**EXTRA CREDIT DONE - Inverted Index (Term Dictionary) Construction with TF-IDF for a Full Feature Set for Each Document. Page (11- 13)**

**Name: Temitope Adeniyi CSU ID: 2866307**

**Programming Language: Python**

**Part 1: Preprocessing to Build Document Vectors for Web Page Content Analysis**

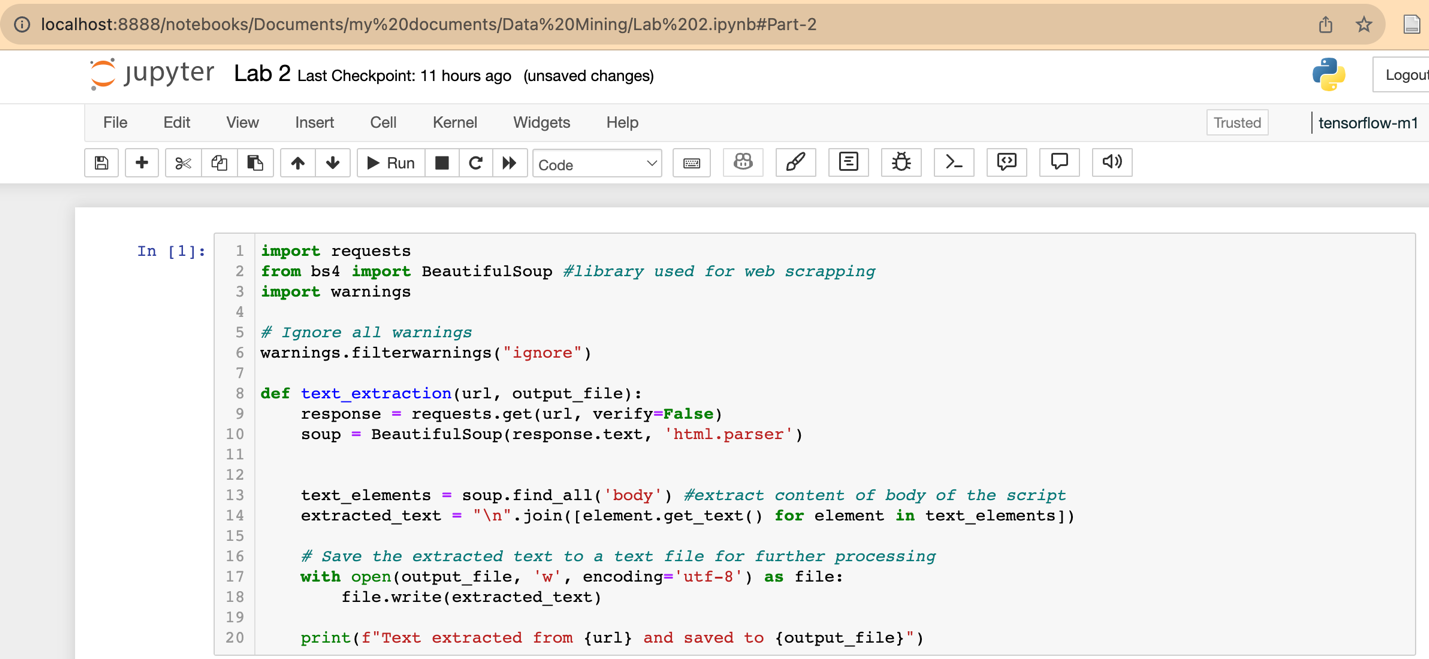
1. **Extracting text from web page.**

**Library used:**

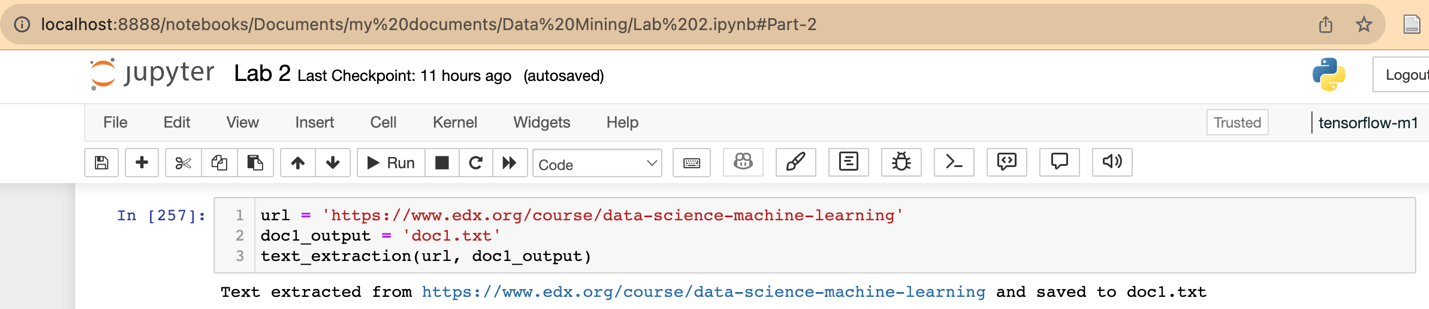
* **requests:** This library is used to send HTTP requests to the provided URL and retrieve the web page's content.
* **BeautifulSoup:** It is a library for web scraping, used to parse and extract data from HTML documents.
* **warnings:** This library is imported to suppress any warning messages during the execution of the script.

Given by the code below, a function designed to extract text from a web page using web scraping techniques. It utilizes the requests library to retrieve the content of a given URL of the different documents and the BeautifulSoup library to parse and extract textual data from the <body> element of the HTML, where the main content of a web page is typically located. This will be done each of the document web page's HTML structure. The extracted text is then saved to a text file for further processing or analysis.

The code below creates a function called text\_extraction that extract only text contents in the body of the script parsing other irrelevant html scripts, anchor or tags.



Doc1 is extracted from the webpage in the code below and saved into doc1.txt.



Doc2 is extracted in the code below and saved into doc2.txt.



Doc3 is extracted in the code below and saved into doc3.txt.



Doc4 is extracted in the code below and saved into doc4.txt.



Doc5 is extracted in the code below and saved into doc5.txt.



Doc6 is extracted in the code below and saved into doc6.txt.

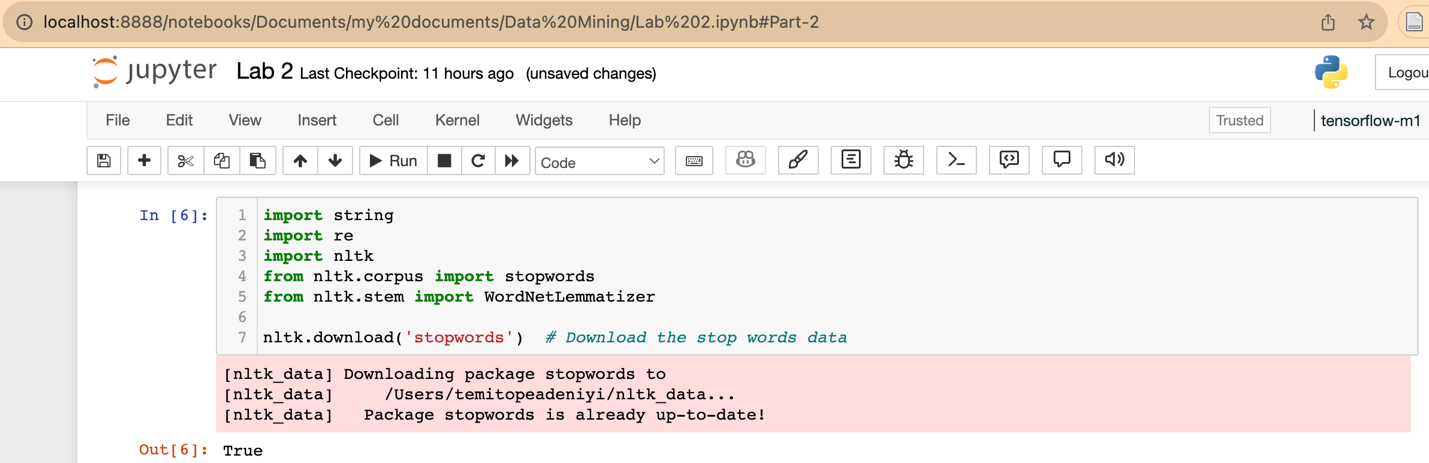


1. **Preprocessing text**

The code below set up the environment for text preprocessing by importing the necessary library used to preprocess the extracted text. This also includes downloading the NLTK stop words resources that will be used to identify and remove stop words.

**Library used:**

* **String:** The string module provides a collection of string constants and functions for string manipulation.
* **re:** The re module is used for regular expression operations.
* **nltk:** The Natural Language Toolkit library for working with human language data.
* **nltk.stem:** used to import NLTK WordNetLemmatizer used for stemming



These are the preprocessing tasks done:

1. Remove punctuation and symbols.
2. Remove stop words.
3. Change text to lower case.
4. Stemming- lemmatization

These preprocessing steps are all in the function process\_text defined in the code below. This function performs all these steps in order and returns a cleaned\_lower\_text for each of the documents

A screenshot of a computer

Description automatically generated

The code below read the extracted text for doc1 and then perform preprocessing using the function process\_text

A screenshot of a computer

Description automatically generated

The code below read the extracted text for doc2 and then perform preprocessing using the function process\_text

A screenshot of a computer

Description automatically generated

The code below read the extracted text for doc3 and then perform preprocessing using the function process\_text

A screenshot of a computer

Description automatically generated

The code below read the extracted text for doc4 and then perform preprocessing using the function process\_text

A screenshot of a computer

Description automatically generated

The code below read the extracted text for doc5 and then perform preprocessing using the function process\_text

A screenshot of a computer

Description automatically generated

The code below read the extracted text for doc6 and then perform preprocessing using the function process\_text

A screenshot of a computer

Description automatically generated

1. **Counting Terms (TF and DF)**

The code below imports all the necessary library used for counting the term occurrences for the topics in each document.

Topic to count: [research, data, mining, analytics, data mining, machine learning, deep learning]

**Library Used:**

* **nltk.tokenize:** tokenization involves breaking down a document into individual words, phrases, or tokens . NLTK's word\_tokenize function is used for this purpose. It takes a text input and returns a list of tokens. This will prepare the document for counting the occurrence of words.
* **nltk bigrams:** Thisis used to import the functions for generating bigrams from text data. These functions are part of the NLTK (Natural Language Toolkit) library and will be used for identifying the 3 bigrams words "data mining", "machine learning", "deep learning" in the document.
* **Pandas:** This library is used to create pandas dataframe to store term counts

A screenshot of a computer

Description automatically generated

A function count\_terms is then created that processes a the documents to count the frequency of specific terms uni-grams, and bi-grams within them. The main objective of this code is to produce two DataFrames:

Term\_freq\_df: A DataFrame that contains the term counts, including uni-grams and bi-grams, for each document.

doc\_freq\_df: A DataFrame that shows the document frequency of each term across the entire document collection.

**Processing followed in count\_terms function (tf and df)**

1. The code begins by defining two sets of terms: keywords and bi\_grams. Keywords represent the uni-gram terms, while bi-grams consist of the terms in pairs. These terms are the elements that will be counted in the documents.
2. The code then iterates through the collection of the documents, tokenizes each document into words using the word\_tokenize function, and converts the words to lowercase to ensure case-insensitivity.
3. For each document, the code counts the frequency of both the uni-gram keywords and bi-grams across the entire document collection and updates the doc\_frequency dictionary accordingly.
4. The counts of one-gram keywords and bi-grams for each document are aggregated into the Term\_freq\_df DataFrame. The document number is also included.
5. The code returns both the Term\_freq\_df and doc\_freq\_df DataFrames as results.

A screenshot of a computer

Description automatically generated

The function count\_terms is used to count the topics for all the six documents. The results for the term frequency and document frequency are save in Term\_freq and doc\_freq respectively.

A screenshot of a computer

Description automatically generated

The code below also shows the output of the Document frequency (df) of each uni-gram and bi-gram topic in the document.

A screenshot of a computer

Description automatically generated

The code below shows the output of the Term frequency (tf) of each uni-gram amd bi-gram topic in the document.

A screenshot of a computer

Description automatically generated

Document Frequency (DF)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | **Term** | **Document Frequency** | | --- | --- | --- | | research | | 6 | | data | | 6 | | mining | | 4 | | analytics | | 4 | | data mining | | 3 | | machine learning | | 4 | | deep learning | | 4 | |

Term Frequency (TF)

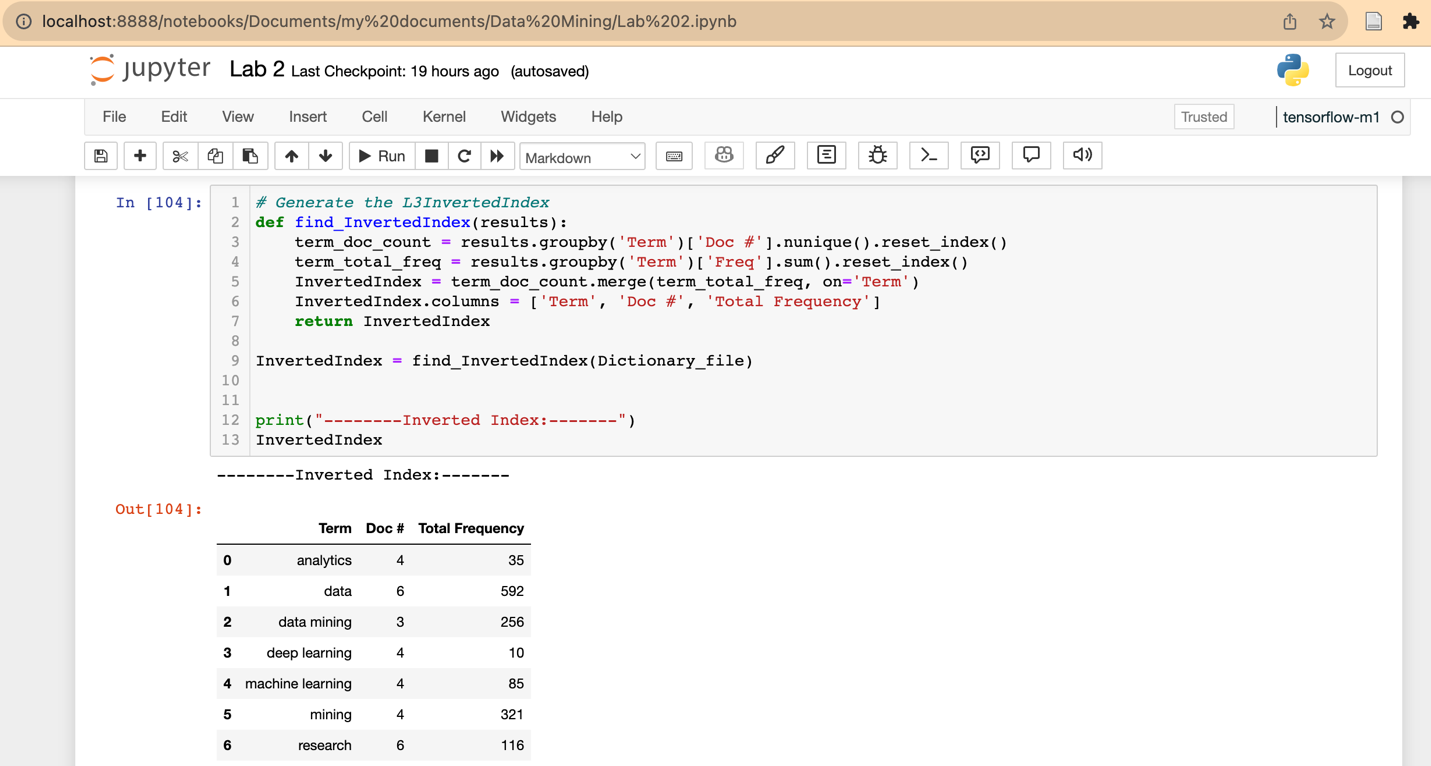
| **Document** | **research** | **data** | **mining** | **analytics** | **data mining** | **machine learning** | **deep learning** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Doc 1 | 7 | 24 | 0 | 1 | 0 | 13 | 0 |
| Doc 2 | 18 | 3 | 8 | 0 | 0 | 0 | 0 |
| Doc 3 | 38 | 1 | 0 | 0 | 0 | 0 | 2 |
| Doc 4 | 10 | 246 | 155 | 6 | 127 | 24 | 1 |
| Doc 5 | 10 | 246 | 155 | 6 | 127 | 24 | 1 |
| Doc 6 | 33 | 72 | 3 | 22 | 2 | 24 | 6 |

The code below defines a function that counts the frequency of the terms.

A screenshot of a computer

Description automatically generated

The code below defines a function that calculates the number of documents in which each term appears and the total frequency of each term across all documents resulting in a InvertedIndex dataframe.



The output of the dictionary file table and the inverted index is shown below as a table with both as subplots.

A screenshot of a computer

Description automatically generated

**Calculating IDF**   
The idf is calculated using the formula below;

Where N = 6 (Total number of documents)

df = document frequency of each term

The code below shows the code calculating idf, the output is a dataframe that includes the term, number of document containing the term, total frequency and the idf value for each term

Term **“data”** and **“research”** has an idf of Zero showing that each term exists in all the six documents. One might say that these terms is not discriminating enough to be considered important or unique topics.

A screenshot of a computer

Description automatically generated

**Calculating TF-IDF**

TF-IDF was calculated with the formula

The value of zero for the terms **“data”** and **“research”** for all the documents is because both terms have an idf of zero calculated from

**A screenshot of a computer

Description automatically generated**

**Part 2 – Cosine Similarity**

Cosine similarity is calculated using the formula below

The code below shows a function named cosine\_similarity for the calculation of the cosine similarity using the term frequency (document vector) table.

A screenshot of a computer

Description automatically generated

The code below show the output of the cosine similarity calculated for the pairwise documents.

A screenshot of a computer

Description automatically generated

**Part 3: Analysis and Discussion of Problems**

The cosine similarity correctly indicates the similarity between the documents

* Doc4 and Doc5 shows a cosine similarity of 1, which means both document are identical

Doc4 is the text from the webpage <https://en.wikipedia.org/wiki/Data_mining>

and Doc5 is from the webpage <https://en.wikipedia.org/wiki/Data_mining#Data_mining>

Manually inspecting these documents shows that both are the same webpage. Therefore, the cosine similarity of 1 is valid.

* Topics of Doc6 is similar to Topics of Doc4 and Doc5 because the cosine similarity between both Doc4 and Doc5 and Doc6 is 0.70 which indicates a high similarity.

Also, in terms of term frequency Doc6 has 72 counts of the term **“data”** with both Doc4 and Doc5 having the count of 246 this is more similar than the other documents. Also, Doc6 has the same count of 24 with Doc4 and Doc5 for the bi-gram term **“machine learning”.** Doc6 also a count of 33 for the term **“research”** while for Doc4 and Doc5 has a count of 10 for both.