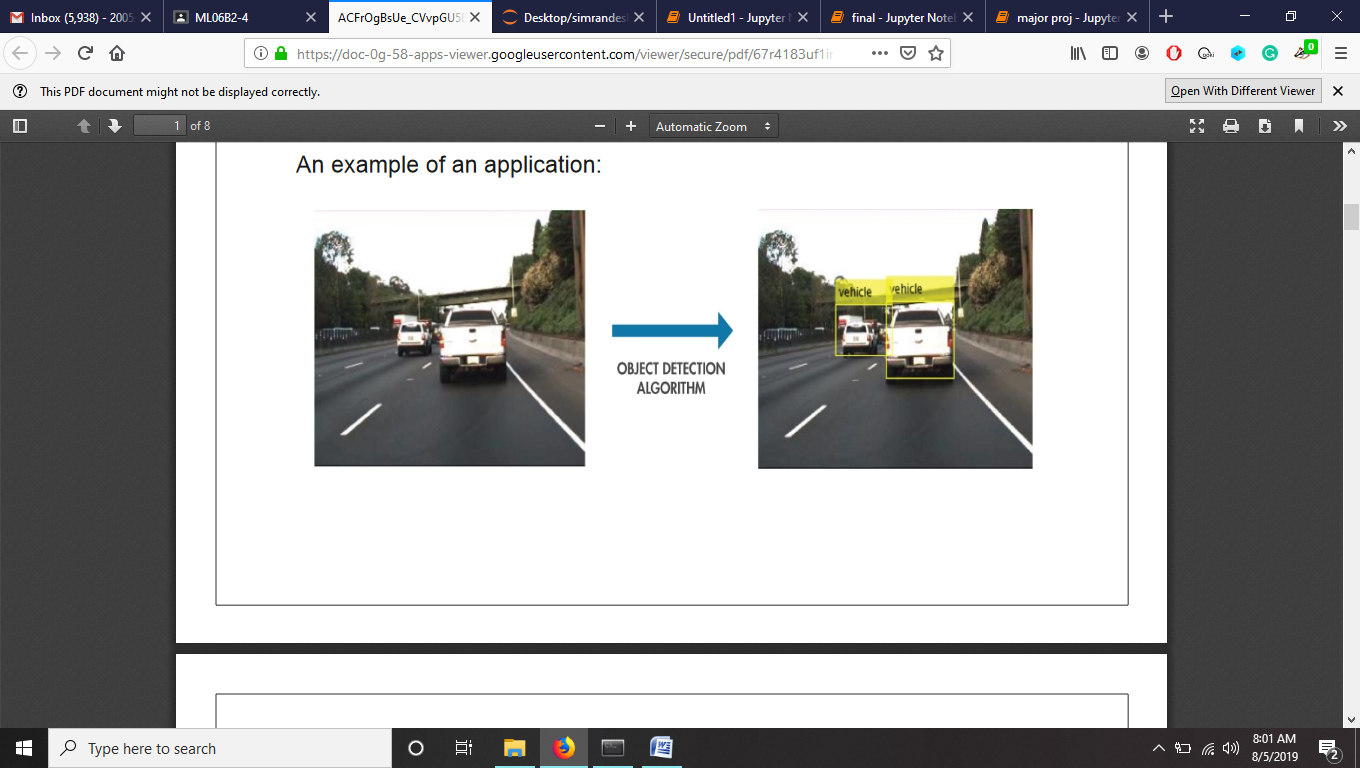
**FACE DETECTION**

**WITH OPENCV**

**Introduction**

Computer vision is a field of study which aims at gaining a deep understanding from digital images or videos. Combined with AI and ML techniques, today many industries are investing in researches and solutions of computer vision. For example: many studies are being carried on to implement security cameras with object detection capabilities. Indeed, imagine a camera in a train station which, depending on the movement captured, is able to detect whether a fight is occurring: it could immediately send a signal to the closest policeman and prevent that fight from getting worse. Object detection is a powerful instrument. Object detection is a computer vision technique for locating instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results. When humans look at images or video, we can recognize and locate objects of interest within a matter of moments. The goal of object detection is to replicate this intelligence using a computer. An example of an application:



## Language used: Python

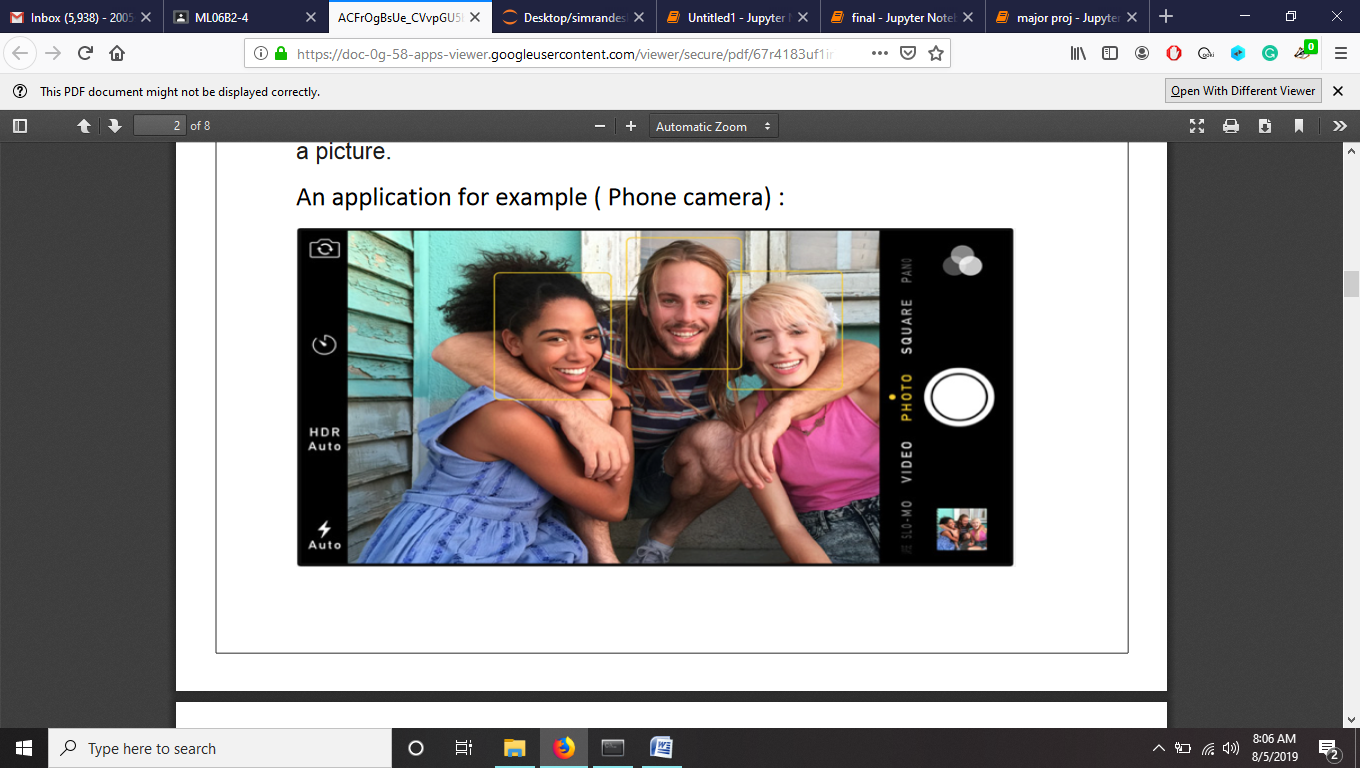
## Library used : Open CV



**Face Detection:**

Face Detection has been one of the hottest topics of computer vision for the past few years. This technology has been available for some years now and is being used all over the place. From cameras that make sure faces are focused before you take a picture, to Facebook when it tags people automatically once you upload a picture.

An application for example ( Phone camera) :



**Theory of face detection classifiers:**

A computer program that decides whether an image is a positive image (face image) or negative image (non-face image) is called a classifier. A classifier is trained on hundreds of thousands of face and non-face images to learn how to classify a new image correctly. OpenCV provides us with two pre-trained and ready to be used for face detection classifiers:

**1.Haar Classifier**

**2.LBP Classifier**

Both of these classifiers process images in gray scales, basically because we don't need color information to decide if a picture has a face or not

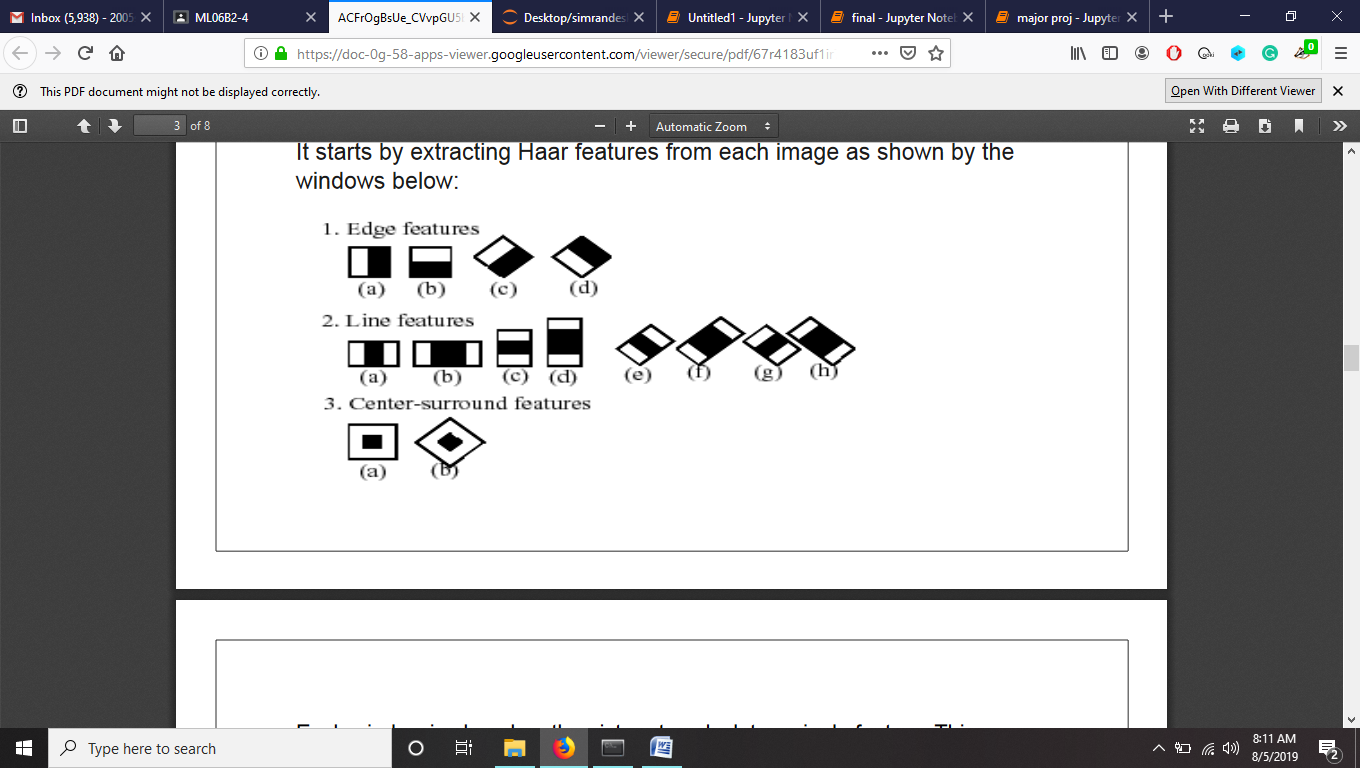
As these are pre-trained in OpenCV, their learned knowledge files also come bundled with OpenCV opencv/data/.

Each file starts with the name of the classifier it belongs to. For example, a Haar cascade classifier starts off as haarcascade\_frontalface\_alt.xml

(All .xml files can be found in the project\_folder)

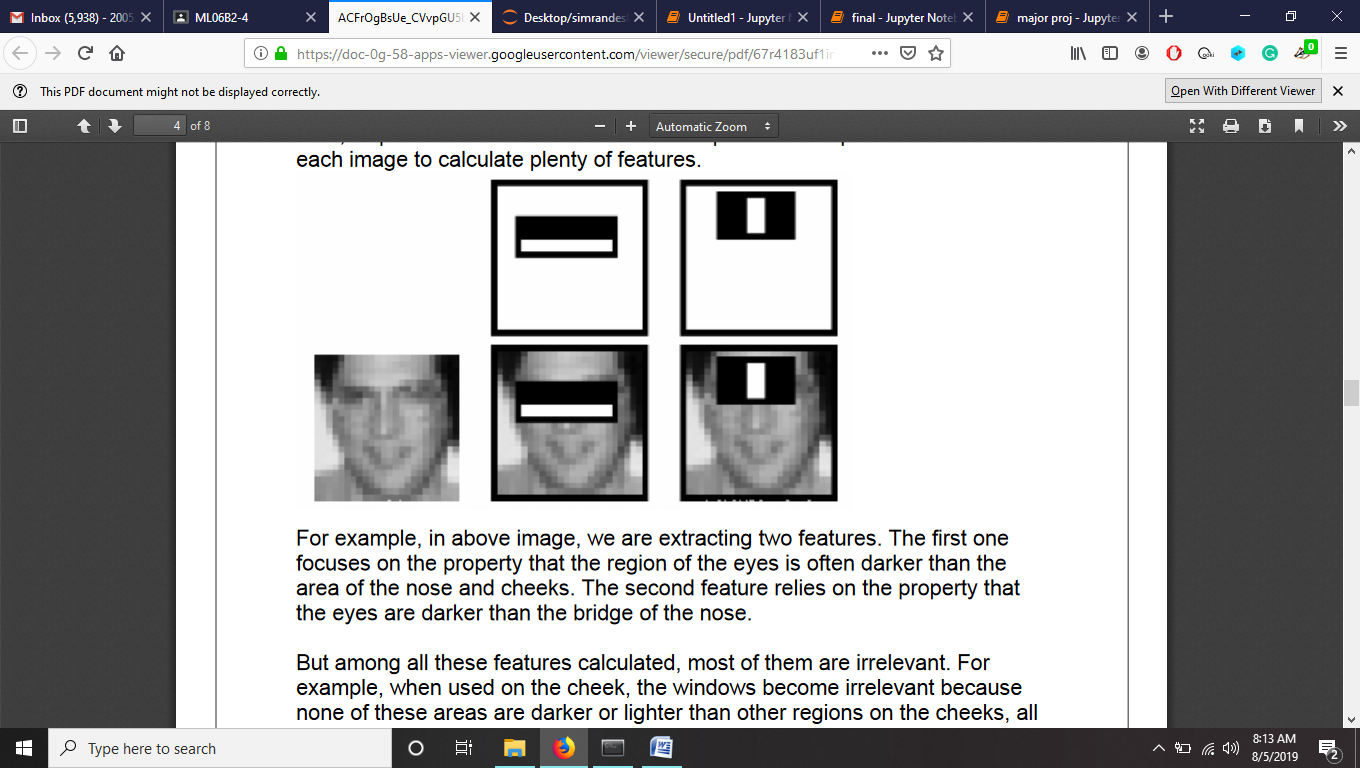
**HAAR CLASSIFIER:**

The Haar Classifier is a machine learning based approach, an algorithm created by Paul Viola and Michael Jones; which (as mentioned before) are trained from many many positive images (with faces) and negatives images (without faces).It starts by extracting Haar features from each image as shown by the windows below:



Each window is placed on the picture to calculate a single feature. This feature is a single value obtained by subtracting the sum of pixels under the white part of the window from the sum of the pixels under the black part of the window.

Now, all possible sizes of each window are placed on all possible locations of each image to calculate plenty of features.



For example, in above image, we are extracting two features. The first one focuses on the property that the region of the eyes is often darker than the area of the nose and cheeks. The second feature relies on the property that the eyes are darker than the bridge of the nose. But among all these features calculated, most of them are irrelevant. For example, when used on the cheek, the windows become irrelevant because none of these areas are darker or lighter than other regions on the cheeks, all sectors here are the same.

So we promptly discard irrelevant features and keep only those relevant with a fancy technique called Adaboost . **AdaBoost** is a training process for face detection, which selects only those features known to improve the classification (face/non-face) accuracy of our classifier.

In the end, the algorithm considers the fact that generally: most of the region in an image is a non-face region. Considering this, it’s a better idea to have a simple method to check if a window is a non-face region, and if it's not, discard it right away and don’t process it again. So we can focus mostly on the area where a face is.

**About the Code:**

**Facedetection\_webcam: This program uses the webcam to give live detection results.**

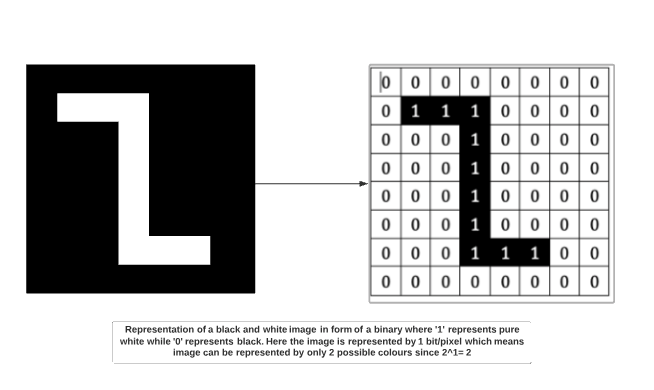
Algorithm and Code used :

# Images as Arrays

An image is nothing but a standard Numpy array containing pixels of data points. More the number of pixels in an image, the better is its resolution. You can think of pixels to be tiny blocks of information arranged in the form of a 2 D grid, and the depth of a pixel refers to the color information present in it. In order to be processed by a computer, an image needs to be converted into a binary form.

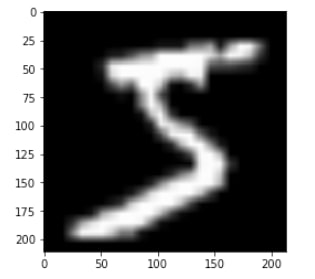
**i. Binary Image**

A binary image consists of 1 bit/pixel and so can have only two possible colors, i.e., black or white. Black is represented by the value 0 while 1 represents white.



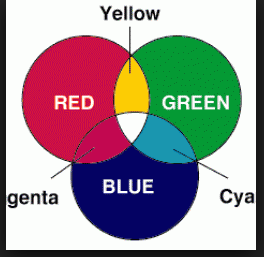
**ii. Grayscale image**

A grayscale image consists of 8 bits per pixel. This means it can have 256 different shades where 0 pixels will represent black color while 255 denotes white. For example, the image below shows a grayscale image represented in the form of an array. A grayscale image has only 1 channel where the channel represents dimension.



**iii. Colored image**

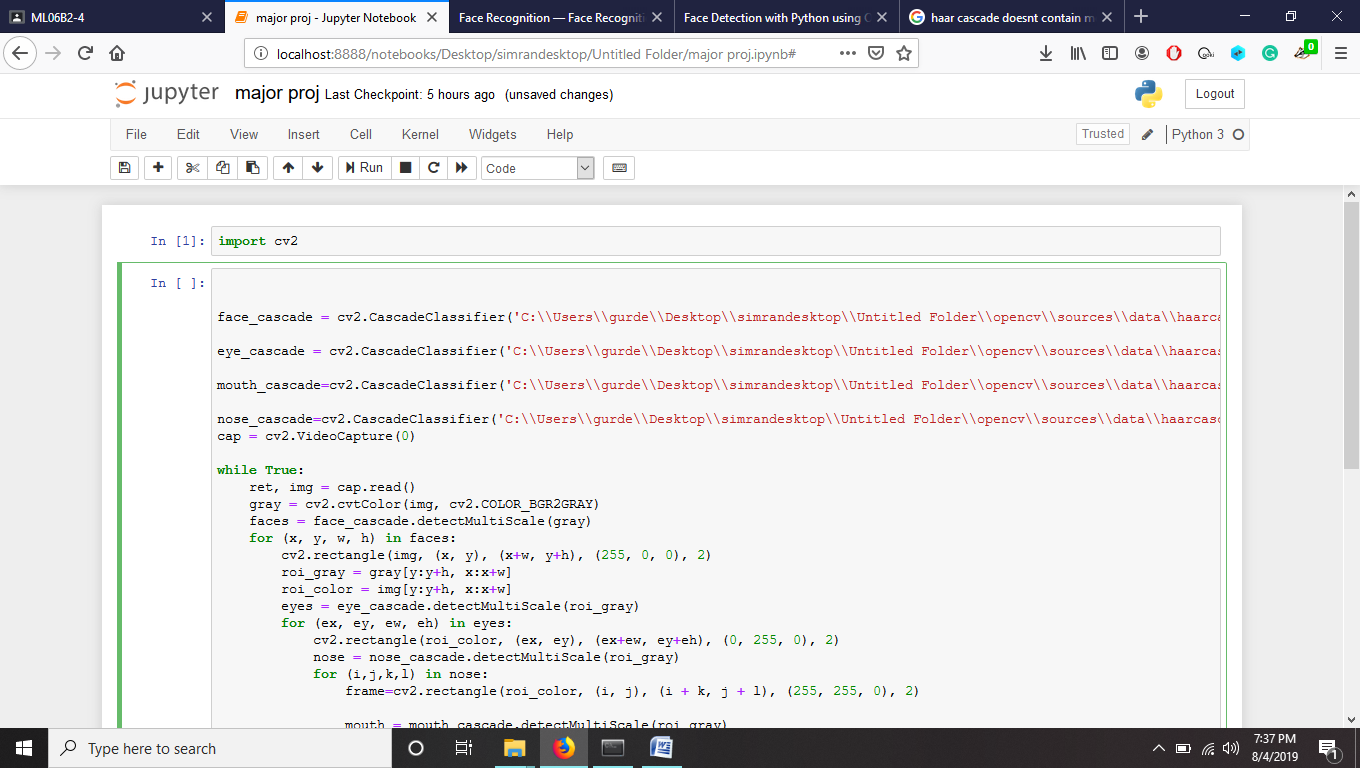
Colored images are represented as a combination of Red, Blue, and Green, and all the other colors can be achieved by mixing these primary colors in correct proportions.



**Face Detection with OpenCV-Python**

Now we have a fair idea about the intuition and the process behind Face

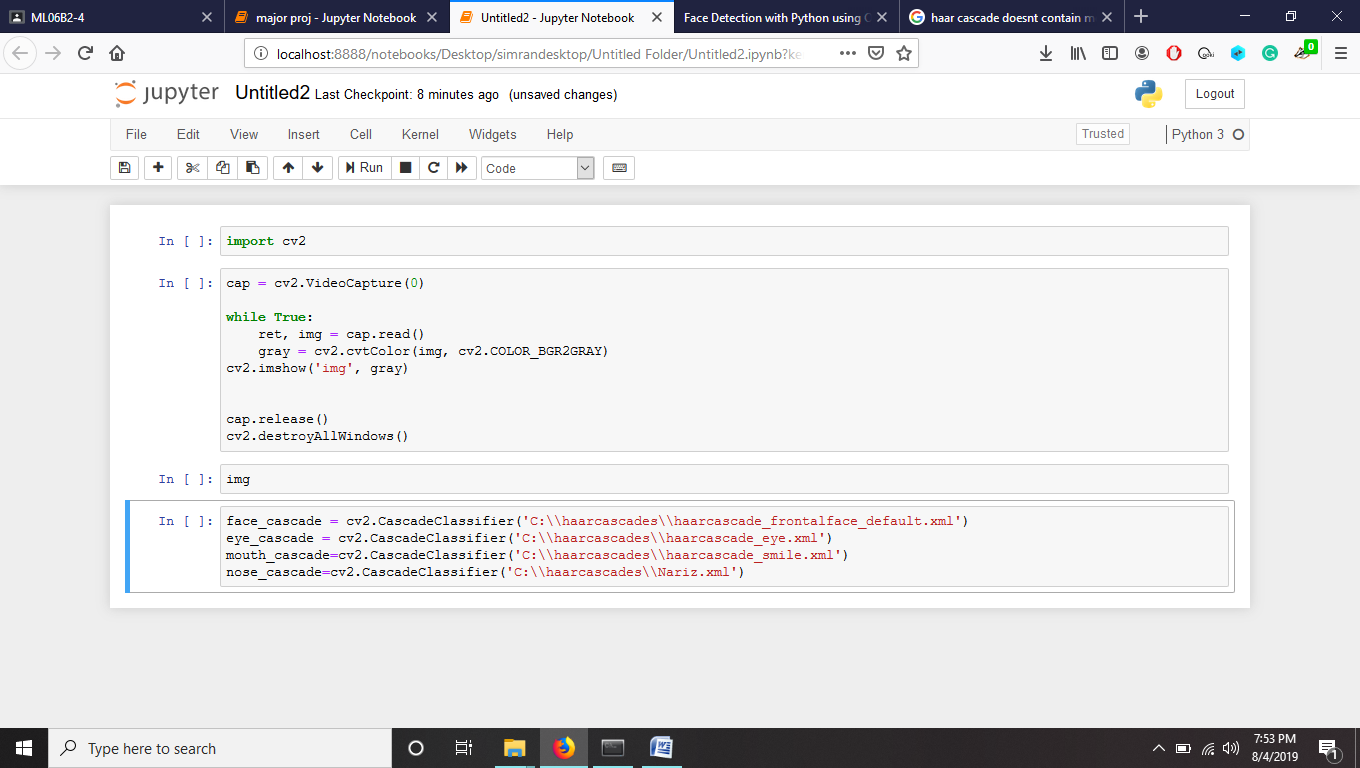
* **Import the necessary libraries**



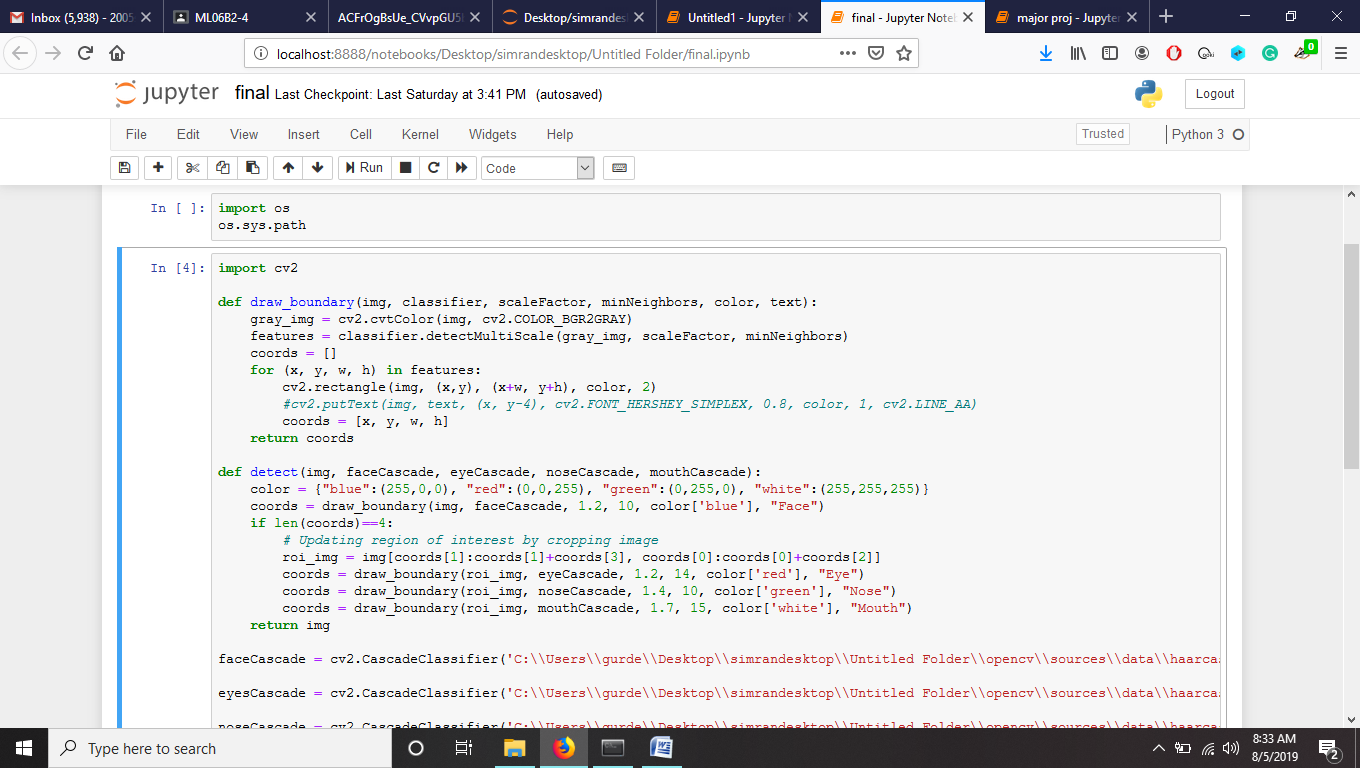
### Haar cascade files

OpenCV comes with a lot of pre-trained classifiers. For instance, there are classifiers for smile, eyes, face, etc. These come in the form of xml files and are located in the opencv/data/haarcascades/ folder. Download the xml files and place them in the data folder in the same working directory as the jupyter notebook.

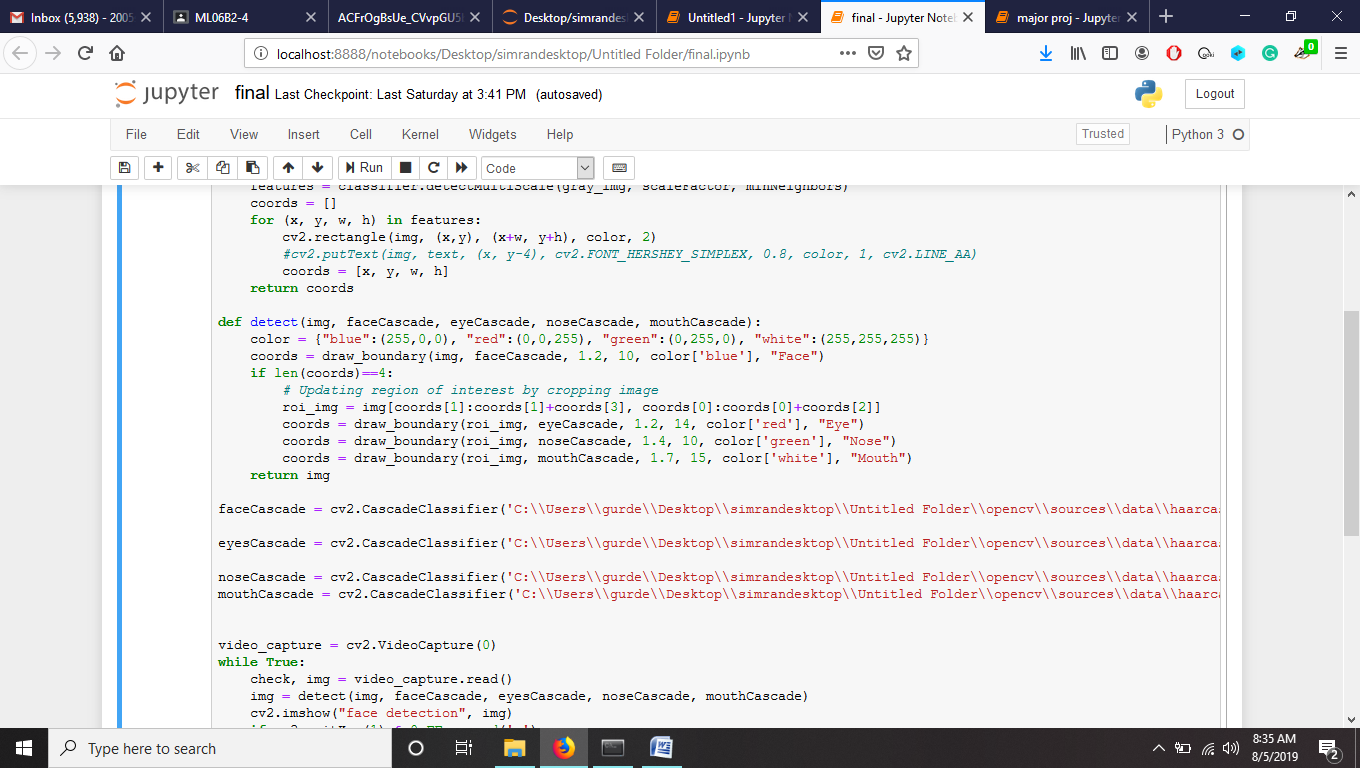
### Loading the classifier



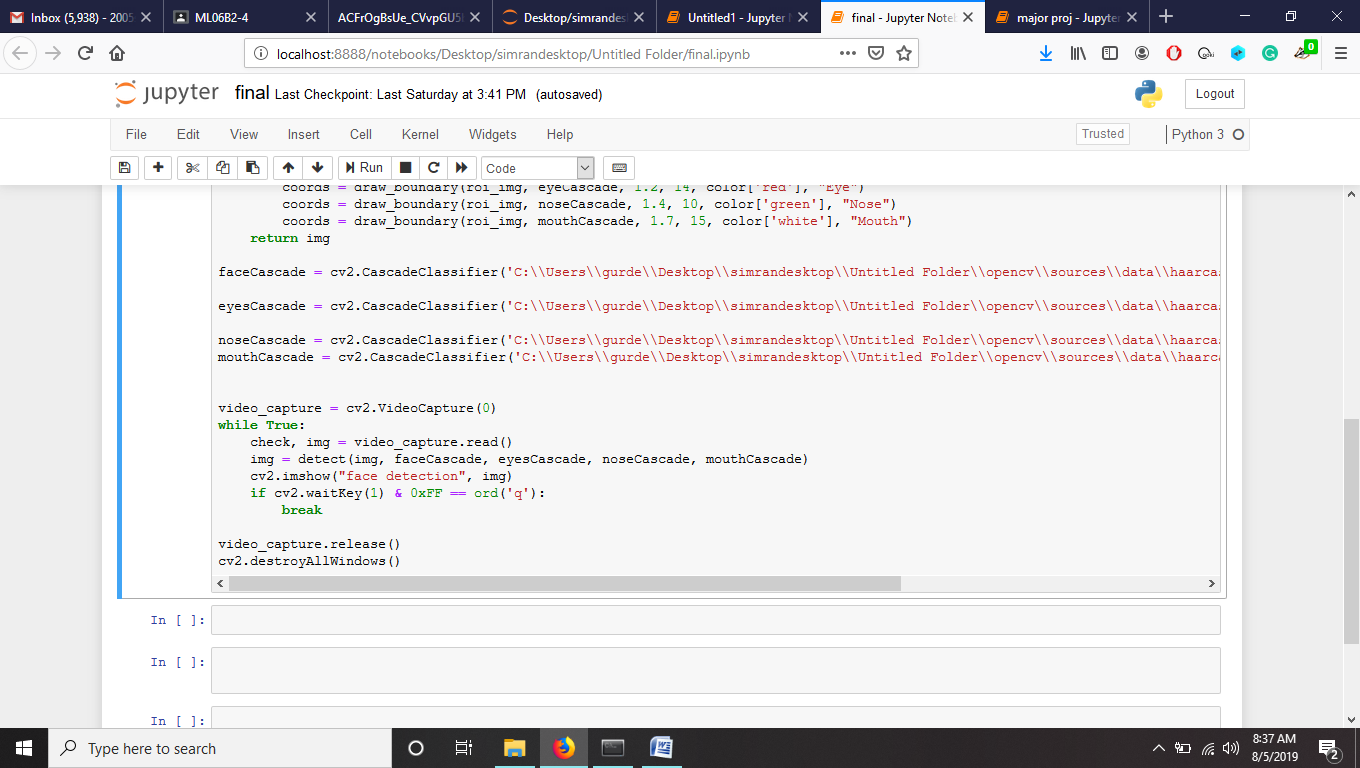
* **Function for making rectangles**



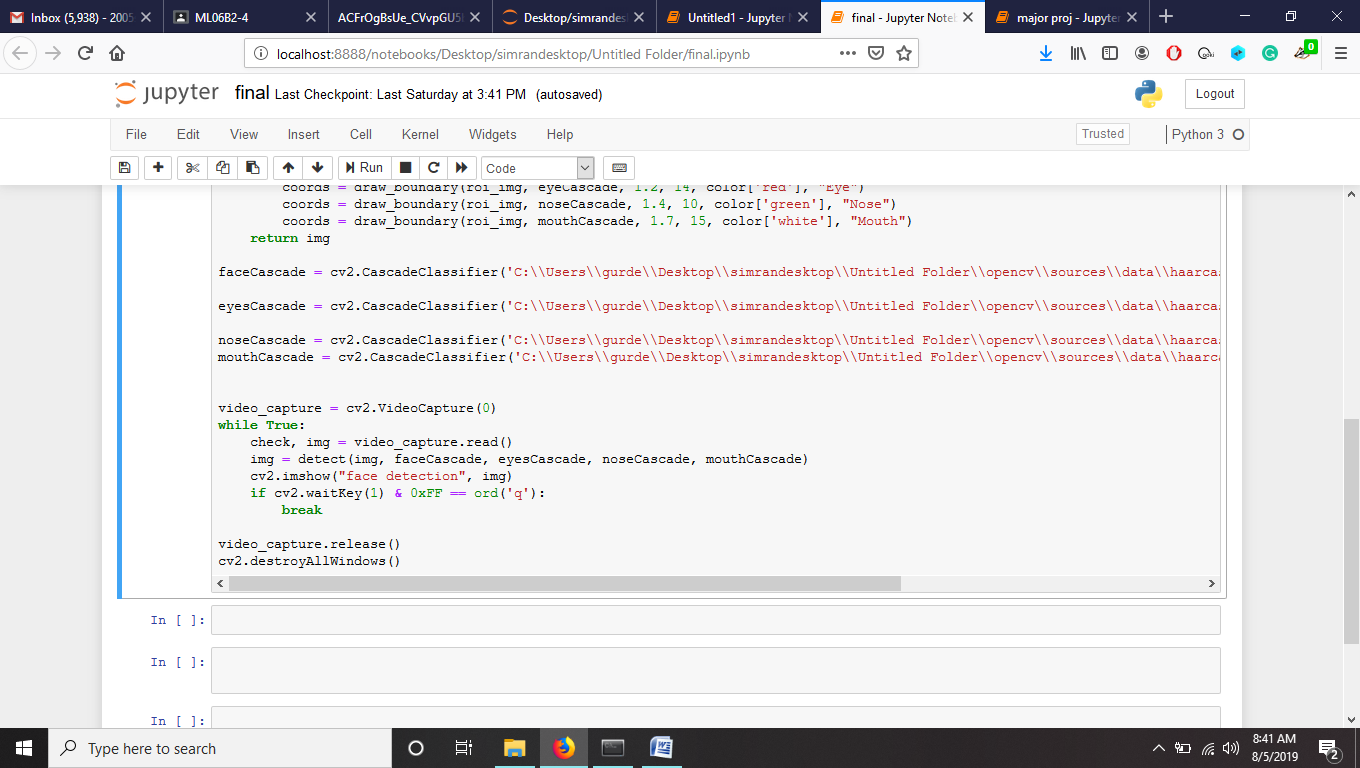
* **Function to detect features**



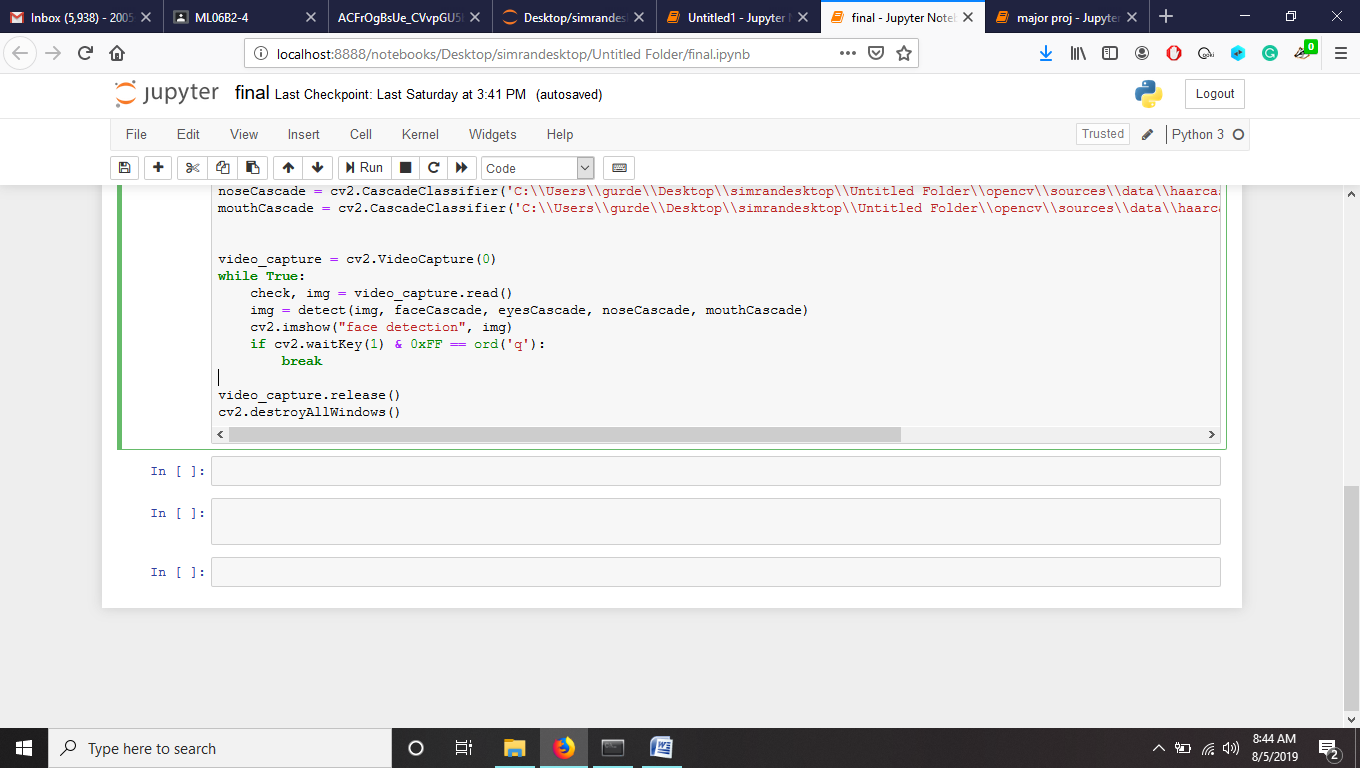
### Connecting the webcam and capturing images.



* **Calling the function “detect” and displaying the image**



* **Quit on pressing ‘q’**



* The [**cv2.waitKey()**](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html) is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues.
* The second condition pertains to the pressing of the Escape key on the keyboard. Thus, if 1 millisecond has passed and the escape key is pressed, the loop will break and program stops.
* [**cv2.destroyAllWindows()**](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html) simply destroys all the windows we created. If you want to destroy any specific window, use the function **cv2.destroyWindow()** where you pass the exact window name as the argument.
  + **Output on live frames (Webcam)**

