INDEX

Practical No.1

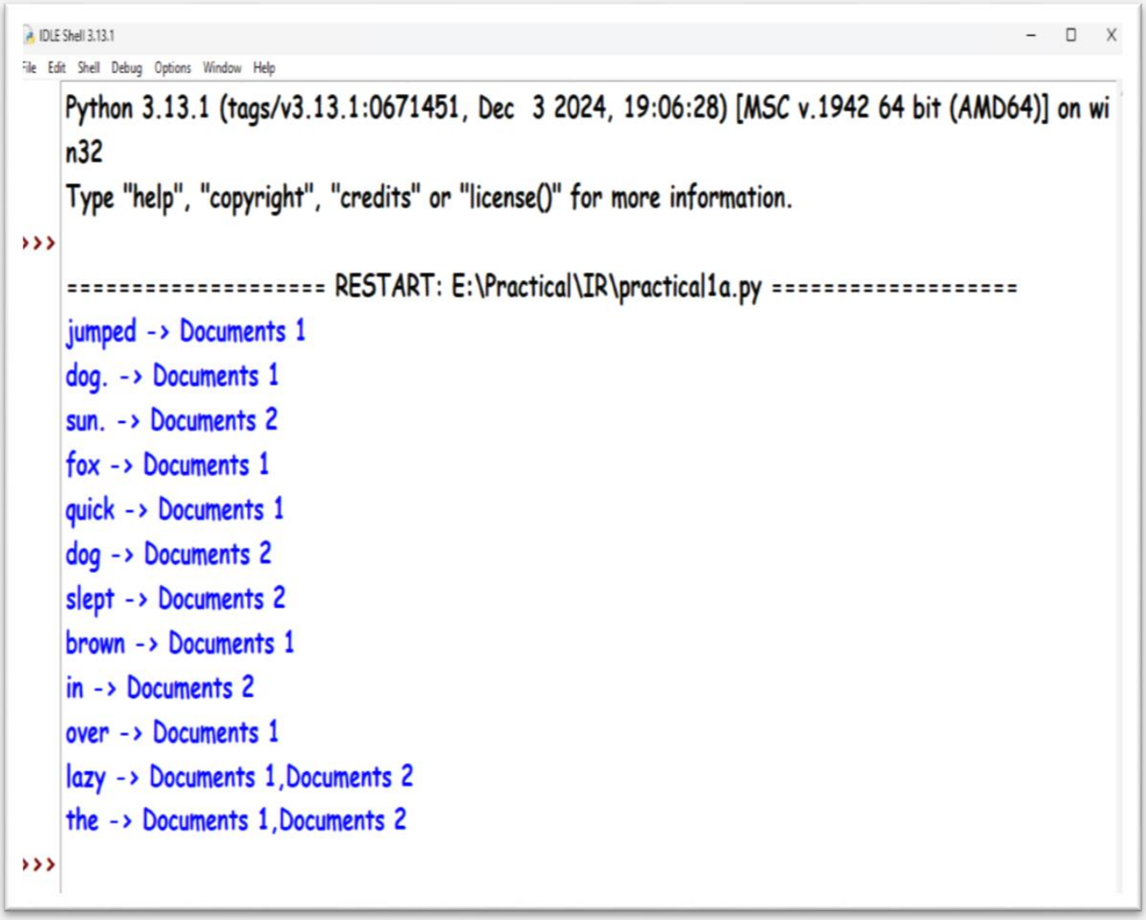
Aim:- Document Indexing and Retrieval

- **Implement an inverted index construction algorithm.**

Code:-

```
document1 ="The quick brown fox jumped over the lazy dog."
document2 = "The lazy dog slept in the sun."


tokens1 = document1.lower().split()
tokens2 = document2.lower().split()
terms = list(set(tokens1 + tokens2))


inverted_index = {}


for term in terms:
    documents = []


    if term in tokens1:
        documents.append("Documents 1")


    if term in tokens2:
        documents.append("Documents 2")

    inverted_index[term] = documents

for term, documents in inverted_index.items():
    print(term, "->",",".join(documents))
```

Output:-

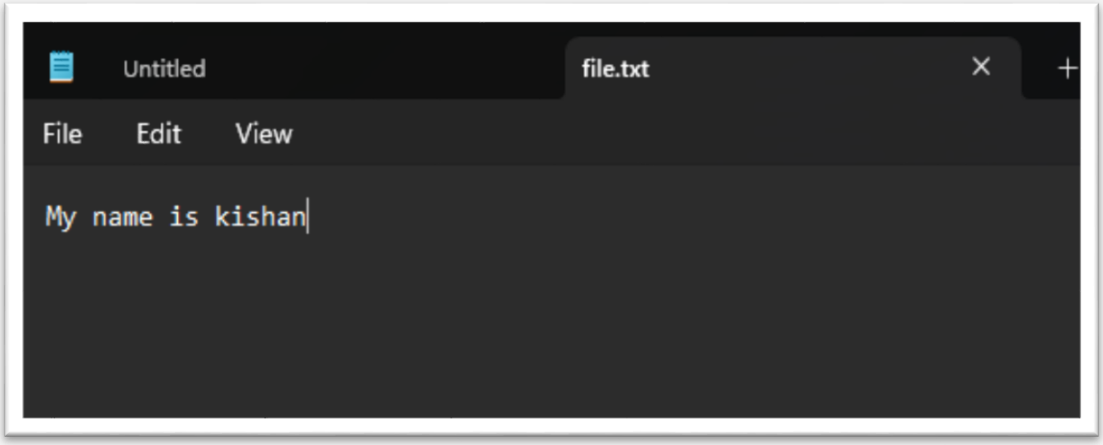


```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on wi
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: E:\Practical\IR\practical1a.py ===================
jumped -> Documents 1
dog. -> Documents 1
sun. -> Documents 2
fox -> Documents 1
quick -> Documents 1
dog -> Documents 2
slept -> Documents 2
brown -> Documents 1
in -> Documents 2
over -> Documents 1
lazy -> Documents 1,Documents 2
the -> Documents 1,Documents 2
>>>
```

- **Build a simple document retrieval system using the constructed index.**

1 B)

Step 1:- Create the text file and file.txt



```
My name is kishan
```

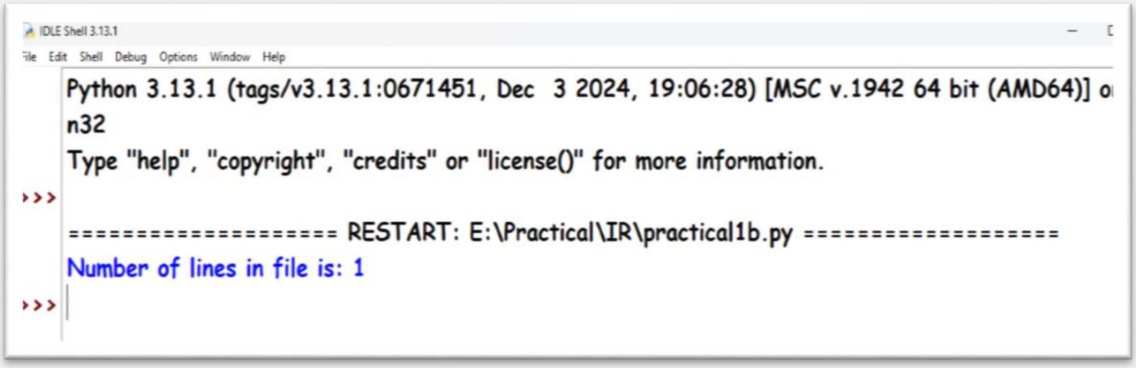
Step 2:- Write python code

Code:-

```
file =open('file.txt',encoding='utf8')
read = file.read()
file.seek(0)
read
line = 1
for word in read:
    if word == '\n':
        line += 1
print("Number of lines in file is:",line)
```

```
array=[]
for i in range(line):
    array.append(file.readline())
```

Output:-



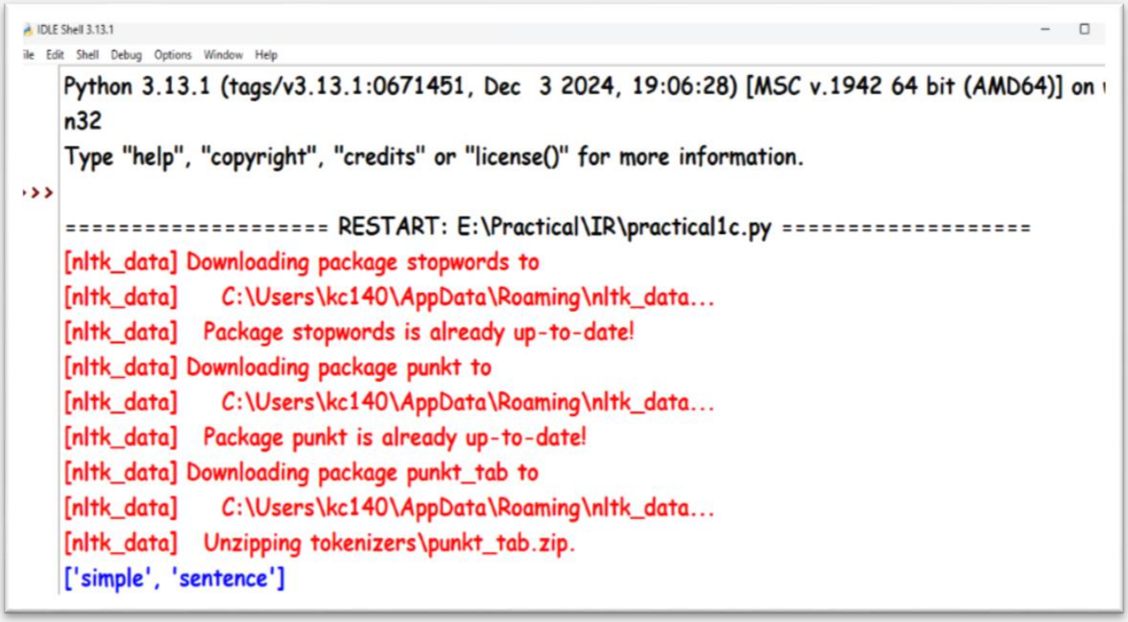
```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] o
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
==================== RESTART: E:\Practical\IR\practical1b.py ===================
Number of lines in file is: 1
>>> 
```

1C)

Code:-

```
from nltk.tokenize import word_tokenize
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('punkt_tab')

read = "This is a simple sentence"

text_tokens = word_tokenize(read)

tokens_without_sw = [word for word in text_tokens if
word.lower() not in stopwords.words('english')]
print(tokens_without_sw)
```

Output:-



```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on 
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: E:\Practical\IR\practical1c.py ===================
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\kc140\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\kc140\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\kc140\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt_tab.zip.
['simple', 'sentence']
```

Practical No.2

Aim:- Retrieval Models

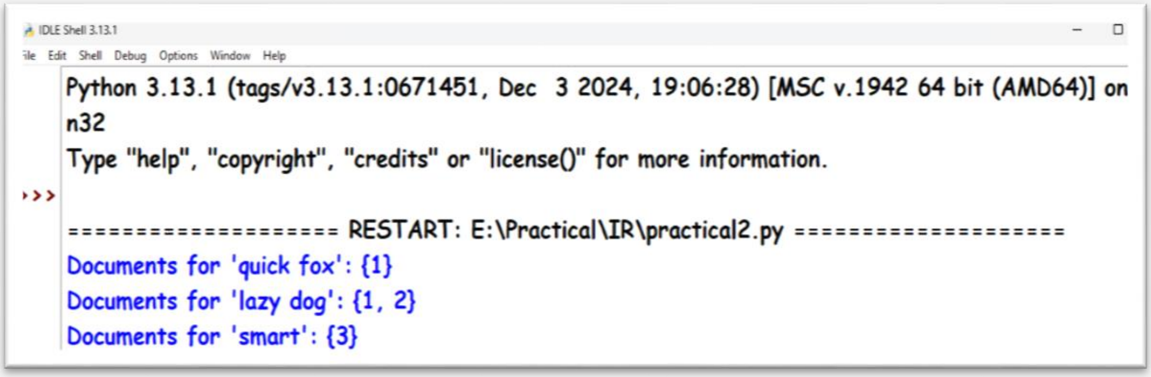
- **Implement the Boolean retrieval model and process queries.**

Code:-

```
from collections import defaultdict
import re
documents = {
    1: "The quick brown fox jumps over the lazy dog.",
    2: "Never jump over the lazy dog quickly.",
    3: "Foxes are quick and smart animals."
}
def preprocess(text):
    text = text.lower()
    text = re.sub(r'\W+', ' ', text)
    return text.split()
def build_inverted_index(docs):
    inverted_index = defaultdict(list)
    for doc_id, text in docs.items():
        words = preprocess(text)
        for word in set(words):
            inverted_index[word].append(doc_id)
    return inverted_index

inverted_index = build_inverted_index(documents)

def search(query, inverted_index):
    query_words = preprocess(query)
    result_sets = [ ]

    for word in query_words:
        result_sets.append(set(inverted_index.get(word, [ ])))

    if result_sets:
        return set.intersection(*result_sets)
    else:
        return set( )

query1 = "quick fox"
query2 = "lazy dog"
query3 = "smart"
```

```
print(f"Documents for '{query1}': {search(query1, inverted_index)}")
print(f"Documents for '{query2}': {search(query2, inverted_index)}")
print(f"Documents for '{query3}': {search(query3, inverted_index)}")
```

Output:-



```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: E:\Practical\IR\practical2.py ====================
Documents for 'quick fox': {1}
Documents for 'lazy dog': {1, 2}
Documents for 'smart': {3}
```

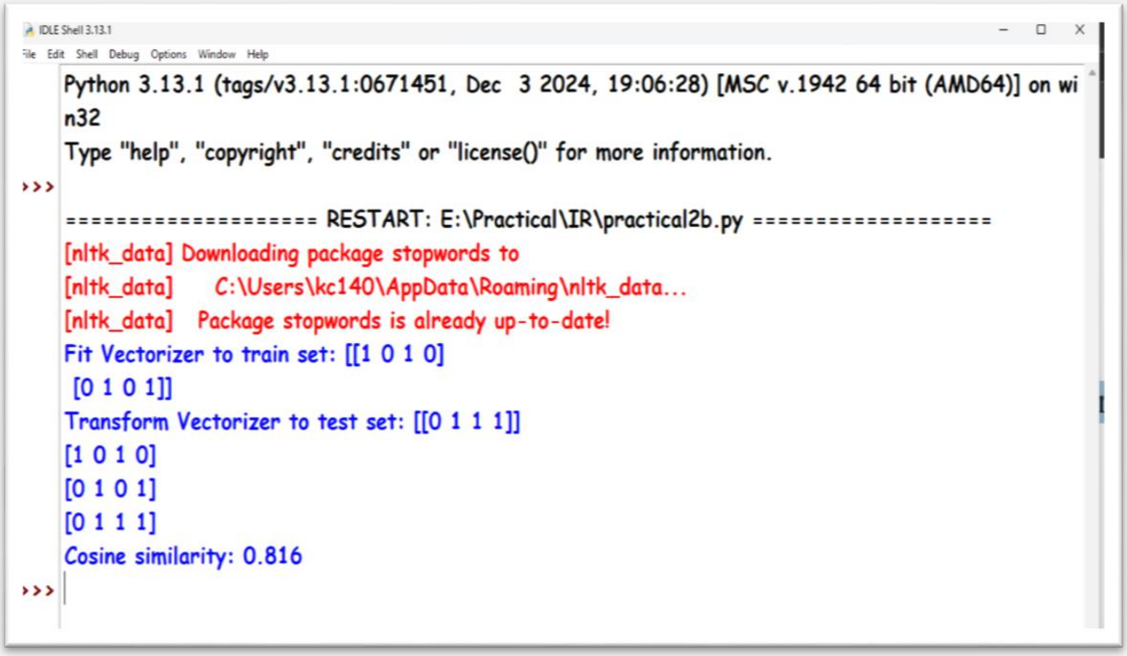
- **Implement the vector space model with TF-IDF weighting and cosine similarity.**

Code:-

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

import nltk

from nltk.corpus import stopwords

import numpy as np

from numpy.linalg import norm

train_set = ["The sky is blue.", "The sun is bright."]  # Documents

test_set = ["The sun in the sky is bright."]  # Query

nltk.download('stopwords')

stopWords = stopwords.words('english')

vectorizer = CountVectorizer(stop_words=stopWords)

transformer = TfidfTransformer()

trainVectorizerArray = vectorizer.fit_transform(train_set).toarray()

testVectorizerArray = vectorizer.transform(test_set).toarray()

print('Fit Vectorizer to train set:', trainVectorizerArray)

print('Transform Vectorizer to test set:', testVectorizerArray)
```

```
cx = lambda a, b: round(np.inner(a, b) / (norm(a) * norm(b)),
3)
for vector in trainVectorizerArray:
    print(vector)
for testV in testVectorizerArray:
    print(testV)
    cosine = cx(vector, testV)
print(f"Cosine similarity: {cosine}")
```

Output:-



```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on wi
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: E:\Practical\IR\practical2b.py ===================
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\kc140\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Fit Vectorizer to train set: [[1 0 1 0]
 [0 1 0 1]]
Transform Vectorizer to test set: [[0 1 1 1]]
[1 0 1 0]
[0 1 0 1]
[0 1 1 1]
Cosine similarity: 0.816
>>>
```

Practical No.3

Aim:- Spelling Correction in IR Systems

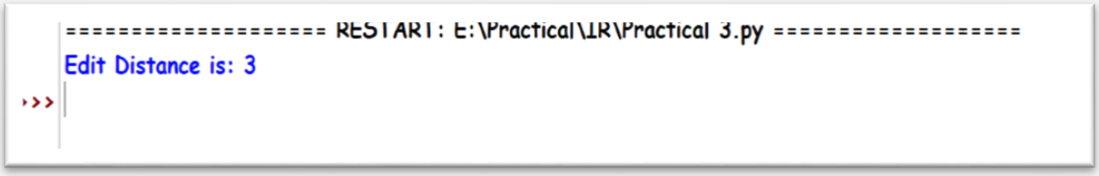
- **Develop a spelling correction module using edit distance algorithms.**
- **Integrate the spelling correction module into an information retrieval system**

Code:-

```
def editDistance(str1,str2, m, n):
    if m == 0:
        return n
    if n == 0:
        return m
    if str1[m-1] == str2[n-1]:
        return editDistance(str1, str2, m-1, n-1)
    return 1 + min(
        editDistance(str1, str2, m, n-1),
        editDistance(str1, str2, m-1, n),
        editDistance(str1, str2, m-1, n-1)
    )
str1 = "sunday"
str2 = "saturday"
```

```
print('Edit Distance is:', editDistance(str1, str2, len(str1), len(str2)))
```

Output:-



```
=================== RESTART: E:\Practical\IR\Practical 3.py ===================
Edit Distance is: 3
>>>
```

Practical No.4

Aim:- Evaluation Metrics for IR Systems.

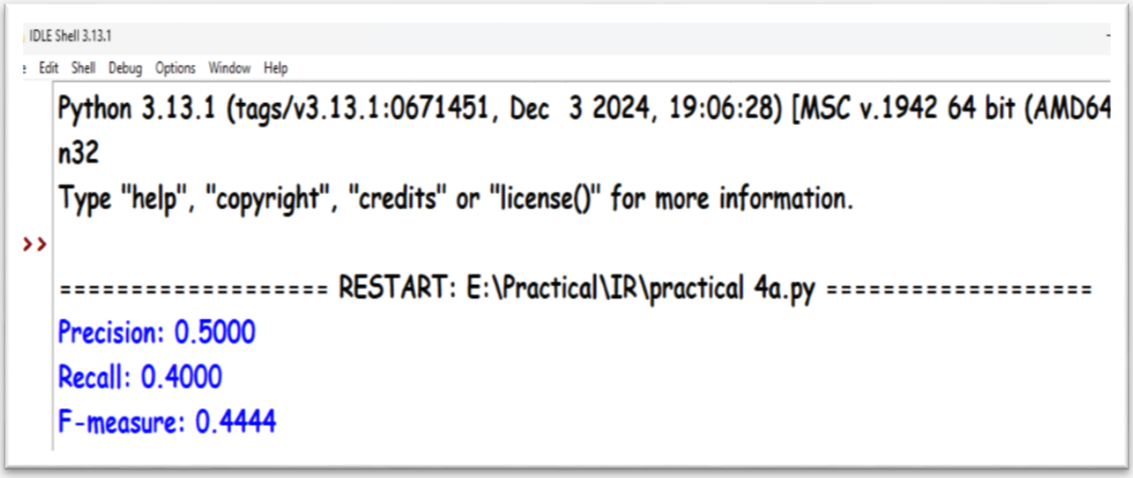
- **Calculate precision, recall, and F-measure for a given set of retrieval results.**

Code:-

```
def calculate_precision_recall_fmeasure(relevant_docs, retrieved_docs):
    relevant_set = set(relevant_docs)
    retrieved_set = set(retrieved_docs)
    relevant_retrieved = len(relevant_set & retrieved_set)
    precision = relevant_retrieved / len(retrieved_set) if len(retrieved_set) > 0 else 0
    recall = relevant_retrieved / len(relevant_set) if len(relevant_set) > 0 else 0
    if precision + recall > 0:
        f_measure = 2 * (precision * recall) / (precision + recall)
    else:
        f_measure = 0
    return precision, recall, f_measure
relevant_docs = [1, 2, 3, 4, 5]
retrieved_docs = [2, 4, 6, 7]
```

```
precision, recall, f_measure = calculate_precision_recall_fmeasure(relevant_docs, retrieved_docs)

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

print(f"F-measure: {f_measure:.4f}")
```

Output:-



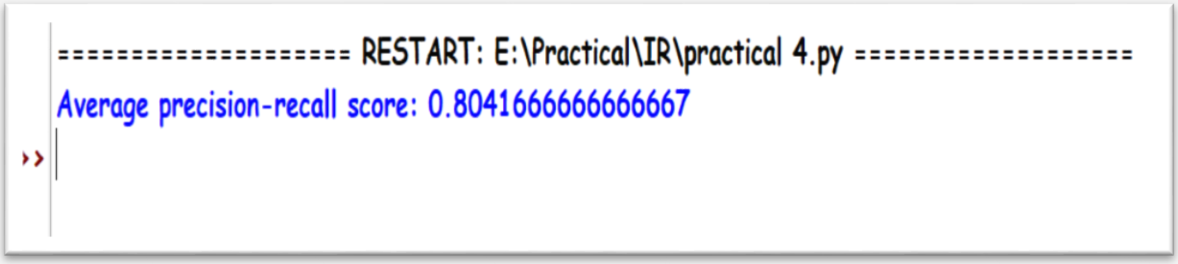
```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>
=================== RESTART: E:\Practical\IR\practical 4a.py ===================
Precision: 0.5000
Recall: 0.4000
F-measure: 0.4444
```

- **Use an evaluation toolkit to measure average precision and other evaluation metrics.**

Code:-

```
from sklearn.metrics import average_precision_score
y_true = [0, 1, 1, 0, 1, 1]
y_scores = [0.1, 0.4, 0.35, 0.8, 0.65, 0.9]
average_precision = average_precision_score(y_true, y_scores)
print(f"Average precision-recall score: {average_precision}")
```

Output:-



```
==================== RESTART: E:\Practical\IR\practical 4.py ===================
Average precision-recall score: 0.8041666666666667
>>
```

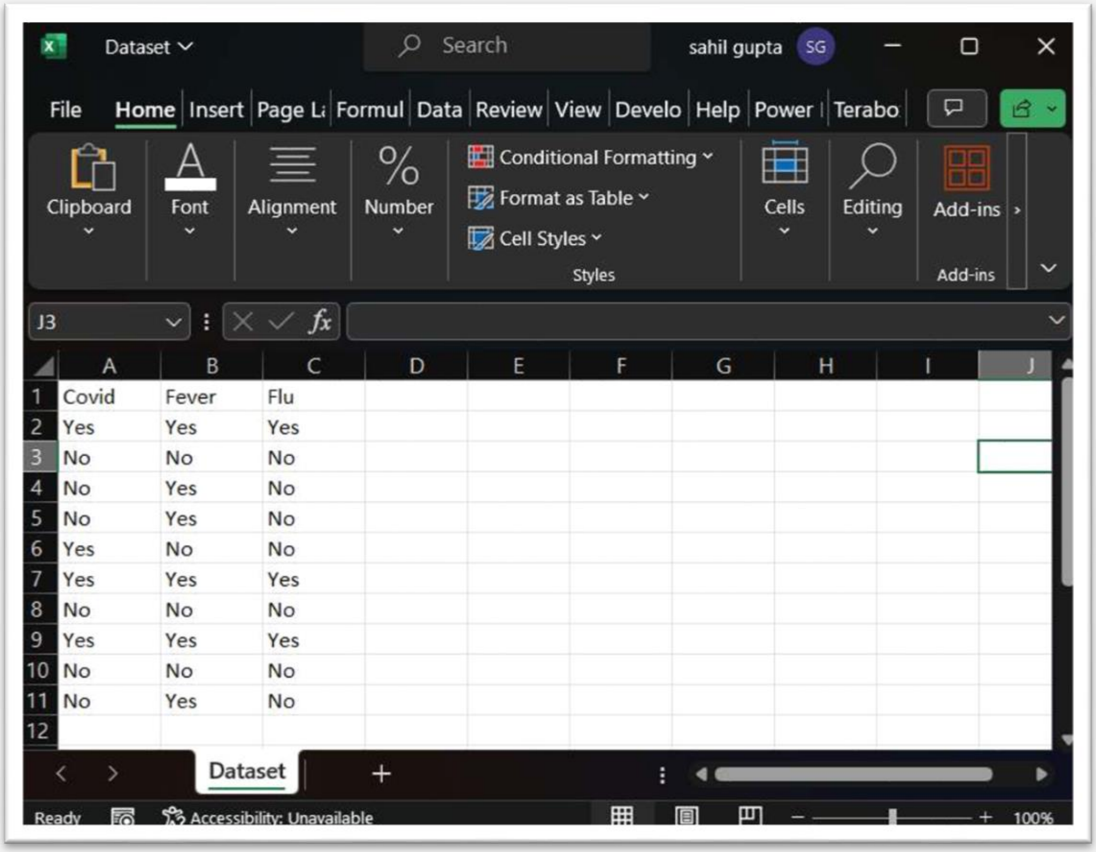
Practical No.5

Aim :- Text Categorization

- **Implement a text classification algorithm (e.g., Naive Bayes or Support Vector Machines).**
- **Train the classifier on a labelled dataset and evaluate its performance.**

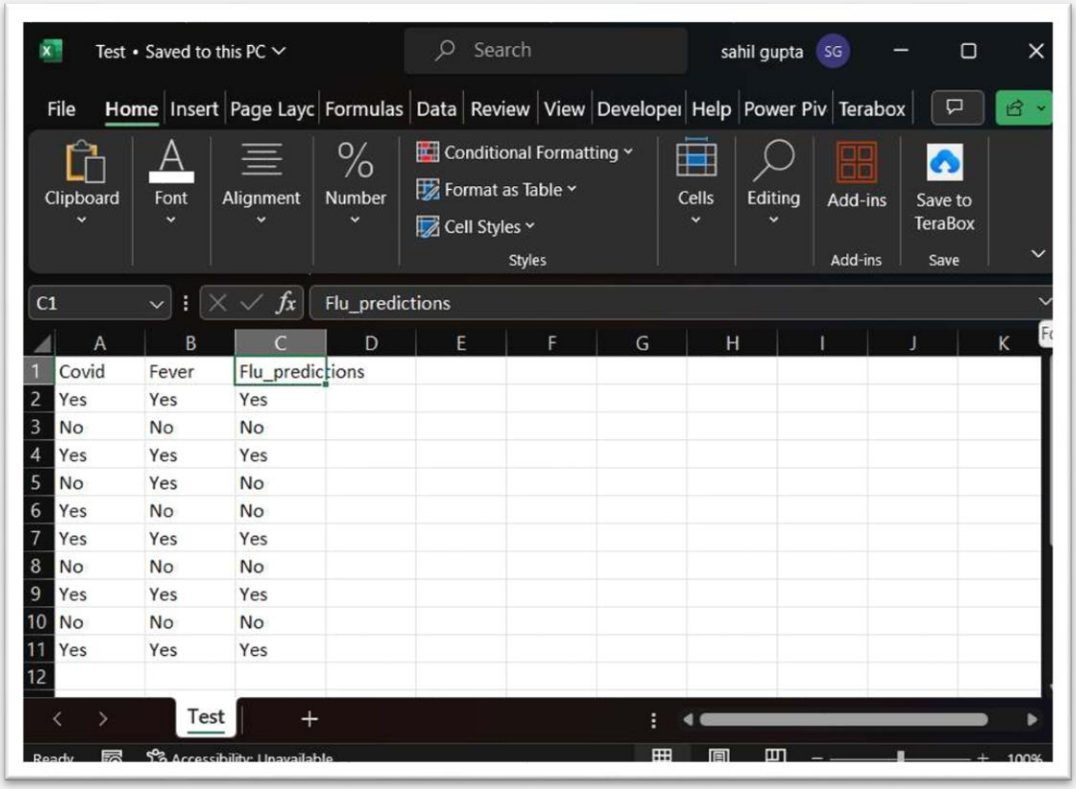
Step 1:- Create a two csv file first is Dataset.csv and second is Test.csv

Dataset.csv



Covid	Fever	Flu
Yes	Yes	Yes
No	No	No
No	Yes	No
No	Yes	No
Yes	No	No
Yes	Yes	Yes
No	No	No
Yes	Yes	Yes
No	No	No
No	Yes	No

Test.csv



C1: Flu_predictions

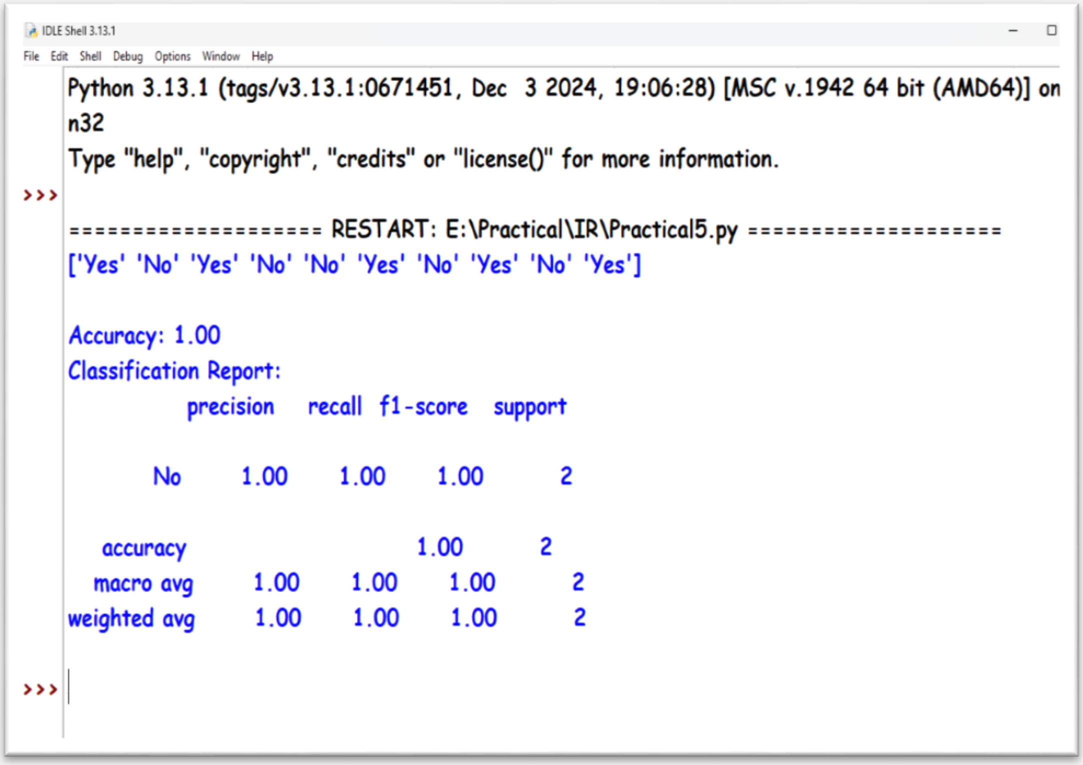
	A	B	C
1	Covid	Fever	Flu_predictions
2	Yes	Yes	Yes
3	No	No	No
4	Yes	Yes	Yes
5	No	Yes	No
6	Yes	No	No
7	Yes	Yes	Yes
8	No	No	No
9	Yes	Yes	Yes
10	No	No	No
11	Yes	Yes	Yes
12			

Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
df = pd.read_csv(r"C:\Users\kc140\OneDrive\Documents\Dataset.csv")
data = df["Covid"] + " " + df["Fever"]
```

```
X = data.astype(str)
y = df['Flu']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)
classifier = MultinomialNB()
classifier.fit(X_train_counts, y_train)
data1 = pd.read_csv(r"C:\Users\kc140\OneDrive\Documents\Test.csv")
new_data = data1["Covid"] + " " + data1["Fever"]
new_data_counts = vectorizer.transform(new_data.astype(str))
predictions = classifier.predict(new_data_counts)
new_data = predictions
print(new_data)
accuracy = accuracy_score(y_test, classifier.predict(X_test_counts))
print(f"\nAccuracy: {accuracy:.2f}")
print("Classification Report:")
print(classification_report(y_test,
classifier.predict(X_test_counts)))
```

Output:-



```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: E:\Practical\IR\Practical5.py ====================
['Yes' 'No' 'Yes' 'No' 'No' 'Yes' 'No' 'Yes' 'No' 'Yes']

Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

          No       1.00      1.00      1.00         2

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2

>>>
```

Practical No.6

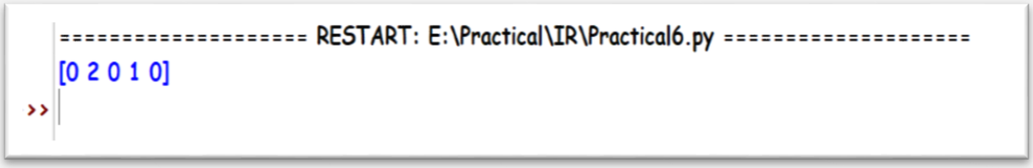
Aim:- Clustering for Information Retrieval

- **Implement a clustering algorithm (e.g., K-means or hierarchical clustering).**
- **Apply the clustering algorithm to a set of documents and evaluate the clustering results.**

Code:-

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
documents = ["Cats are known for their agility and grace",
"Dogs are often called ‘man’s best friend’.",
"Some dogs are trained to assist people with disabilities.",
"The sun rises in the east and sets in the west.",
"Many cats enjoy climbing trees and chasing toys.",
]
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
print(kmeans.labels_)
```

Output:-



```
=================== RESTART: E:\Practical\IR\Practical6.py ===================
[0 2 0 1 0]
>>
```

Practical No.7

Aim:- Web Crawling and Indexing.

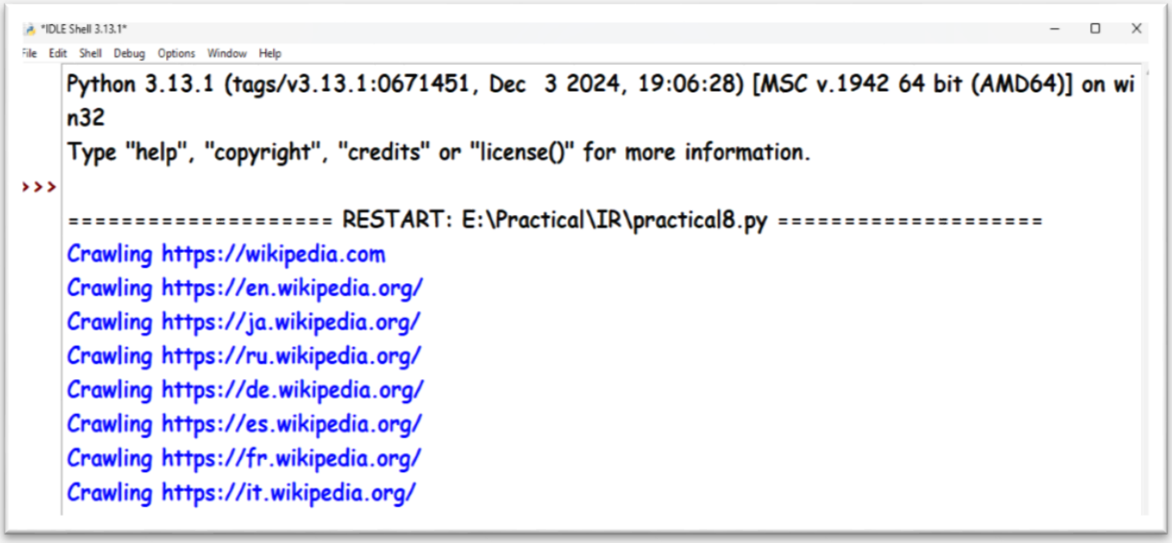
- **Develop a web crawler to fetch and index web pages.**
- **Handle challenges such as robots.txt, dynamic content, and crawling delays.**

Code:-

```
import requests

from bs4 import BeautifulSoup

import time

from urllib.parse import urljoin, urlparse

from urllib.robotparser import RobotFileParser


def get_html(url):

    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'}

    try:

        response = requests.get(url, headers=headers)

        response.raise_for_status()

        return response.text

    except requests.exceptions.HTTPError as errh:

        print(f"HTTP Error: {errh}")
    except requests.exceptions.RequestException as err:
        print(f"Request Error: {err}")
    return None

def save_robots_txt(url):
    try:
        robots_url = urljoin(url, '/robots.txt')
        robots_content = get_html(robots_url)
        if robots_content:
            with open('robots.txt', 'wb') as file:
                file.write(robots_content.encode('utf-8-sig'))
    except Exception as e:
        print(f"Error saving robots.txt: {e}")

def load_robots_txt():
    try:
        with open('robots.txt', 'rb') as file:
            return file.read().decode('utf-8-sig')
    except FileNotFoundError:
        return None
```

```
def extract_links(html, base_url):
    soup = BeautifulSoup(html, 'html.parser')
    links = []
    for link in soup.find_all('a', href=True):
        absolute_url = urljoin(base_url, link.get('href'))
        links.append(absolute_url)
    return links

def is_allowed_by_robots(url, robots_content):
    parser = RobotFileParser()
    parser.parse(robots_content.split('\n'))
    return parser.can_fetch('*', url)

def crawl(start_url, max_depth=3, delay=1):
    visited_urls = set()

    def recursive_crawl(url, depth, robots_content):
        if depth > max_depth or url in visited_urls or not
is_allowed_by_robots(url, robots_content):
            return
        visited_urls.add(url)
        time.sleep(delay)
        html = get_html(url)
        if html:
            print(f"Crawling {url}")
            links = extract_links(html, url)
            for link in links:
                recursive_crawl(link, depth + 1, robots_content)

    save_robots_txt(start_url)
    robots_content = load_robots_txt()
    if not robots_content:
        print("Unable to retrieve robots.txt. Crawling without
restrictions.")

    recursive_crawl(start_url, 1, robots_content)
    print("Performed by 740_Pallavi & 743_Deepak")

# Start crawling from Wikipedia with max depth 2 and delay of 2
seconds
crawl('https://wikipedia.com', max_depth=2, delay=2)
```

Output :-



```
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on wi
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: E:\Practical\IR\practical8.py ====================
Crawling https://wikipedia.com
Crawling https://en.wikipedia.org/
Crawling https://ja.wikipedia.org/
Crawling https://ru.wikipedia.org/
Crawling https://de.wikipedia.org/
Crawling https://es.wikipedia.org/
Crawling https://fr.wikipedia.org/
Crawling https://it.wikipedia.org/
```

Practical No.8

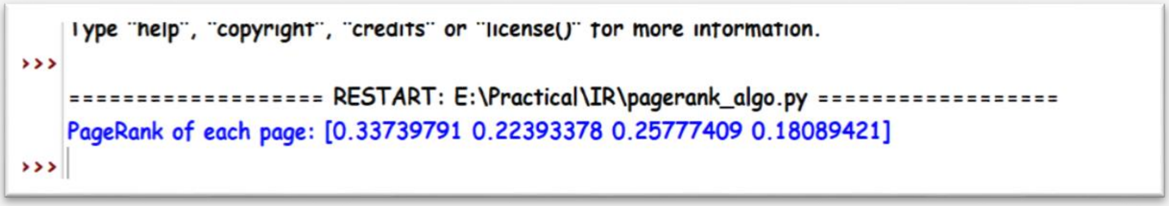
Aim :- Link Analysis and PageRank.

- **Implement the PageRank algorithm to rank web pages based on link analysis.**

Code:-

```
import numpy as np
def pagerank(graph, d=0.85, max_iter=100, tol=1e-6):
    n = len(graph)
    ranks = np.ones(n) / n
    M = np.array(graph, dtype=float)
    outdegree = M.sum(axis=0)
    M = np.divide(M, outdegree, where=outdegree != 0)
    for _ in range(max_iter):
        new_ranks = (1 - d) / n + d * np.dot(M, ranks)
        if np.linalg.norm(new_ranks - ranks, 1) < tol:
            break
        ranks = new_ranks
    return ranks
graph = [
    [0, 1, 1, 0],
    [0, 0, 1, 1],
    [1, 0, 0, 1],
    [1, 0, 0, 0],
]
ranks = pagerank(graph)
print("PageRank of each page:", ranks)
```

Output:-



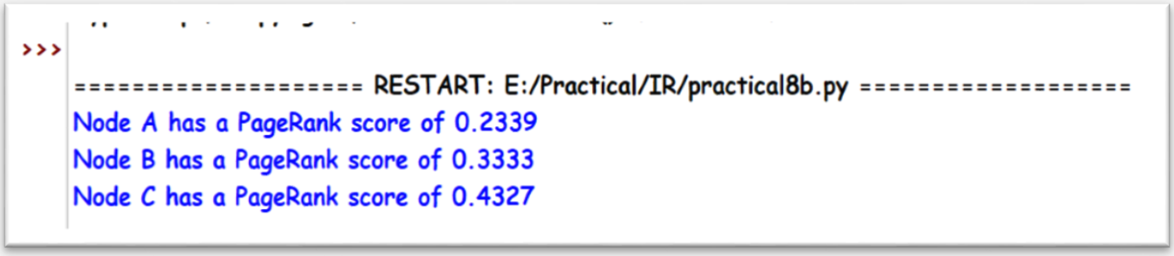
```
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
=================== RESTART: E:\Practical\IR\pagerank_algo.py ==================
PageRank of each page: [0.33739791 0.22393378 0.25777409 0.18089421]
>>> 
```

• Apply the PageRank algorithm to a small web graph and analyze the results.

Code:-

```
import networkx as nx
G = nx.DiGraph()
G.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'C'), ('C', 'A'), ('C', 'B')])
pagerank_scores = nx.pagerank(G, alpha=0.85)
for node, score in pagerank_scores.items():
    print(f"Node {node} has a PageRank score of {score:.4f}")
```

Output :-



```
>>>
==================== RESTART: E:/Practical/IR/practical8b.py ===================
Node A has a PageRank score of 0.2339
Node B has a PageRank score of 0.3333
Node C has a PageRank score of 0.4327
```

Practical No.9

Aim:- Learning to Rank

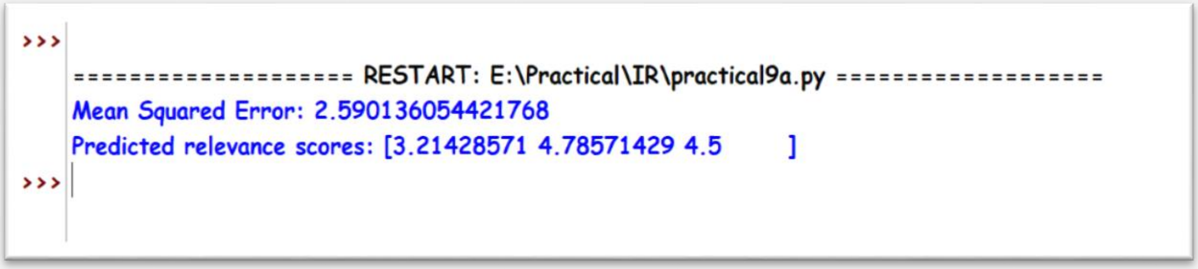
- **Implement a learning to rank algorithm (e.g. RankSVM or RankBoost).**

Code:-

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
queries = [
    "best programming languages",
    "machine learning tutorials",
    "artificial intelligence books"
]
documents = [
    "Python is a versatile language used for machine learning and web development.",
    "This guide teaches machine learning using scikit-learn and TensorFlow.",
    "AI books are essential for learning deep learning techniques and neural networks.",
    "Java is widely used for enterprise applications and Android development.",
    "Books on AI cover topics from basic to advanced deep learning algorithms."
]
relevance_scores = [
    [3, 5, 2, 1, 4],
    [4, 5, 3, 2, 1],
    [5, 4, 3, 1, 2]

]
vectorizer = TfidfVectorizer(stop_words='english')
X_documents = vectorizer.fit_transform(documents)
y = np.array(relevance_scores).flatten()
X_query_doc = []
for query in queries:
    query_vector = vectorizer.transform([query])  # Vectorize the query
    query_doc_pairs = [np.concatenate([query_vector.toarray()[0], doc.toarray()[0]]) for doc in X_documents]
    X_query_doc.extend(query_doc_pairs)
```

```
X_query_doc = np.array(X_query_doc)



X_train, X_test, y_train, y_test = train_test_split(X_query_doc, y, test_size=0.2, random_state=42)

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

print(f"Mean Squared Error: {mse}")

print("Predicted relevance scores:", y_pred)
```

Output:-



```
>>>
    ==================== RESTART: E:\Practical\IR\practical9a.py ===================
    Mean Squared Error: 2.590136054421768
    Predicted relevance scores: [3.21428571 4.78571429 4.5       ]
>>>
```

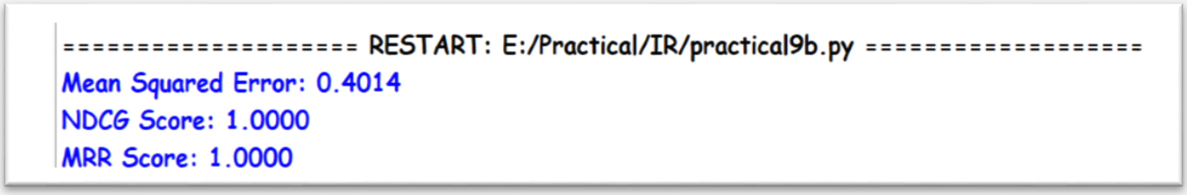
- **Train the ranking model using labelled data and evaluate its effectiveness.**

Code:-

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, ndcg_score
data = {
    'doc_length': [100, 200, 150, 80, 120, 220, 90, 300, 250, 180],
    'keyword_freq': [10, 15, 8, 5, 9, 18, 4, 20, 14, 10],
    'avg_sentence_length': [10, 15, 12, 8, 9, 20, 7, 25, 18, 16],
    'relevance': [2, 3, 1, 0, 1, 3, 0, 3, 2, 2]
}
df = pd.DataFrame(data)
X = df[['doc_length', 'keyword_freq', 'avg_sentence_length']]
y = df['relevance']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3)
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
y_true = np.array([[0, 1, 2], [2, 1, 0]])
y_pred_reshaped = np.array([[0.1, 0.2, 0.7], [0.7, 0.2, 0.1]])
ndcg = ndcg_score(y_true, y_pred_reshaped)
print(f"NDCG Score: {ndcg:.4f}")
def mean_reciprocal_rank(y_true, y_pred):
    mrr = 0
    for true, pred in zip(y_true, y_pred):
        rank = np.argsort(pred)[::-1]
        for i, r in enumerate(rank):
            if true[r] > 0:
                mrr += 1 / (i + 1)
                break
    return mrr / len(y_true)
mrr_score = mean_reciprocal_rank(y_true, y_pred_reshaped)
print(f"MRR Score: {mrr_score:.4f}")
```

Output :-



```
==================== RESTART: E:/Practical/IR/practical9b.py ===================
Mean Squared Error: 0.4014
NDCG Score: 1.0000
MRR Score: 1.0000
```

Practical No.10

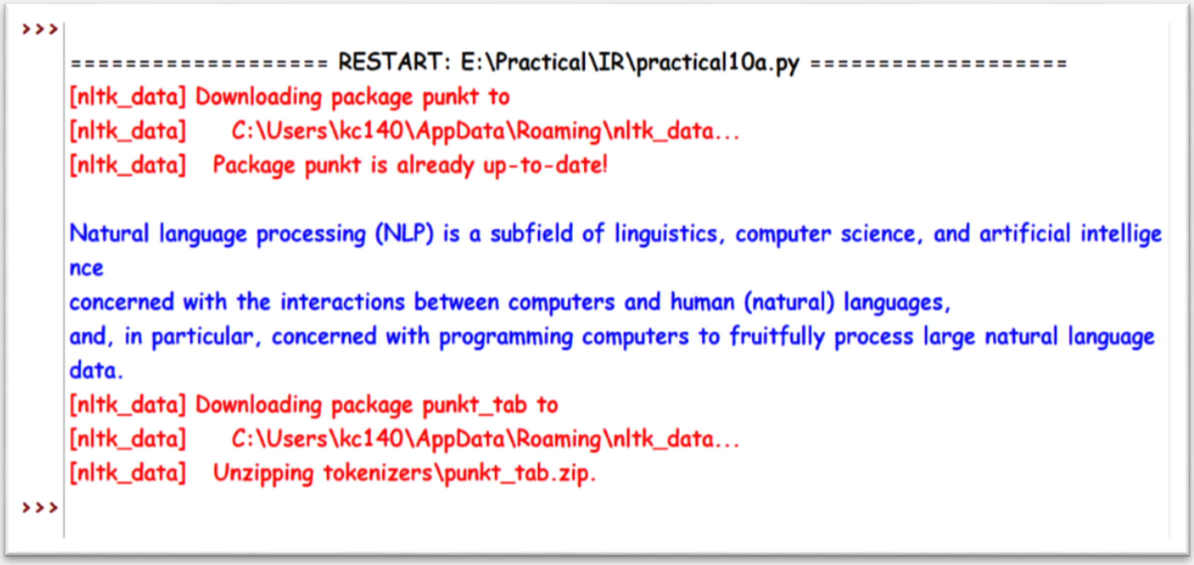
Aim:- Advanced Topics in Information Retrieval

- **Implement a text summarization algorithm (e.g., extractive or abstractive).**

Code:-

```
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import sent_tokenize
import heapq
nltk.download('punkt')
def summarize_text(text, summary_length=3):
  sentences = sent_tokenize(text)
  vectorizer = TfidfVectorizer(stop_words='english')
  tfidf_matrix = vectorizer.fit_transform(sentences)
  sentence_scores = {}
  for i, sentence in enumerate(sentences):
    sentence_scores[i] = tfidf_matrix[i].sum()
    ranked_sentences = heapq.nlargest(summary_length,
sentence_scores, key=sentence_scores.get)
    ranked_sentences = sorted(ranked_sentences)
    summary = [sentences[i] for i in ranked_sentences]
    return ' '.join(summary)

text = """
Natural language processing (NLP) is a subfield of linguistics,
computer science, and artificial intelligence
concerned with the interactions between computers and
human (natural) languages,
and, in particular, concerned with programming computers to
fruitfully process large natural language data.
Challenges in natural language processing frequently involve
natural language understanding, natural
language generation frequently from formal, machine-
readable logical forms), connecting language and machine
perception,
managing or analyzing large amounts of unstructured data
(such as "big data"), and often statistical machine learning.
"""
summary = summarize_text(text)
print(summary)
nltk.download('punkt_tab')
```

Output:-



```
>>>
    ================== RESTART: E:\Practical\IR\practical10a.py ==================
    [nltk_data] Downloading package punkt to
    [nltk_data]     C:\Users\kc140\AppData\Roaming\nltk_data...
    [nltk_data]   Package punkt is already up-to-date!

    Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intellige
    nce
    concerned with the interactions between computers and human (natural) languages,
    and, in particular, concerned with programming computers to fruitfully process large natural language
    data.
    [nltk_data] Downloading package punkt_tab to
    [nltk_data]     C:\Users\kc140\AppData\Roaming\nltk_data...
    [nltk_data]   Unzipping tokenizers\punkt_tab.zip.
>>>
```

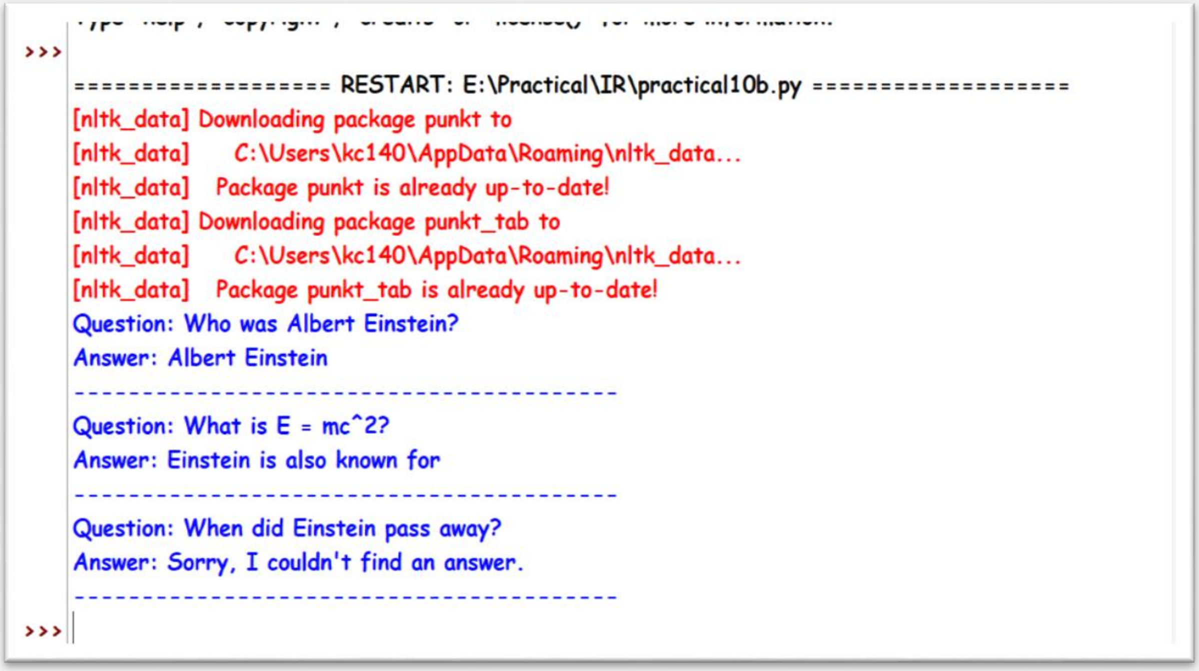
- **Build a question-answering system using techniques such as information extraction**

Code:-

```
import nltk
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.tokenize import sent_tokenize
import re
nltk.download('punkt')
def preprocess_text(text):
    return sent_tokenize(text)
def extract_relevant_sentences(text, question, top_n=3):
    sentences = preprocess_text(text)
    if not sentences:
        return []
    vectorizer = TfidfVectorizer(stop_words='english')
    tfidf_matrix = vectorizer.fit_transform([question] +
sentences)
    cosine_similarities = cosine_similarity(tfidf_matrix[0:1],
tfidf_matrix[1:]).flatten()
```

```
    relevant_sentences_idx = cosine_similarities.argsort()[-top_n:][::-1]
    return [sentences[i] for i in relevant_sentences_idx]
def extract_entities(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    return entities
def extract_answer_from_sentence(sentence, question):
    if "who" in question.lower():
        entities = extract_entities(sentence)
        for entity, label in entities:
            if label == "PERSON":
                return entity
    elif "what" in question.lower():
        words = re.findall(r'\b\w+\b', sentence)
        return " ".join(words[:5])
    return None
def answer_question(text, question, top_n=3):
    relevant_sentences = extract_relevant_sentences(text, question, top_n)
    for sentence in relevant_sentences:
        answer = extract_answer_from_sentence(sentence,
question)
        if answer:
            return answer
    return "Sorry, I couldn't find an answer."
text = """
Albert Einstein was born in Ulm, Germany, in 1879. He was a
theoretical physicist who developed the theory of relativity.
Einstein is also known for his famous equation, E = mc^2. He
won the Nobel Prize in Physics in 1921 for his discovery of
the photoelectric effect. Einstein passed away in 1955 at the
age of 76 in Princeton, New Jersey.
"""
questions = [
    "Who was Albert Einstein?",
    "What is E = mc^2?",
    "When did Einstein pass away?"
]
nltk.download('punkt_tab')
nlp = spacy.load("en_core_web_sm")
for question in questions:
    print(f"Question: {question}")
    print(f"Answer: {answer_question(text, question)}")
    print("-" * 40)
```

Output:-



```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ================== RESTART: E:\Practical\IR\practical10b.py ==================
    [nltk_data] Downloading package punkt to
    [nltk_data]     C:\Users\kc140\AppData\Roaming\nltk_data...
    [nltk_data]   Package punkt is already up-to-date!
    [nltk_data] Downloading package punkt_tab to
    [nltk_data]     C:\Users\kc140\AppData\Roaming\nltk_data...
    [nltk_data]   Package punkt_tab is already up-to-date!
    Question: Who was Albert Einstein?
    Answer: Albert Einstein
    ----------------------------------------
    Question: What is E = mc^2?
    Answer: Einstein is also known for
    ----------------------------------------
    Question: When did Einstein pass away?
    Answer: Sorry, I couldn't find an answer.
    ----------------------------------------
>>>
```