**Title:** Fingerprint Matching using Python

**Introduction:**

Fingerprint recognition is a widely used biometric technique for identifying individuals. It is based on the unique patterns found on an individual's fingertips. Fingerprint matching is a critical component of fingerprint recognition systems, which involves comparing a query fingerprint with a set of stored fingerprints to determine the identity of an individual. With the increasing use of biometric systems in various applications, the need for efficient and accurate fingerprint matching algorithms has become more pressing. This project aims to develop a fingerprint matching system using Python, which is a popular programming language known for its simplicity and flexibility.

**Existing System:**

Traditional fingerprint matching systems use manual methods, which are time-consuming and prone to errors. These systems rely on human experts to compare fingerprints, which can lead to inconsistencies and inaccuracies. Moreover, manual methods are not scalable and cannot handle large databases of fingerprints. Automated fingerprint matching systems have been developed to overcome these limitations, but they often require specialized hardware and software, making them expensive and inaccessible to many organizations.

**Proposed System:**

The proposed system uses Python to develop a fingerprint matching algorithm that is efficient, accurate, and scalable. The system consists of the following components:

1. **Fingerprint Acquisition:** The system uses a fingerprint scanner to capture fingerprint images.
2. **Image Preprocessing:** The captured images are preprocessed to enhance the quality and remove noise.
3. **Feature Extraction:** The system extracts features from the preprocessed images, such as minutiae points (ridge endings and bifurcations).
4. **Matching:** The system compares the features of the query fingerprint with the features of the stored fingerprints to determine the best match.

**Improvisation:**

The proposed system improvises on existing fingerprint matching systems in several ways:

1. **Efficiency:** The system uses Python, which is a fast and efficient language, to reduce the processing time and improve the overall performance.
2. **Accuracy:** The system uses advanced image processing techniques to enhance the quality of the fingerprint images, leading to more accurate feature extraction and matching.
3. **Scalability:** The system can handle large databases of fingerprints, making it suitable for large-scale applications.

**Advantages:**

1. **Efficient:** The system is fast and efficient, making it suitable for real-time applications.
2. **Accurate:** The system uses advanced image processing techniques to improve the accuracy of feature extraction and matching.
3. **Scalable:** The system can handle large databases of fingerprints, making it suitable for large-scale applications.
4. **Cost-effective:** The system uses Python, which is an open-source language, making it cost-effective.

**Disadvantages:**

1. **Limited Robustness:** The system may not perform well in cases where the fingerprint images are of poor quality or have been tampered with.
2. **Dependence on Hardware:** The system requires a fingerprint scanner to capture fingerprint images, which may not be available in all environments.

**Code:**

import cv2

import numpy as np

# Define the extract\_features function

def extract\_features(image):

# Convert the image to grayscale

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Apply Gaussian filter to enhance the image

gray\_image = cv2.GaussianBlur(gray\_image, (5, 5), 0)

# Apply thresholding to binarize the image

\_, binary\_image = cv2.threshold(gray\_image, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

# Extract minutiae points

minutiae\_points = []

for i in range(binary\_image.shape[0]):

for j in range(binary\_image.shape[1]):

if binary\_image[i, j] == 255:

minutiae\_points.append((i, j))

return minutiae\_points

# Define the match\_features function

def match\_features(query\_features, template\_features):

# Calculate the distance between the query and template features

distance = 0

for i in range(len(query\_features)):

distance += np.linalg.norm(np.array(query\_features[i]) - np.array(template\_features[i]))

# Calculate the matching score

score = 1 - (distance / len(query\_features))

return score

# Add the file paths to the query and template fingerprints

query\_image\_path = "C:/Users/priyadarshini/OneDrive/Desktop/aiml project/fingerprint1.jpg"

template\_image\_path = "C:/Users/priyadarshini/OneDrive/Desktop/aiml project/fingerprint2.jpg"

# Read the fingerprint images from their file paths

query\_image = cv2.imread(query\_image\_path)

template\_image = cv2.imread(template\_image\_path)

# Preprocess the images

query\_image = cv2.resize(query\_image, (512, 512))

template\_image = cv2.resize(template\_image, (512, 512))

# Extract features from the images

query\_features = extract\_features(query\_image)

template\_features = extract\_features(template\_image)

# Match the features

score = match\_features(query\_features, template\_features)

# Print the matching score

print('Matching Score:', score)

**Output:**

The output of the system is a matching score, which indicates the similarity between the query fingerprint and the stored fingerprints. The matching score ranges from 0 to 1