```
from scipy.spatial import distance as dist
#to measure distance between centroids
import numpy as np
import argparse
#to parse arguments through command line
#import imutils
import time
import cv2
import os

ap = argparse.ArgumentParser()
#preparing the argument parser
ap.add_argument("-i", "--input", required=True,
        help="path to input video")
ap.add_argument("-o", "--output", required=True,
        help="path to output video")
ap.add_argument("-y", "--yolo", required=True,
        help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
        help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
        help="threshold when applyong non-maxima suppression")
#passing arguments to ap
args = vars(ap.parse_args())
#getting the arguments from ap

labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
#loading the yolo class labels in labelspath
LABELS = open(labelsPath).read().strip().split("\n")
#saving classes names in LABELS

np.random.seed(42)
#make the random numbers predictable
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
        dtype="uint8")
#initialize a list of colors for every class in the YOLO model

weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])
#deriving the path to YOLO weights and configuration of the trained model

print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
# load our YOLO object detector trained on COCO dataset (80 classes)
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
# determine only the *output* layer names that we need from YOLO

vs = cv2.VideoCapture(args["input"])
writer = None
(W, H) = (None, None)
#dimensions of frame

try:
        prop = cv2.CAP_PROP_FRAME_COUNT
        total = int(vs.get(prop))
        print("[INFO] {} total frames in video".format(total))
#determining number of frames in the input video
except:
        print("[INFO] could not determine # of frames in video")
        print("[INFO] no approx. completion time can be provided")
        total = -1

while True:

        (grabbed, frame) = vs.read()
        #read the next frame from the file
        if not grabbed:
                break
                #end of the stream
        violate = set()


        if W is None or H is None:
                (H, W) = frame.shape[:2]
                #grabbing the dimensions of the frame

        blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
                swapRB=True, crop=False)
                #creating a blob
        net.setInput(blob)
        #computing blob input

        start = time.time()
        layerOutputs = net.forward(ln)
        #computing output of layers in ln
        end = time.time()
        elap = (end - start)

        boxes = []
        confidences = []
        classIDs = []
        centroids = []
        # initialize our lists of detected bounding boxes, confidences,
        # and class IDs, respectively

        for output in layerOutputs:
                #loop over layer outputs

                for detection in output:
                        #loop over each detaction

                        scores = detection[5:]
                        #collecting classID and probabilitiy from detection in the array(i.e.,
scores)
                        #for the current object detected
                        classID = np.argmax(scores)
                        #getting classID from scores
                        confidence = scores[classID]
                        # extract confidence (i.e., probability)
                        # of the current object detection

                        personId = LABELS.index("person")
                        #get the person class from the LABELS

                        if classID == personId and confidence > args["confidence"]:
                                #filter detections for person as object
                                box = detection[0:4] * np.array([W, H, W, H])
                                #extract coordinates for centroid and dimensions of box
                                (centerX, centerY, width, height) = box.astype("int")

                                x = int(centerX - (width / 2))
                                y = int(centerY - (height / 2))
                                #coordinates for top left corner of bounding box

                                boxes.append([x, y, int(width), int(height)])
                                confidences.append(float(confidence))
                                centroids.append((centerX, centerY))
                                classIDs.append(classID)
                                #update lists of boxes, confidences, centroids and classIDs

        if len(centroids) >= 2:
                #verifying that atleast there are two identifiable objects

                D = dist.cdist(centroids, centroids, metric="euclidean")
                #calculating euclidean distance between centroids
                for i in range(0, D.shape[0]):

                        for j in range(i + 1, D.shape[1]):

                                if D[i, j] < 50:

                                        violate.add(i)
                                        violate.add(j)
                #checking the distance between centroids in all sets

        idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
                args["threshold"])
                #applying non maxima suppression

        if len(idxs) > 0:
                #ensuring a detection

                for i in idxs.flatten():
                        #collapsing it in one dimension for convenient extraction

                        (x, y) = (boxes[i][0], boxes[i][1])
                        (w, h) = (boxes[i][2], boxes[i][3])
                        (cX, cY) = (centroids[i])
                        #extracting bounding box coordinates with respective centroids

                        color = [int(c) for c in COLORS[classIDs[i]]]
                        #assigning color to classIDs
                        if i in violate:
                                color = (0,0,255)
                                #changing color to red in case of violation

                        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
                        #draw the bounding box
                        cv2.circle(frame, (cX , cY), 5, color, 1)
                        #draw the centroid
                        text = "{}: {:.4f}".format(LABELS[classIDs[i]],
                                confidences[i])
                        cv2.putText(frame, text, (x, y - 5),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
                                #display the labels and confidences over bounding boxes

        text = "Social Distancing Violations: {}".format(len(violate))
        cv2.putText(frame, text, (10, frame.shape[0] - 25),
                cv2.FONT_HERSHEY_SIMPLEX, 0.85, (0, 0, 255), 3)
                #display total number of violations

        if writer is None:
                #initialize videoWriter
                fourcc = cv2.VideoWriter_fourcc(*"XVID")
                writer = cv2.VideoWriter(args["output"], fourcc, 30,
                        (frame.shape[1], frame.shape[0]), True)

                if total > 0:
                        print("[INFO] single frame took {:.4f} seconds".format(elap))
                        print("[INFO] estimated total time to finish: {:.4f} seconds".format(
                                elap * total))

        writer.write(frame)
        #write the frame to output video file

print("[INFO] cleaning up...")
writer.release()

#release the writer
vs.release()
#release input
```