PLAGIARISM CHECKER

**PLAGIARISM:**Plagiarism means using someone else’s work without giving them credit.In writing, plagiarizing involves uses of words, ideas, or information from a source without citing it correctly.

**CODE:**

Import the libraries

import os

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

* The OS is for file handling.
* The ‘TfidfVectorizer’ from sklearn is used to convert text data into numerical vectors.
* The ‘cosine-similarity’ from sklearn is used to compute similarity between vectors.

def read\_file(file\_path):

with open(file\_path, 'r', encoding='utf-8') as file:

return file.read()

* It's a file reading function by uploading a file path.

def vectorize(texts):

return TfidfVectorizer().fit\_transform(texts).toarray()

* This function takes a list of text documents, converts them into TF-IDF vectors and returns these vectors as arrays.

def similarity(doc1, doc2):

return cosine\_similarity([doc1, doc2])

* This function computes the cosine similarity between two documents represented as vectors.

def compare\_files(file1\_path, file2\_path):

text1 = read\_file(file1\_path)

text2 = read\_file(file2\_path)

similarity\_score = compute\_similarity(text1, text2)

return (os.path.basename(file1\_path), os.path.basename(file2\_path), similarity\_score)

* This function compares two files by:
  + Reading their contents.
  + Computing the similarity between them using ‘compute\_similarity’.
  + Returning a tuple containing the filenames and their similarity score.

def compute\_similarity(text1, text2):

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform([text1, text2])

similarity\_matrix = cosine\_similarity(tfidf\_matrix)

return similarity\_matrix[0, 1]

* This function computes the similarity between two text strings by:
  + Vectorizing the texts.
  + Calculating the cosine similarity between the resulting TF-IDF vectors.
  + Returning the similarity score.

def check\_plagiarism(files):

student\_notes = [read\_file(file) for file in files]

vectors = vectorize(student\_notes)

s\_vectors = list(zip(files, vectors))

plagiarism\_results = set()

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((os.path.basename(student\_a), os.path.basename(student\_b)))

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

plagiarism\_results.add(score)

return plagiarism\_results

* This function checks for plagiarism among multiple files by:
  + Reading the contents of the files.
  + Vectorizing the texts.
  + Pairwise comparing each document with every other document.
  + Storing the similarity scores in a set to avoid duplicates.
  + Returning the results as a set of tuples containing filenames and similarity scores.

def main():

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

initial\_results = check\_plagiarism(student\_files)

for result in initial\_results:

print(result)

while True:

another\_files = input("\nDo you want to compare other files not included in the initial set? (yes/no): ").strip().lower()

if another\_files != 'yes':

break

new\_file1\_path = input("Enter the path for the first new file: ").strip()

new\_file2\_path = input("Enter the path for the second new file: ").strip()

result = compare\_files(new\_file1\_path, new\_file2\_path)

print(f"('{result[0]}', '{result[1]}', {result[2]})")

if \_\_name\_\_ == "\_\_main\_\_":

main()

* The ‘main’ function:
* Lists all text files in the current directory.
* Calls ‘check\_plagiarism’ on these files and prints the results.
* Provides an interactive loop to allow the user to compare additional pairs of files, repeatedly asking for file paths and comparing them until the user chooses to stop.