**Face Recognition using opencv**

**Project Description:**

This project aims to develop a robust face recognition system utilizing the capabilities of MTCNN (Multi-task Cascaded Convolutional Networks) and FaceNet. The system will be designed to identify or verify individuals based on their facial features captured in images.

Employing MTCNN, the system will accurately detect faces within input images, even under challenging conditions like variations in pose, lighting, and occlusion.

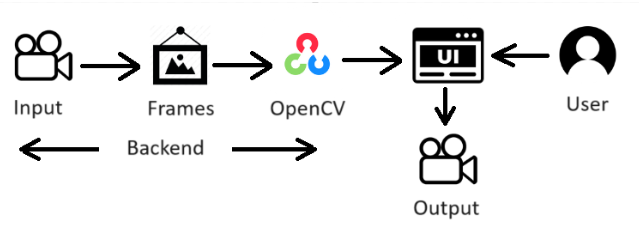
Once a face is detected, FaceNet, a deep learning model, will come into play. FaceNet will extract a low-dimensional facial embedding vector that captures the essential characteristics of the face. This vector serves as a unique identifier for each individual.

the system will compare the embedding vector of a known person against a new image of the same person. A high similarity score between the embeddings will indicate a successful verification

Benefits:

* High Accuracy: The combination of MTCNN's precise face detection and FaceNet's powerful feature extraction capabilities leads to a highly accurate recognition system
* Flexibility: The system can be adapted for various applications like access control, security surveillance, or personalized user experiences.

**Technical Architecture:**



**Pre-requisites:**

**To complete this project, you must require the following software, concepts, and packages.**

1. **IDE Installation**:

VS Code IDE is Ideal to complete this project

To install **VS Code**, please refer to [VS Code IDE Installation Steps](https://www.youtube.com/watch?v=naL0cZNQh1g)

1. **Python Packages**

If you are using **anaconda navigator**, follow the below steps to download the required packages:

Open the Anaconda prompt

* Type “pip install tensorflow==2.15.0” and click enter.
* Type “pip install opencv-python==4.9.0.80” and click enter
* Type “pip install mtcnn==0.1.1” and click enter
* Type “pip keras-facenet==0.3.2” and click
* Type “scikit-learn==1.3.2” and click
* Type "pip install Flask” and click enter.

**Prior Knowledge:**

You must have prior knowledge of the following topics to complete this project.

* OpenCV - <https://www.youtube.com/watch?v=WQeoO7MI0Bs>
* Flask - <https://www.youtube.com/watch?v=lj4I_CvBnt0>

# Project Objectives:

By the end of this project, you will:

* Know fundamental concepts and techniques used for computer vision.
* Gain knowledge of CNN and transfer learning.

# Project Flow:

* The user interacts with the UI to enter the input.
* Entered input is analyzed by the model which is integrated.
* Once the model analyses the input the summary is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

**Create app.py python file :**

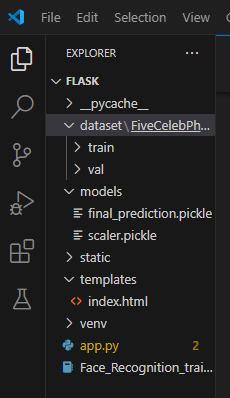
* Import the required libraries

**UI Integration:**

* Building HTML Pages
* Build Python code
* Run the application

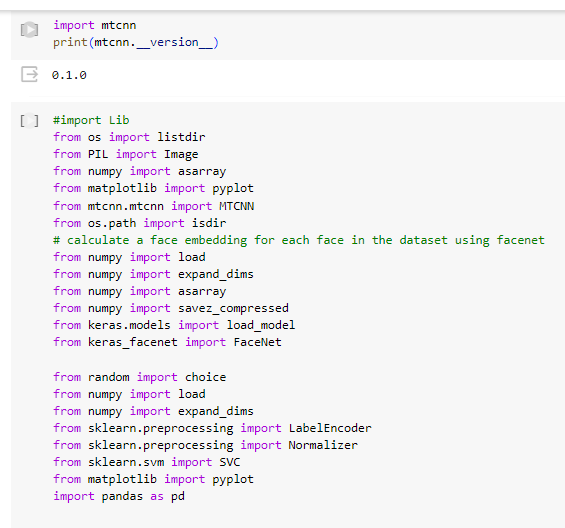
**Project Structure:**

The Template folder contains HTML pages. The app.py contains the python code used . you'll find HTML pages, while app.py comprises the python code responsible for counting the pushups.



**Milestone 1:** **Create app.py python file**

**Activity 1: Import the required libraries**

We will be importing the necessary packages initially.

**Activity 2: Face Recognition experience with Python:**

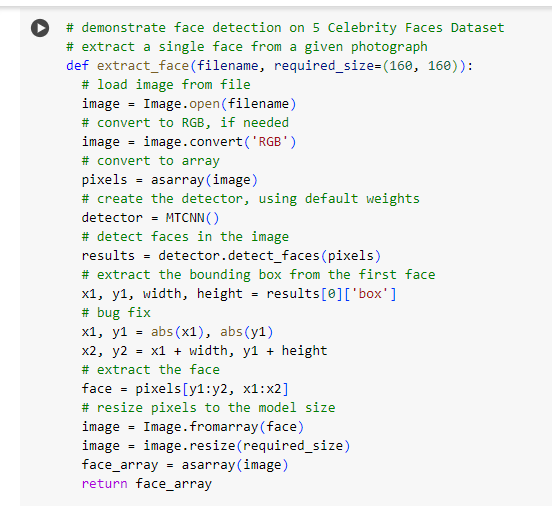
**About the dataset :** [**Link**](https://www.kaggle.com/datasets/dansbecker/5-celebrity-faces-dataset?resource=download)

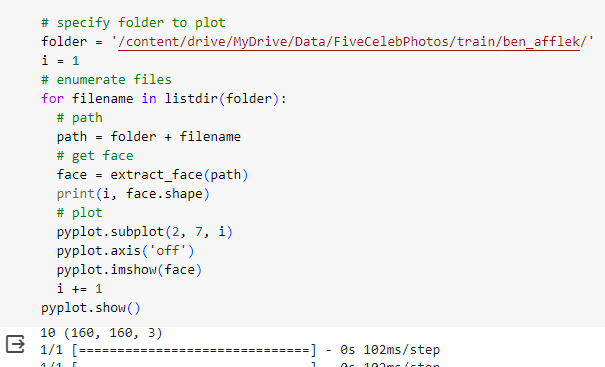
It is a small dataset for experimenting with computer vision techniques. It has a training directory containing 14-20 photos each of the celebrities

* Ben Afflek
* Elton John
* Jerry Seinfeld
* Madonna
* Mindy Kaling

The validation directory has 5 photos of each celebrity.

The photos haven't been cropped for consistent aspect ratios. With so few training photos, this an especially interesting test of computer vision techniques.

****

****

Explanation of above code :

1. Function Definition:

extract\_face(filename, required\_size=(160, 160)): This function extracts a single face from a given image file.

2. Image Loading:

Loads the image from the specified filename using Image.open(filename).

Ensures the image is in RGB format using image.convert('RGB').

Converts the image to a NumPy array for further processing using asarray(image).

3. Face Detection:

Creates an MTCNN face detector instance using detector = MTCNN().

Detects faces in the image using results = detector.detect\_faces(pixels).

4. Face Extraction:

Extracts the bounding box coordinates for the first detected face (x1, y1, width, height).

Applies a bug fix to ensure positive coordinates using x1, y1 = abs(x1), abs(y1).

Calculates the bounding box's ending coordinates (x2, y2).

Extracts the face region from the image array using the bounding box coordinates.

5. Face Resizing:

Converts the extracted face region back to an image using Image.fromarray(face).

Resizes the image to a standard size (160x160 by default) using image = image.resize(required\_size).

Converts the resized image back to a NumPy array using face\_array = asarray(image).

6. Face Visualization:

Iterates through image files in a specified folder (folder).

For each file:

Calls the extract\_face function to extract the face.

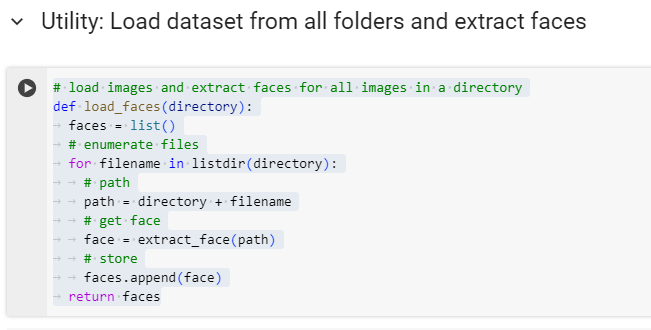
Prints the extracted face's dimensions.

Creates a subplot using pyplot.subplot(2, 7, i) to display the face.

Turns off-axis labels using pyplot.axis('off').

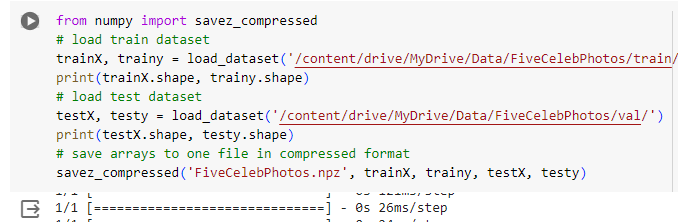
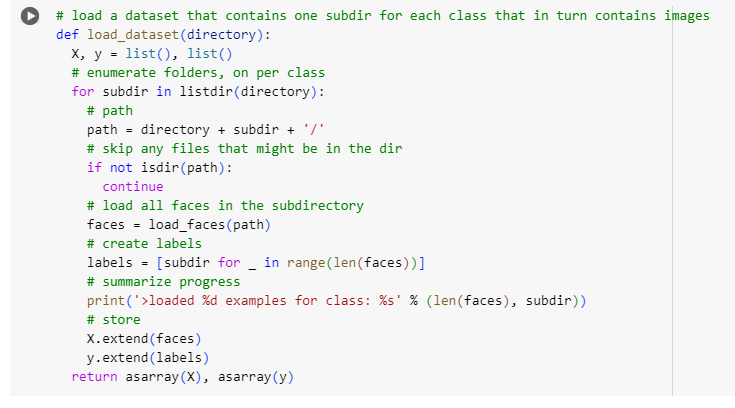
Displays the face using pyplot.imshow(face).

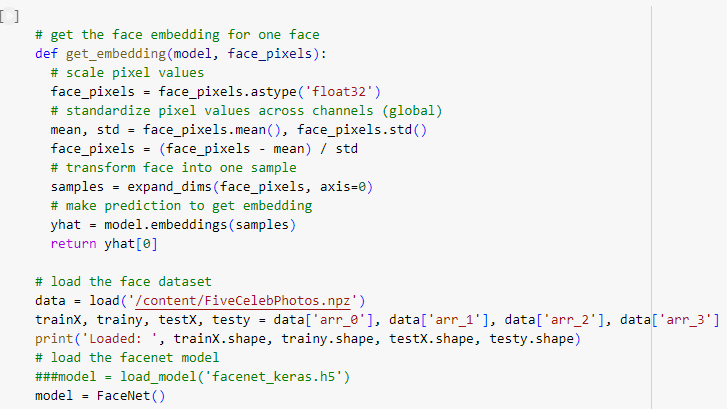
Shows all subplots using pyplot.show().

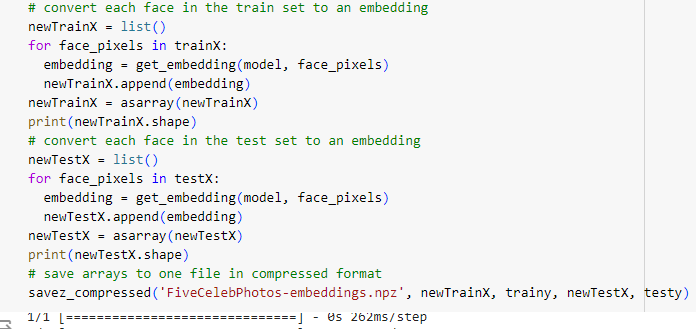


Explanation of the above code :

Load dataset from all the folders and extract faces for each image in training and test dataset.





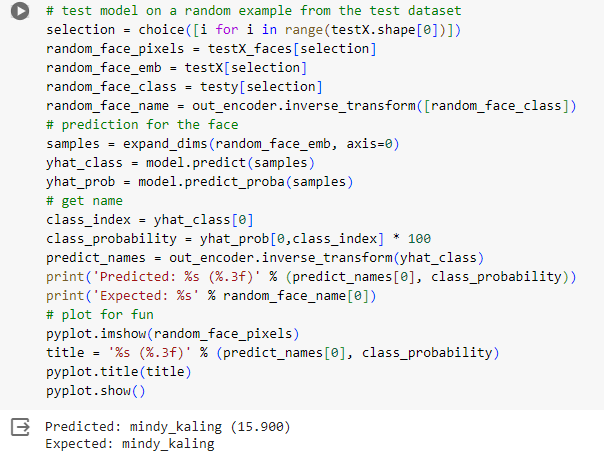


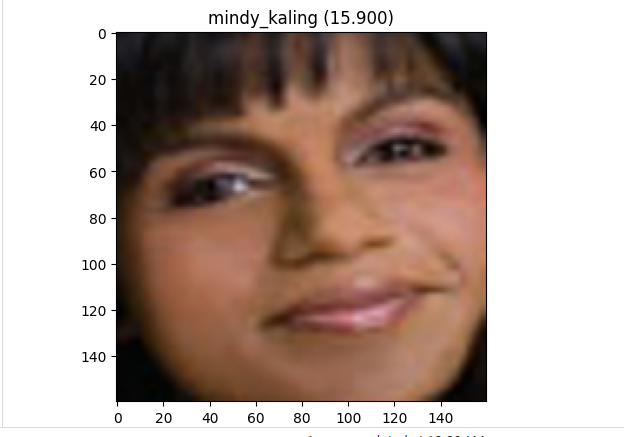
Explanation of the above code : Get the face embedding for each of the faces.



Explanation of above code

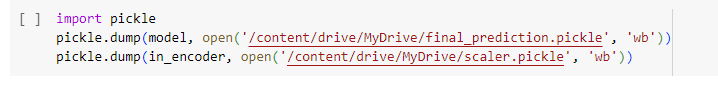
Apply normalizer on train and test data and encode the class label.  
train the model on SVM linear.





Explanation of above code:

Test the model on giving random input from test data.



Explanation of above code:

Save the model and normalizer file to use in flask app.

# Milestone 2: UI Integration

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the users where he/she has to navigate to open the web cam.

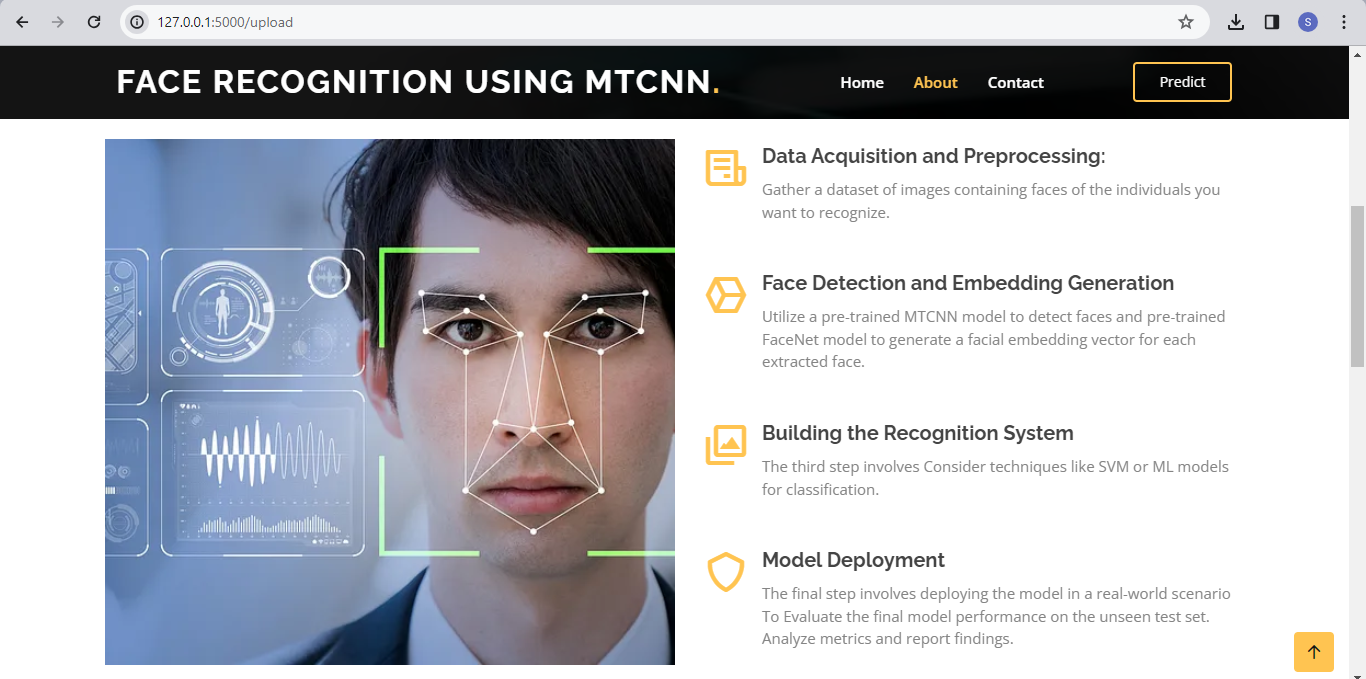
This section has the following tasks

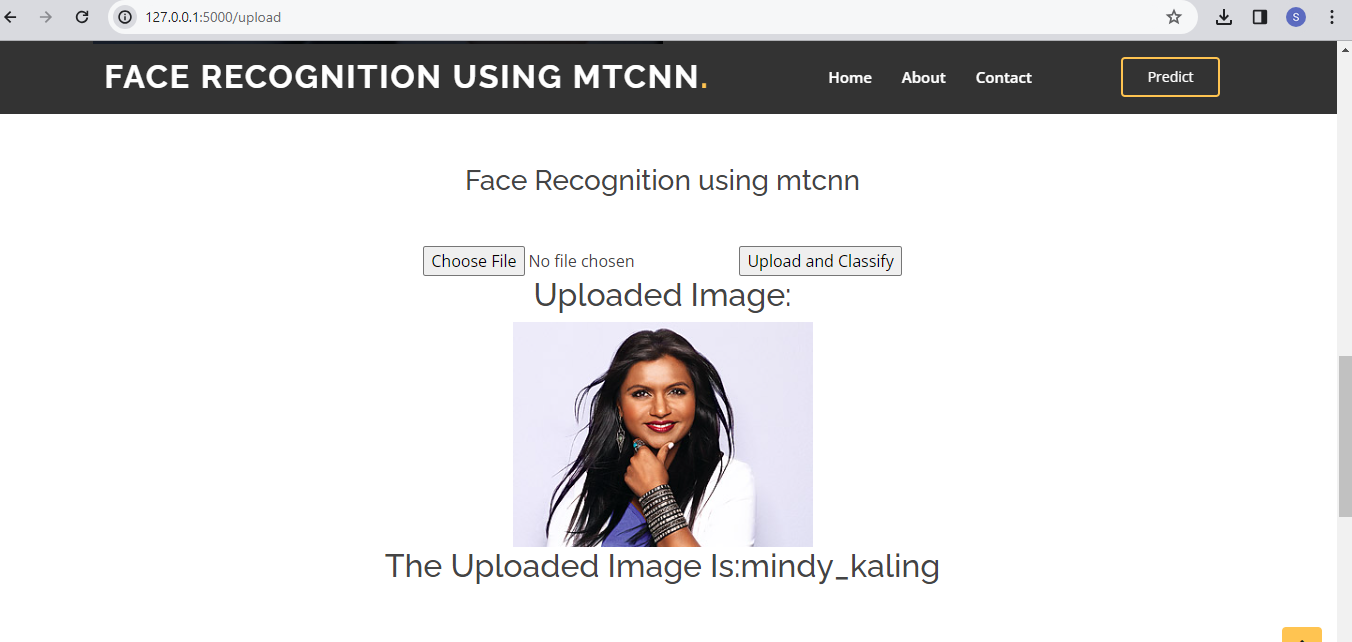
* Building HTML Pages
* Building server-side script

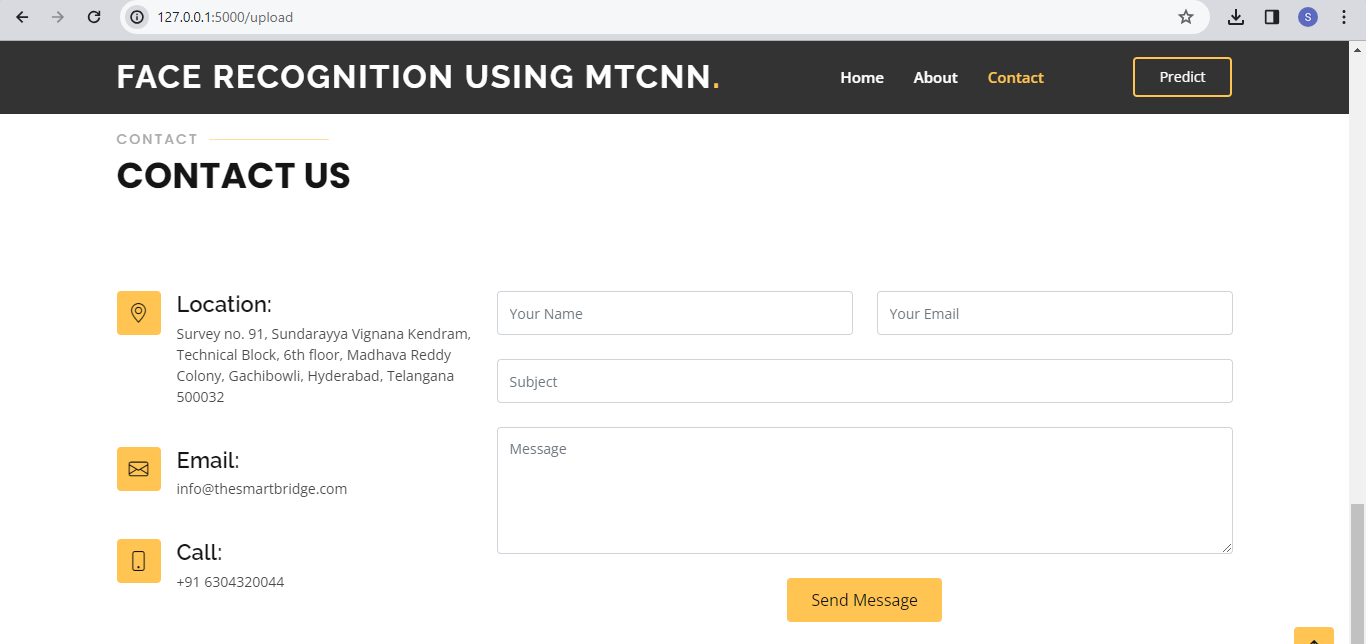
**Activity1: Building Html Pages:**

We have created HTML files for this project and saved them in the templates folder.

Let’s see how those html pages looks like:

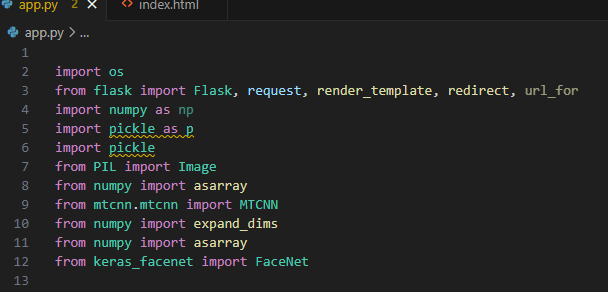




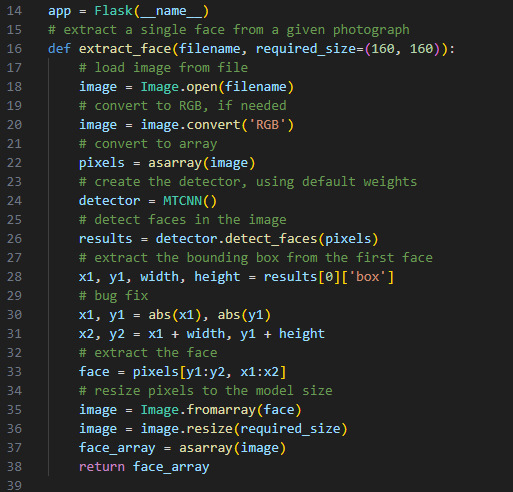


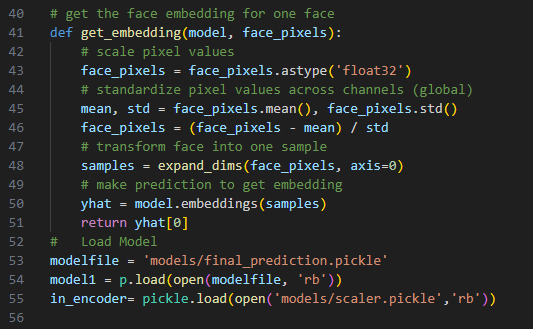
**Activity 2: Build Python code:**

* Import the libraries



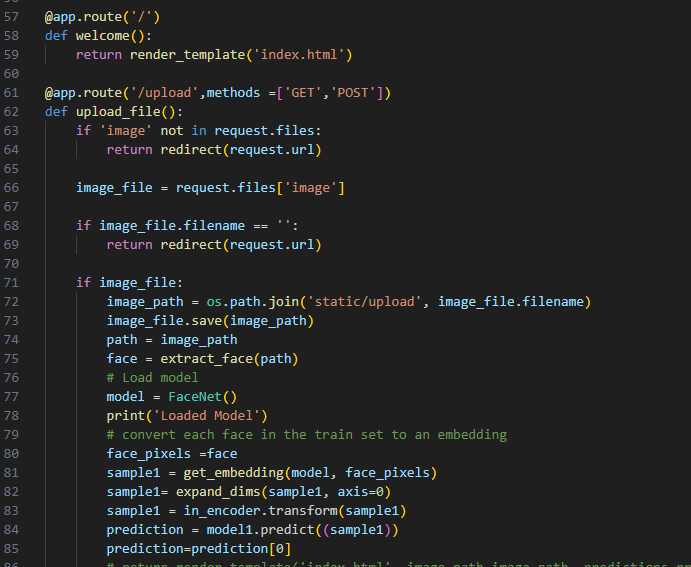
* Importing the Flask module into the project is mandatory. An object of the Flask class is our WSGI application. The Flask constructor takes the name of the current module (\_\_name\_\_) as an argument.

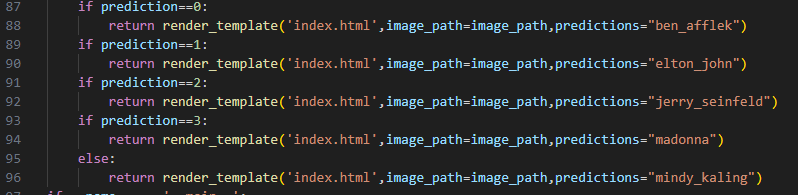




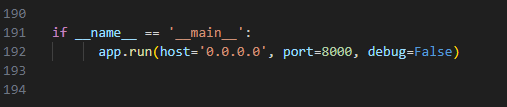
Render HTML page:

* Here we will be using the declared constructor to route to the HTML page that we have created earlier. In the above example, the ‘/’ URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.



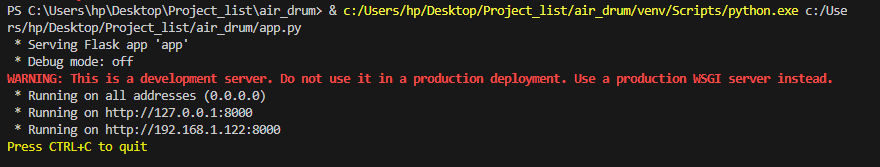


Main Function:



**Activity 3: Run the application**

* Open the anaconda prompt from the start menu
* Navigate to the folder where your Python script is.
* Now type the “python app.py” command
* Navigate to the localhost where you can view your web page.
* Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



Output:

