

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**FACULTY OF SCIENCE AND TECHNOLOGY**

(Formerly SRM University, Under section 3 of UGC Act, 1956)

**S.R.M NAGAR, KATTANKULATHUR – 603 203,**

**KANCHEEPURAM DISTRICT**

**SCHOOL OF COMPUTING**

**DEPARTMENT OF NETWORKING AND COMMUNICATIONS**

**Course Code:** 18CSE305J

**Course Name:** Artificial Intelligence

**Course Project**

**Title:** Plagiarism Checker

**Team Members:**

1. RA1911030010069 – Praveen Kumar
2. RA1911030010090 – Tejas Ashok
3. RA1911030010103 – Vinoth S

**Date:** 18-04-2022

**Title:** Plagiarism Checker - Python

**Problem Statement:**

We all know that computers are good at numbers, so in order to compute the similarity between on two text documents, the textual raw data is transformed into vectors => arrays of numbers and then from that we are going to use a basic knowledge vector to compute the similarity between them.

This python application identifies similarities between a test.txt file that can be created in the local directory and a website. The link to the website can be provided at the start of the program and the application scrapes the website using requests and beautifulsoup. The driver code then compares the scraped data and test.txt for similarities and produces the results.

**Working:**

The Plagiarism Checker uses the request module to send requests to a website (Wikipedia currently supported), to receive all the html data from the website.

Beautifulsoup then parses the raw html and organizes the data so that it could be manipulated.

The data is then split according to headings on the website and then are stored in sperate text files according to their headings.

The data of all the files are then compared with each other by converting the data to vectors and then comparing it with cosine\_similarity.

The output is then displayed. The files with similarity close to 1 are considered copied.

**Code:**

import os

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import requests

from bs4 import BeautifulSoup as bs

# Directory of the files

DIR = "files"

# URL and Headers

url = input("Enter the URL (Wikipedia Only): ")

headers = {

    "Host": "en.wikipedia.org",

    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0"

}

# Request the url with headers and convert to text

r = requests.get(url, headers=headers).text

# Creates a bs object to perform parsing

soup = bs(r, 'lxml')

TITLE = soup.find('h1').get\_text()

# Dictionary to store parsed string

parsed\_dic = {}

# Parsing

para = soup.find('p', class\_=None)

for d in para.find\_all('sup'):

    d.decompose()

parsed\_dic["Introduction"] = para.get\_text()

for tag in soup.find\_all('h2'):

    sib = tag.find\_next\_sibling('p')

    if sib is None:

        continue

    p = ""

    while(sib is not None and sib.find\_previous\_sibling('h2').find('span').get\_text() == tag.find('span').get\_text()):

        for d in sib.find\_all('sup'):

            d.decompose()

        p += sib.get\_text()

        sib = sib.find\_next\_sibling('p')

    parsed\_dic[tag.find('span').get\_text()] = p

# Creates seperate txt files for every heading in Wikipedia

for key, value in parsed\_dic.items():

    with open(DIR + os.sep + key + '.txt', 'w', encoding="utf-8") as f:

        f.write(value)

# Creates a list of files and its data

student\_files = [doc for doc in os.listdir(DIR) if doc.endswith('.txt')]

student\_notes = [open(DIR + os.sep + \_file, encoding='utf-8').read()

                 for \_file in student\_files]

# Creates vectors of the data of each file

def vectorize(Text): return TfidfVectorizer().fit\_transform(Text).toarray()

def similarity(doc1, doc2): return cosine\_similarity([doc1, doc2])

# Compares every files vector with each other

vectors = vectorize(student\_notes)

s\_vectors = list(zip(student\_files, vectors))

plagiarism\_results = set()

# Function to compare the vectors

def check\_plagiarism():

    global s\_vectors

    for student\_a, text\_vector\_a in s\_vectors:

        new\_vectors = s\_vectors.copy()

        current\_index = new\_vectors.index((student\_a, text\_vector\_a))

        del new\_vectors[current\_index]

        for student\_b, text\_vector\_b in new\_vectors:

            sim\_score = similarity(text\_vector\_a, text\_vector\_b)[0][1]

            student\_pair = sorted((student\_a, student\_b))

            score = (student\_pair[0], student\_pair[1], sim\_score)

            plagiarism\_results.add(score)

    return plagiarism\_results

# Print the result

print("Not very Similar:")

for data in check\_plagiarism():

    if data[1].split('.')[0] == 'test' and data[2] <= 0.5:

        print(data)

print()

print("Are kind of Similar:")

for data in check\_plagiarism():

    if data[1].split('.')[0] == 'test' and data[2] > 0.5 and data[2] <= 0.75:

        print(data)

print()

print("A lot Similar:")

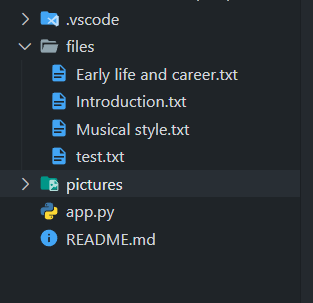
for data in check\_plagiarism():

    if data[1].split('.')[0] == 'test' and data[2] > 0.75:

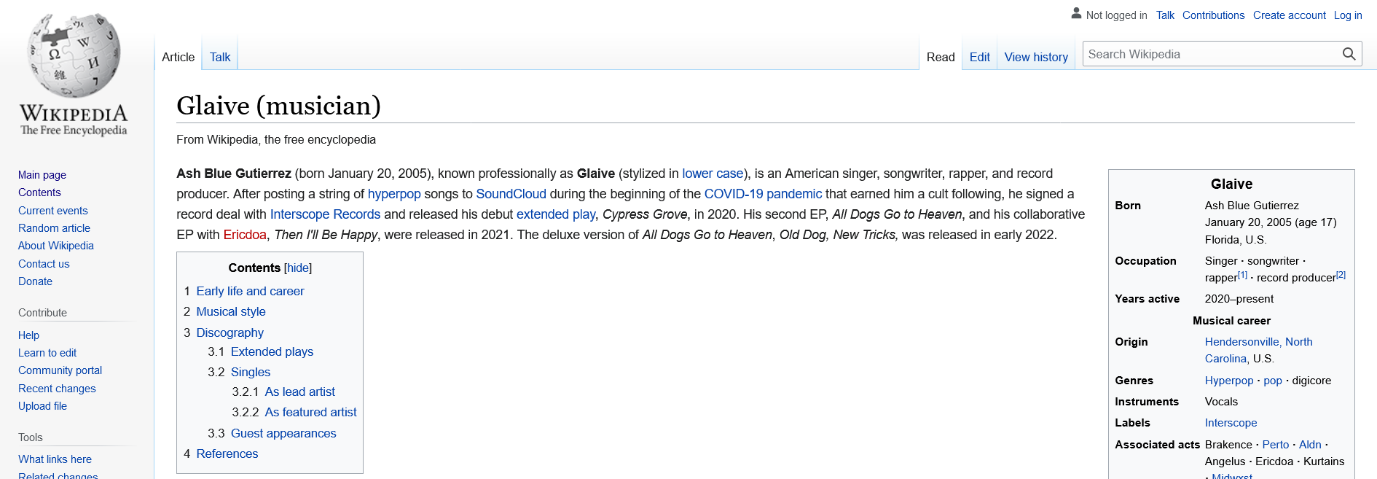
        print(data)

**Test Case #1:**

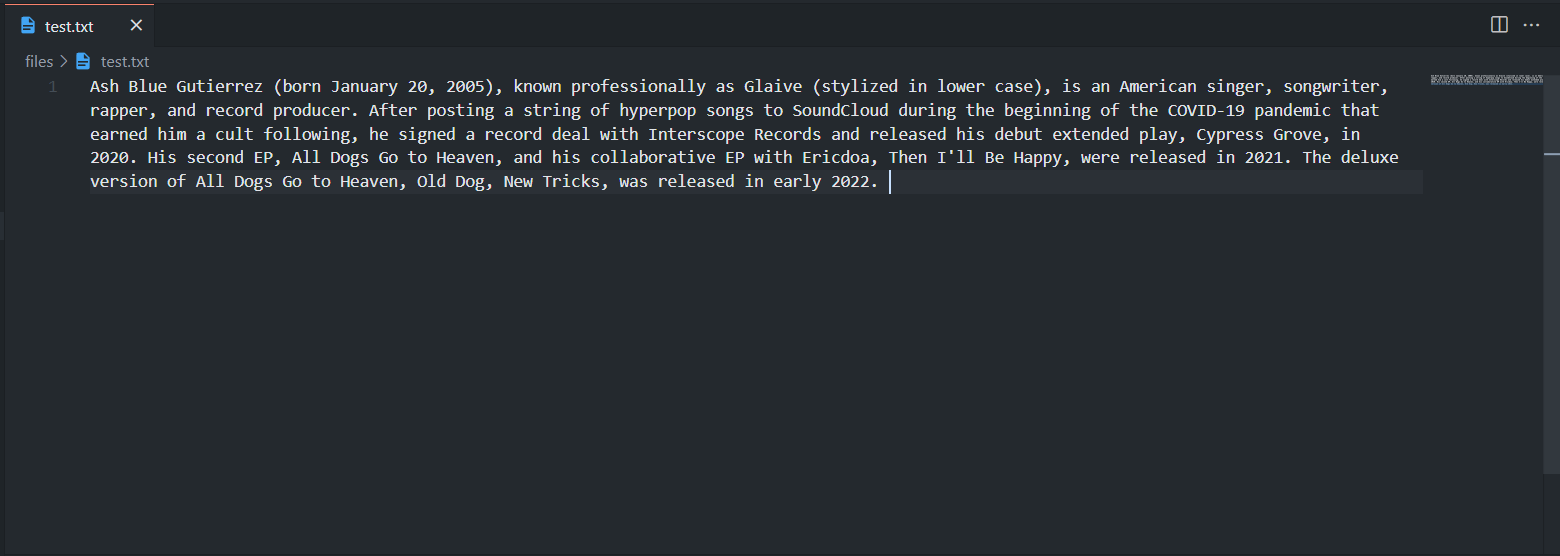
**Directory**



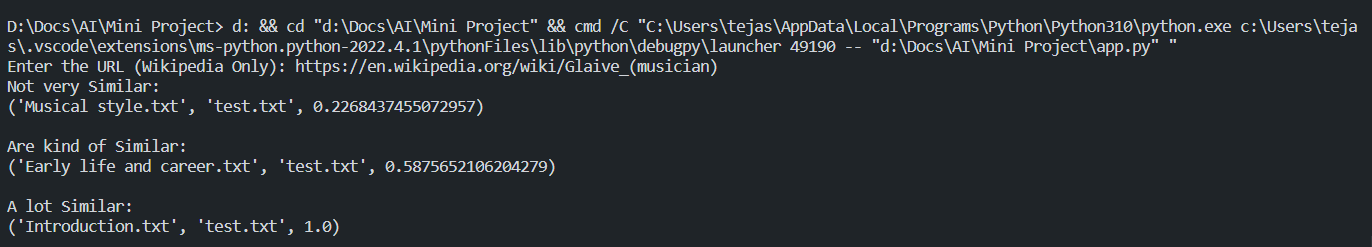
**Webpage**



**Input**

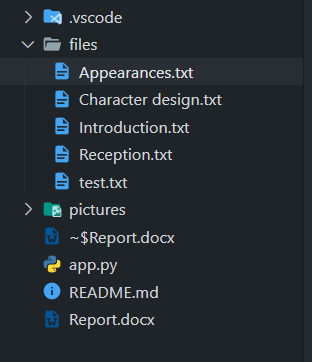


**Output**

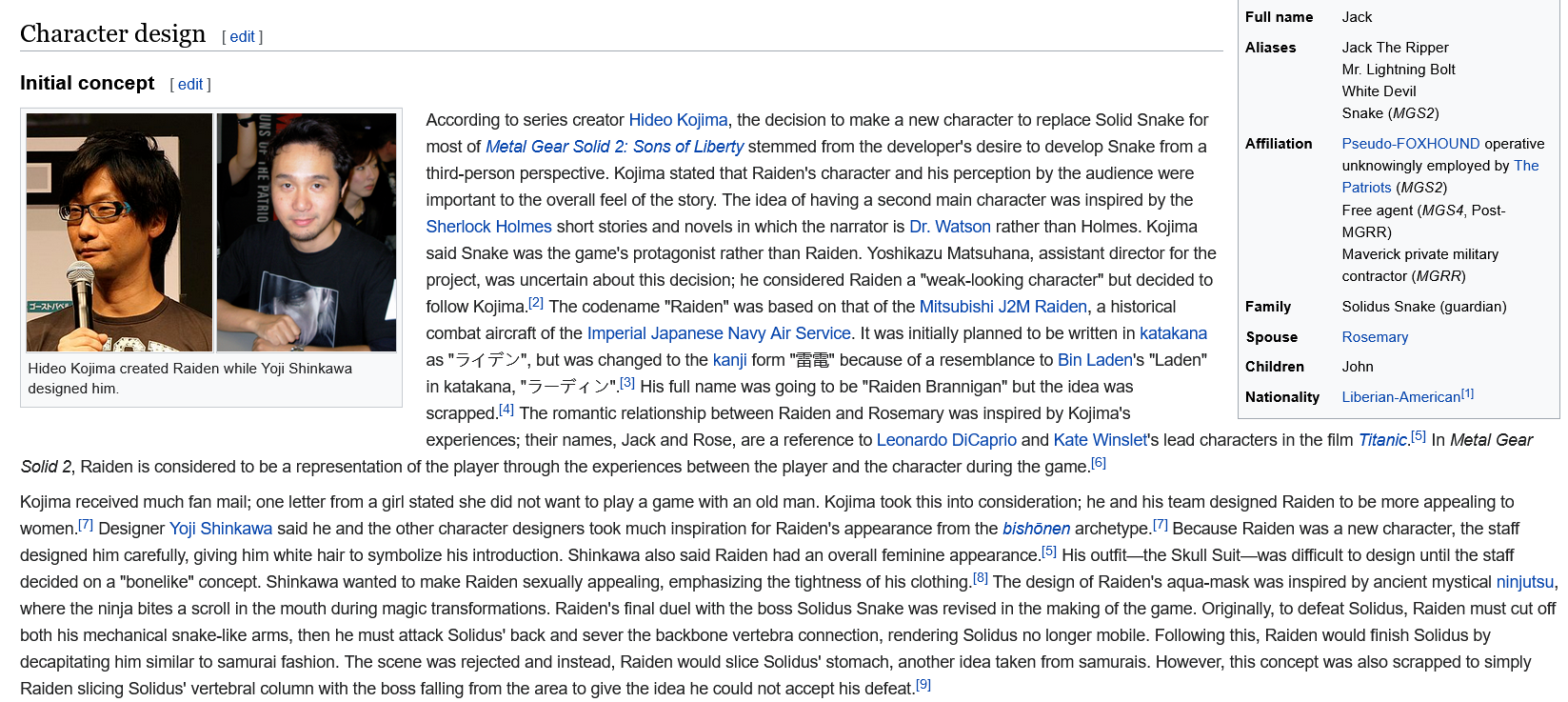


**Test Case #2:**

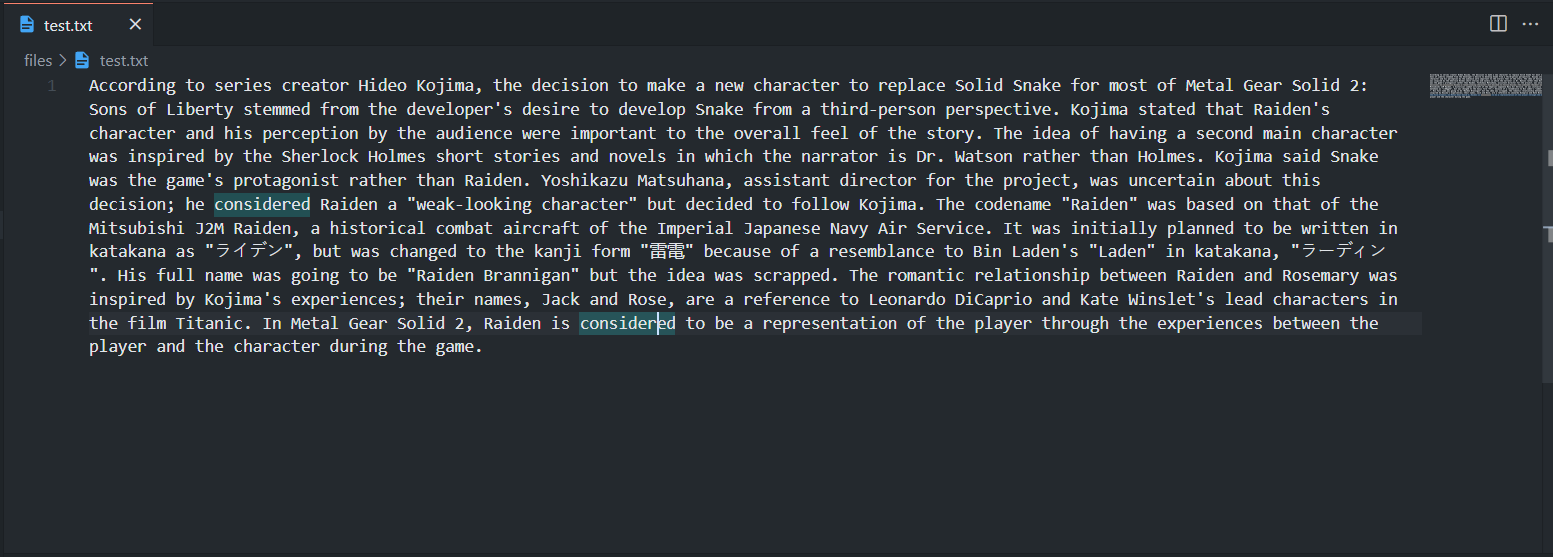
**Directory**



**Webpage**



**Input**



**Output**

