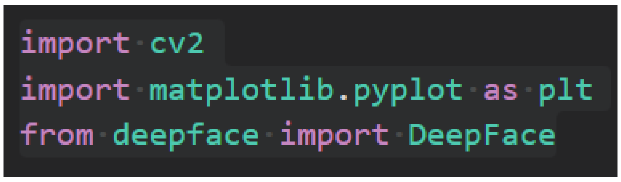
Face-Emotion Detection

I made two projects:

1. Through Image using Jupyter Notebook
2. Through Real-Time Video Using Python

For the one through Image:



```
import cv2
import matplotlib.pyplot as plt
from deepface import DeepFace
```

Import cv2:

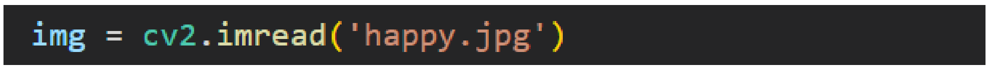
I implemented face recognition using OpenCV and Jupyter Notebook. OpenCV is a video and image processing library and it is used for image and video analysis, like facial detection, license plate reading, photo editing, advanced robotic vision, and many more thus I used it here

Import matplotlib.pyplot as plt:

Pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. here we need it to display the image and make necessary changes to it

Installed deepface and imported DeepFace:

Deepface is a lightweight facial analysis framework including face recognition and demography (age, gender, emotion and race) for Python and since we need these details for facial recognition I imported DeepFace.



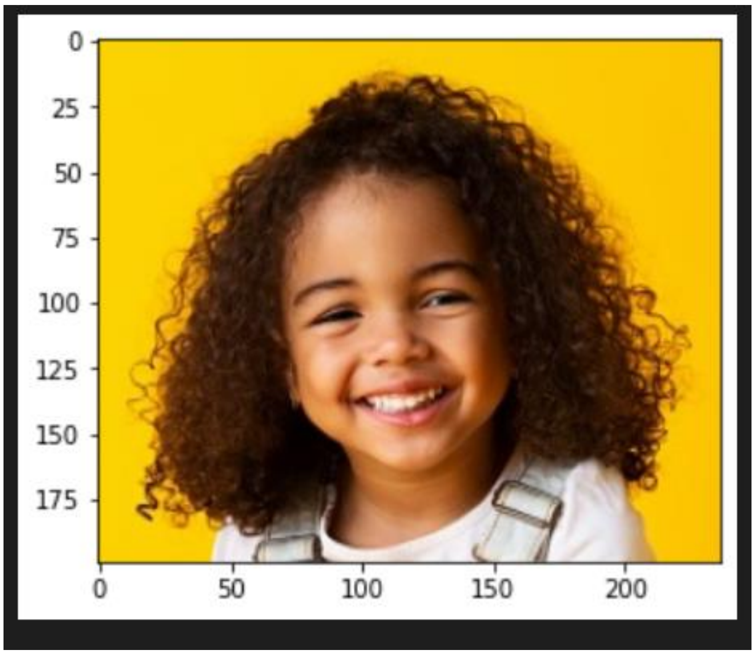
```
img = cv2.imread('happy.jpg')
```

In the variable 'img' this line of code stores the image 'happy.jpg' by reading it

cv2 is the library used and im means image

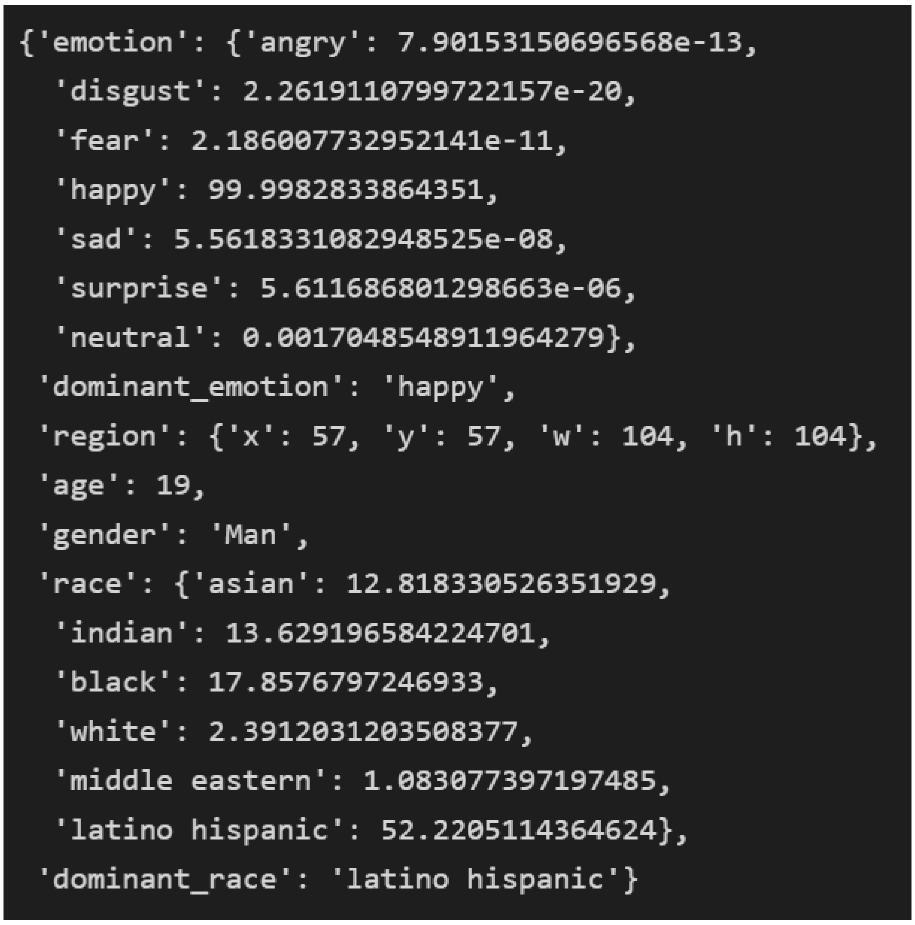
```
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

Here I used matplotlib.pyplot library which I named plt for easier usage and displayed i.e. 'imshow' the image which is read by the cv2 library using cv2.cvtColor where img is the source and cv2.COLOR_BGR2RGB changes BGR image into RGB i.e. BlueGreenRed image to RedGreenBlue image because original image was in BGR format.

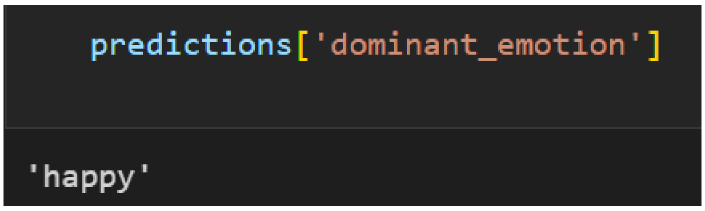


```
predictions=DeepFace.analyze(img)
predictions
```

A variable 'predictions' is assigned to store the analysis of the image img using DeepFace library which gives age, gender, emotion etc as output with accuracy as high as possible when I printed predictions



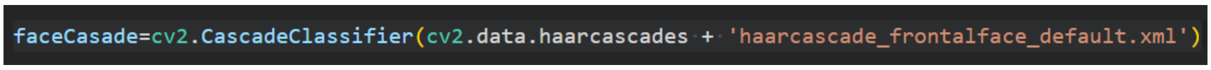
```
{'emotion': {'angry': 7.90153150696568e-13,
  'disgust': 2.2619110799722157e-20,
  'fear': 2.186007732952141e-11,
  'happy': 99.9982833864351,
  'sad': 5.5618331082948525e-08,
  'surprise': 5.611686801298663e-06,
  'neutral': 0.0017048548911964279},
 'dominant_emotion': 'happy',
 'region': {'x': 57, 'y': 57, 'w': 104, 'h': 104},
 'age': 19,
 'gender': 'Man',
 'race': {'asian': 12.818330526351929,
  'indian': 13.629196584224701,
  'black': 17.8576797246933,
  'white': 2.3912031203508377,
  'middle eastern': 1.083077397197485,
  'latino hispanic': 52.2205114364624},
 'dominant_race': 'latino hispanic'}
```



```
predictions['dominant_emotion']
```

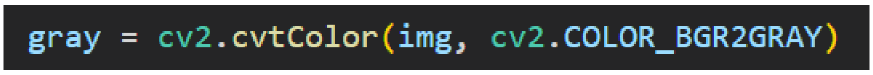
```
'happy'
```

Here I printed the dominant_emotion from predicitons and the output was ‘happy’ because as per the predictions output the emotion ‘happy’ had 99.998% resemblance with the image img



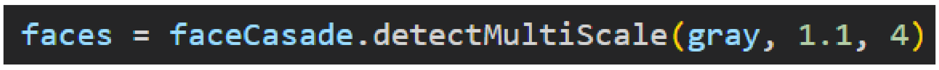
```
faceCasade=cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. One of those XML files is haarcasade_frontalface_default



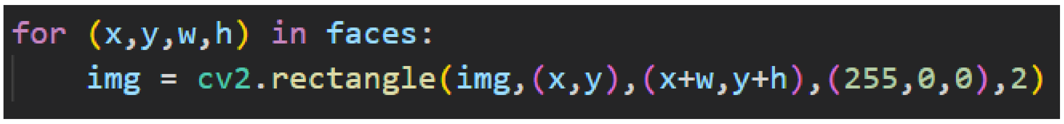
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Now, we need to load input image in grayscale mode so that it can been read easily without using much time



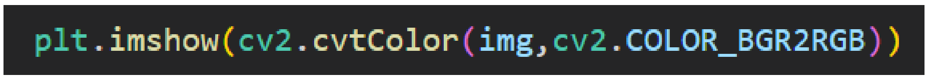
```
faces = faceCasade.detectMultiScale(gray, 1.1, 4)
```

It is used to find faces or eyes



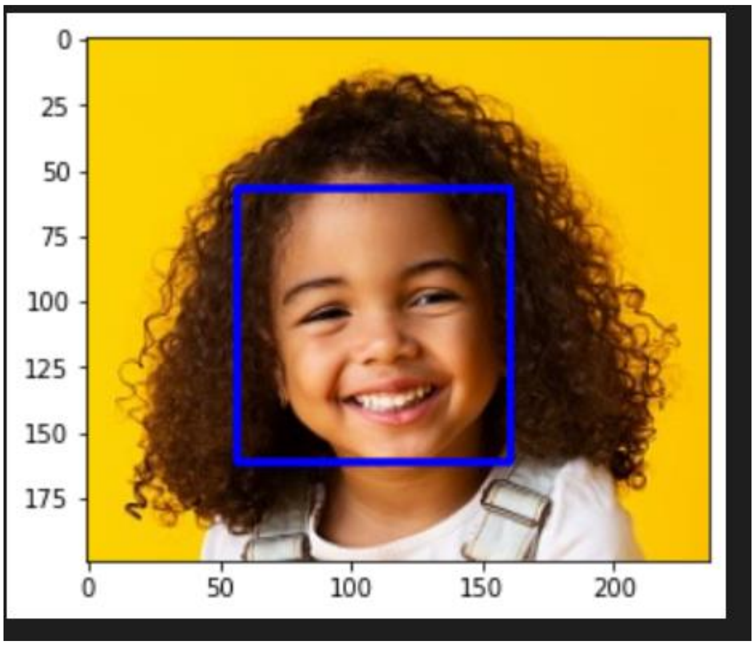
```
for (x,y,w,h) in faces:
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
```

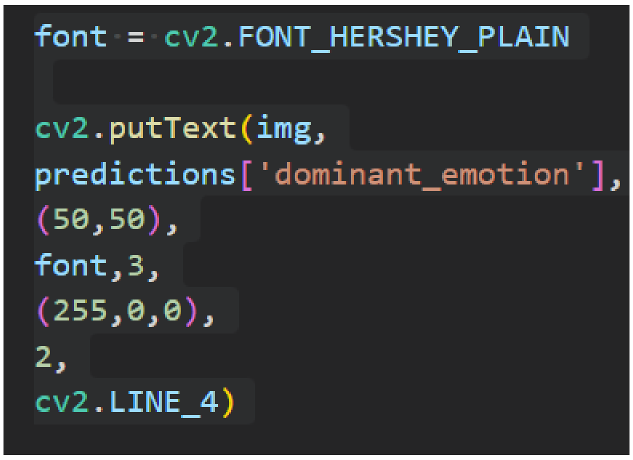
If faces are found, it returns the positions of detected faces as rectangle(x,y,w,h)



```
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

Here I used matplotlib.pyplot library which I named plt for easier usage and displayed i.e. ‘imshow’ the image which is read by the cv2 library using cv2.cvtColor where img is the source and cv2.COLOR_BGR2RGB changes BGR image into RGB i.e. BlueGreenRed image to RedGreenBlue image because original image was in BGR format





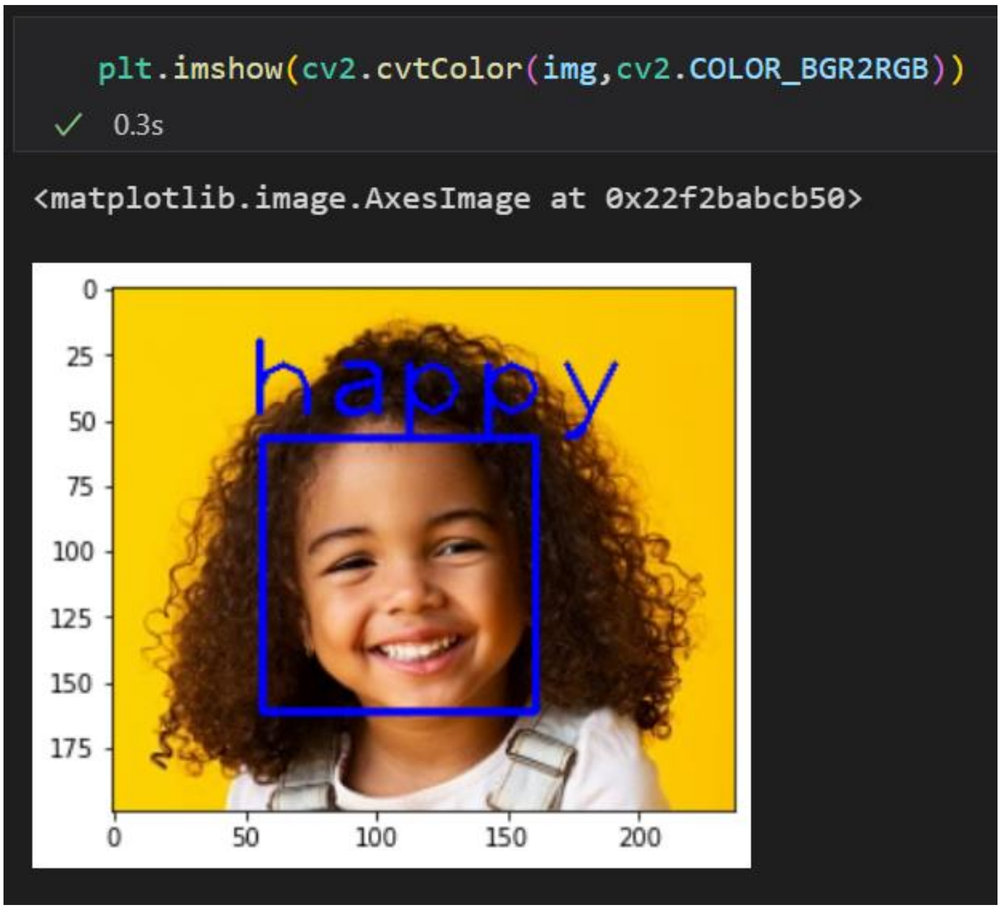
```
font = cv2.FONT_HERSHEY_PLAIN

cv2.putText(img,
predictions['dominant_emotion'],
(50,50),
font,3,
(255,0,0),
2,
cv2.LINE_4)
```

This is used to display text

Syntax: cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])

image: It is the image on which text is to be drawn i.e. ‘img’
text: Text string to be drawn i.e. the dominant emotion in predictions which is ‘happy’ here
org: It is the coordinates of the bottom-left corner of the text string in the image. The coordinates are represented as tuples of two values i.e. (50,50).
font: It denotes the font type. We already assigned font_hershey_plain to font
fontScale: Font scale factor that is multiplied by the font-specific base size.
color: It is the color of text string to be drawn. For BGR, we pass a tuple (255, 0, 0) for blue color.
thickness: It is the thickness of the line in px i.e. 2px here.
lineType: This is an optional parameter.It gives the type of the line to be used.
bottomLeftOrigin: This is an optional parameter. When it is true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.
Return Value: It returns an image.

Final Output:

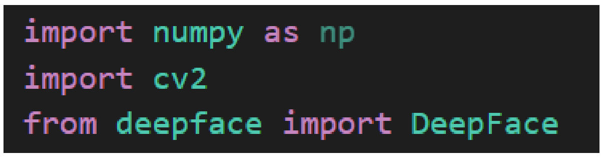


```
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

✓ 0.3s

```
<matplotlib.image.AxesImage at 0x22f2babcb50>
```

Through Real-Time Video Using Python



```
import numpy as np
import cv2
from deepface import DeepFace
```

<u>Import cv2</u>:

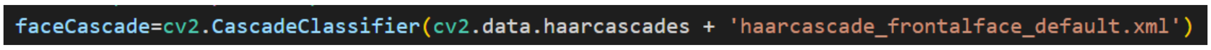
I implemented face recognition using OpenCV and Jupyter Notebook. OpenCV is a video and image processing library and it is used for image and video analysis, like facial detection, license plate reading, photo editing, advanced robotic vision, and many more thus I used it here

<u>Import matplotlib.pyplot as plt:</u>

Pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. here we need it to display the image and make necessary changes to it

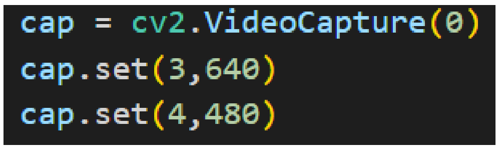
<u>Installed deepface and imported DeepFace</u>:

Deepface is a lightweight facial analysis framework including face recognition and demography (age, gender, emotion and race) for Python and since we need these details for facial recognition I imported DeepFace.



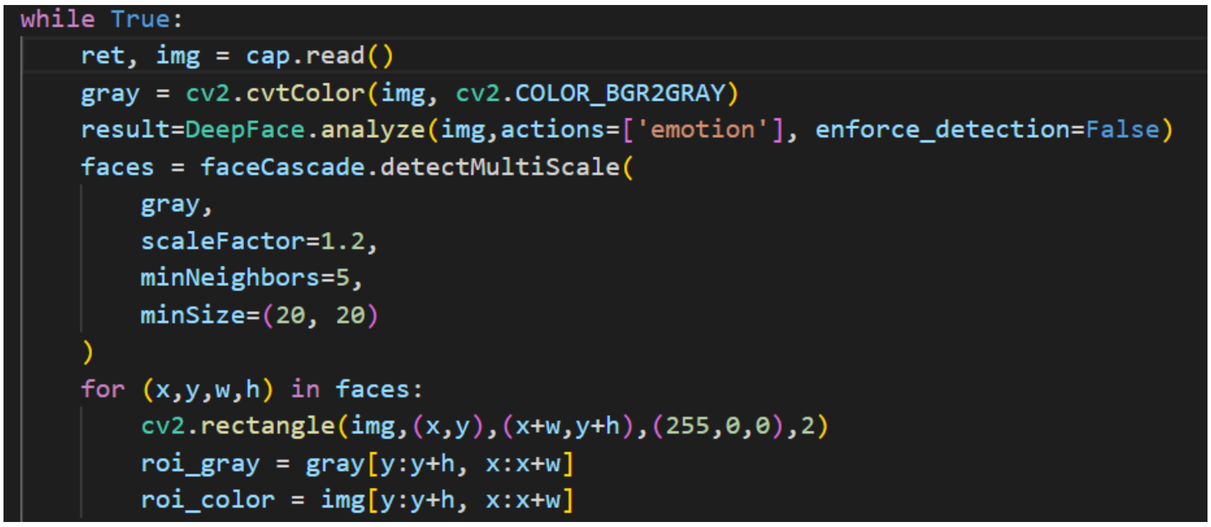
```
faceCascade=cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. One of those XML files is haarcasade_frontalface_default



```
cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)
```

This captures the video from the webcam and the reolution of the video is set to 640x480



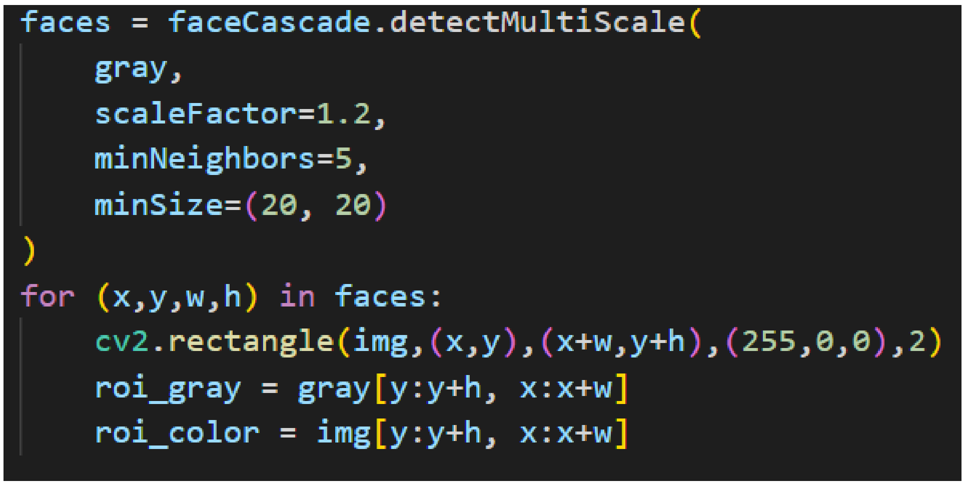
```
while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    result=DeepFace.analyze(img,actions=['emotion'], enforce_detection=False)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
```

ret is a boolean variable that returns true if the frame is available.

img is an image array vector captured based on the default frames per second defined explicitly or implicitly

Now, we need to load input image in grayscale mode so that it can been read easily without using much time

Variable result is assigned to the analysis of the image done by DeepFace library and it is done on the action ‘emotion’ and enforce_detection is set to false because DeepFace.analyze has its default value is true. If you don’t set it, then it returns exception if it cannot detect any faces. Setting it to false skips exception when it cannot detect any faces.



```
faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.2,
    minNeighbors=5,
    minSize=(20, 20)
)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
```

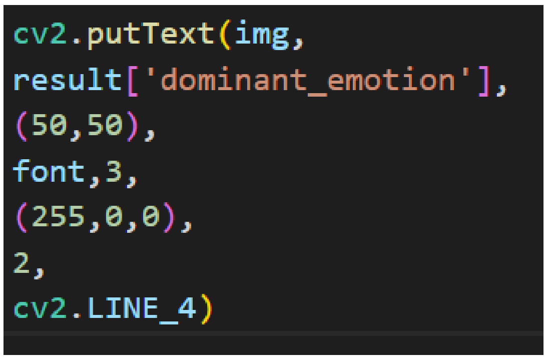
It is used to find faces or eyes

First img is converted to grayscale

Then the scale factor i.e. parameter specifying how much the image size is reduced at each image scale is set to 1.2

minNeighbours is the parameter specifying how many neighbors each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces: higher value results in less detections but with higher quality. We're using 5 in the code.

minSize is the minimum possible object size. Objects smaller than that are ignored.

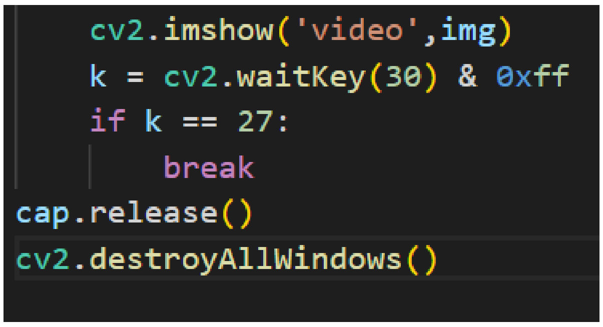


```
cv2.putText(img,
result['dominant_emotion'],
(50,50),
font,3,
(255,0,0),
2,
cv2.LINE_4)
```

This is used to display text

Syntax: cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])

image: It is the image on which text is to be drawn i.e. ‘img’
text: Text string to be drawn i.e. the dominant emotion in results which depends on the person in the video
org: It is the coordinates of the bottom-left corner of the text string in the image. The coordinates are represented as tuples of two values i.e. (50,50).
font: It denotes the font type. We already assigned font_hershey_plain to font
fontScale: Font scale factor that is multiplied by the font-specific base size.
color: It is the color of text string to be drawn. For BGR, we pass a tuple (255, 0, 0) for blue color.
thickness: It is the thickness of the line in px i.e. 2px here.
lineType: This is an optional parameter.It gives the type of the line to be used.
bottomLeftOrigin: This is an optional parameter. When it is true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.
Return Value: It returns an image.



```
    cv2.imshow('video',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

First line is used to show i.e. display the video on screen

cv2.waitKey() returns a 32 Bit integer value (might be dependent on the platform). The key input is in ASCII which is an 8 Bit integer value. So you only care about these 8 bits and want all other bits to be 0. This you can achieve with k = cv2.waitKey(30) & 0xff

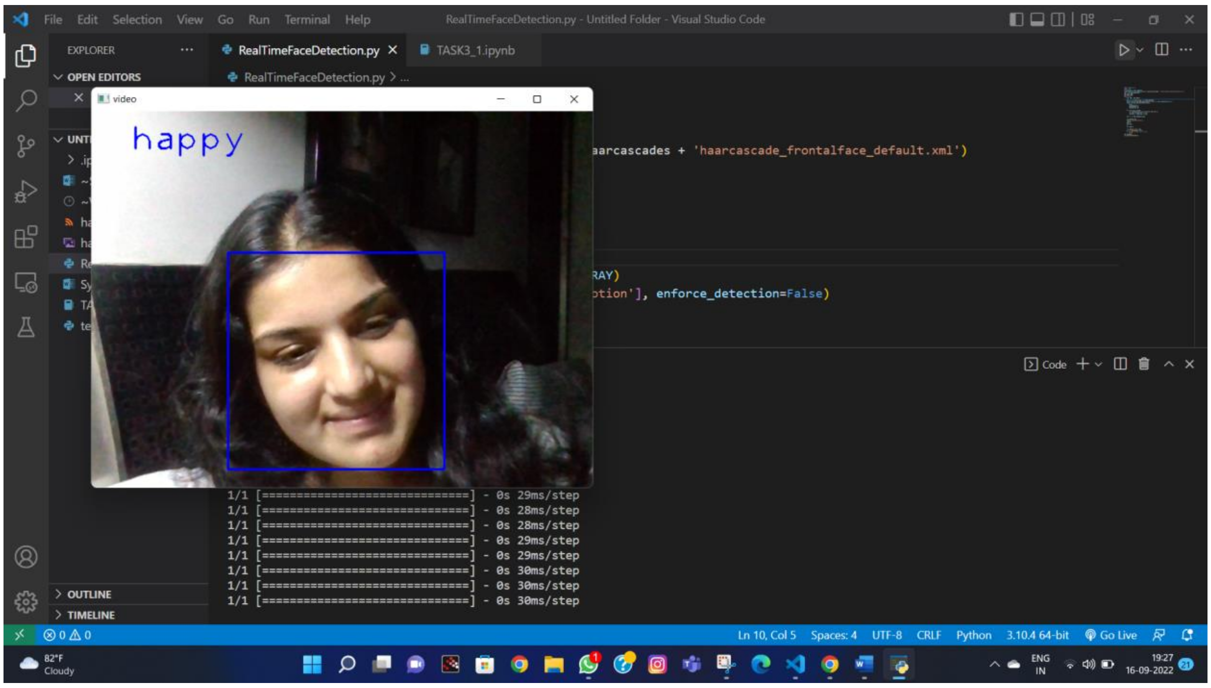
0xFF is a hexadecimal constant which is 11111111 in binary. By using bitwise AND (&) with this constant, it leaves only the last 8 bits of the original (in this case, whatever cv2.waitKey(30) is)

When you call cap.release(), then:

1. release software resource
2. release hardware resource

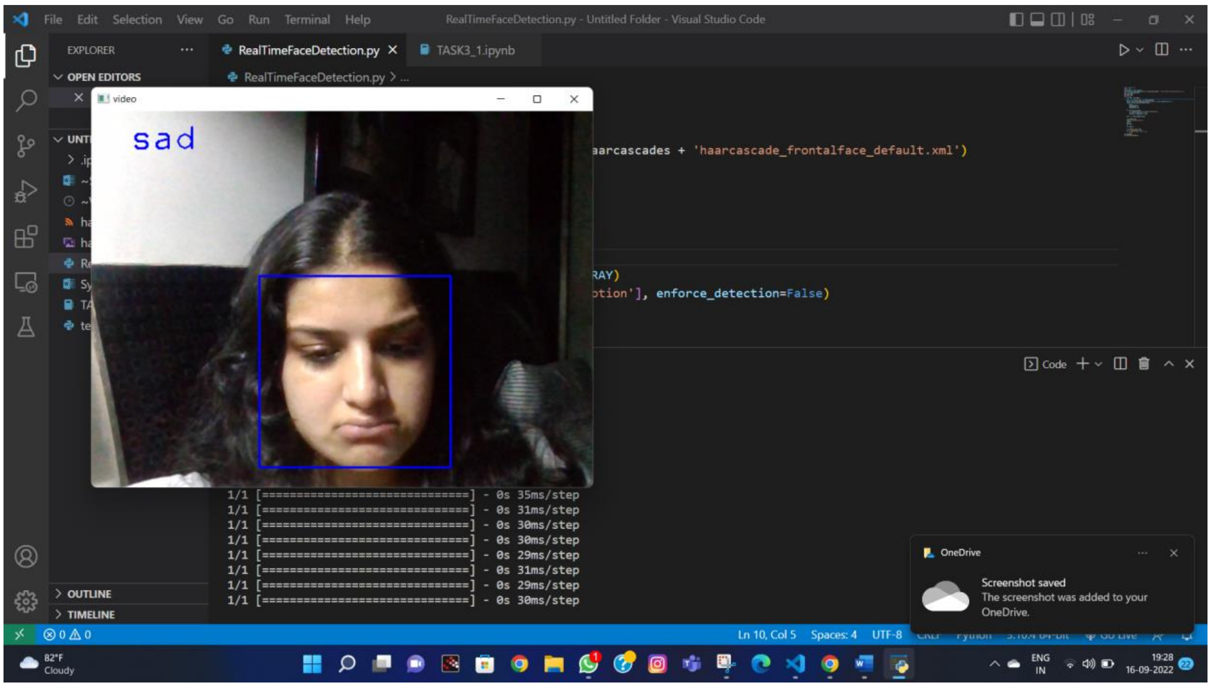
cv2.destroyAllWindows()
closes all the windows after the program is executed and the user pressed crtl+c

Final output:


File Edit Selection View Go Run Terminal Help
RealTimeFaceDetection.py - Untitled Folder - Visual Studio Code
EXPLORER
OPEN EDITORS
RealTimeFaceDetection.py
TASK3_1.ipynb
video
happy
OUTLINE
TIMELINE
Ln 10, Col 5 Spaces: 4 UTF-8 CRLF Python 3.10.4 64-bit Go Live
82°F Cloudy
ENG IN
19:27 16-09-2022

```
aarcascades + 'haarcascade_frontalface_default.xml')

RAY)
ption'], enforce_detection=False)
```

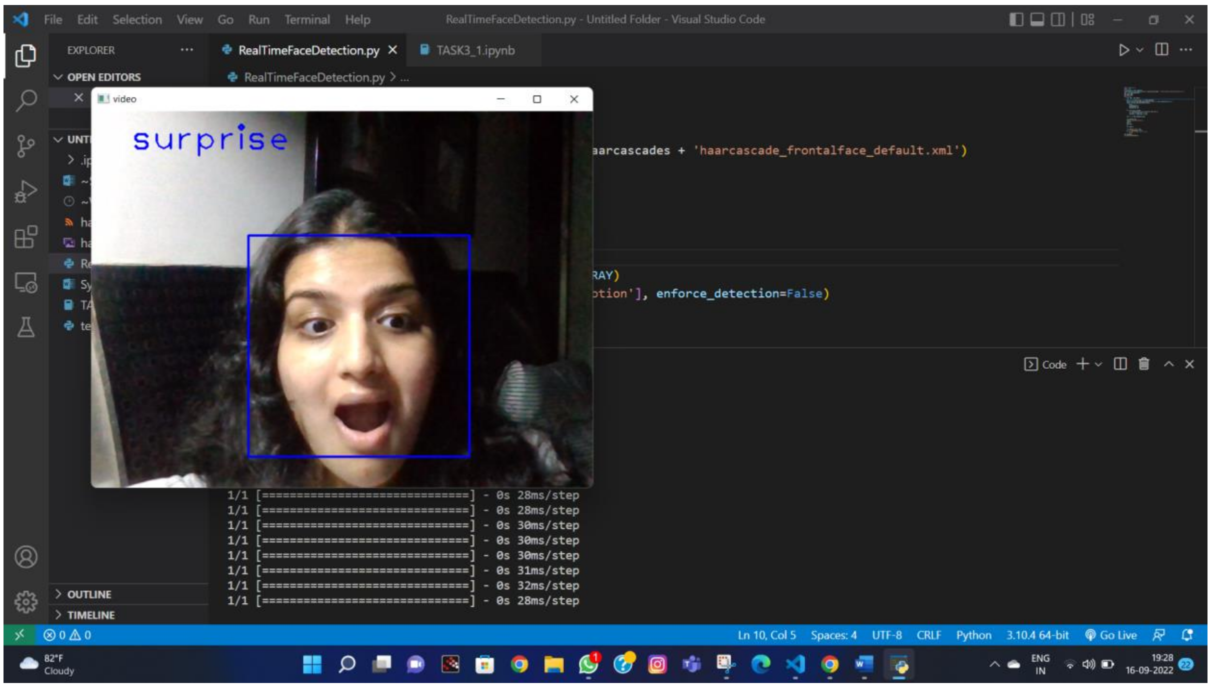
```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
```


File Edit Selection View Go Run Terminal Help
RealTimeFaceDetection.py - Untitled Folder - Visual Studio Code
EXPLORER
OPEN EDITORS
RealTimeFaceDetection.py
TASK3_1.ipynb
video
sad
OUTLINE
TIMELINE
Ln 10, Col 5 Spaces: 4 UTF-8
82°F Cloudy
ENG IN
19:28 16-09-2022

```
aarcascades + 'haarcascade_frontalface_default.xml')

RAY)
ption'], enforce_detection=False)
```

```
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
```

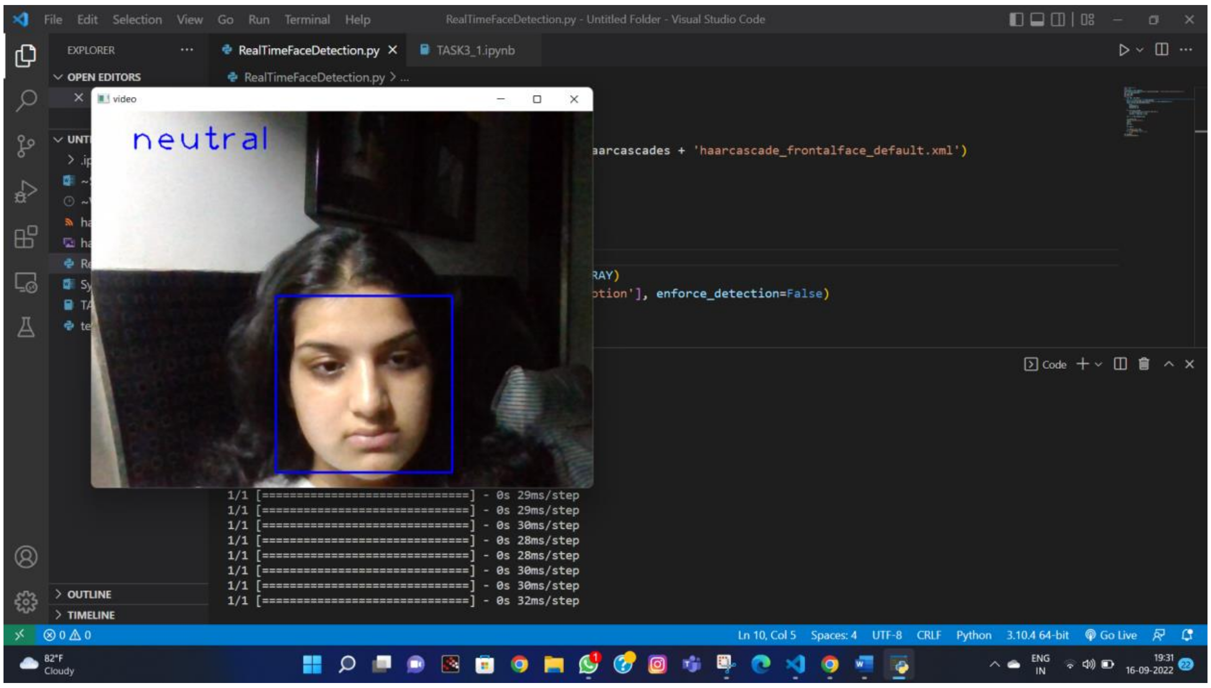
OneDrive
Screenshot saved
The screenshot was added to your OneDrive.



```
aarcascades + 'haarcascade_frontalface_default.xml')

RAY)
ption'], enforce_detection=False)
```

```
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 28ms/step
```

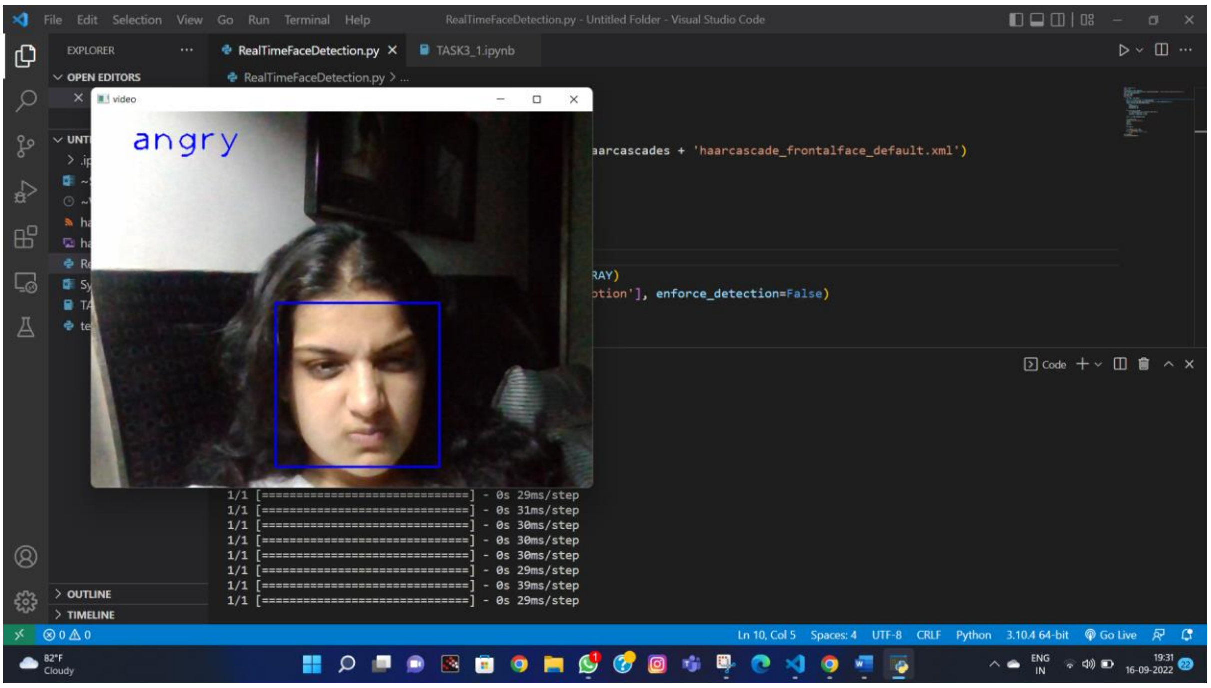
Ln 10, Col 5 Spaces: 4 UTF-8 CRLF Python 3.10.4 64-bit Go Live



```
aarcascades + 'haarcascade_frontalface_default.xml')

RAY)
ption'], enforce_detection=False)
```

```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
```

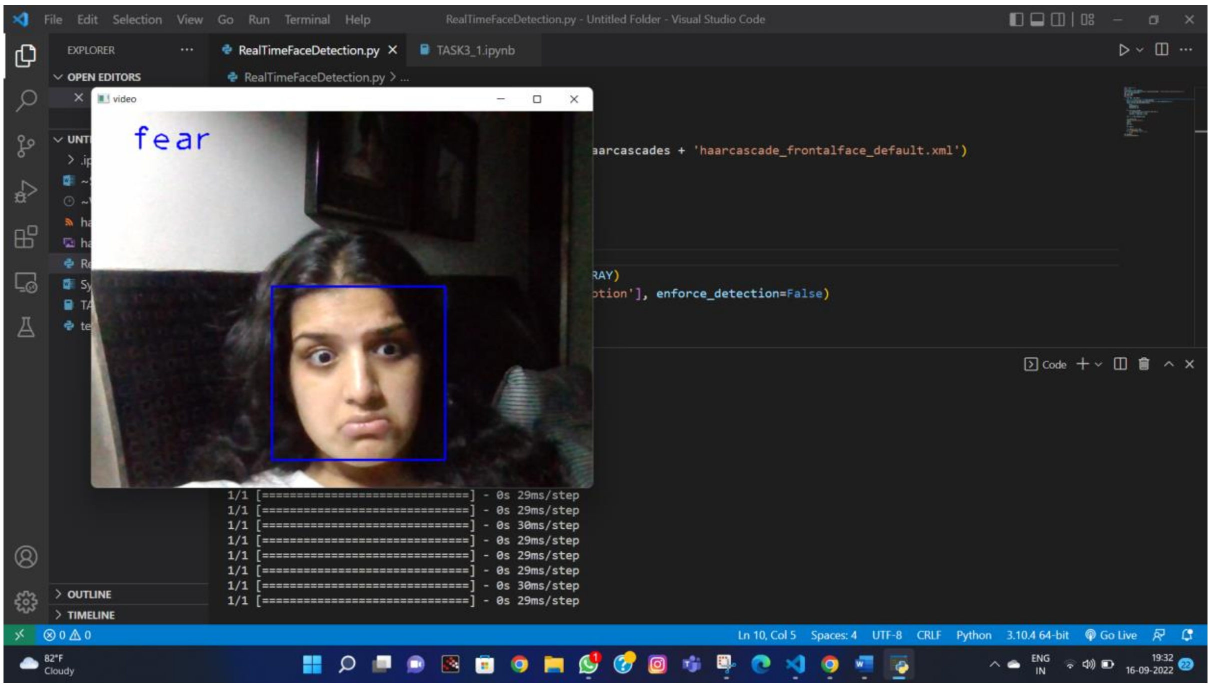
Ln 10, Col 5 Spaces: 4 UTF-8 CRLF Python 3.10.4 64-bit Go Live


RealTimeFaceDetection.py - Untitled Folder - Visual Studio Code
RealTimeFaceDetection.py
TASK3_1.ipynb
video
angry

```
aarcascades + 'haarcascade_frontalface_default.xml')
RAY)
ption'], enforce_detection=False)
```

```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 29ms/step
```

\


RealTimeFaceDetection.py - Untitled Folder - Visual Studio Code
RealTimeFaceDetection.py
TASK3_1.ipynb
video
fear

```
aarcascades + 'haarcascade_frontalface_default.xml')
RAY)
ption'], enforce_detection=False)
```

```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
```

I selected this algorithm and used the inbuilt DeepFace library because the data is already fed inside and is really reliable. This save my time during coding and also cut down all the surveys required to fetch in a proper and reliable data set.