import docx

import csv

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

# Load the Word document

doc = docx.Document("/content/resume1.docx")

# Initialize a list to store the extracted content

csv\_data = []

# Iterate through paragraphs and extract text

for paragraph in doc.paragraphs:

    csv\_data.append([paragraph.text])

# Write the extracted content to a CSV file

csv\_file\_path = "/content/output1.csv"

with open(csv\_file\_path, 'w', newline='', encoding='utf-8') as csvfile:

    csv\_writer = csv.writer(csvfile)

    csv\_writer.writerows(csv\_data)

print("Conversion completed.")

# Load the CSV file into a DataFrame

df = pd.read\_csv(r'/content/output1.csv', header=None)

# Concatenate the content into a single column

content = df[0].str.cat(sep='\n')

# Find the indices of "Skills:" and "Experience:"

skills\_index = content.find("Skills:")

experience\_index = content.find("Experience:")

# Extract the skills data

skills\_data = content[skills\_index + len("Skills:"):experience\_index].strip()

# Split the skills data into individual skills

skills\_list = [skill.strip("- ").strip() for skill in skills\_data.split('\n') if skill.strip()]

resumes = skills\_list

# Print the extracted skills

for skill in skills\_list:

    print(skill)

job\_descriptions = [

    "We are looking for a software engineer to develop web applications.",

    "Join our team as a software developer to create innovative software solutions."

]

# Combine job descriptions and resumes

documents = job\_descriptions + resumes

# Create a CountVectorizer instance

vectorizer = CountVectorizer().fit\_transform(documents)

# Calculate cosine similarity between job descriptions and resumes

cosine\_similarities = cosine\_similarity(vectorizer[0:2], vectorizer[2:])

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

def suggest\_skills(job\_description, skills\_set):

    # Prepare job descriptions and skills set

    documents = [job\_description] + resumes

    # Create a CountVectorizer instance

    vectorizer = CountVectorizer().fit\_transform(documents)

    # Calculate cosine similarity between job description and skills

    cosine\_similarities = cosine\_similarity(vectorizer[0:1], vectorizer[1:])

    # Find the most similar skill based on cosine similarity

    most\_similar\_skill\_idx = cosine\_similarities.argmax()

    # Suggest skills from the skills set

    suggested\_skills = set(skills\_set) - set(skills\_set[most\_similar\_skill\_idx])

    return suggested\_skills

# Example job description and skills set

job\_description = "We are looking for a software engineer to develop web applications."

skills\_set = [

    "programming",

    "web development",

    "software engineering",

    "problem solving",

    "communication"

]

# Print suggested skills

if suggested\_skills:

    print("Suggested skills for this job:")

    for skill in suggested\_skills:

        print(skill)

else:

    print("No additional skills suggested.")

    from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

def suggest\_development\_plan(resumes, job\_description\_skills):

    # Find the skills gap

    skills\_gap = set(skills\_set) - set(resumes)

    if not skills\_gap:

        return "You already possess all the necessary skills for this role."

    # Suggest a tailored development plan

    suggested\_plan = f"To excel in the new role, you should focus on developing the following skills:\n"

    for skill in skills\_gap:

        suggested\_plan += f"- {skill}\n"

    return suggested\_plan

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

def rank\_candidates(job\_description, candidate\_skills):

    # Create a CountVectorizer instance

    vectorizer = CountVectorizer().fit\_transform([job\_description] + candidate\_skills)

    # Calculate cosine similarity between job description and candidate skills

    cosine\_similarities = cosine\_similarity(vectorizer[0:1], vectorizer[1:])

    # Combine candidates with their cosine similarity scores

    ranked\_candidates = list(zip(candidate\_skills, cosine\_similarities[0]))

    # Sort candidates by similarity score in descending order

    ranked\_candidates.sort(key=lambda x: x[1], reverse=True)

    return ranked\_candidates

# Example job description and candidate skills

job\_description = "We are looking for a software engineer to develop web applications."

candidates = [

    "Experienced software engineer with expertise in web development and problem solving.",

    "Junior developer with strong programming skills and a passion for web development.",

    "Data analyst with skills in data processing and visualization."

]

# Rank the candidates based on similarity to the job description

ranked\_candidates = rank\_candidates(job\_description, candidates)

# Print the ranked candidates

print("Ranked candidates:")

for candidate, similarity in ranked\_candidates:

    print(f"Candidate: {candidate}")

    print(f"Similarity to job description: {similarity:.2f}")

    print()

    # Find the best matching job for each resume

for i, resume\_similarities in enumerate(cosine\_similarities):

    best\_matching\_job\_idx = resume\_similarities.argmax()

    if resume\_similarities[best\_matching\_job\_idx] > 0:

        best\_matching\_job = job\_descriptions[best\_matching\_job\_idx]

        print(f"Best matching job for Resume {i+1}: {best\_matching\_job}")

    else:

        print(f"No suitable job found for Resume {i+1}")

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

def suggest\_career\_paths(individual\_abilities, career\_opportunities):

    suggested\_paths = []

    for career, career\_skills in career\_opportunities.items():

        # Create a CountVectorizer instance

        vectorizer = CountVectorizer().fit\_transform([', '.join(individual\_abilities), ', '.join(career\_skills)])

        # Calculate cosine similarity between individual abilities and career skills

        cosine\_similarity\_score = cosine\_similarity(vectorizer[0:1], vectorizer[1:])[0][0]

        if cosine\_similarity\_score > 0:

            suggested\_paths.append((career, cosine\_similarity\_score))

    # Sort suggested paths by similarity score in descending order

    suggested\_paths.sort(key=lambda x: x[1], reverse=True)

    return suggested\_paths

# Example individual abilities and career opportunities

individual\_abilities = ["programming", "problem solving", "communication"]

career\_opportunities = {

    "Software Engineer": ["programming", "web development", "problem solving"],

    "Data Analyst": ["data analysis", "statistics", "communication"],

    "Marketing Specialist": ["marketing", "communication", "creativity"]

}

# Suggest career paths based on individual abilities

suggested\_career\_paths = suggest\_career\_paths(individual\_abilities, career\_opportunities)

# Print suggested career paths

if suggested\_career\_paths:

    print("Suggested career paths:")

    for career, similarity in suggested\_career\_paths:

        print(f"Career: {career}")

        print(f"Similarity to individual abilities: {similarity:.2f}")

        print()

else:

    print("No suitable career paths found.")