

# Print the most frequently occurring word
print(f"The most frequently occurring word is: {most_common_word}, and the count is: {word_count[most_common_word]}")
```

```
The most frequently occurring word is: the, and the count is: 19227
CPU times: user 55.8 s, sys: 1.09 s, total: 56.9 s
Wall time: 59.1 s
```

2. The least frequent characters in English are: x, z, j, q. How many articles are there in the dataset that contain all these 4 characters (at least one occurrence of each)? **(5 marks)**

```
%%time
# Function created to find the number of articles containing the least frequent characters
def articles_containing_least_frequent_chars(dataset, least_frequent_chars):
    articles_with_least_frequent_chars = 0

    for article in dataset:
        text = article['text']
        preprocessed_text = preprocess_text(text)

        # Check if all least frequent characters are present
        has_all_chars = all(char in preprocessed_text for char in least_frequent_chars)

        if has_all_chars:
            articles_with_least_frequent_chars += 1

    return articles_with_least_frequent_chars

# Least frequent characters as a set
least_frequent_chars = set(['x', 'z', 'j', 'q'])

# Count the articles containing all four least frequent characters using the revised function above
count_least_frequent_chars = articles_containing_least_frequent_chars(wiki_articles, least_frequent_chars)

# Output the count of articles containing all four least frequent characters
print(f"Number of articles containing all four least frequent characters: {count_least_frequent_chars}")
```

```
Number of articles containing all four least frequent characters: 14688
CPU times: user 32.3 s, sys: 103 ms, total: 32.4 s
Wall time: 32.5 s
```

3. What is the longest alphabetical word in the dataset that starts with the letter 'p', and what is the length of this word? (alphabetical word = containing only a-z letters and no numbers or other characters) **(10 marks)**

```
%%time

# Function created to find the longest_alphabetical word starting with p
def longest_alphabetical_word_starting_with_p(dataset):
    longest_word = ''
    longest_word_length = 0

    for article in dataset:
        text = article['text']
        preprocessed_text = preprocess_text(text)
        words = preprocessed_text.split()

        for word in words:
            if word.startswith('p') and word.isalpha():
                if len(word) > longest_word_length:
                    longest_word = word
```

```

    longest_word_length = len(word)

    return longest_word, longest_word_length

# Find the longest word starting with 'p' in the dataset...
longest_p_word, longest_p_word_length = longest_alphabetical_word_starting_with_p(wiki_articles)

# Output the longest word starting with 'p' and its length...
print(f"The longest alphabetical word in the dataset that starts with the letter 'p' is '{longest_p_word}' with a length of {longest_p_word_length}")

The longest alphabetical word in the dataset that starts with the letter 'p' is 'pneumonoultramicroscopicsilicovolcanoconiosis' with
CPU times: user 46.1 s, sys: 211 ms, total: 46.3 s
Wall time: 46.5 s

```

4. What is the longest sequence of capitalised words in the dataset? Only consider if the first character of each word is upper case, regardless of whether the rest of the characters are upper or lower case. (15 marks)

Toy example: if we had the following sentence: "Queen Mary University is located in the United Kingdom, specifically in London". This example has 3 sequences of capitalised words: Queen Mary University, United Kingdom and London. Of these, the longest sequence of capitalised words is Queen Mary University, with 3 words.

```

%%time

def longest_sequence_capitalised_words(dataset):
    longest_sequence = []
    current_sequence = []

    for article in dataset:
        text = article['text']
        translator = str.maketrans('', '', string.punctuation)
        text_no_punctuation = text.translate(translator)
        words = text_no_punctuation.split()


        for word in words:
            if word[0].isupper():
                current_sequence.append(word)
            else:
                if len(current_sequence) > len(longest_sequence):
                    longest_sequence = current_sequence
                current_sequence = []

        # Check if the last sequence was the longest in the article
        if len(current_sequence) > len(longest_sequence):
            longest_sequence = current_sequence

    return longest_sequence

longest_capitalised_sequence = longest_sequence_capitalised_words(wiki_articles)
longest_sequence = ' '.join(longest_capitalised_sequence)
print(f'Longest Sequence: {longest_sequence}')
print(f'Length: {len(longest_capitalised_sequence)}')

```

 Longest Sequence: Havilland, Michael Curtiz, James Cagney, David O. Selznick, William Wyler, Richard Brooks, Harry Cohn, Jane Wyman, Length: 134
CPU times: user 43.8 s, sys: 130 ms, total: 44 s
Wall time: 44.1 s

5. The same word can be written using different case (i.e. variations of lower and upper letters). For example, 'house' and 'HouSe' use the same letters to form the exact same word but with different case. Write the python code that identifies the word in the entire dataset that has the largest number of case variations. How many variations does it have? (15 marks)

Toy example: if our dataset had only the following words: "HouSe, House, HOUSE, YES, yes, YeS, Yes, yEs, yEs, yEs." We would then identify that 'house' has 3 variations and 'yes' has 6 variations. Therefore, 'yes' is the word with the largest number of variations, with 6.

```

%%time

def word_with_most_case_variations(dataset):
    case_variations = {}

    for article in dataset:
        text = article['text']
        translator = str.maketrans('', '', string.punctuation)

```

```

text_no_punctuation = text.translate(translator)
words = text_no_punctuation.split()

for word in words:
    standardised_word = word.lower()
    if standardised_word not in case_variations:
        case_variations[standardised_word] = {word}
    else:
        case_variations[standardised_word].add(word)

variation_count = 0
word = ''

for key, value in case_variations.items():
    if len(value) > variation_count:
        variation_count = len(value)
        word = key

return word, variation_count

# Example usage
word, variations = word_with_most_case_variations(wiki_articles)
print(f'The word in the entire dataset that has the largest number of case variations: {word}, and the number of variations is: {variation_count}')

The word in the entire dataset that has the largest number of case variations: pops, and the number of variations is: 6
CPU times: user 57.6 s, sys: 409 ms, total: 58 s
Wall time: 58.2 s

```

6. Two words are anagrams of each other if they can be written using the same characters but in a different order, including repeated characters. For example, 'cheap' is an anagram of 'peach', 'reacting' is an anagram of 'creating' and 'resistance' is an anagram of 'ancestries'.

Likewise, **n-word anagram sets** include **n** words that are anagrams of one another:

- 'asleep', 'please' and 'elapse' form a 3-word anagram set.
- 'aspired', 'despair', 'diapers' and 'praised' form a 4-word anagram set.

By only considering words that are *6 characters or longer*, list all the anagram sets of 4 words or more (that is n-word anagram sets where $n \geq 4$) that can be found within any article, i.e. we are not interested in checking anagrams across articles. **(15 marks)**

```

%%time
with open('wiki-articles.json', 'r') as wiki_file:
    for article in wiki_file:
        entry = json.loads(article)
        text = entry['text'].translate(str.maketrans('', '', string.punctuation))
        words = text.lower().split()

        anagrams = {}

        for word in words:
            if len(word) >= 6 and word.isalpha():
                sorted_word = ''.join(sorted(word))
                if sorted_word not in anagrams:
                    anagrams[sorted_word] = set()
                anagrams[sorted_word].add(word)

        anagram_sets = [words for words in anagrams.values() if len(words) >= 4]

        for anagram_set in anagram_sets:
            print(anagram_set)

{'siphlodora', 'drosophila', 'phloridosa', 'lordiphosa', 'dorsilopha', 'psilodorha'}
{'vftqsbporuzwyxhgdiecjalmnk', 'xdybpwosmuzriqgenlhvjtfack', 'jsrhfenduazyqgxtmcbbpiwvolk', 'ofrjvmazhqnbxpykculgswetdi', 'fcbjqawtvc'}
{'integral', 'altinger', 'triangle', 'relating'}
CPU times: user 1min 15s, sys: 695 ms, total: 1min 15s
Wall time: 1min 16s

```

7. With the aim of building a small search engine, we want to index the contents of the articles in the dataset (i.e. mapping which articles contain each word), where we also want to store the position(s) in which content appears in each article. For example, if an article ID is 7 and its text is "python coding is so much fun", we would say that the word 'coding' appears in position 1 of article ID 7 and the word 'fun' appears in position 5 of article ID 7. A toy example of an index could be:

- 'house' can be found in: position 97 of article 7, positions 32 and 221 of article 13, and position 63 of article 27
- 'London' can be found in: positions 17 and 97 of article 7, position 21 of article 25, and position 157 of article 42.

- etc.

(35 marks total) (see specific marks for each of the subquestions of question 7)

7. (a) Create a data structure that indexes the dataset contents following the above objective. You should ignore the case (i.e. lowercase everything) for this exercise. **(15 marks)**

Note: questions 7b and 7c need to be implemented using the indexing structure created here

```
%%time
```

```
def index(dataset):
    word_index = {}

    for article in dataset:
        article_id = article['id']
        text = article['text']
        processed_text = preprocess_text(text)
        words = processed_text.split()

        position = 0
        for word in words:
            position += 1
            if word not in word_index:
                word_index[word] = {}
            if article_id not in word_index[word]:
                word_index[word][article_id] = []
            word_index[word][article_id].append(position)

    return word_index

resulting_index = index(wiki_articles)

CPU times: user 1min 45s, sys: 5.75 s, total: 1min 50s
Wall time: 1min 52s
```

7. (b) Given a word as input parameter, write a function that outputs the article IDs in which the word can be found. As an example, produce the output for the word 'python'. **(5 marks)**

```
%%time
```

```
def find_word(word, word_index):
    word = word.lower() # Ensure the word is in lowercase so we are comparing lowercase to lowercase
    if word in word_index:
        return list(word_index[word].keys())
    else:
        return []

# Find the article IDs for the word 'python'
word_to_search = 'python'
article_ids_for_word = find_word(word_to_search, resulting_index)
print(f"Article IDs where '{word_to_search}' can be found: {article_ids_for_word}")
print(len(article_ids_for_word))

Article IDs where 'python' can be found: ['594', '809', '1063', '1437', '2581', '2807', '3054', '3382', '4015', '4147', '4373', '457210']
CPU times: user 344 µs, sys: 206 µs, total: 550 µs
Wall time: 407 µs
```

7. (c) Given two words and a distance value (integer) as input, return the list of article IDs where the two words occur within the distance. As an example, produce the output for words 'python' and 'coding' within a distance of 20. **(15 marks)**

For example, if we have a dataset with five articles:

- article ID 1: "python coding"
- article ID 2: "I am coding in python"
- article ID 3: "To learn coding, I have decided to start with python"
- article ID 4: "python is a fantastic programming language"
- article ID 5: "it's sunny today and I like it"

If we are given two words: 'python' and 'coding', and a distance of 2:

- article ID 1: the condition is true as 'python' and 'coding' occur within a distance of 1.
- article ID 2: the condition is true as 'python' and 'coding' occur within a distance of 2.

- article ID 3: the condition is false as 'python' and 'coding' occur within a larger distance of 7.
- article ID 4: the condition is false because it only contains 'python', no 'coding'.
- article ID 5: the condition is false because it doesn't contain any of the two keywords 'python' and 'coding'.

Therefore, in this case, `within_distance_ids('python', 'coding', 2)` would output `[1, 2]` as the list of matching article IDs.

```
%%time
```

```
def find_words_within_distance(word1, word2, distance, word_index):
    word1 = word1.lower()
    word2 = word2.lower()

    articles_with_both_words = set()

    if word1 in word_index and word2 in word_index:
        for article_id in word_index[word1]:
            if article_id in word_index[word2]:
                positions_word1 = word_index[word1][article_id]
                positions_word2 = word_index[word2][article_id]

                for pos1 in positions_word1:
                    for pos2 in positions_word2:
                        if abs(pos1 - pos2) <= distance:
                            articles_with_both_words.add(article_id)

    return list(articles_with_both_words)

# Example usage for 'python' and 'coding' within a distance of 20
word1_to_search = 'python'
word2_to_search = 'coding'
distance_value = 20

article_ids_for_words_within_distance = find_words_within_distance(word1_to_search, word2_to_search, distance_value, resulting_index)
print(f"Article IDs where '{word1_to_search}' and '{word2_to_search}' occur within a distance of {distance_value}: {article_ids_for_words}")

Article IDs where 'python' and 'coding' occur within a distance of 20: ['23862', '6974']
CPU times: user 691 µs, sys: 354 µs, total: 1.05 ms
Wall time: 1.05 ms
```