# Tribhuvan University

# Bhaktapur Multiple Campus

Dudhpati-7, Bhaktapur, Nepal


# Lab Report of

# Information Retrieval (CSC 413)

**Prepared By:**

**Suman Khadka**

B.Sc. CSIT 076 Batch, 7th Semester

Roll no: 23275

**Submitted To:**

**Nabin Ghimire**

Lecturer

CSIT Department

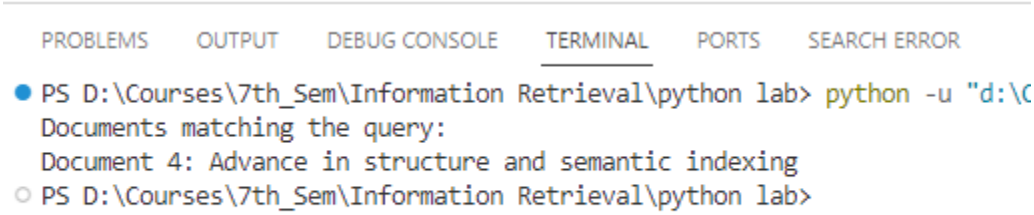
**1. Program to demonstrate the Boolean Retrieval Model and Vector Space Model.**

**booleanRetrieval.py**

```
# A simple dictionary to represent the documents
documents = {
    1: "English tutorial and fast track",
    2: "Book on semantic analysis",
    3: "Learning latent semantic indexing",
    4: "Advance in structure and semantic indexing",
    5: "Analysis of latent structures"
}

# A dictionary to store the inverted index
inverted_index = {}

# Populate the inverted index
for doc_id, text in documents.items():
    for word in text.split():
        if word not in inverted_index:
            inverted_index[word] = set()
        inverted_index[word].add(doc_id)

# The query
query = "advance AND structure AND NOT analysis”"

# Parse the query
query_terms = query.split()
query_sets = []
current_set = set()
for term in query_terms:
    if term == "and":
        query_sets.append(current_set)
        current_set = set()
    elif term == "or":
        query_sets.append(current_set)
        current_set = set()
    elif term in inverted_index:
        current_set.update(inverted_index[term])
else:
    query_sets.append(current_set)

# Intersection of query sets
result_set = query_sets[0]
for i in range(1, len(query_sets)):
    result_set = result_set.intersection(query_sets[i])
```

```
# Get the document ids from the result set
document_ids = list(result_set)

# Print the result
print("Documents matching the query:")
for doc_id in document_ids:
    print(f"Document {doc_id}: {documents[doc_id]}")
```

**Output**




---

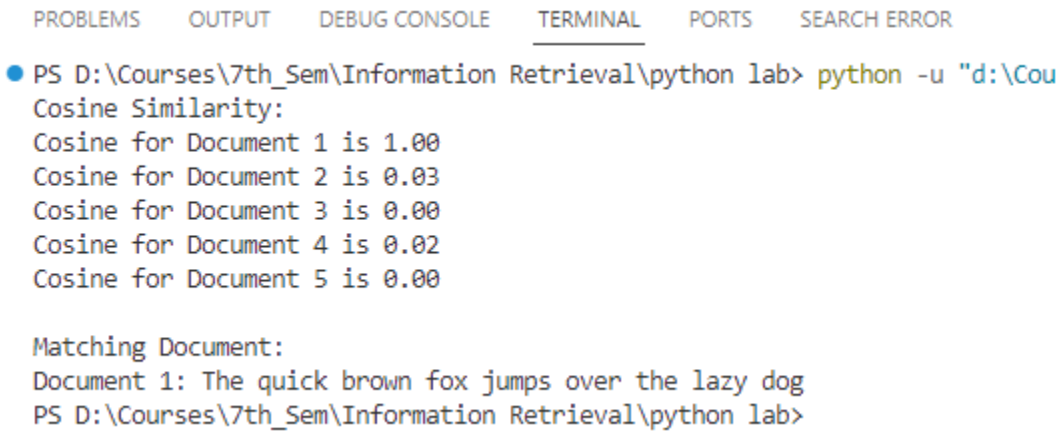
**vectorSpace.py**

```
import math
import string

# Define the document collection with IDs
documents = [
    (1, "The quick brown fox jumps over the lazy dog"),
    (2, "The five boxing wizards jump quickly"),
    (3, "Pack my box with five dozen liquor jugs"),
    (4, "How quickly daft jumping zebras vex"),
    (5, "Two driven jocks help fax my big quiz")
]

# Define the query
query = "The quick brown fox jumps over the lazy dog"

# Preprocess the documents and the query
def preprocess(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))  # Remove
punctuation
    words = text.split()
    return words
```

```
# Calculate term frequency (TF)
def term_frequency(word, document):
    return document.count(word)

# Calculate document frequency (DF)
def document_frequency(word, documents):
    return sum([word in document for _, document in documents])

# Calculate inverse document frequency (IDF)
def inverse_document_frequency(word, documents):
    return math.log(len(documents) / (1 + document_frequency(word, documents)))

# Calculate TF-IDF
def tf_idf(word, document, documents):
    tf = term_frequency(word, document)
    idf = inverse_document_frequency(word, documents)
    return tf * idf

# Create a TF-IDF matrix for all words in all documents
def create_tf_idf_matrix(documents):
    tf_idf_matrix = {}
    for _, document in documents:
        for word in preprocess(document):
            if word not in tf_idf_matrix:
                tf_idf_matrix[word] = [tf_idf(word, document, documents) for _,
document in documents]
    return tf_idf_matrix

# Create the TF-IDF matrix
tf_idf_matrix = create_tf_idf_matrix(documents)

# Create the query vector
query_vector = [tf_idf(word, query, documents) for word in tf_idf_matrix.keys()]

# Calculate the cosine similarity between the query vector and each document
vector
def cosine_similarity(query_vector, document_vector):
    dot_product = sum(q * d for q, d in zip(query_vector, document_vector))
    magnitude_query = math.sqrt(sum(q ** 2 for q in query_vector))
    magnitude_document = math.sqrt(sum(d ** 2 for d in document_vector))
    return dot_product / (magnitude_query * magnitude_document)

# Print the cosine similarity for each document
print("Cosine Similarity:")
```

```
for doc_id, document in documents:
    document_vector = [tf_idf(word, document, documents) for word in
tf_idf_matrix.keys()]
    cosine_similarity_value = cosine_similarity(query_vector, document_vector)
    print(f"Cosine for Document {doc_id} is {cosine_similarity_value:.2f}")

# Print the document matching the query
print("\nMatching Document:")
matching_document = max(documents, key=lambda x: cosine_similarity(query_vector,
[tf_idf(word, x[1], documents) for word in tf_idf_matrix.keys()]))
print(f"Document {matching_document[0]}: {matching_document[1]}")
```

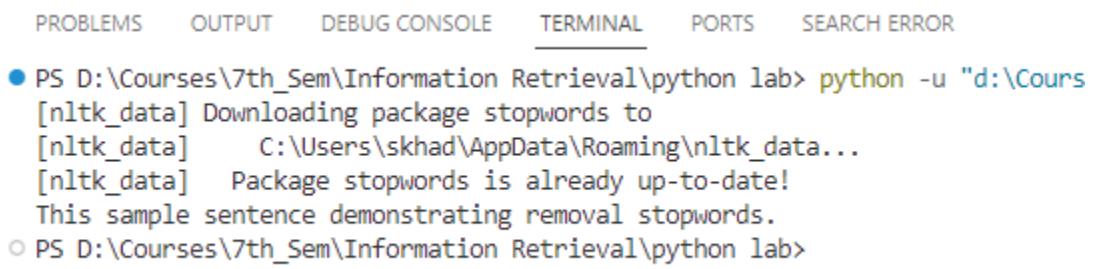
**Output**



## 2. Remove Stops words from a given random sentence using NLTK library.

```
import nltk
from nltk.corpus import stopwords

# Download the stopwords if not already downloaded
nltk.download('stopwords')

# Get the stopwords in English
stop_words = set(stopwords.words('english'))

# Define the input sentence
input_sentence = "This is a sample sentence demonstrating the removal of
stopwords."

# Tokenize the sentence into words
words = input_sentence.split()

# Remove the stop words
filtered_words = [word for word in words if word not in stop_words]

# Join the filtered words back into a sentence
filtered_sentence = " ".join(filtered_words)

# Print the filtered sentence
print(filtered_sentence)
```

## Output



**3. Perform the following text operation with the help of suitable paragraph of your own choice**

**a. Lowercasing**

**b. Tokenization**

**c. Stemming**

**d. Punctuation removal**

**e. Stop words removal**

**f. Lemmatization**

```
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

nltk.download("wordnet")

# Initialize the stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Initialize the stop word list
stop_words = set(stopwords.words("english"))

# Define the paragraph
paragraph = "The quick brown fox jumps over the lazy dog, while the sun sets
behind the tall, majestic mountains. The birds chirp as the day comes to an end,
and the stars begin to twinkle in the sky. The night is peaceful, and the world
is still."

# Perform lowercasing
lowercased_paragraph = paragraph.lower()
print("After performing Lowercasing :")
print(lowercased_paragraph)
print()

# Perform tokenization
tokenized_paragraph = lowercased_paragraph.split()
print("After Tokenization:")
print(tokenized_paragraph)
```

```
print()

# Perform stemming
stemmed_paragraph = []
for word in tokenized_paragraph:
    stemmed_paragraph.append(stemmer.stem(word))
print("After Stemming:")
print(stemmed_paragraph)
print()

# Perform punctuation removal
punctuation_removed_paragraph = [
    word.strip(string.punctuation)
    for word in stemmed_paragraph
    if word.strip(string.punctuation)
]
print("After Punctuation removal:")
print(punctuation_removed_paragraph)
print()

# Perform stop words removal
stop_words_removed_paragraph = [
    word for word in punctuation_removed_paragraph if word not in stop_words
]
print("After Stopwords removal:")
print(stop_words_removed_paragraph)
print()

# Perform lemmatization
lemmatized_paragraph = []
for word in stop_words_removed_paragraph:
    lemmatized_paragraph.append(lemmatizer.lemmatize(word))
print("After Lemmatization:")
print(lemmatized_paragraph)
```

## Output

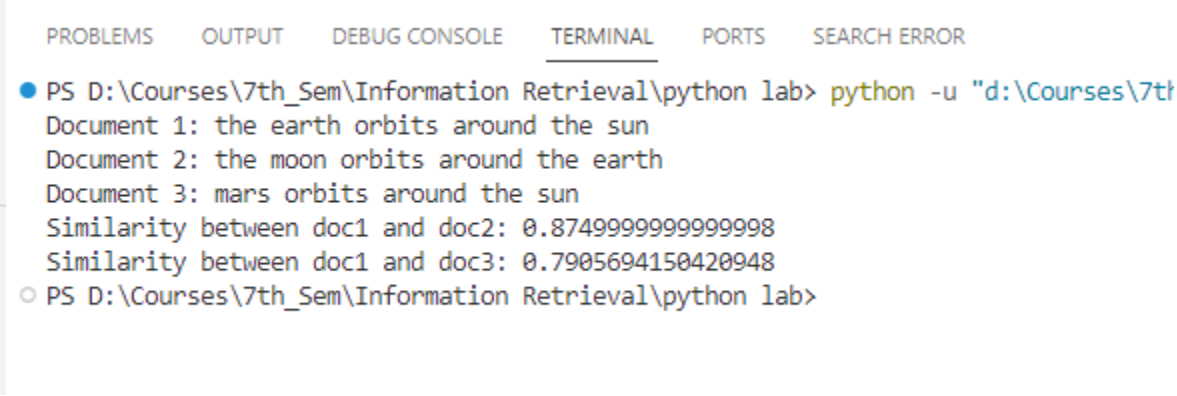
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR                                Code

PS D:\Courses\7th_Sem\Information Retrieval\python lab> python -u "d:\Courses\7th_Sem\Information Retrieval\python lab\IRo
ations.py"
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\skhad\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
After performing Lowercasing :
the quick brown fox jumps over the lazy dog, while the sun sets behind the tall, majestic mountains. the birds chirp as th
ay comes to an end, and the stars begin to twinkle in the sky. the night is peaceful, and the world is still.

After Tokenization:
['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog,', 'while', 'the', 'sun', 'sets', 'behind', 'the', '
l,', 'majestic', 'mountains.', 'the', 'birds', 'chirp', 'as', 'the', 'day', 'comes', 'to', 'an', 'end,', 'and', 'the', 'st
', 'begin', 'to', 'twinkle', 'in', 'the', 'sky.', 'the', 'night', 'is', 'peaceful,', 'and', 'the', 'world', 'is', 'still.'

After Stemming:
['the', 'quick', 'brown', 'fox', 'jump', 'over', 'the', 'lazi', 'dog,', 'while', 'the', 'sun', 'set', 'behind', 'the', 'tall,'
, 'majest', 'mountains.', 'the', 'bird', 'chirp', 'as', 'the', 'day', 'come', 'to', 'an', 'end,', 'and', 'the', 'star', 'begin
', 'to', 'twinkl', 'in', 'the', 'sky.', 'the', 'night', 'is', 'peaceful,', 'and', 'the', 'world', 'is', 'still.']

After Punctuation removal:
['the', 'quick', 'brown', 'fox', 'jump', 'over', 'the', 'lazi', 'dog', 'while', 'the', 'sun', 'set', 'behind', 'the', 'tall',
'majest', 'mountains', 'the', 'bird', 'chirp', 'as', 'the', 'day', 'come', 'to', 'an', 'end', 'and', 'the', 'star', 'begin', '
to', 'twinkl', 'in', 'the', 'sky', 'the', 'night', 'is', 'peaceful', 'and', 'the', 'world', 'is', 'still']

After Stopwords removal:
['quick', 'brown', 'fox', 'jump', 'lazi', 'dog', 'sun', 'set', 'behind', 'tall', 'majest', 'mountains', 'bird', 'chirp', 'day'
, 'come', 'end', 'star', 'begin', 'twinkl', 'sky', 'night', 'peaceful', 'world', 'still']

After Lemmatization:
['quick', 'brown', 'fox', 'jump', 'lazi', 'dog', 'sun', 'set', 'behind', 'tall', 'majest', 'mountain', 'bird', 'chirp', 'day',
 'come', 'end', 'star', 'begin', 'twinkl', 'sky', 'night', 'peaceful', 'world', 'still']
PS D:\Courses\7th_Sem\Information Retrieval\python lab>
```

**4. Program to find the similarity between documents**

```
import string

def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    return text

def create_bow(text):
    words = text.split()
    word_count = {}
    for word in words:
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1
    return word_count

def calculate_similarity(bow1, bow2):

    all_words = set(bow1.keys()).union(set(bow2.keys()))

    dot_product = sum(bow1.get(word, 0) * bow2.get(word, 0) for word in
all_words)
    magnitude1 = sum(val ** 2 for val in bow1.values()) ** 0.5
    magnitude2 = sum(val ** 2 for val in bow2.values()) ** 0.5

    # Calculate cosine similarity
    if magnitude1 == 0 or magnitude2 == 0:
        return 0
    else:
        return dot_product / (magnitude1 * magnitude2)

# Example usage
def main():
    doc1 = "The Earth orbits around the Sun."
    doc2 = "The Moon orbits around the Earth."
    doc3 = "Mars orbits around the Sun."

    doc1 = preprocess_text(doc1)
    doc2 = preprocess_text(doc2)
    doc3 = preprocess_text(doc3)

    bow1 = create_bow(doc1)
```

```
    bow2 = create_bow(doc2)
    bow3 = create_bow(doc3)

    similarity_doc1_doc2 = calculate_similarity(bow1, bow2)
    similarity_doc1_doc3 = calculate_similarity(bow1, bow3)

    print("Document 1:", doc1)
    print("Document 2:", doc2)
    print("Document 3:", doc3)
    print("Similarity between doc1 and doc2:", similarity_doc1_doc2)
    print("Similarity between doc1 and doc3:", similarity_doc1_doc3)

if __name__ == "__main__":
    main()
```

**Output**



## 5. Implement Porter stemmer algorithm

```
import re

class PorterStemmer:
    def __init__(self):
        self.step2list = {
            'ational': 'ate',
            'tional': 'tion',
            'enci': 'ence',
            'anci': 'ance',
            'izer': 'ize',
            'bli': 'ble',
            'alli': 'al',
            'entli': 'ent',
            'eli': 'e',
            'ousli': 'ous',
            'ization': 'ize',
            'ation': 'ate',
            'ator': 'ate',
            'alism': 'al',
            'iveness': 'ive',
            'fulness': 'ful',
            'ousness': 'ous',
            'aliti': 'al',
            'iviti': 'ive',
            'biliti': 'ble',
        }

        self.step3list = {
            'icate': 'ic',
            'ative': '',
            'alize': 'al',
            'iciti': 'ic',
            'ical': 'ic',
        }

    def stem(self, word):
        word = word.lower()
        word = re.sub(r"'s?$", "", word)
        word = re.sub(r"(ss|ies)$", r"\1", word)
        word = re.sub(r"(.*?[^aeiou])ies$", r"\1y", word)

        for suffix in self.step2list.keys():
            if word.endswith(suffix):
```

```
                stem = word[:-len(suffix)] + self.step2list[suffix]
                if self.m(stem) > 0:
                    return stem

        for suffix in self.step3list.keys():
            if word.endswith(suffix):
                stem = word[:-len(suffix)] + self.step3list[suffix]
                if self.m(stem) > 0:
                    return stem

        return word

    def m(self, word):
        return len(re.findall(r"[aeiou]", word))

# Example usage
stemmer = PorterStemmer()
words = ['rationalization', 'nationalization', 'catalyst', 'realization',
'sensational', 'ability', 'productivity', 'optimization']
print("\nAfter applying Porter Alogirithm: \n")
for word in words:
    print(f"{word}: {stemmer.stem(word)}")
```

**Output**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR
PS D:\Courses\7th_Sem\Information Retrieval\python lab> python -u "d:\Courses\

After applying Porter Alogirithm:

rationalization: rationalize
nationalization: nationalize
catalyst: catalyst
realization: realize
sensational: sensate
ability: ability
productivity: productivity
optimization: optimize
PS D:\Courses\7th_Sem\Information Retrieval\python lab>
```

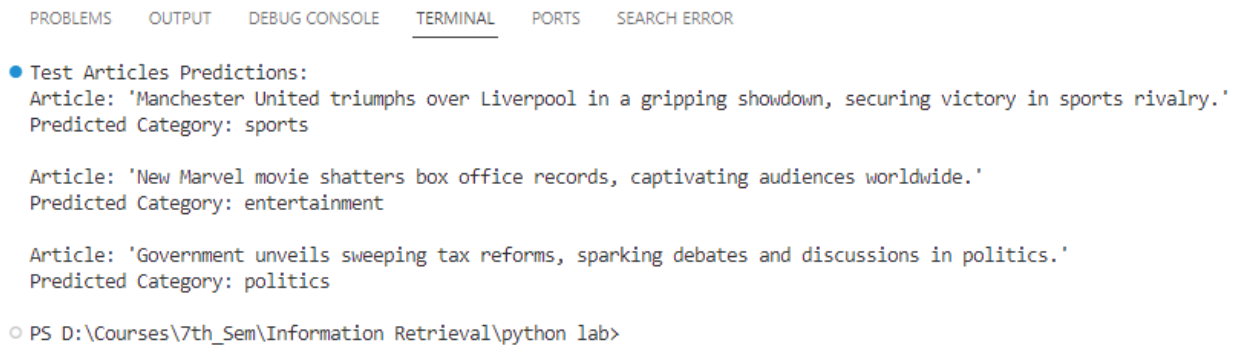
**6. Group the online news onto different categorize like sports, entertainment, politics.**

```
import string
news_data = [
    ("Manchester United defeats Liverpool in a thrilling match", "sports"),
    ("New Marvel movie breaks box office records", "entertainment"),
    ("Government announces new tax reforms", "politics"),
    ("Political leaders discuss trade deal", "politics"),
    ("Real Madrid wins the Champions League", "sports"),
    ("New album released by popular band", "entertainment"),
    ("Election results announced by officials", "politics"),
    ("Nadal advances to semi-finals in Wimbledon", "sports"),
    ("Movie premiere attended by Hollywood stars", "entertainment"),
]

def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    return text.split()

def build_vocabulary(data):
    vocabulary = set()
    for news_item, _ in data:
        words = preprocess_text(news_item)
        vocabulary.update(words)
    return vocabulary

def calculate_class_priors(data):
    class_counts = {}
    total_count = len(data)
    for _, category in data:
        if category in class_counts:
            class_counts[category] += 1
        else:
            class_counts[category] = 1
    priors = {category: count/total_count for category, count in
class_counts.items()}
    return priors

def calculate_word_probabilities(data, vocabulary):
    word_counts = {category: {word: 0 for word in vocabulary} for _, category in
data}
    class_counts = {category: 0 for _, category in data}

    for news_item, category in data:
```

```
        words = preprocess_text(news_item)
        class_counts[category] += len(words)
        for word in words:
            word_counts[category][word] += 1
    word_probs = {category: {word: (count + 1) / (class_counts[category] +
len(vocabulary))
                             for word, count in word_counts[category].items()}
                  for category in class_counts}

    return word_probs

def naive_bayes_classifier(news_item, class_priors, word_probs):
    words = preprocess_text(news_item)
    scores = {category: class_priors[category] for category in class_priors}
    for category in scores:
        for word in words:
            if word in word_probs[category]:
                scores[category] *= word_probs[category][word]

    predicted_category = max(scores, key=scores.get)
    return predicted_category

def main():
    # Test news
    test_articles = [
        "Manchester United triumphs over Liverpool in a gripping showdown,
securing victory in sports rivalry.",
        "New Marvel movie shatters box office records, captivating audiences
worldwide.",
        "Government unveils sweeping tax reforms, sparking debates and
discussions in politics."
    ]
    # Build vocabulary and calculate probabilities
    vocabulary = build_vocabulary(news_data)
    class_priors = calculate_class_priors(news_data)
    word_probs = calculate_word_probabilities(news_data, vocabulary)
    print("Test Articles Predictions:")
    for article in test_articles:
        predicted_category = naive_bayes_classifier(article, class_priors,
word_probs)
        print(f"Article: '{article}'")
        print(f"Predicted Category: {predicted_category}")
        print()

if __name__ == "__main__":
```

```
main()
```

## Output


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SEARCH ERROR
Test Articles Predictions:
Article: 'Manchester United triumphs over Liverpool in a gripping showdown, securing victory in sports rivalry.'
Predicted Category: sports
Article: 'New Marvel movie shatters box office records, captivating audiences worldwide.'
Predicted Category: entertainment
Article: 'Government unveils sweeping tax reforms, sparking debates and discussions in politics.'
Predicted Category: politics
PS D:\Courses\7th_Sem\Information Retrieval\python lab>

**7. Build a recommender system for online music store.**

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

# Sample music data (user-item matrix)
music_data = {
    'User': ['User1', 'User2', 'User3', 'User4', 'User5'],
    'Pop': [5, 4, 0, 0, 3],
    'Rock': [0, 0, 5, 4, 0],
    'Hip-Hop': [4, 0, 0, 0, 5],
    'Classical': [0, 5, 3, 0, 4],
    'Flok': [0, 0, 4, 5, 0],
}

# Convert data to DataFrame
df = pd.DataFrame(music_data)

# Calculate similarity matrix using cosine similarity
similarity_matrix = cosine_similarity(df.drop('User', axis=1))

# Function to recommend tracks for a given user
def recommend_tracks(user_id, similarity_matrix, num_recommendations=3):
    user_index = df[df['User'] == user_id].index[0]
    user_similarities = similarity_matrix[user_index]
    similar_users_indices = user_similarities.argsort()[-num_recommendations-1:-
1][::-1]  # Exclude user's own index
    recommended_tracks = []
    for index in similar_users_indices:
        similar_user_id = df.iloc[index]['User']
        user_tracks = df.iloc[index][1:]
        recommended_tracks.extend(user_tracks[user_tracks > 0].index)  # Only
recommend tracks with rating > 0
    return list(set(recommended_tracks))[:num_recommendations]  # Remove
duplicates and limit recommendations

# Example usage

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

# Sample music data (user-item matrix)
music_data = {
    'User': ['User1', 'User2', 'User3', 'User4', 'User5'],
    'Pop': [5, 4, 0, 0, 3],
```

```
    'Rock': [0, 0, 5, 4, 0],
    'Hip-Hop': [4, 0, 0, 0, 5],
    'Classical': [0, 5, 3, 0, 4],
    'Flok': [0, 0, 4, 5, 0],
}

# Convert data to DataFrame
df = pd.DataFrame(music_data)

# Calculate similarity matrix using cosine similarity
similarity_matrix = cosine_similarity(df.drop('User', axis=1))

# Function to recommend tracks for a given user
def recommend_tracks(user_id, similarity_matrix, num_recommendations=3):
    user_index = df[df['User'] == user_id].index[0]
    user_similarities = similarity_matrix[user_index]
    similar_users_indices = user_similarities.argsort()[-num_recommendations-1:-
1][::-1]  # Exclude user's own index
    recommended_tracks = []
    for index in similar_users_indices:
        similar_user_id = df.iloc[index]['User']
        user_tracks = df.iloc[index][1:]
        recommended_tracks.extend(user_tracks[user_tracks > 0].index)  # Only
recommend tracks with rating > 0
    return list(set(recommended_tracks))[:num_recommendations]  # Remove
duplicates and limit recommendations

# Example usage

user_id1 = 'User5'
recommended_tracks = recommend_tracks(user_id1, similarity_matrix)
print("Recommended tracks for", user_id1, ":", recommended_tracks)
```

## Output

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   SEARCH ERROR

PS D:\Courses\7th_Sem\Information Retrieval\python lab> python -u "d:\Courses\7th_Sem
Recommended tracks for User1 : ['Classical', 'Rock', 'Hip-Hop']
PS D:\Courses\7th_Sem\Information Retrieval\python lab> python -u "d:\Courses\7th_Sem
Recommended tracks for User3 : ['Flok', 'Hip-Hop', 'Classical']
PS D:\Courses\7th_Sem\Information Retrieval\python lab> python -u "d:\Courses\7th_Sem
Recommended tracks for User5 : ['Flok', 'Pop', 'Classical']
PS D:\Courses\7th_Sem\Information Retrieval\python lab>
```