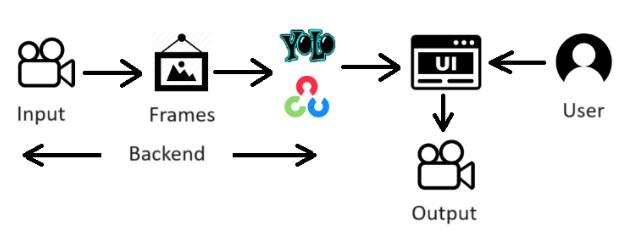
**Automated Milk Bottles Counting with YOLOv8**

**Project Description:**

The integration of YOLOv8 technology into automated milk bottle counting on conveyor belts marks a significant advancement in dairy production efficiency and accuracy. This cutting-edge solution, which seamlessly combines YOLOv8's real-time object detection capabilities with conveyor belt systems, streamlines the monitoring and tracking of milk bottles throughout the production process. By leveraging YOLOv8's advanced algorithms, this system optimizes production workflows, reduces errors, and maximizes productivity in dairy manufacturing. With its innovative approach, this solution not only enhances existing infrastructure but also sets a new standard for milk bottle counting precision, offering unparalleled insights and opportunities for automation in the dairy industry.

**Technical Architecture:**



**Pre-requisites:**

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

1. **IDE Installation**:

Spyder/ PyCharm IDE is Ideal to complete this project

To install **Spyder IDE**, please refer to [Spyder IDE Installation Steps](https://www.youtube.com/watch?v=ou65T_mC8Z8)

To install **PyCharm IDE**, please refer to the [PyCharm IDE Installation steps](https://www.youtube.com/watch?v=YxHplztMQMc)

1. **Python Packages**

If you are using **Anaconda Navigator**, follow the below steps to download the required packages:

Open the Anaconda prompt and create a virtual environment.

* Type “conda with base environment python=3.11”.
* Type "pip install opencv-python==4.7.0.68” and click enter.
* Type "pip install flask” and click enter.

**Prior Knowledge:**

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

* YOLO - <https://www.youtube.com/watch?v=ag3DLKsl2vk>
* 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 a broad understanding of the YOLO.
* Gain knowledge of OpenCV.

# 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,

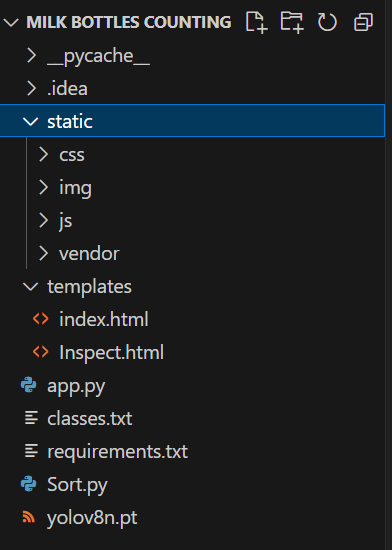
Create app1.py file

* Import the required libraries
* Flask Application Setup and Video Capture
* Model Initialization and Object Detection
* SORT Tracker Initialization and Line Setup
* Object Detection and Tracking Loop

Application building

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

**Project Structure:**

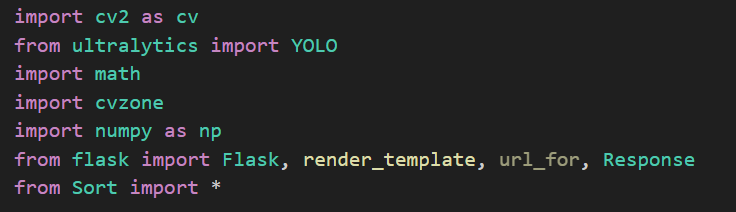


The "  Milk bottles counting " project consists of a Template folder containing HTML pages, and the app1.py file holds the Flask code responsible for Conveyor Craft and the bot.mp4 is input video and yolov8n.pt this is Pretrained model which we are using for our project and sort.py module is used for object tracking in the Flask application

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

**Activity 1: Import the required libraries**

We will be importing the necessary packages initially.



Import the OpenCV library as CV for image processing

Import cv zone library for additional image processing functionalities

Import the YOLO object detection model from ultralytics

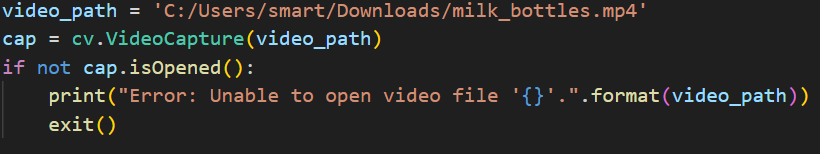
Import the math module for mathematical operations

Import the sort module for object tracking

Import the NumPy library for numerical operations

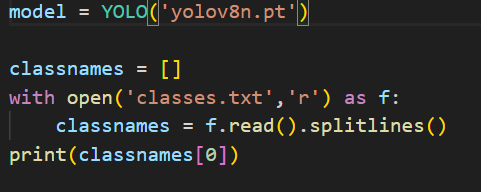
**Activity 2: Flask Application Setup and Video Capture**

This track sets up a Flask web application using the **Flask** library and initializes a video capture object (**cap**) to read frames from a video file named **'milk\_bottles.mp4'**. It checks if the video file can be opened and exits the program if it cannot.



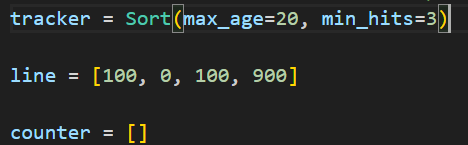
**Activity 3: Model Initialization and Object Detection:**

This track initializes the YOLO object detection model (model) using the YOLO class from the ultralytics library. It also reads class names from a file named 'classes.txt' and prints the first class name. This part of the code sets up the necessary components for object detection.



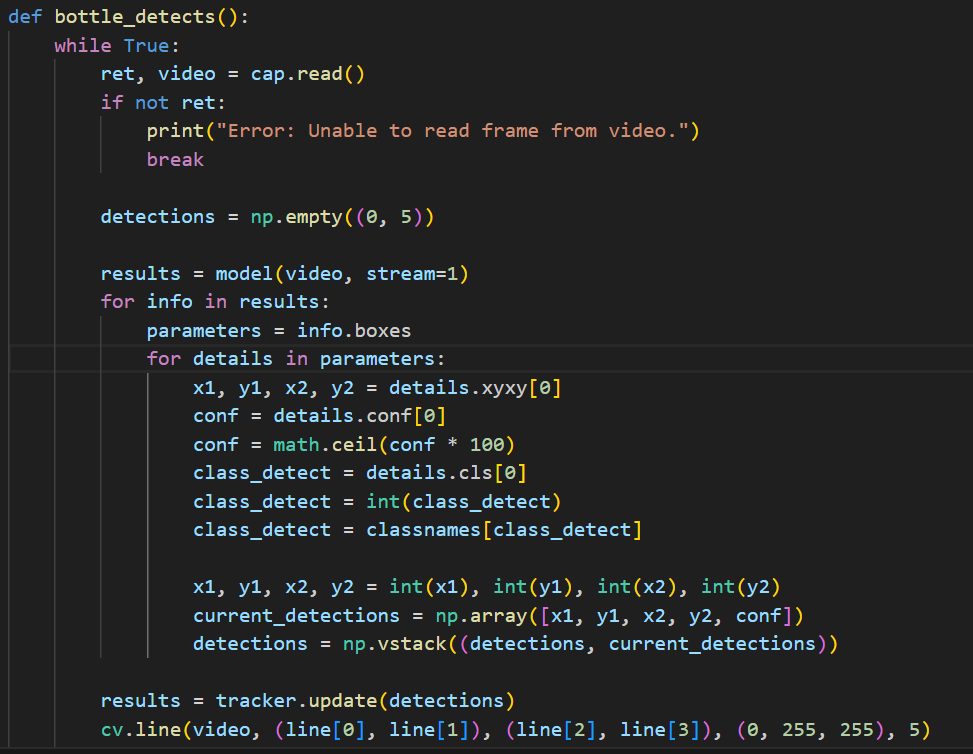
**Activity 4: SORT Tracker Initialization and Line Setup**

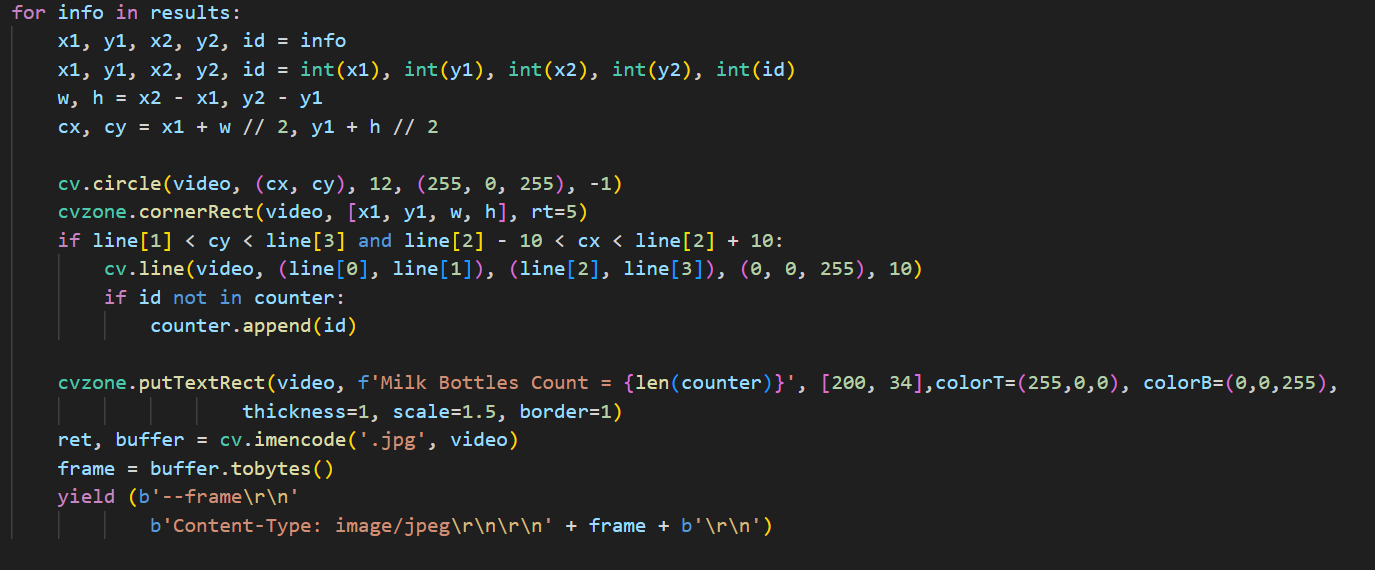
This track initializes the SORT (Simple Online and Realtime Tracking) tracker (tracker) using the Sort class from the sort module. It also defines the coordinates of a line (line) on the video frame, which will be used to count objects that cross the line.



**Activity 5: Object Detection and Tracking Loop:**

* while True: This starts an infinite loop, ensuring continuous processing of video frames until the program is interrupted or the video ends.
* ret, video = cap.read(): This line reads a frame (video) from the video capture object (cap) and returns a boolean value (ret) indicating whether the frame was successfully read.
* if not ret: This condition checks if the frame was not successfully read. If true, it means the end of the video or an error occurred, and the loop breaks.
* print("Error: Unable to read frame from video."):This prints an error message if a frame cannot be read from the video.
* break: This statement exits the loop if an error occurs while reading the frame.
* detections = np.empty((0, 5)): This initializes an empty NumPy array to store object detections. Each row of the array represents a detected object, with five columns for bounding box coordinates and confidence score.
* results = model(video, stream=1): This line performs object detection on the current frame (video) using the YOLO model (model). The stream=1 parameter enables streaming mode, providing real-time detection results.
* for info in results: This loop iterates over the detected objects in the results.
* parameters = info.boxes: This extracts the bounding box parameters of each detected object from the detection results.
* for details in parameters: This nested loop iterates over the details of each detected object, including its bounding box coordinates and confidence score.
* x1, y1, x2, y2 = details.xyxy[0]: This line extracts the coordinates of the bounding box from the details object.
* conf = details.conf[0]: This extracts the confidence score of the detection from the details object.
* conf = math.ceil(conf \* 100): This converts the confidence score to a percentage by multiplying it by 100 and rounding up using the math.ceil function.
* class\_detect = details.cls[0]: This extracts the class index of the detection from the details object.
* class\_detect = int(class\_detect): This converts the class index to an integer.
* class\_detect = classnames[class\_detect]: This retrieves the class name corresponding to the class index from the classnames list.
* x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2): This line converts the bounding box coordinates to integers.
* current\_detections = np.array([x1, y1, x2, y2, conf]): This creates an array containing the bounding box coordinates and confidence score of the detection.
* detections = np.vstack((detections, current\_detections)): This appends the current detection to the detections array using np.vstack, vertically stacking arrays.
* results = tracker.update(detections): This line updates the SORT tracker (tracker) with the detected objects stored in the detections array.
* cv.line(video, (line[0], line[1]), (line[2], line[3]), (0, 255, 255), 5): This draws a line on the video frame using OpenCV's cv.line function. The line's coordinates are defined by the line list, and its color is specified as (0,255,255).





# Milestone 2: Application Building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he/she has to navigate to detect button. Then the video will be showcased on the UI.

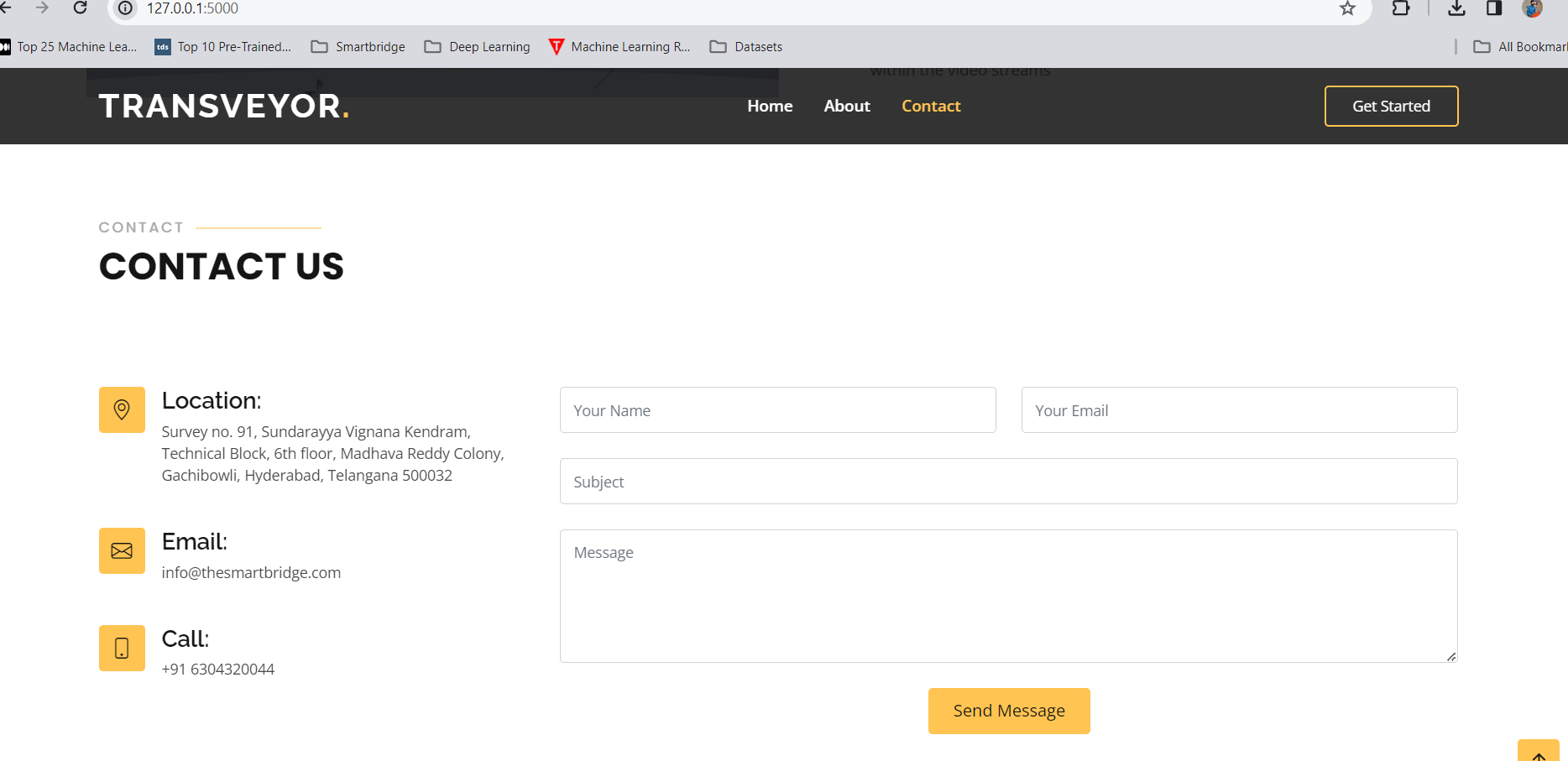
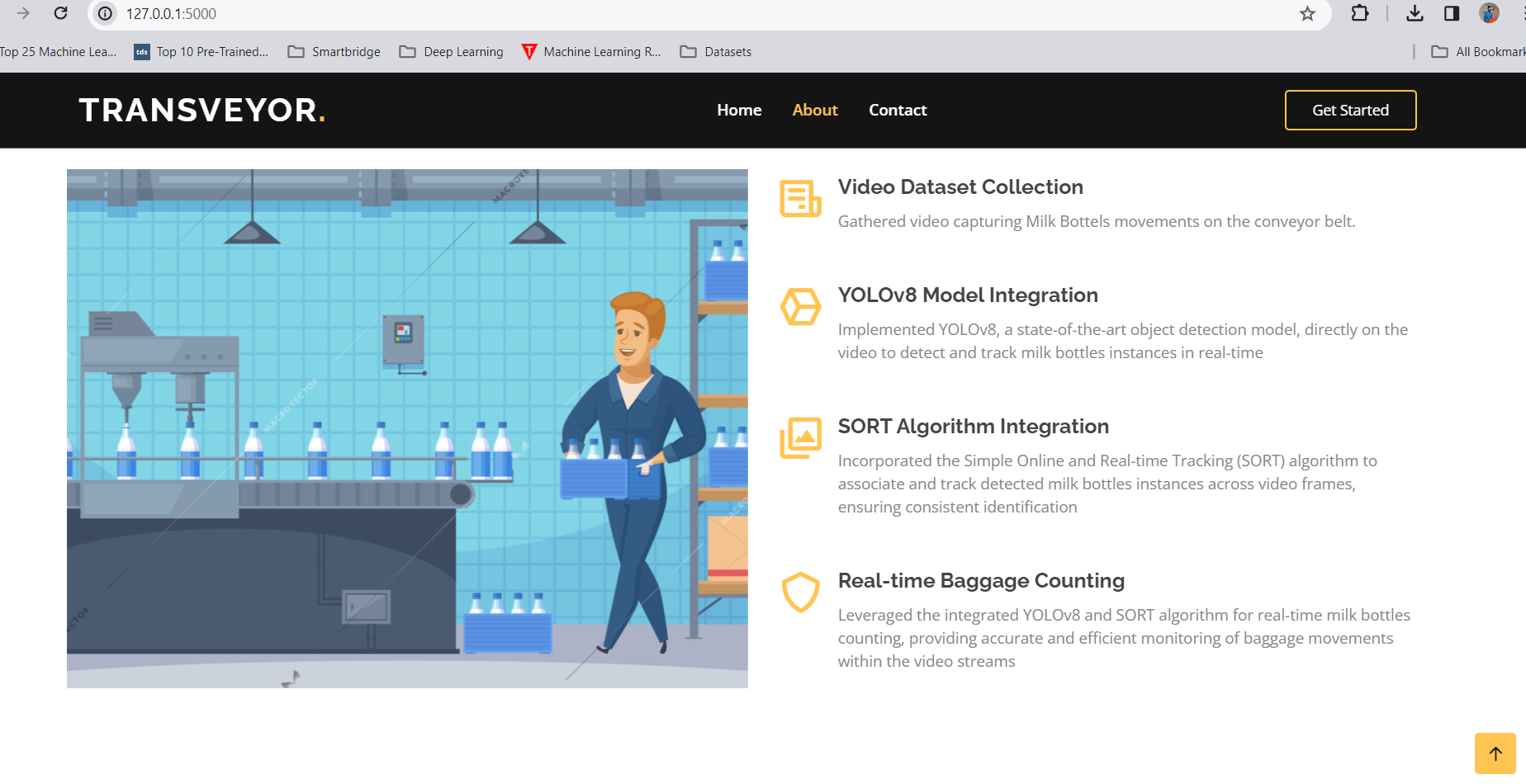
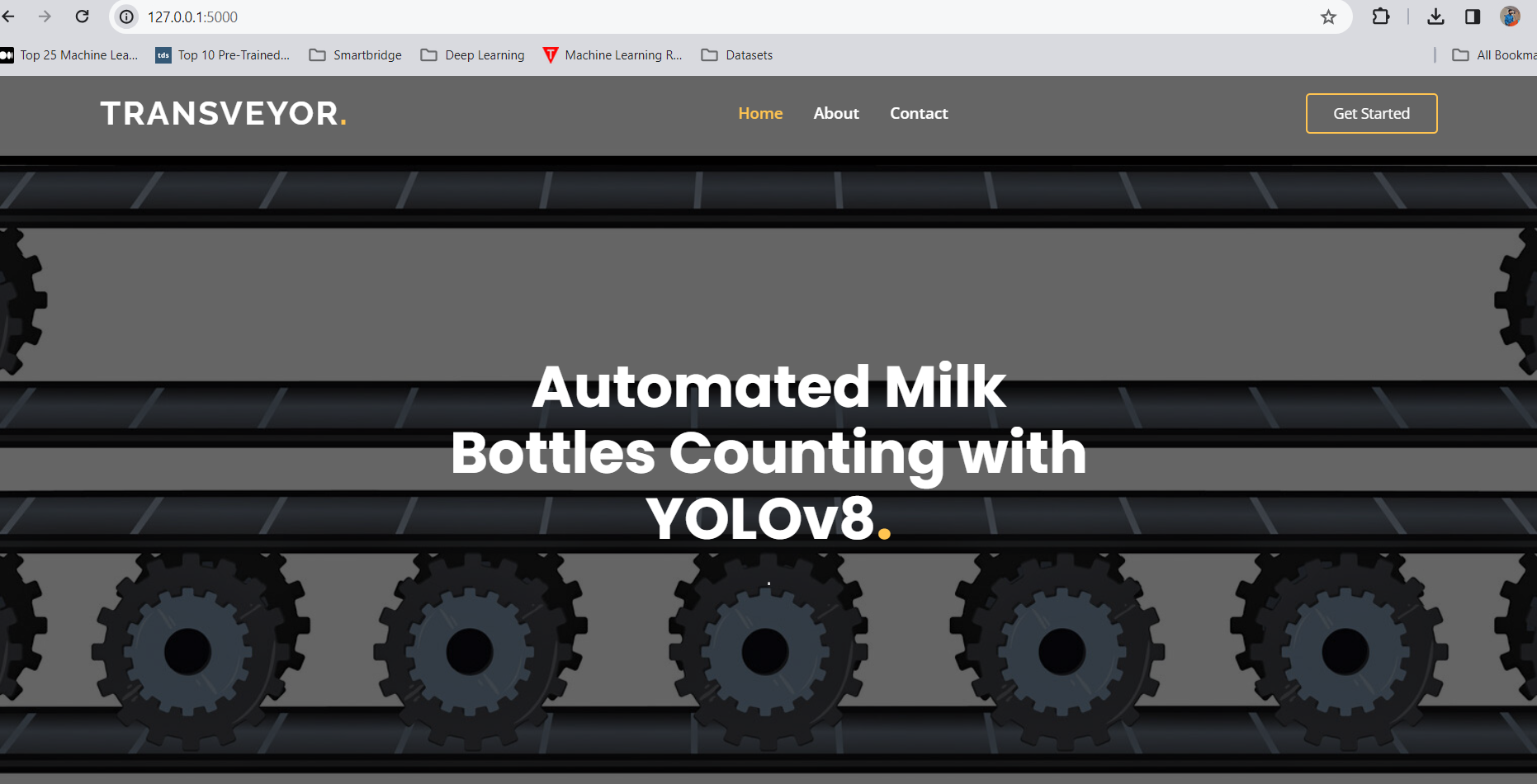
This section has the following tasks

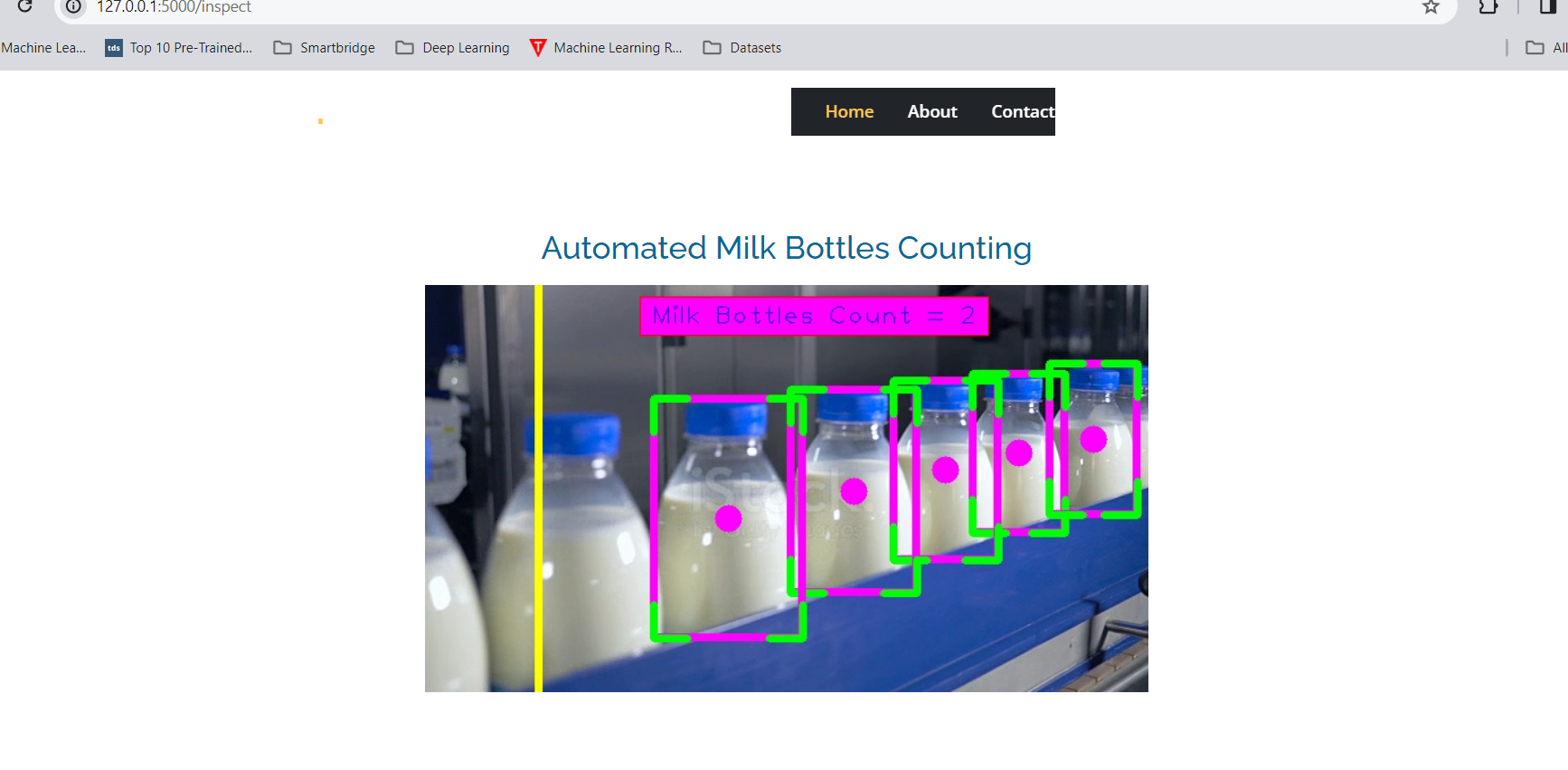
* Building HTML Pages
* Building server-side script

**Activity1: Building Html Pages:**

For this project we have created 2 HTML files and saved them in the templates folder.

Let’s see the HTML pages





**Activity 2: Build Python code:**

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

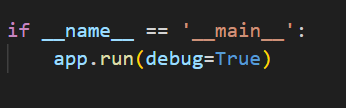


Render HTML page:

* 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.

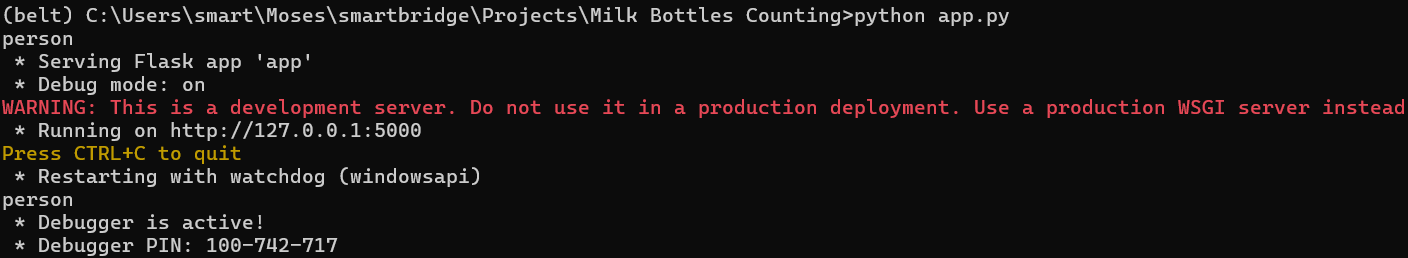


Main Function:

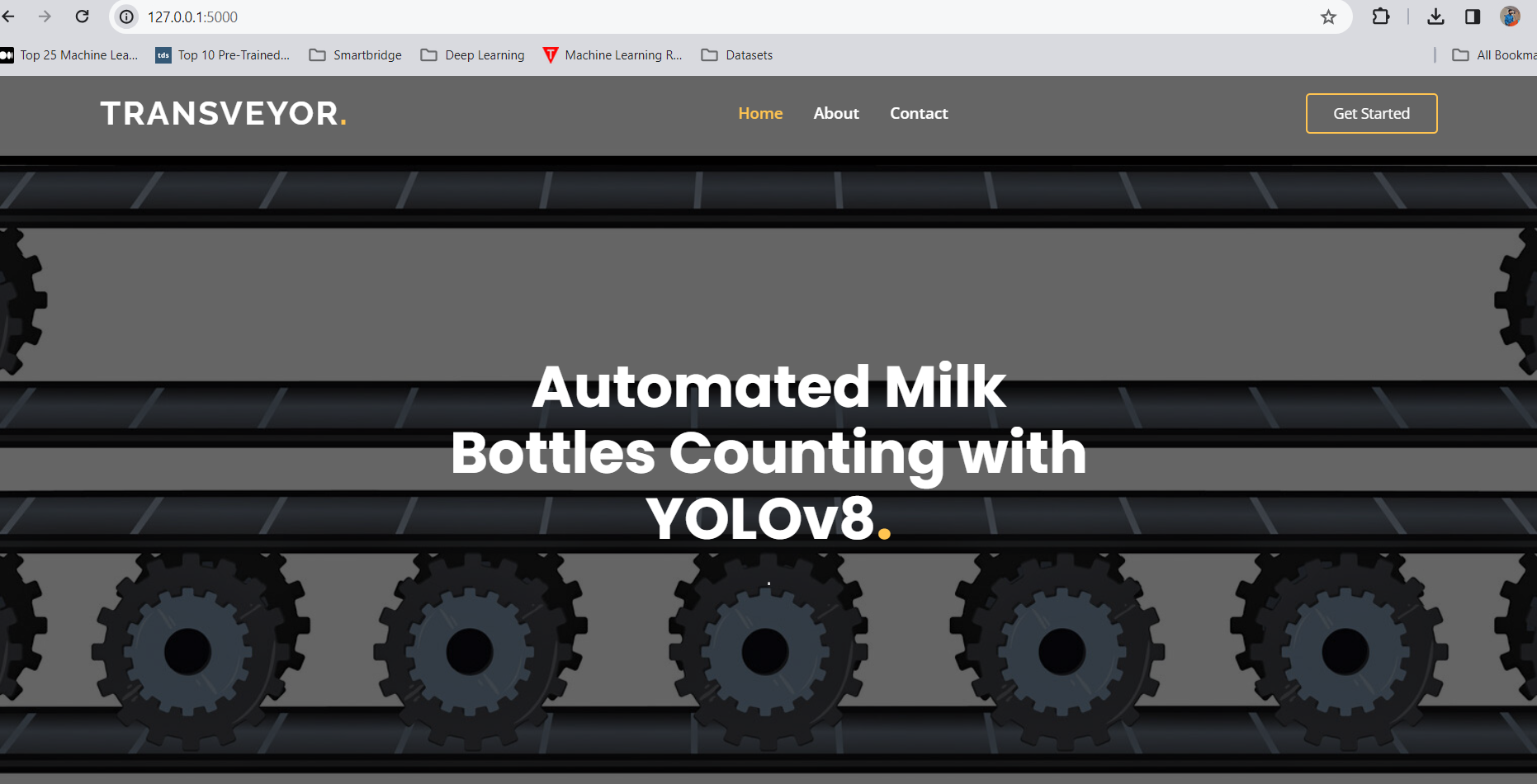


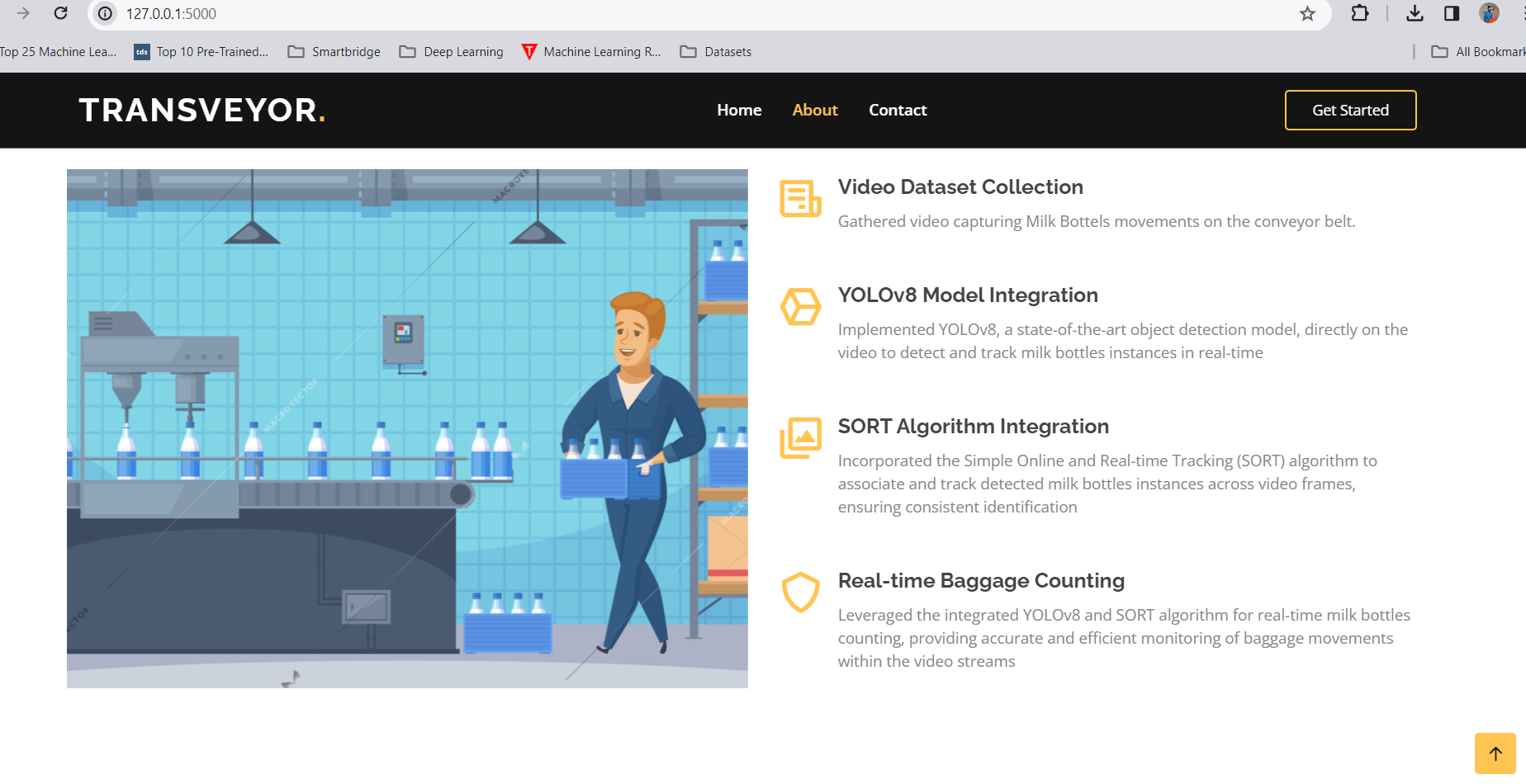
**Activity 3: Run the application**

* Open the anaconda prompt from the start menu
* Navigate to the folder where your python script is.
* type pytho app.py
* Navigate to the localhost where you can view your web page.
* Click on the inspect button from the top right corner, you will enter on result page where through web cam we can detect person and verify social distance between them

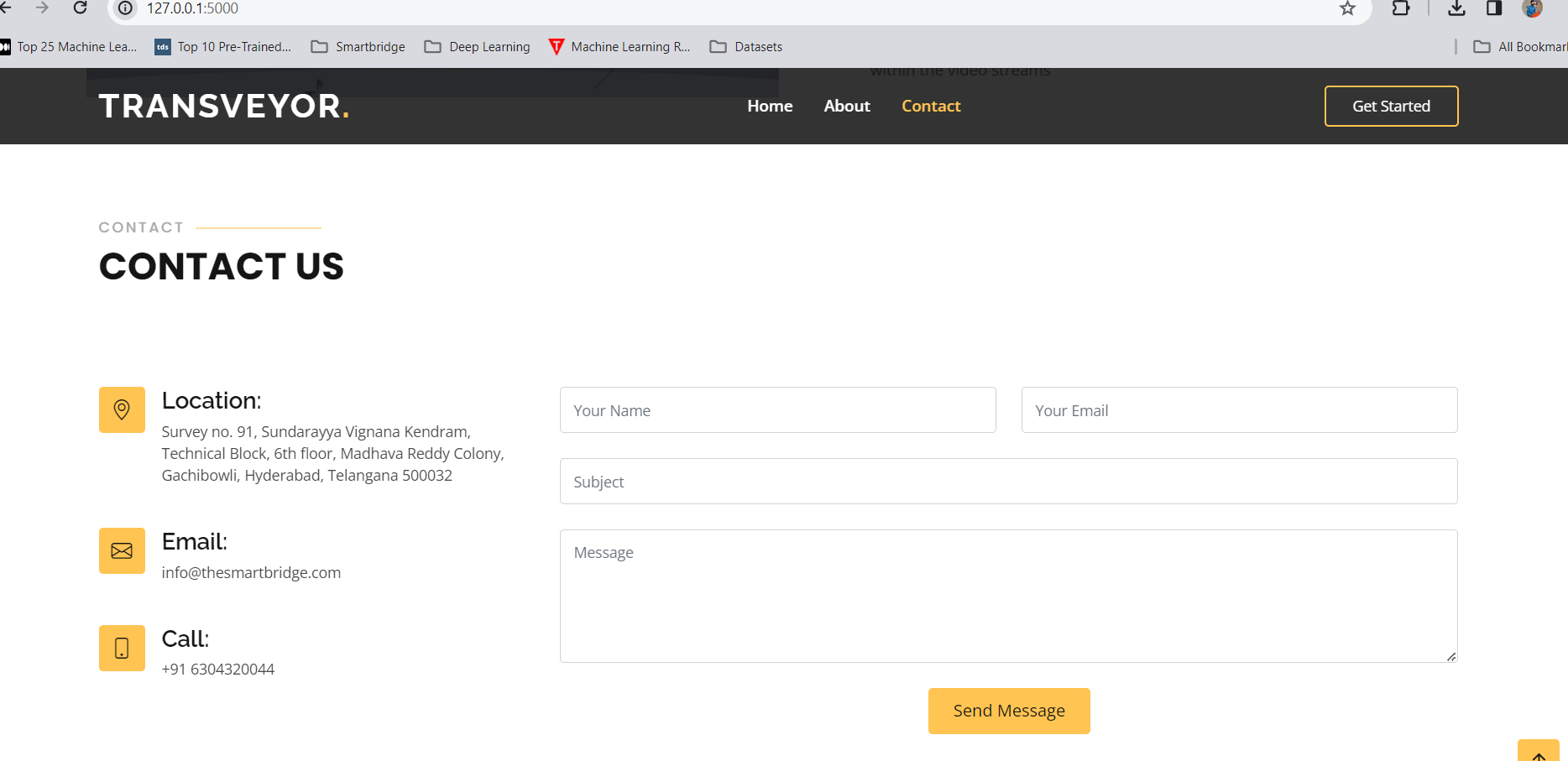


Home page:



About:

Contact :



OUTPUT :

