S.I.E.S College of Arts, Science and Commerce (Empowered Autonomous)

Sion(W), Mumbai – 400 022.

DEPARTMENT OF INFORMATION TECHNOLOGY

MSc (IT), SEMESTER I

Practical Journal

For the Subject: -

Web Mining – I

Submitted by YOGESH CHATROORAM SAHU

FMIT2526179

For the Academic Year

2025-2026

College of Arts, Science & Commerce

SIES

RISE WITH EDUCATION

NAAC REACCREDITED - ‘A’ GRADE

ISO 9001 : 2008

S.I.E.S College of Arts, Science and Commerce (Empowered Autonomous) Sion(W), Mumbai – 400 022.

CERTIFICATE

This is to certify that Mr. **YOGESH CHATROORAM SAHU**, of **MSc [Information Technology] Semester - I**, Seat No. **FMIT2526179** has successfully completed the necessary course of experiments in the subject of **Web Mining-I** as a partial fulfilment of the degree **M.Sc. (I.T.)** during the academic year **2025-2026**.

Asst. Prof. In-Charge
MR. RAJESH YADAV

Examination date:

Examiner’s Signature & Date:

Signature of HOD
Sudha Bhagavatheeswaran College Seal Date:

INDEX

Practical.No.	Title	Signature
1	Write a program for Pre-processing of Text Document A1) Stop Word Removal. A2) Tokenization and Filtering.	
2	Write a program to implement Boolean retrieval model for the given document.	
3	Write a program to implement Vector Space Model and compute cosine Similarity between a query and documents.	
4	Write a program to detect spam keywords (keyword stuffing) in web content using word frequency analysis.	
5	Write a program to perform Text summarization using Extractive(LexRank) and Abstractive (Transformers) methods.	
6	Write a program to create an inverted index for a collection of documents.	
7	Write a program to identify opinion spam in user reviews.	

Practical No.1

Aim – Write a program for Pre-processing of a Text Document.

A1) To remove stopwords (stop word removal)

Description:

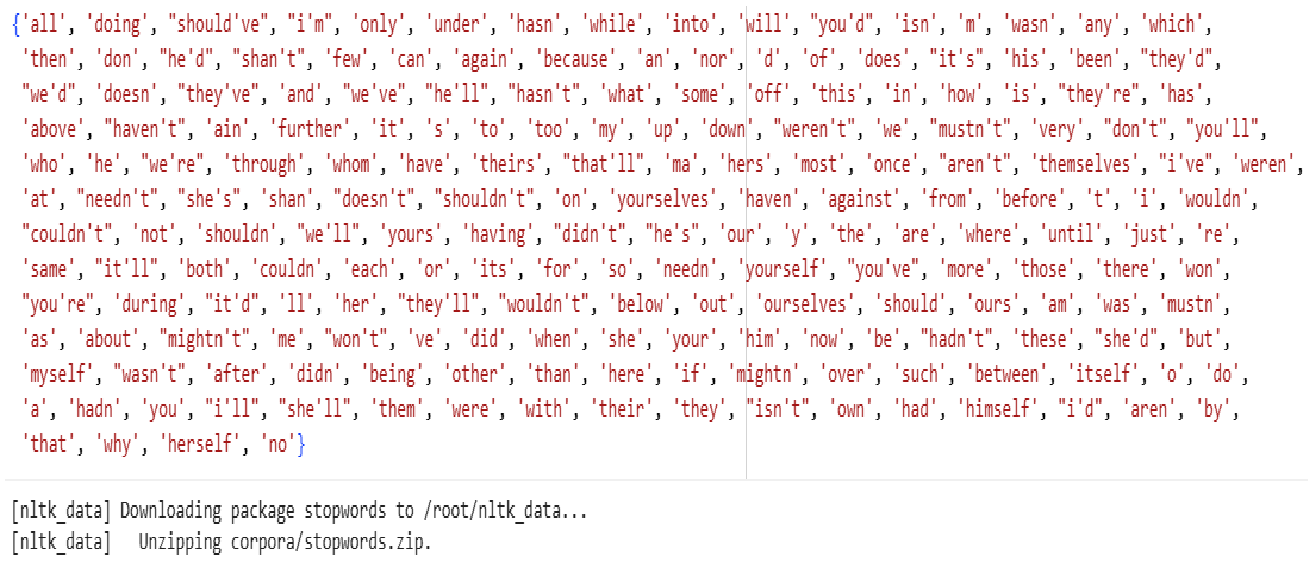
Stopwords are common words such as *"is, the, in, and"* which occur frequently in text but carry very little meaning. Removing them helps reduce data size and improves the focus on important words.

- **NLTK (Natural Language Toolkit):** A Python library used for text processing tasks.
- **corpus:** A collection of datasets (like stopwords) provided by NLTK.
- **set ():** Converts a list of stopwords into a set for faster lookup and to avoid duplicates.

Code: -

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

# Load English stopwords
stop_words = set(stopwords.words('english'))

# Display stopwords
print(stop_words)
```

Output: -



```
{'all', 'doing', "should've", "i'm", 'only', 'under', 'hasn', 'while', 'into', 'will', "you'd", 'isn', 'm', 'wasn', 'any', 'which',
 'then', 'don', "he'd", "shan't", 'few', 'can', 'again', 'because', 'an', 'nor', 'd', 'of', 'does', "it's", 'his', 'been', "they'd",
 "we'd", 'doesn', "they've", 'and', "we've", "he'll", "hasn't", 'what', 'some', 'off', 'this', 'in', 'how', 'is', "they're", 'has',
 'above', "haven't", 'ain', 'further', 'it', 's', 'to', 'too', 'my', 'up', 'down', "weren't", 'we', "mustn't", 'very', "don't", "you'll",
 'who', 'he', "we're", 'through', 'whom', 'have', 'theirs', "that'll", 'ma', 'hers', 'most', 'once', "aren't", 'themselves', "i've", 'weren',
 'at', "needn't", "she's", 'shan', "doesn't", "shouldn't", 'on', 'yourselves', 'haven', 'against', 'from', 'before', 't', 'i', 'wouldn',
 "couldn't", 'not', 'shouldn', "we'll", 'yours', 'having', "didn't", "he's", 'our', 'y', 'the', 'are', 'where', 'until', 'just', 're',
 'same', "it'll", 'both', 'couldn', 'each', 'or', 'its', 'for', 'so', 'needn', 'yourself', "you've", 'more', 'those', 'there', 'won',
 "you're", 'during', "it'd", 'll', 'her', "they'll", "wouldn't", 'below', 'out', 'ourselves', 'should', 'ours', 'am', 'was', 'mustn',
 'as', 'about', "mightn't", 'me', "won't", 've', 'did', 'when', 'she', 'your', 'him', 'now', 'be', "hadn't", 'these', "she'd", 'but',
 'myself', "wasn't", 'after', 'didn', 'being', 'other', 'than', 'here', 'if', 'mightn', 'over', 'such', 'between', 'itself', 'o', 'do',
 'a', 'hadn', 'you', "i'll", "she'll", 'them', 'were', 'with', 'their', 'they', "isn't", 'own', 'had', 'himself', "i'd", 'aren', 'by',
 'that', 'why', 'herself', 'no'}

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

A2) To tokenize and filter out sentence

Description:-

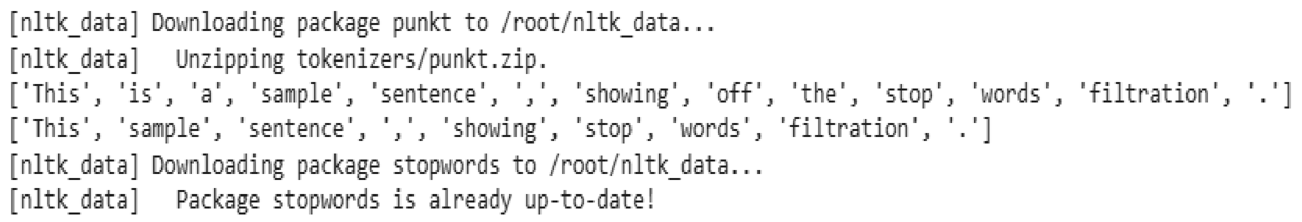
Tokenization is the process of breaking text into smaller units called *tokens* (words). After tokenization, stopwords are removed so that only meaningful words remain.

- **word_tokenize():** Splits text into individual words.
- **Filtering:** Removes stopwords from the tokenized list.
- **append():** Adds words to a list one by one.

Code :-

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
example_sent = "This is a sample sentence, showing off the stop words filtration."
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)
# Removed duplicate reinitialization
filtered_sentence = []
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
print(word_tokens)
print(filtered_sentence)
```

Output:-



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words', 'filtration', '.']
['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Practical No.02

Aim: write a program to implement boolean retrieval model for the given example.

Example Documents:-

- **Doc1:** Information Retrieval has 2 models and information.
- **Doc2:** Boolean is a basic Information Retrieval classic model.
- **Doc3:** Information is a data that processed, Information.
- **Doc4:** When a Data Processed the result is Information, Data.

Query (Q): $(Data \wedge Information) \vee (\neg Retrieval)$

Description:

The **Boolean Retrieval Model** is one of the simplest retrieval models in Information Retrieval. Documents are represented as sets of words (terms), and queries are expressed using Boolean operators.

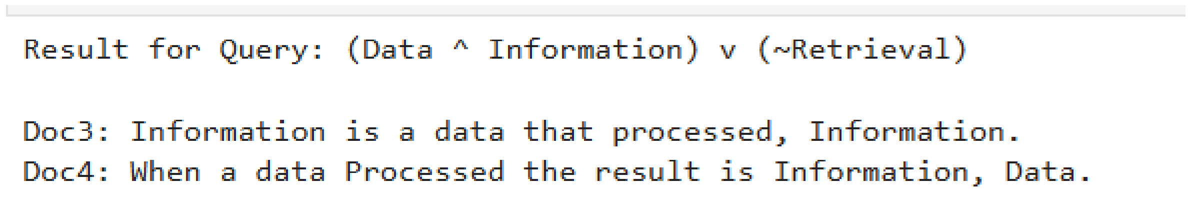
- **Operators used:**
  - **AND (∧)** → Retrieves documents containing both terms.
  - **OR (∨)** → Retrieves documents containing at least one of the terms.
  - **NOT (¬)** → Excludes documents containing the specified term.

- **Python concepts used:**
  - **import re** – Loads Python's regular expression module, used for text pattern matching.
  - **re.findall()** – Finds all words (tokens) in a document by matching word patterns.
  - **lower()** – Converts text into lowercase to make search case-insensitive.
  - **set(docs.keys())** – Converts document keys into a set for easy logical operations.
  - **has_term()** – A helper function that checks whether a word (term) exists in a document.

Code:-

```
import re

docs = {
    1: "Information Retrieval has 2 models and information.",
    2: "Boolean is a basic Information Retrieval classic model.",
    3: "Information is a data that processed, Information.",
    4: "When a Data Processed the result is Information, Data."
}

# All document IDs
all_docs = set(docs.keys())

# Terms we are interested in
terms = ["information", "data", "retrieval"]

# Function to check if a term exists in a document
def has_term(doc, term):
    words = set(re.findall(r'\w+', doc.lower()))
    return term in words

# Collect documents containing each term
data_docs = set(i for i, doc in docs.items() if has_term(doc, "data"))
info_docs = set(i for i, doc in docs.items() if has_term(doc, "information"))
retrieval_docs = set(i for i, doc in docs.items() if has_term(doc, "retrieval"))

# Evaluate query: (Data AND Information) OR (NOT Retrieval)
result = (data_docs & info_docs) | (all_docs - retrieval_docs)

print("Result for Query: (Data ^ Information) v (~ Retrieval)\n")
for i in sorted(result):
    print(f"Doc{i}:", docs[i])
```

Output:-



```
Result for Query: (Data ^ Information) v (~Retrieval)

Doc3: Information is a data that processed, Information.
Doc4: When a data Processed the result is Information, Data.
```

Practical No.03

Aim : Write a program to implement vector space model.

Description:

The **Vector Space Model (VSM)** is an important model in Information Retrieval. It represents both documents and queries as vectors in a common vector space. The similarity between a document and a query is calculated using **Cosine Similarity**.

1. **Math Module** – Provides mathematical functions like sqrt() which is used to calculate vector length (magnitude).
2. **Collections Module** – Provides Counter, which counts word frequencies in documents.
3. **Vectorization** – Each document or query is converted into a vector of term frequencies.
4. **Cosine Similarity Formula:**

$$\text{Cosine Similarity} = \frac{A \cdot B}{|| A || \times || B ||}$$

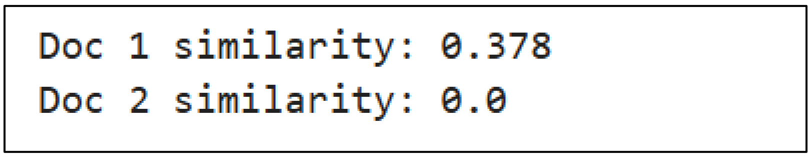
   - **A · B** → Dot product of vectors A and B
   - **||A||, ||B||** → Magnitude (length) of vectors

- If similarity = **1**, the query and document are very similar.
- If similarity = **0**, they are not similar.

Code :-

```
import math, collections

docs = ["A man and a woman.", "A baby."]
vocab = sorted(set(w.lower().strip('.,') for d in docs for w in d.split()))

def vectorize(text):
    c = collections.Counter(w.lower().strip('.,') for w in text.split())
    return [c[t] for t in vocab]

def cosine_sim(a, b):
    dot = sum(x * y for x, y in zip(a, b))
    mag_a = math.sqrt(sum(x * x for x in a))
    mag_b = math.sqrt(sum(y * y for y in b))
    return dot / (mag_a * mag_b) if mag_a and mag_b else 0

query = "woman"
q_vec = vectorize(query)
doc_num = 1
for doc in docs:
    sim = cosine_sim(vectorize(doc), q_vec)
    print("Doc", doc_num, "similarity:", round(sim, 3))
    doc_num += 1
```

Output :-



```
Doc 1 similarity: 0.378
Doc 2 similarity: 0.0
```

Practical no: 04

Aim : write a program for implementing web spamming.

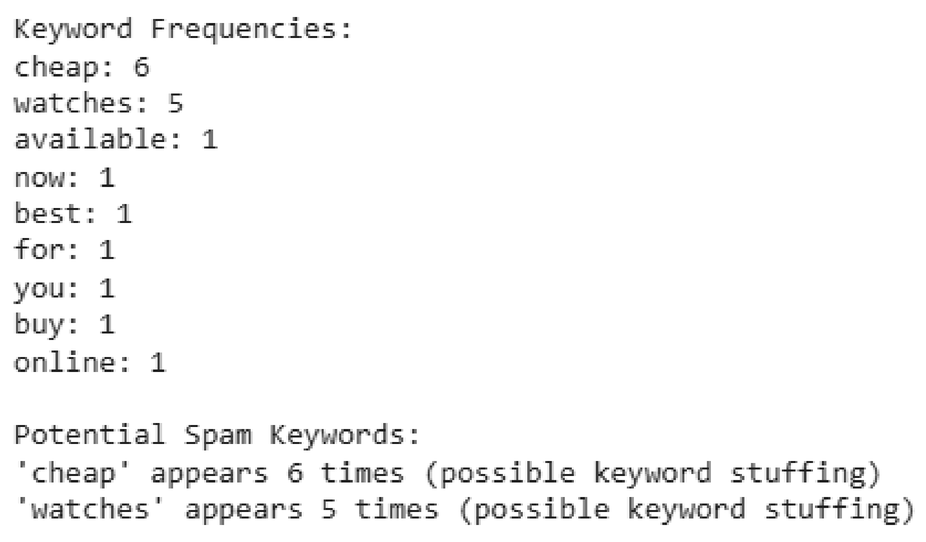
Description:

1. **Web Spamming:**
   Web spamming is the use of unfair techniques to manipulate search engine rankings and attract more visitors. Common methods include:
   - **Keyword stuffing:** Repeating important words too many times.
   - **Hidden content/links:** Adding invisible text or links for search engines.
   - **Link farming:** Creating fake backlinks to boost rankings.
   - **Deceptive content:** Misleading users with irrelevant or low-quality material.
2. **Collections Module:**
   Python’s collections module provides advanced data structures, making tasks like counting words easier and faster.
3. **Counter Class:**
   Counter counts the frequency of items in a list.
   Example: ['apple', 'apple', 'banana'] → apple:2, banana:1
4. **Regular Expressions (re):**
   The re module helps search for patterns in text efficiently. It can find words, numbers, or email patterns without scanning manually.

CODE :-

```
from collections import Counter
import re

web_content = """Cheap watches available now! Best cheap watches for you.
Buy cheap watches online. Cheap cheap cheap watches watches!"""

# Extract words (lowercase, ignore punctuation)
words = re.findall(r'\b\w+\b', web_content.lower())

# Count word frequencies
keyword_counts = Counter(words)

# Threshold for spam detection
SPAM_THRESHOLD = 4

# Print frequencies
print("Keyword Frequencies:")
for word, count in keyword_counts.items():
    print(f"{word}: {count}")

# Detect potential spam keywords
print("\nPotential Spam Keywords:")
for word, count in keyword_counts.items():
    if count >= SPAM_THRESHOLD:
        print(f"'{word}' appears {count} times (possible keyword stuffing)")
```

OUTPUT :-



```
Keyword Frequencies:
cheap: 6
watches: 5
available: 1
now: 1
best: 1
for: 1
you: 1
buy: 1
online: 1

Potential Spam Keywords:
'cheap' appears 6 times (possible keyword stuffing)
'watches' appears 5 times (possible keyword stuffing)
```

Practical No:05

Aim : Write a program to implement summarization.

Theory :

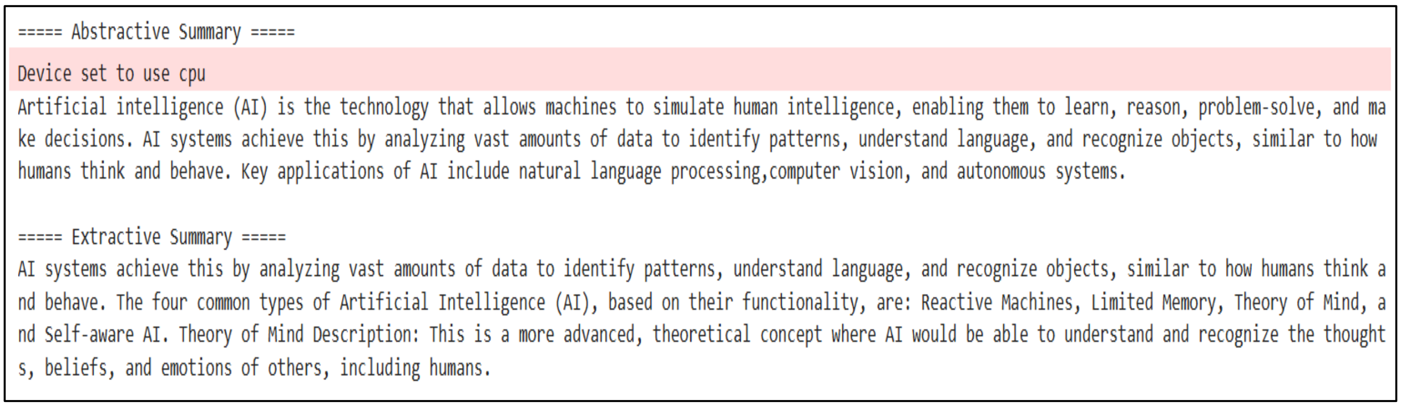
1. **Summarization:**
   Summarization is the process of condensing a large text into a shorter version while keeping its main meaning. It helps quickly understand articles, reports, or research papers. There are two main types:
   - **Extractive Summarization:** Selects the most important sentences directly from the text. Techniques like frequency counts or graph-based ranking (e.g., LexRank) are used.
   - **Abstractive Summarization:** Generates new sentences in human-like wording to convey the main ideas. Transformer models are commonly used.
2. **PlaintextParser (sumy.parsers.plaintext):**
   Parses raw text into a structured format suitable for summarization. It can read text from strings or files.
3. **Tokenizer (sumy.nlp.tokenizers):**
   Breaks text into sentences or words. Essential for analyzing text and improving summarization accuracy.
4. **LexRankSummarizer (sumy.summarizers.lex_rank):**
   Implements a graph-based algorithm to select important sentences for extractive summarization. Produces concise summaries.
5. **Transformers Pipeline:**
   High-level API for NLP tasks like summarization. Uses pre-trained models for abstractive summarization, generating readable and coherent summaries.
6. **Libraries:**
   - sumy – For extractive summarization.
   - transformers – For abstractive summarization using transformer models.
   - torch – Required backend for transformer models.

Code :-

```
# Install necessary packages (for Jupyter/Colab)
!pip install sumy transformers torch

import nltk
nltk.download('punkt_tab')
from sumy.parsers.plaintext import PlaintextParser


from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lex_rank import LexRankSummarizer
from transformers import pipeline
```

```
# Sample text
text = """ Artificial intelligence (AI) is the technology that allows machines to simulate human
intelligence, enabling them to learn, reason, problem-solve, and make decisions. AI systems
achieve this by analyzing vast amounts of data to identify patterns, understand language, and
recognize objects, similar to how humans think and behave. Key applications of AI include
natural language processing, computer vision, and autonomous systems, impacting various
industries by automating tasks and improving decision-making. The four common types of
Artificial Intelligence (AI), based on their functionality, are: Reactive Machines, Limited
Memory, Theory of Mind, and Self-aware AI. Reactive machines, like the IBM Deep Blue chess
program, act on the present situation but don't store memories. Limited Memory AI, such as self-
driving cars, can use past data to inform current decisions. Theory of Mind and Self-aware AI
are currently conceptual stages of AI that would possess human-like understanding of emotions
and consciousness, respectively.
"""

# Extractive Summarization
def extractive_summary(text, num_sentences=3):
    parser = PlaintextParser.from_string(text, Tokenizer("english"))
    summarizer = LexRankSummarizer()
    summary = summarizer(parser.document, num_sentences)
    return ' '.join(str(sentence) for sentence in summary)

# Abstractive Summarization
def abstractive_summary(text):
    summarizer = pipeline("summarization")
    summary = summarizer(text, max_length=100, min_length=25, do_sample=False)
    return summary[0]['summary_text']

# Run both summarizations
print("===== Abstractive Summary =====")
print(abstractive_summary(text))

print("\n===== Extractive Summary =====")
print(extractive_summary(text))
```

Output :-



```
===== Abstractive Summary =====
Device set to use cpu
Artificial intelligence (AI) is the technology that allows machines to simulate human intelligence, enabling them to learn, reason, problem-solve, and ma
ke decisions. AI systems achieve this by analyzing vast amounts of data to identify patterns, understand language, and recognize objects, similar to how
humans think and behave. Key applications of AI include natural language processing,computer vision, and autonomous systems.

===== Extractive Summary =====
AI systems achieve this by analyzing vast amounts of data to identify patterns, understand language, and recognize objects, similar to how humans think a
nd behave. The four common types of Artificial Intelligence (AI), based on their functionality, are: Reactive Machines, Limited Memory, Theory of Mind, a
nd Self-aware AI. Theory of Mind Description: This is a more advanced, theoretical concept where AI would be able to understand and recognize the thought
s, beliefs, and emotions of others, including humans.
```

Practical No:06

Aim : Write a program to implement inverted index

Theory:-

1. **Inverted Index:**
   An inverted index is a data structure widely used in search engines and information retrieval systems. It maps each word (or term) to the list of documents in which it appears. Unlike a normal index that stores documents sequentially, an inverted index allows quick lookup of documents containing a specific word.

Example:
If the word *"document"* occurs in Doc1, Doc3, and Doc5, the inverted index will store.

2. **defaultdict(set):**
   defaultdict is a special type of dictionary from Python's *collections* module. When a key is not present, it automatically initializes it with a default value. Using *defaultdict(set*) allows us to automatically create an empty set for new words, which helps in storing document IDs without duplicates.
   Use case: Building inverted indexes, adjacency lists in graphs, or grouping items.

3. **enumerate():**
   enumerate() is a built-in Python function that returns both the index and the value of items in an iterable. It is useful to keep track of document IDs while looping over a list of documents.

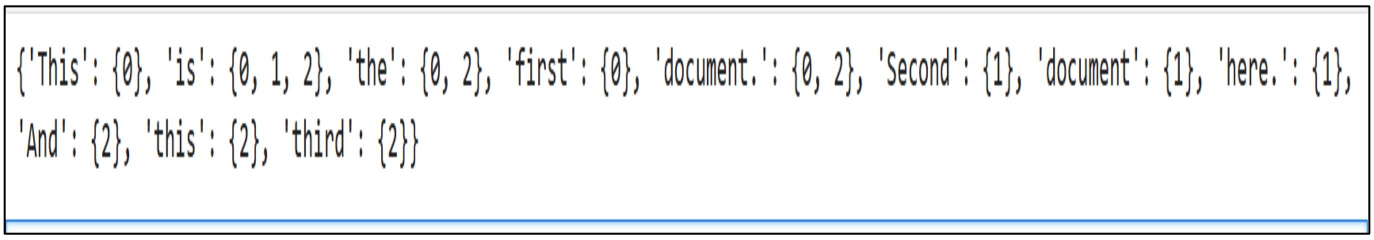
4. **split() and add():**
   - *split()* breaks a sentence into words (tokens) using whitespace by default.
   - *add(doc_id)* adds the current document ID to the set corresponding to a word, avoiding duplicate entries.

Code :-

```
from collections import defaultdict
# Function to create an inverted index
def create_inverted_index(documents):
    inverted_index = defaultdict(set)
    for doc_id, document in enumerate(documents):
        for word in document.split():
            inverted_index[word].add(doc_id)
    return inverted_index
```

```
# Sample documents
documents = [
    "This is the first document.",
    "Second document is here.",
    "And this is the third document."
]

# Create inverted index
inverted_index = create_inverted_index(documents)

# Display the inverted index
print(dict(inverted_index))
```

Output :-



```
{'This': {0}, 'is': {0, 1, 2}, 'the': {0, 2}, 'first': {0}, 'document.': {0, 2}, 'Second': {1}, 'document': {1}, 'here.': {1},
'And': {2}, 'this': {2}, 'third': {2}}
```

Practical No:07

Aim : Write a program to identify opinion spam in user reviews.

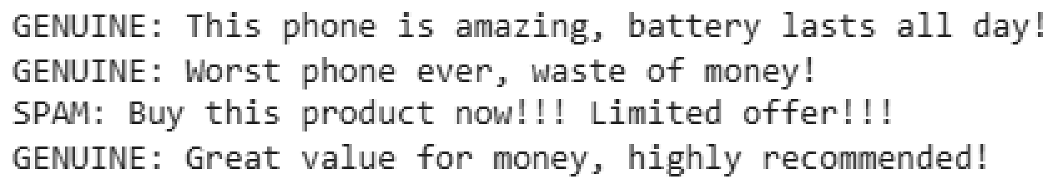
Theory:-

1. **Opinion Spam:**
   Opinion spam refers to fake or misleading reviews written to manipulate the perception of a product or service. Spam reviews may exaggerate benefits, provide false complaints, or include promotional messages. Detecting spam helps users trust online reviews and improves overall credibility.
   Examples of spam indicators:
   - Promotional phrases like "Buy now", "Limited offer", "Click here".
   - Excessive use of exclamation marks or repeated words.
   - Irrelevant content not related to the product.
2. **any():**
   - any() is a built-in Python function that returns True if **at least one element** of an iterable satisfies the condition.
   - Here, it is used to check if any spam keyword exists in a review.
3. **lower():**
   - Converts text to lowercase to ensure **case-insensitive matching**.
   - For example, "BUY NOW" and "buy now" are treated the same.
4. **Looping through reviews:**
   - Each review is checked against the list of spam keywords.
   - If any spam keyword is present, it is flagged as "SPAM". Otherwise, it is "GENUINE".

Code :-

```
# List of sample reviews
reviews = [
    "This phone is amazing, battery lasts all day!",
    "Worst phone ever, waste of money!",
    "Buy this product now!!! Limited offer!!!",
    "Great value for money, highly recommended!"
]

# List of spam keywords
spam_words = ["buy now", "limited offer", "click here", "free"]

# Detect spam reviews
for r in reviews:
    if any(word in r.lower() for word in spam_words):
        print("SPAM:", r)
    else:
        print("GENUINE:", r)
```

Output:-



```
GENUINE: This phone is amazing, battery lasts all day!
GENUINE: Worst phone ever, waste of money!
SPAM: Buy this product now!!! Limited offer!!!
GENUINE: Great value for money, highly recommended!
```