**Step-by-Step Explanation of the Code and Libraries Used**

This Python script implements a Face Recognition Attendance System using **OpenCV, Face Recognition, MySQL, and Tkinter** for the graphical interface.

**1. Importing Libraries**

python

import cv2

import numpy as np

import face\_recognition

import mysql.connector

from datetime import datetime

import tkinter as tk

from tkinter import simpledialog, messagebox

from PIL import Image, ImageTk

import pickle**Why did we add these libraries?**

| **Library** | **Purpose** |
| --- | --- |
| cv2 (OpenCV) | Used for capturing video from the camera and processing images. |
| numpy (NumPy) | Used for handling numerical computations (e.g., face distance calculation). |
| face\_recognition | Used for detecting and recognizing faces. |
| mysql.connector | Used to connect and interact with the MySQL database. |
| datetime | Used to get the current date and time for attendance marking. |
| tkinter (Tk) | Used to create a graphical user interface (GUI) for the application. |
| PIL (Image, ImageTk) | Used to convert OpenCV frames into images for Tkinter display. |
| pickle | Used to store and retrieve face encodings in binary format from the database. |

**Why We Use These Libraries?**

* **cv2 (OpenCV)** → Used for capturing live video and processing images.
* **numpy** → Used for numerical computations, especially for face encoding comparisons.
* **face\_recognition** → Core library for detecting and recognizing faces.
* **mysql.connector** → Used for connecting to the MySQL database to store face encodings and attendance records.
* **datetime** → Helps in recording attendance timestamps.
* **tkinter** → Used for creating the graphical user interface (GUI).
* **PIL (Pillow)** → Used to convert OpenCV images into formats compatible with Tkinter.
* **pickle** → Used to serialize and deserialize face encodings before storing in MySQL.

**2. Connecting to the MySQL Database**

python

conn = mysql.connector.connect(

host="52.20.7.147",

user="petalhosuser",

password="PetalHosDb@20#19",

database="face\_attendance"

)

cursor = conn.cursor()

* This establishes a connection to the **MySQL database**.
* The cursor object is used to execute SQL queries.

**3. GUI Design Using Tkinter**

python

class FaceRecognitionApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Face Recognition Attendance")

self.root.geometry("800x500")

self.root.configure(bg="white")

* Creates the **main application window** with Tkinter.
* Sets the **title**, **size**, and **background color**.

**4. Adding a Video Feed to GUI**

python

self.video\_label = tk.Label(self.root)

self.video\_label.pack(side=tk.LEFT, padx=10, pady=10)

* Creates a label where the **live camera feed** will be displayed.

def update\_video(self):

ret, frame = self.cap.read()

if ret:

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frame = cv2.resize(frame, (400, 300))

img = Image.fromarray(frame)

imgtk = ImageTk.PhotoImage(image=img)

self.video\_label.imgtk = imgtk

self.video\_label.config(image=imgtk)

self.root.after(10, self.update\_video)

* **Purpose:** Converts the captured OpenCV frame to an image format compatible with Tkinter and updates it every 10 milliseconds.

**5. Register a Face**

python

def register\_face(self):

"""Capture and register face into the database"""

ret, frame = self.cap.read()

if ret:

name = simpledialog.askstring("Input", "Enter Name:")

* **Purpose:** Captures a frame from the webcam and prompts the user to enter a name.

python

if name:

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

face\_encodings = face\_recognition.face\_encodings(rgb\_frame)

* \*\*Converts the frame from **BGR (OpenCV format) to RGB (Face Recognition format)**.
* **Extracts face encodings** using face\_recognition.face\_encodings().

python

if face\_encodings:

face\_encoding\_blob = pickle.dumps(face\_encodings[0])

* **Serializes the face encoding** into binary format using pickle.dumps(), making it compatible for database storage.

python

# Check if face is already registered

cursor.execute('SELECT name FROM users')

records = cursor.fetchall()

known\_encodings = []

known\_names = []

* **Fetches all existing names and face encodings** from the database.

python

for record in records:

cursor.execute('SELECT face\_encoding FROM users WHERE name = %s', (record[0],))

encoding\_blob = cursor.fetchone()[0]

encoding = pickle.loads(encoding\_blob)

known\_encodings.append(encoding)

known\_names.append(record[0])

* **Loads each stored face encoding** and name into separate lists.

python

matches = face\_recognition.compare\_faces(known\_encodings, face\_encodings[0])

if True in matches:

matched\_idx = matches.index(True)

matched\_name = known\_names[matched\_idx]

messagebox.showinfo("Already Registered", f"This face is already registered under the name: {matched\_name}")

* **Checks if the captured face is already registered**.
* If **matched**, displays a warning message.

python

else:

# Insert into database

cursor.execute('INSERT INTO users (name, face\_encoding) VALUES (%s, %s)', (name, face\_encoding\_blob))

conn.commit()

messagebox.showinfo("Success", f"{name} registered successfully!")

* If **not matched**, the face is inserted into the database.

**6. Mark Attendance**

python

def mark\_attendance(self):

"""Mark attendance for recognized faces"""

known\_encodings, known\_names, known\_ids = self.fetch\_registered\_faces()

* **Fetches all registered faces from the database**.

python

ret, frame = self.cap.read()

if ret:

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

face\_locations = face\_recognition.face\_locations(rgb\_frame)

face\_encodings = face\_recognition.face\_encodings(rgb\_frame, face\_locations)

* Captures the webcam frame, **detects face locations and encodes faces**.

python

for encoding in face\_encodings:

matches = face\_recognition.compare\_faces(known\_encodings, encoding)

face\_distances = face\_recognition.face\_distance(known\_encodings, encoding)

best\_match\_index = np.argmin(face\_distances) if matches else -1

* **Compares the detected face against stored faces**.

python

if best\_match\_index != -1 and matches[best\_match\_index]:

name = known\_names[best\_match\_index]

user\_id = known\_ids[best\_match\_index]

date = datetime.now().strftime('%Y-%m-%d')

time = datetime.now().strftime('%H:%M:%S')

* **If a match is found**, fetches the corresponding **user ID and name**.

python

cursor.execute('SELECT \* FROM attendance WHERE user\_id = %s AND date = %s', (user\_id, date))

result = cursor.fetchone()

if result is None:

cursor.execute('INSERT INTO attendance (user\_id, date, time) VALUES (%s, %s, %s)', (user\_id, date, time))

conn.commit()

messagebox.showinfo("Attendance Marked", f"Attendance recorded for {name}")

* **Checks if attendance was already marked today** and inserts a new record if not.

**7. Delete a Registered Face**

python

def delete\_face(self):

"""Delete a registered face from the database"""

name = simpledialog.askstring("Delete Face", "Enter Name to Delete:")

* **Prompts the user to enter the name** of the person to be deleted.

python

if name:

cursor.execute('SELECT id FROM users WHERE name = %s', (name,))

user = cursor.fetchone()

* **Fetches the user ID from the database**.

python

if user:

user\_id = user[0]

confirm = messagebox.askquestion("Delete", f"Are you sure you want to delete {name}?", icon='warning')

if confirm == 'yes':

cursor.execute('DELETE FROM attendance WHERE user\_id = %s', (user\_id,))

cursor.execute('DELETE FROM users WHERE id = %s', (user\_id,))

conn.commit()

messagebox.showinfo("Success", f"{name} deleted successfully!")

* **Deletes both user data and attendance records**.

**Conclusion**

This program allows users to:

1. **Register Faces** (Avoids duplicates).
2. **Mark Attendance** (Records date and time).
3. **Delete Faces** (Removes user and their attendance).
4. **Live Video Feed** using OpenCV.