**Practical Session: Concepts of Search Engine**

* **Part 1: Indexing Process** 
  + Implement a basic indexing align with the key concepts of search engine architecture.

1. Write a function that take a list of documents (text strings) and creates an inverted index. The inverted index should map terms to the list of document IDs.

documents = {

"Article: Pandas Basics": "This article covers the basics of using Pandas in Python.",

"Tutorial: Data Visualization": "Learn about data visualization techniques with Python libraries.",

"Case Study: Sales Analysis": "Analyze sales data using Python for a real-world case study."

}

import re

from collections import defaultdict

def create\_inverted\_index(documents)**:**

    inverted\_index **=** defaultdict(list)

    for doc\_id**,** content in documents**.**items()**:**

# Tokenize the content (convert to lowercase and split on

non-word characters)

        tokens **=** re**.**findall(r**'**\w+**',** content**.**lower())

        # Add each token to the inverted index

        for token in set(tokens)**:**  # Using set to avoid duplicates

            inverted\_index[token]**.**append(doc\_id)

    return dict(inverted\_index)

# Example usage

documents **=** {

**"**Article: Pandas Basics**":** **"**This article covers the basics of using Pandas in Python.**",**

**"**Tutorial: Data Visualization**":** **"**Learn about data visualization techniques with Python libraries.**",**

**"**Case Study: Sales Analysis**":** **"**Analyze sales data using Python for a real-world case study.**"** }

inverted\_index **=** create\_inverted\_index(documents)

# Print the inverted index

for term**,** doc\_ids in inverted\_index**.**items()**:**

    print(f"{term}: {doc\_ids}")

=========================================================================

Output:

pandas: ['Article: Pandas Basics']

basics: ['Article: Pandas Basics']

covers: ['Article: Pandas Basics']

the: ['Article: Pandas Basics']

python: ['Article: Pandas Basics', 'Tutorial: Data Visualization', 'Case Study: Sales Analysis']

in: ['Article: Pandas Basics']

of: ['Article: Pandas Basics']

article: ['Article: Pandas Basics']

this: ['Article: Pandas Basics']

using: ['Article: Pandas Basics', 'Case Study: Sales Analysis']

visualization: ['Tutorial: Data Visualization']

with: ['Tutorial: Data Visualization']

learn: ['Tutorial: Data Visualization']

data: ['Tutorial: Data Visualization', 'Case Study: Sales Analysis']

techniques: ['Tutorial: Data Visualization']

libraries: ['Tutorial: Data Visualization']

about: ['Tutorial: Data Visualization']

study: ['Case Study: Sales Analysis']

world: ['Case Study: Sales Analysis']

real: ['Case Study: Sales Analysis']

a: ['Case Study: Sales Analysis']

analyze: ['Case Study: Sales Analysis']

case: ['Case Study: Sales Analysis']

sales: ['Case Study: Sales Analysis']

for: ['Case Study: Sales Analysis']

* **Part 2: Query Process**
  + Implement and simulate a basic query align with the key concepts of search engine architecture.

1. Write a function that takes a query and an inverted index, then returns the list of document IDs that contain all terms in the query.

query = "analyze sales"

inverted\_index = {

“this”: [0], “article”: [0], “covers”: [0], “the”: [0], “basics”: [0], “of”: [0], “using”: [0, 2], “pandas”: [0], “in”: [0], “python”: [0, 1, 2], “learn”: [1], “about”: [1], “data”: [1, 2], “visualization”: [1], “techniques”: [1], “with”: [1], “libraries”: [1], “analyze”: [2], “sales”: [2], “for”: [2], “a”: [2], “real-world”: [2], “case”: [2], “study”: [2] }

def search\_documents(query**,** inverted\_index)**:**

    # Tokenize and lowercase the query

    query\_terms **=** query**.**lower()**.**split()

    # Get the set of document IDs for each query term

    doc\_sets **=** [set(inverted\_index**.**get(term**,** [])) for term in query\_terms]

    # Find the intersection of all document sets

    if doc\_sets**:**

        result **=** set**.**intersection(\*doc\_sets)

        return list(result)

    else**:**

        return []

# Example usage

query **=** **"**analyze sales**"**

inverted\_index **=** {

**"**this**":** [0]**,**

**"**article**":** [0]**,**

**"**covers**":** [0]**,**

**"**the**":** [0]**,**

**"**basics**":** [0]**,**

**"**of**":** [0]**,**

**"**using**":** [0**,** 2]**,**

**"**pandas**":** [0]**,**

**"**in**":** [0]**,**

**"**python**":** [0**,** 1**,** 2]**,**

**"**learn**":** [1]**,**

**"**about**":** [1]**,**

**"**data**":** [1**,** 2]**,**

**"**visualization**":** [1]**,**

**"**techniques**":** [1]**,**

**"**with**":** [1]**,**

**"**libraries**":** [1]**,**

**"**analyze**":** [2]**,**

**"**sales**":** [2]**,**

**"**for**":** [2]**,**

**"**a**":** [2]**,**

**"**real-world**":** [2]**,**

**"**case**":** [2]**,**

**"**study**":** [2]

}

result **=** search\_documents(query**,** inverted\_index)

print(f"Documents containing '{query}': {result}")

=======================================================

Output:

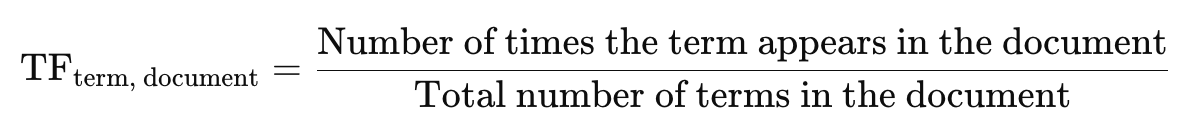
Documents containing 'analyze sales': [2]

* **Part 3: Ranking**
  + Implement a single scoring align with the key concepts of search engine architecture.

1. Write a function that calculates a simple term frequency score for a specific document based on a query.

documents = "Article: Pandas Basics: This article covers the basics of using Pandas in Python."

query = “Pandas”



# Implement a simple term frequency scoring function

def calculate\_tf\_score(document**,** query)**:**

**"""**

    Calculate a simple term frequency score for a document based on a query.

    Args:

    document (str): The text content of the document.

    query (str): The search query.

    Returns:

    float: The term frequency score.

**"""**

    # Convert document and query to lowercase for case-insensitive matching

    document **=** document**.**lower()

    query **=** query**.**lower()

    # Split the document into words

    doc\_words **=** document**.**split()

    # Count the occurrences of query terms in the document

    query\_terms **=** query**.**split()

    term\_count **=** sum(doc\_words**.**count(term) for term in query\_terms)

    # Calculate the score as term frequency

    score **=** term\_count **/** len(doc\_words)

    return score

# Example usage

document **=** **"**Article: Pandas Basics: This article covers the basics of using Pandas in Python.**"**

query **=** **"**Pandas**"**

tf\_score **=** calculate\_tf\_score(document**,** query)

print(f"Term Frequency Score for '{query}' in the document: {tf\_score:.4f}")

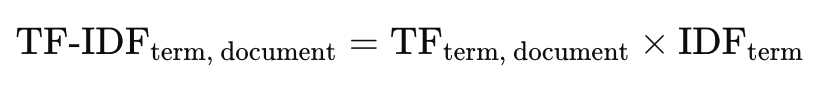
===========================================================================

Otput:

Term Frequency Score for 'Pandas' in the document: 0.1538

* **Part 4: TF-IDF Scoring**
  + Implement a simple TF-IDF scoring system to rank documents based on a query.

1. Write a function that calculates the Term Frequency (TF) for each document concerning a given query term.
2. Implement the Inverse Document Frequency (IDF) calculation.
3. Use both TF and IDF to compute the TF-IDF score for each document.
4. Return a ranked list of documents based on their TF-IDF scores.



import math

from collections import Counter

def calculate\_tf(document**,** term)**:**

**"""**Calculate Term Frequency for a term in a document.**"""**

    words **=** document**.**lower()**.**split()

    return words**.**count(term**.**lower()) **/** len(words)

def calculate\_idf(documents**,** term)**:**

**"""**Calculate Inverse Document Frequency for a term across all documents.**"""**

    num\_documents **=** len(documents)

    num\_documents\_with\_term **=** sum(1 for doc in documents if term**.**lower() in doc**.**lower())

    return math**.**log(num\_documents **/** (1 **+** num\_documents\_with\_term))

def calculate\_tf\_idf(documents**,** query)**:**

**"""**Calculate TF-IDF scores for documents based on a query.**"""**

    scores **=** []

    query\_terms **=** query**.**lower()**.**split()

    for i**,** doc in enumerate(documents)**:**

        score **=** 0

        for term in query\_terms**:**

            tf **=** calculate\_tf(doc**,** term)

            idf **=** calculate\_idf(documents**,** term)

            score **+=** tf **\*** idf

        scores**.**append((i**,** score))

    return sorted(scores**,** key**=**lambda x: x[1]**,** reverse**=**True)

def rank\_documents(documents**,** query)**:**

**"""**Rank documents based on TF-IDF scores.**"""**

    ranked\_docs **=** calculate\_tf\_idf(documents**,** query)

    return [documents[i] for i**,** \_ in ranked\_docs]

# Example usage

documents **=** [

**"**Pandas is a powerful data manipulation library for Python.**",**

**"**Python is a versatile programming language used in data science.**",**

**"**Data analysis often involves using libraries like Pandas and NumPy.**",**

**"**Machine learning algorithms can be implemented using Python libraries.**"**

]

query **=** **"**Pandas data analysis**"**

ranked\_documents **=** rank\_documents(documents**,** query)

print(**"**Ranked documents based on the query '{}':**".**format(query))

for i**,** doc in enumerate(ranked\_documents**,** 1)**:**

    print(f"{i}. {doc}")

==============================================================================

Output:

Ranked documents based on the query 'Pandas data analysis':

1. Data analysis often involves using libraries like Pandas and NumPy.

2. Pandas is a powerful data manipulation library for Python.

3. Python is a versatile programming language used in data science.

4. Machine learning algorithms can be implemented using Python libraries.

* **Part 5: Ranking with Multiple Query Terms**
  + Extend the ranking function to handle multiple query terms and return the top N ranked documents.

1. Modify the existing rank\_documents function to accept an additional parameter for the number of top documents to return.
2. Ensure that the TF-IDF calculation considers all terms in the query.
3. Return only the top N ranked documents.

# Modify the rank\_documents function to handle multiple query terms and return top N documents

def rank\_documents(documents**,** query**,** top\_n=None)**:**

**"""**

    Rank documents based on TF-IDF scores for multiple query terms.

    Args:

    documents (list): List of document strings.

    query (str): Query string containing one or more terms.

    top\_n (int, optional): Number of top ranked documents to return. If None, return all.

    Returns:

    list: Top N ranked documents.

**"""**

    ranked\_docs **=** calculate\_tf\_idf(documents**,** query)

    # If top\_n is not specified or is greater than the number of documents,

    # return all ranked documents

    if top\_n is None or top\_n **>** len(documents)**:**

        top\_n **=** len(documents)

    # Return only the top N ranked documents

    return [documents[i] for i**,** \_ in ranked\_docs[**:**top\_n]]

# Example usage

query **=** **"**Python data analysis**"**

top\_n **=** 2

top\_ranked\_documents **=** rank\_documents(documents**,** query**,** top\_n)

print(f"Top {top\_n} ranked documents based on the query '{query}':")

for i**,** doc in enumerate(top\_ranked\_documents**,** 1)**:**

    print(f"{i}. {doc}")

=========================================================================

Output:

Top 2 ranked documents based on the query 'Python data analysis':

1. Data analysis often involves using libraries like Pandas and NumPy.

2. Pandas is a powerful data manipulation library for Python.