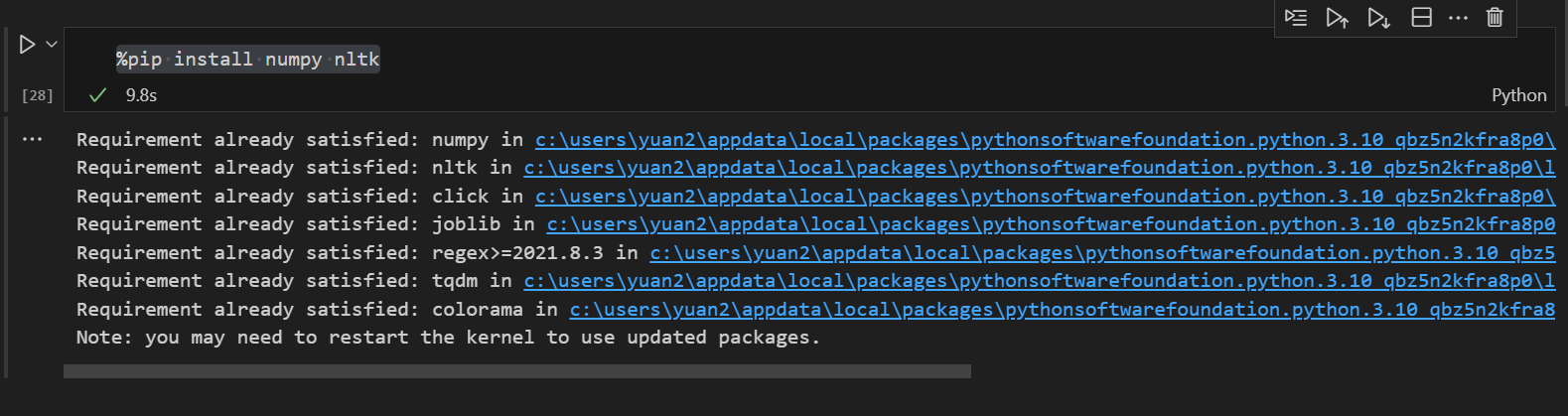
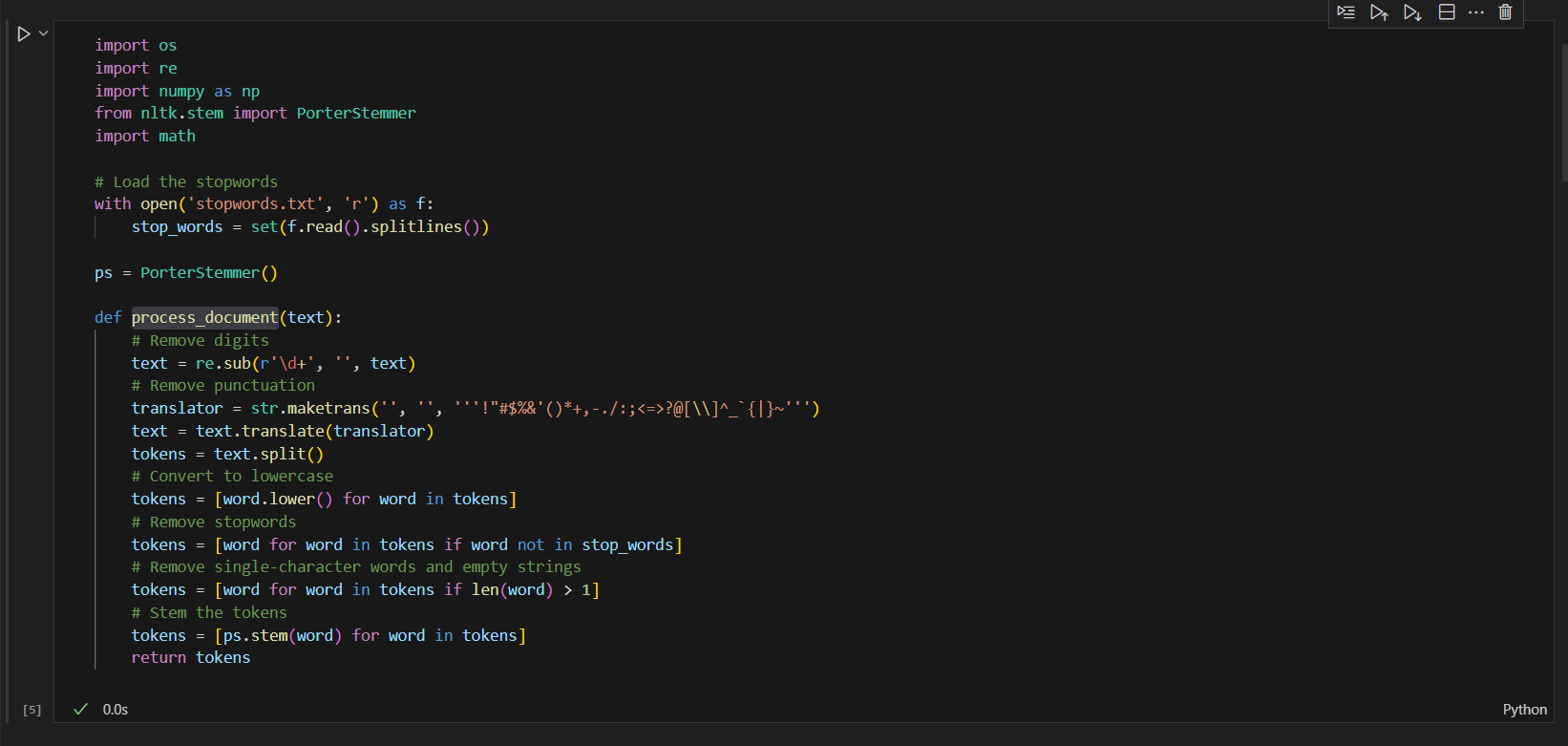
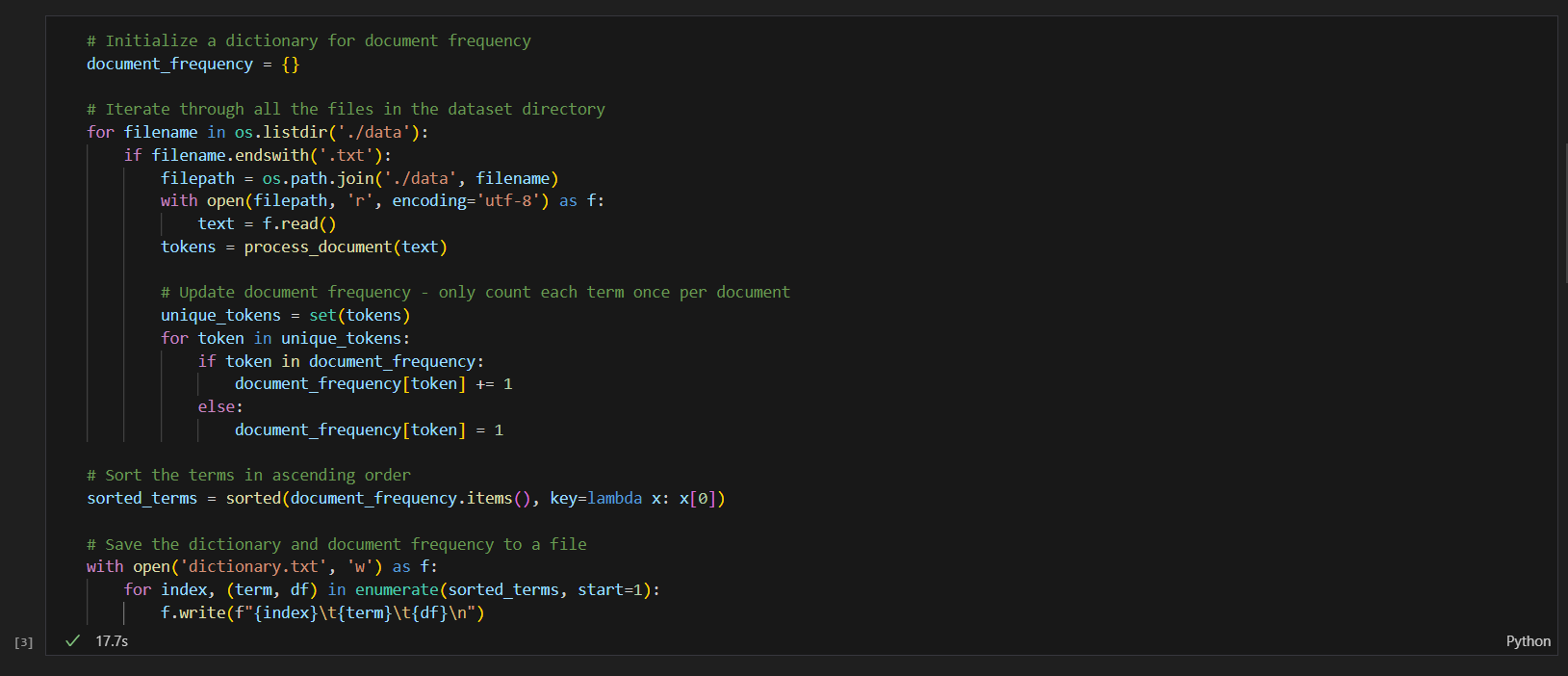
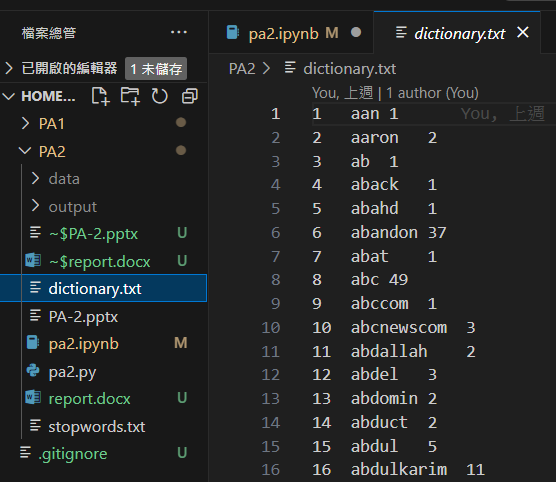
Introduction to Information Retrieval and Text Mining

**Programming Assignment 3**  R12725026 秦孝媛

1. **執行環境**：VS code
2. **程式語言**：python 3.10
3. **執行方式**：
   1. 使用 VS code 或 Jupyter Notebook 打開 pa3.py
   2. 執行第一個 cell 安裝需要的 packages（numpy, nltk）



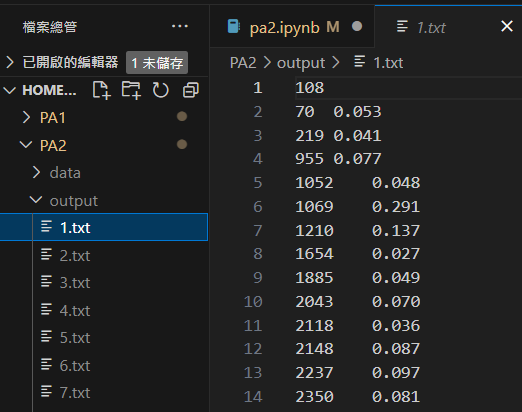
* 1. 確保stopwords.txt在相同的目錄中，執行第二個 cell，import 必要的 packages 以及 pa1 tokenize 的處理函數  
     
  2. 執行第三個 cell，會在目錄下產生 dictionary.txt

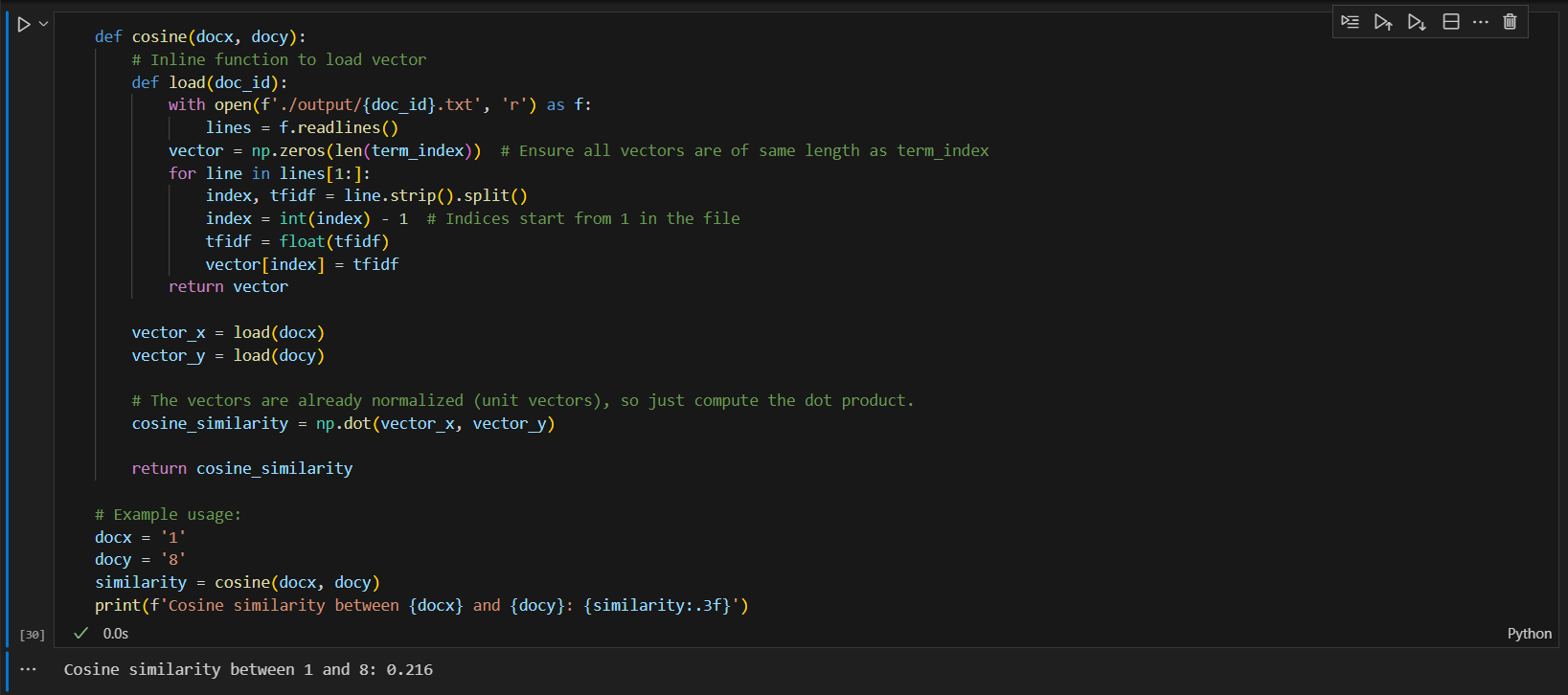
dictionary.txt 截圖  


* 1. 執行第四個 cell，會在 ./output 資料夾中，產生每一個文件的 {DocID.txt}，裡面包含所有的 unit tfidf vectors



1.txt 截圖



* 1. 執行第五個 cell，會將指定 documents 的 vector 載入並且計算 cosine similarity（更改不同的 docx 以及 docy 編號，可以計算不同的文章相似性）  
     

1. **作業處理邏輯說明**：

***Construct a dictionary based on the terms extracted from the given documents.***

* 1. 先將 pa1 的 tokenization 寫成一個函數，處理依序為，移除數字、移除標點符號、Lowercasing、移除 stopwords、移除長度為一或零的字串、使用 NLTK library中的PorterStemmer 進行 stemming。

def process\_document(text):

    # Remove digits

    text = re.sub(r'\d+', '', text)

    # Remove punctuation

    translator = str.maketrans('', '', '''!"#$%&'()\*+,-./:;<=>?@[\\]^\_`{|}~''')

    text = text.translate(translator)

    tokens = text.split()

    # Convert to lowercase

    tokens = [word.lower() for word in tokens]

    # Remove stopwords

    tokens = [word for word in tokens if word not in stop\_words]

    # Remove single-character words and empty strings

    tokens = [word for word in tokens if len(word) > 1]

    # Stem the tokens

    tokens = [ps.stem(word) for word in tokens]

    return tokens

* 1. 計算 document frequency 以及建立 dictionary.txt

將 ./data 中 1095 個 txt 檔案依序讀入，對於單一個 document，先使用前述process\_document的函數進行 tokenization，獲得單一 document 的 tokens 之後，用 set 轉換為 unique\_tokens，並更新整個 collection 的 document frequency。

document\_frequency = {}

# Iterate through all the files in the dataset directory

for filename in os.listdir('./data'):

    if filename.endswith('.txt'):

        filepath = os.path.join('./data', filename)

        with open(filepath, 'r', encoding='utf-8') as f:

            text = f.read()

        tokens = process\_document(text)

        # Update document frequency - only count each term once per document

        unique\_tokens = set(tokens)

        for token in unique\_tokens:

            if token in document\_frequency:

                document\_frequency[token] += 1

            else:

                document\_frequency[token] = 1

最後以 ascending order 重新排序 document frequency，並存入 dictionary.txt 中

# Sort the terms in ascending order

sorted\_terms = sorted(document\_frequency.items(), key=lambda x: x[0])

# Save the dictionary and document frequency to a file

with open('dictionary.txt', 'w') as f:

    for index, (term, df) in enumerate(sorted\_terms, start=1):

        f.write(f"{index}\t{term}\t{df}\n")

***Transfer each document into a tfidf unit vector.***

* 1. 計算 term frequency（tf），從 ./data 中 1095 個 txt 檔案中，再次經由 process\_document 提取出單一 document 的 tokens，接著計算這些 document 的 term frequency。

# Build a term index dictionary for easy lookup

term\_index = {term: index for index, (term, df) in enumerate(sorted\_terms, start=1)}

# Now compute the tf-idf vectors for each document

for filename in os.listdir('./data'):

    if filename.endswith('.txt'):

        filepath = os.path.join('./data', filename)

        with open(filepath, 'r', encoding='utf-8') as f:

            text = f.read()

        tokens = process\_document(text)

        # Compute term frequency

        tf = {}

        for token in tokens:

            if token in tf:

                tf[token] += 1

            else:

                tf[token] = 1

接著，使用先前已經在計算 document frequency 時，建立好的 term\_index 以及 document frequency 來計算 tfidf 的值。

 # Create a zero vector of length equal to the number of terms

        tfidf\_vector = np.zeros(len(term\_index))

        for term, freq in tf.items():

            if term in term\_index:

                tf\_t = freq

                df\_t = document\_frequency[term]

                N = len(os.listdir('./data'))  # Assuming all files in the dataset directory are text documents

                idf\_t = math.log10(N / df\_t)

                tfidf\_t = tf\_t \* idf\_t

                tfidf\_vector[term\_index[term] - 1] = tfidf\_t  # -1 because indices start from 1

將計算計算好的 tfidf 值轉換為 unit-tfidf，並且防止除以零的情況。

# Normalize the tf-idf vector to unit length

        norm = np.linalg.norm(tfidf\_vector)

        if norm > 0:

            tfidf\_vector\_unit = tfidf\_vector / norm

        else:

            tfidf\_vector\_unit = tfidf\_vector  # avoid division by zero

最後將單一 document 中有的 term，依照 term index 以及 tfidf 的值，儲存在 {DocID.txt} 中。

        # Get non-zero entries for the sparse representation

        non\_zero\_entries = [(index + 1, tfidf) for index, tfidf in enumerate(tfidf\_vector\_unit) if tfidf > 0]

        # Save the tf-idf unit vector to a file

        doc\_id = os.path.splitext(filename)[0]  # Assuming filename is 'DocID.txt'

        with open(f'./output/{doc\_id}.txt', 'w') as f:

            f.write(f"{len(non\_zero\_entries)}\n")  # Write the number of non-zero entries

            for index, tfidf in non\_zero\_entries:

                f.write(f"{index}\t{tfidf:.3f}\n")  # Write the term index and tf-idf value, formatted to 3 decimal places

***Write a function cosine(Docx, Docy) which loads the tf-idf vectors of documents x and y and returns their cosine similarity.***

* 1. 最後，寫一個函數，先載入單一 document 的 unit tfidf vectors，然後再計算 cosine similarity。

def cosine(docx, docy):

    # Inline function to load vector

    def load(doc\_id):

        with open(f'./output/{doc\_id}.txt', 'r') as f:

            lines = f.readlines()

        vector = np.zeros(len(term\_index))  # Ensure all vectors are of same length as term\_index

        for line in lines[1:]:

            index, tfidf = line.strip().split()

            index = int(index) - 1  # Indices start from 1 in the file

            tfidf = float(tfidf)

            vector[index] = tfidf

        return vector

    vector\_x = load(docx)

    vector\_y = load(docy)

    # The vectors are already normalized (unit vectors), so just compute the dot product.

    cosine\_similarity = np.dot(vector\_x, vector\_y)

    return cosine\_similarity

以下為範例用法，輸出為Cosine similarity between 1 and 2: 0.195

# Example usage:

docx = '1'

docy = '2'

similarity = cosine(docx, docy)

print(f'Cosine similarity between {docx} and {docy}: {similarity:.3f}')