Appendix 8: Code Developed for this Investigation Project Compiled and Written by Jesse van der Merwe

This code can be found at: https://github.com/JessWhosBack/EIE-Investigation-22G05.git

It should be noted that the appropriate licenses and full credit for all code that has been taken from an outside source is provided at the top of each script.

CONTENTS:

8A: isolateSpirals.py

```
# EIE Investigation: "Which Hand?"
# Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
# ELEN4012A NOVEMBER 2022

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# COPYRIGHT NOTICE:
# This code is taken from Kelvin da Silva's Master's project which involved the classification of tremors.
# This can be found at: https://github.com/kdasilva835842/tremor_classification
#
# Kelvin's code is based on the OpenCV Text Detection (EAST text detector) article by Adrian Rosebrock
(20/08/2021).
# This can be found at: https://pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/
#
# The code has been further modified, with permission from Kelvin, to suit the needs of this project.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

# IMPORTS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
import numpy as np
import cv2
import imutils
from imutils.object_detection import non_max_suppression
from pdf2image import convert_from_path
import glob
import os
import tempfile

# Read in the PDFs and convert to .jpg- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
image_array = []
image_names = []
image_path = []

counter = 0
rawCount = 0
sizeErrorCountA = 0
sizeErrorCountB = 0
sizeErrorCountC = 0

with tempfile.TemporaryDirectory() as tempDir:
    print('created temporary directory', tempDir)
    for outer_foldername in glob.glob('Data/Original/*'):
        for foldername in glob.glob(outer_foldername + '/*'):
            newFolderName = foldername.replace("Original", "Cropped")
            os.makedirs(newFolderName, exist_ok = True)
            os.makedirs(newFolderName+'/DrawingA', exist_ok = True)
            os.makedirs(newFolderName+'/DrawingB', exist_ok = True)
            os.makedirs(newFolderName+'/DrawingC', exist_ok = True)
            for filename in glob.glob(foldername + '/*.pdf'):
                arrayName = os.path.basename(filename)
                arrayName = arrayName.replace(".pdf","")
                newName = filename.replace(".pdf","")
                newName = str(tempDir) + "/"+str(arrayName)
                newPath = foldername.replace("Original", "Cropped")
                image_path.append(newPath)

                def convertPDFtoJGP(originalImage,finalImage,dpi=200):
                    pages = convert_from_path(originalImage, dpi)
                    for page in pages:
                        page.save(finalImage, 'JPEG')

                convertPDFtoJGP(filename,newName+'.jpg',150)
                image = cv2.imread(newName+'.jpg')
                image_array.append(image)
                image_names.append(arrayName)
                 rawCount = rawCount + 1

cropToleranceA = 0.9
cropToleranceB = 0.95
cropToleranceC = 0.9

MidPlaneFractionX = 0.4
MidPlaneFractionY = 0.5

# Desired dimensions of Drawings A and B (square) or Drawing C (rectangle)
squareDimension = 300
rectangleDimension = 600

# Array used to save the various widths of Drawing A, to better calculate the average width
array_width = []
array_width.append(475)
```

```python
# Arrays used to save the various x and y coordinates, to better calculate the average coordinates
array_xStartA = []
array_xStartA.append(140)
array_xStartB = []
array_xStartB.append(710)
array_xStartC = []
array_xStartC.append(140)
array_yEndA = []
array_yEndA.append(490)
array_yEndB = []
array_yEndB.append(490)
array_yEndC = []
array_yEndC.append(1032)

for counter,image in enumerate(image_array):
    print("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -")
    print("Starting image ", image_names[counter])
    copied = image.copy()

    orientedImage = copied.copy()
    finalA = orientedImage.copy()
    finalB = orientedImage.copy()
    finalC = orientedImage.copy()
    image = orientedImage.copy()

    (total_h,total_w) = image.shape[:2]
    mask = np.zeros((total_h,total_w), np.uint8)
    cv2.rectangle   (mask,  (0, int(0.245*total_h)),    (total_w, int(total_h*0.8)),    255,    -1)

    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.bitwise_and(image,mask)
    kernel = np.ones((3,3),np.uint8)
    image = cv2.erode(image,kernel,iterations = 1)
    image = cv2.cvtColor(image, cv2.COLOR_BAYER_BG2BGR)

    # Important: The EAST text requires that your input image dimensions be multiples of 32
    newW = 704
    newH = 704
    minConf = 0.8
    (H,W) = image.shape[:2]
    rW = W / float(newW)
    rH = H / float(newH)
    image = cv2.resize(image, (newW, newH))
    (H, W) = image.shape[:2]

    # Define the two output layer names for the EAST detector model -- the first is the output probabilities and the second can be used to derive the bounding box coordinates of text
    layerNames = [
        "feature_fusion/Conv_7/Sigmoid",
        "feature_fusion/concat_3"]

    # Load the pre-trained EAST text detector
    net = cv2.dnn.readNet("Cropping/frozen_east_text_detection.pb")

    # Construct a blob from image and then perform a forward pass of the model to obtain the two output layer sets
    blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
        (123.68, 116.78, 103.94), swapRB=True, crop=False)

    net.setInput(blob)
    (scores, geometry) = net.forward(layerNames)

    # Show timing information on text prediction grab the number of rows and columns from the scores volume, then initialize our set of bounding box rectangles and corresponding confidence scores
    (numRows, numCols) = scores.shape[2:4]
    rects = []
    confidences = []

    for y in range(0, numRows):
        # Extract scores (probabilities), followed by geometrical data to derive bounding box coordinates
        scoresData = scores[0, 0, y]
        xData0 = geometry[0, 0, y]
        xData1 = geometry[0, 1, y]
        xData2 = geometry[0, 2, y]
        xData3 = geometry[0, 3, y]
        anglesData = geometry[0, 4, y]

    # Loop over the number of columns
        for x in range(0, numCols):
            # If score does not have sufficient probability, ignore it
            if scoresData[x] < minConf:
                Continue
            # Compute the offset factor as resulting feature maps will be 4x smaller than the input image
            (offsetX, offsetY) = (x * 4.0, y * 4.0)

            # Extract the rotation angle for the prediction and then compute the sin and cosine
            angle = anglesData[x]
            cos = np.cos(angle)
            sin = np.sin(angle)

            # Use the geometry volume to derive the width and height of the bounding box
            h = xData0[x] + xData2[x]
            w = xData1[x] + xData3[x]

            # Compute both the starting and ending (x, y)-coordinates for the text prediction bounding box
            endX = int(offsetX + (cos * xData1[x]) + (sin * xData2[x]))
            endY = int(offsetY - (sin * xData1[x]) + (cos * xData2[x]))
            startX = int(endX - w)
            startY = int(endY - h)

            # Add the bounding box coordinates and probability score to respective lists
            rects.append((startX, startY, endX, endY))
            confidences.append(scoresData[x])

    # Apply non-maxima suppression to suppress weak, overlapping bounding boxes
    boxes = non_max_suppression(np.array(rects), probs=confidences)

    xStart = []
    xEnd = []
    yStart = []
    yEnd = []

    # Loop over the bounding boxes
    for (startX, startY, endX, endY) in boxes:
        # Scale the bounding box coordinates based on the respective ratios
        startX = int(startX * rW)
        startY = int(startY * rH)
        endX = int(endX * rW)
        endY = int(endY * rH)

        # Draw the bounding box on the image
        xStart.append(startX) # vector of x coords
        yStart.append(startY)
        xEnd.append(endX)
        yEnd.append(endY)

    (height,width) = finalA.shape[:2]
    orientation = 0

    # Set array position of the Drawing's to be -1 (i.e. not found)
    A_position = -1
    B_position = -1
    C_position = -1

    # If the coordinates match where a drawing should be, save that position to the corresponding drawing's
X_position variable
    for i in range(0, len(xStart)):
        if xStart[i] < 400:
            if yEnd[i] < total_h/2:
                if A_position == -1:
                    A_position = i
                else:
                    if xEnd[i] - xStart[i] < 100:
                        A_position = i
            elif yEnd[i] < 1300 and xEnd[i] < 500:
                C_position = i
        elif yStart[i] < total_h/2:
            if B_position == -1:
                B_position = i
            else:
                if xEnd[i] - xStart[i] < 100:
                    B_position = i

    # In case a drawing is not found, use the average values instead
    width = np.mean(array_width)
    xStart_A = np.mean(array_xStartA)
    xStart_B = np.mean(array_xStartB)
    xStart_C = np.mean(array_xStartC)
    yEnd_A = np.mean(array_yEndA)
    yEnd_B = np.mean(array_yEndB)
    yEnd_C = np.mean(array_yEndC)



    # Check if the drawing was found, and save the coordinates respectively
    if A_position != -1:
        xStart_A = xStart[A_position]
        yEnd_A = yEnd[A_position]
        array_xStartA.append(xStart_A)
        array_yEndA.append(yEnd_A)
    else:
        print("THERE IS NO A TEXT DETECTED - USING AVERAGE VALUES")

    if B_position != -1:
        xStart_B = xStart[B_position]
        yEnd_B = yEnd[B_position]
        array_xStartB.append(xStart_B)
        array_yEndB.append(yEnd_B)
    else:
        print("THERE IS NO B TEXT DETECTED - USING AVERAGE VALUES")

    if C_position != -1:
        xStart_C = xStart[C_position]
        yEnd_C = yEnd[C_position]
        array_xStartC.append(xStart_C)
        array_yEndC.append(yEnd_C)
    else:
        print("THERE IS NO C TEXT DETECTED - USING AVERAGE VALUES")

    # Using other detected Drawings to find better coordinates for missing Drawings
    if A_position == -1 and B_position != -1:
        xStart_A = xStart_B-total_w/2
        yEnd_A = yEnd[B_position]

    if B_position == -1 and A_position != -1:
        xStart_B = xStart_A*0.5 + total_w/2
        yEnd_B = yEnd[A_position]

    if A_position == -1 and B_position == -1:
        xStart_A = xStart_C
        xStart_B = xStart_A*0.5 + total_w/2

        yEnd_A = yStart[C_position] - total_h/3
        yEnd_B = yStart[C_position] - total_h/3

    # Calculate the width of Drawing A, either with average or coordinate values
    width = (xStart_B*cropToleranceA-xStart_A)
    array_width.append(width)

    # CROPPING OUT SPIRAL A - - - - - - - - - - - #
    try:
        finalA = cv2.cvtColor(finalA, cv2.COLOR_BGR2GRAY)
        finalA = finalA[int(yEnd_A):int(yEnd_A+width), int(xStart_A):int(xStart_A + width)]
        finalA = imutils.resize(finalA, width=squareDimension)

        new_image_path = str(image_path[counter])+'/DrawingA/'+str(image_names[counter])+"_A"+".jpg"
        cv2.imwrite(new_image_path, finalA)
    except Exception as e:
        print("Unable to save to file A as image size is empty: ",str(e))

    # CROPPING OUT SPIRAL B - - - - - - - - - - - #
    try:
        finalB = cv2.cvtColor(finalB, cv2.COLOR_BGR2GRAY)
        finalB = finalB[ int(yEnd_B) : int((yEnd_B+width)), int(xStart_B*cropToleranceB): int((xStart_B+width)*cropToleranceB)]
        finalB = imutils.resize(finalB, width=squareDimension)

        new_image_path = str(image_path[counter])+'/DrawingB/'+str(image_names[counter])+"_B"+".jpg"
        cv2.imwrite(new_image_path, finalB)
    except Exception as e:
        print("Unable to save to file B as image size is empty: ",str(e))

    # CROPPING SPIRAL C - - - - - - - - - - - #
    try:
        finalC = cv2.cvtColor(finalC, cv2.COLOR_BGR2GRAY)
        finalC = finalC[ int(yEnd_C) : int(yEnd_C+0.75*width), int(xStart_C): int(xStart_B+width)]
        finalC = imutils.resize(finalC, width=rectangleDimension)

        new_image_path = str(image_path[counter])+'/DrawingC/'+str(image_names[counter])+"_C"+".jpg"
        cv2.imwrite(new_image_path, finalC)
    except Exception as e:
        print("Unable to save to file C as image size is empty: ",str(e))
    print("Finished image ", image_names[counter])

# END OF CODE - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
```

8B: extractRectangle.py

```python
# EIE Investigation: "Which Hand?"
# Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
# ELEN4012A NOVEMBER 2022

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# COPYRIGHT NOTICE:
# This code contains snippets from Adrian Rosebrock's article "OpenCV shape detection" (08/02/2016)
# Which can be found at: https://pyimagesearch.com/2016/02/08/opencv-shape-detection/
#
# This code also contains snippets from jdhao's article "Cropping Rotated Rectangles from Image with OpenCV" (23/02/2019)
# Which can be found at: https://jdhao.github.io/2019/02/23/crop_rotated_rectangle_opencv/
# This code is under the license: CC BY-NC-ND 4.0 (https://creativecommons.org/licenses/by-nc-nd/4.0/)
# It has further been modified and combined to suit the needs of this project, but will not be further distributed.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

# IMPORTS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
import imutils
import cv2
import glob
import os
from PIL import Image
import numpy as np

# Read in the .jpgs and save into an image array- - - - - - - - - - - - - - - - - - - - - - - - - - - - #
image_array = []
image_names = []
image_path = []

for outer_foldername in glob.glob('Data\Cropped\*'):
    for foldername in glob.glob(outer_foldername + '\*'):
        for filename in glob.glob(foldername + "\DrawingC\*.jpg"):
            im = Image.open(filename).convert('L') # convert image to grayscale
            res = im.point((lambda p: 256 if p>=200 else 0)) # convert each pixel into either black or white
            res.save(filename)

            arrayName = os.path.basename(filename)
            arrayName = arrayName.replace(".jpg","")
            newPath = foldername + '\DrawingC\Rectangles'
            os.makedirs(newPath, exist_ok = True)
            image_path.append(newPath)
            image = cv2.imread(filename)
            image = cv2.bitwise_not(image) # invert the colors of the image
            image_array.append(image)
            image_names.append(arrayName)

# Loop through the image array and extract the top-most hand drawn rectangle- - - - - - - - - - - - - - - - #
for image_counter,images in enumerate(image_array):
    image = images.copy()
    ratio = image.shape[0] / float(image.shape[0])
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    thresh = cv2.threshold(blurred, 200, 255, cv2.THRESH_BINARY)[1]

    contours = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)
    coordinate_1 = []
    coordinate_2 = []
    coordinate_3 = []
    coordinate_4 = []

    for c in contours:
        if cv2.contourArea(c) > 50:
            try:
                x1, y1, w, h = cv2.boundingRect(c)
                if w > 350:
                    M = cv2.moments(c)
                    cX = int((M["m10"] / M["m00"]) * ratio)
                    cY = int((M["m01"] / M["m00"]) * ratio)

                    c = c.astype("float")
                    c *= ratio # multiply the contour (x, y)-coordinates by the resize ratio
                    c = c.astype("int")

                    rect = cv2.minAreaRect(c)
                    box = cv2.boxPoints(rect)
                    box = np.int0(box)

                    b_counter = 0
                    b_left_1 = -1
                    b_left_2 = -1
                    b_right_1 = -1
                    b_right_2 = -1

                    # Sort the boxes so that the coordinates are in the correct order
                    for b in box:
                        if b[0] < 100:
                            if b_left_1 == -1:
                                b_left_1 = b_counter
                            else:
                                b_left_2 = b_counter
                        else:
                            if b_right_1 == -1:
                                b_right_1 = b_counter
                            else:
                                b_right_2 = b_counter
                         b_counter = b_counter + 1

                    if box[b_left_1][1] < box[b_left_2][1]:
                        coordinate_1.append(box[b_left_1])
                        coordinate_4.append(box[b_left_2])
                    else:
                        coordinate_1.append(box[b_left_2])
                        coordinate_4.append(box[b_left_1])

                    if box[b_right_1][1] < box[b_right_2][1]:
                        coordinate_2.append(box[b_right_1])
                        coordinate_3.append(box[b_right_2])
                    else:
                        coordinate_2.append(box[b_right_2])
                        coordinate_3.append(box[b_right_1])
            except:
                print("ERROR")
     try:
        counter = 0
        number_boxes = len(coordinate_1)

        # Since we detected the black surrounding boxes, we can now work out the coordinates of inner drawing
        new_coordinate_1 = coordinate_4[number_boxes-1]
        new_coordinate_2 = coordinate_3[number_boxes-1]
        new_coordinate_3 = coordinate_2[number_boxes-2]
        new_coordinate_4 = coordinate_1[number_boxes-2]
        new_coordinate_array = np.array([
            [new_coordinate_1],
            [new_coordinate_2],
            [new_coordinate_3],
            [new_coordinate_4]
        ])
        rect = cv2.minAreaRect(new_coordinate_array)
        new_box = cv2.boxPoints(rect)
        new_box = np.int0(new_box)

        # get width and height of the detected rectangle
        new_width = int(rect[1][0])
        new_height = int(rect[1][1])
        new_box_float = new_box.astype("float32")

        # coordinate of the points in box points after the rectangle has been straightened
        straightened_array = np.array([[0, new_height-1], [0, 0], [new_width-1, 0],
                            [new_width-1, new_height-1]], dtype="float32")

        M = cv2.getPerspectiveTransform(new_box_float, straightened_array) # the perspective transformation matrix

        # directly warp the rotated rectangle to get the straightened rectangle
        warped = cv2.warpPerspective(image.copy(), M, (new_width, new_height))

        (warped_H,warped_W) = warped.shape[:2]
        if warped_H > warped_W: warped = cv2.rotate(warped, cv2.ROTATE_90_CLOCKWISE)

        warped = cv2.bitwise_not(imutils.resize(warped, width=600))
        (warped_H,warped_W) = warped.shape[:2]
        warped = warped[5 : warped_H - 5, 5: warped_W-5]

        cv2.imwrite(str(image_path[image_counter]) + "/"+str(image_names[image_counter])+"_RECT.jpg", warped)
        cv2.waitKey(0)
    except:
        print("ERROR")

    # END OF CODE - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
```

8C: method2.py

```python
# EIE Investigation: "Which Hand?"
# Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
# ELEN4012A NOVEMBER 2022

# IMPORTS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
import cv2
import glob
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from numpy import trapz
from enum import Enum
import pandas as pd
from IPython.display import display
from scipy import signal
from scipy.fft import rfft, rfftfreq, irfft

class Results(Enum):
    FILENAME = 0
    PATIENT_NUMBER = 1
    DOMINANT_HAND = 2
    TREATED_HAND = 3
    TIME_PERIOD = 4
    PD_HAND = 5
    AREA_TRAPZ = 6
    MAX = 7
    STDDEV = 8
    AVG_AREA_TRAPZ = 9
    AVG_AREA_MAX = 10
    AVG_AREA_STDDEV = 11
    NUM_PEAKS = 12
    AVG_PEAK_DIST = 13

# Read in the .jpgs and save into an image array, ensuring grayscale- - - - - - - - - - - - - - - - - - - - - #
image_array = []
image_names = []
image_patient_number = []
image_time_frame = []
image_dominant_hand = []
image_treated_hand = []
image_PD_hand = []
file_count = 0

def get_time_period(image_name):
    image_name = str.lower(image_name)

    if image_name.find("w") != -1:
        period = image_name[0:image_name.find("w")]
        time =  [int(s) for s in period.split() if s.isdigit()]

        if period.find("one") != -1:
            output = "1W"
        else:
            output = str(time[0]) + "W"
        return output

    if image_name.find("m") != -1:
        period = image_name[0:image_name.find("m")]
        time =  [int(s) for s in period.split() if s.isdigit()]

        if period.find("one") != -1:
            output = "1M"
            return output
         else:
            try:
                output = str(time[0]) + "M"
                return output
            except:
                pass # Presentation purposes, delete later
                # print("ERROR - m but no time")

    if image_name.find("y") != -1:
        period = image_name[0:image_name.find("y")]
        time =  [int(s) for s in period.split() if s.isdigit()]

        if period.find("one") != -1:
            output = "1Y"
            return output
        else:
            try:
                output = str(time[0]) + "Y"
                return output
            except:
                print("ERROR - y but no time - " + str(image_name))

    if image_name.find("before") != -1:
        return "before"
    return -1

def is_treated_hand(image_name, patient_number):
    image_name = str.lower(image_name)

    if patient_number == 6 or patient_number == 17 or patient_number == 22 or patient_number == 32 or patient_number == 74 or patient_number == 85 or patient_number == 87 or patient_number == 109 or patient_number == 111 or patient_number == 115 or patient_number == 116  or patient_number == 8 or patient_number == 16 or patient_number == 31 or patient_number == 39 or patient_number == 40 or patient_number == 47 or patient_number == 56 or patient_number == 57 or patient_number == 124:
        left_position = image_name.find("lt")
        if left_position == -1:
            left_position = image_name.find("left")
            if left_position == -1:
                return False
            else:
                return True
        else:
            return True
    else:
        right_position = image_name.find("rt")
        if right_position == -1:
            right_position = image_name.find("right")
            if right_position == -1:
                return False
            else:
                return True
        else:
            return True

def is_dominant_hand(image_name, patient_number):
    image_name = str.lower(image_name)

    if patient_number == 17 or patient_number == 22 or patient_number == 32 or patient_number == 56 or patient_number == 74 or patient_number == 87 or patient_number == 102 or patient_number == 109 or patient_number == 111 or patient_number == 115 or patient_number == 116 or patient_number == 117:
        left_position = image_name.find("lt")
        if left_position == -1:
            left_position = image_name.find("left")
            if left_position == -1:
                return False
            else:
                return True
        else:
            return True
    else:
        right_position = image_name.find("rt")
        if right_position == -1:
            right_position = image_name.find("right")
            if right_position == -1:
                return False
            else:
                return True
        else:
            return True

def is_PD_hand(patient_number):
    if patient_number == 3 or patient_number == 4 or patient_number == 7 or patient_number == 8 or patient_number == 16 or patient_number == 18 or patient_number == 28 or patient_number == 31 or patient_number == 33 or patient_number == 37 or patient_number == 38 or patient_number == 39 or patient_number == 40 or patient_number == 43 or patient_number == 44 or patient_number == 47 or patient_number == 48 or patient_number == 55 or patient_number == 56 or patient_number == 57 or patient_number == 59 or patient_number == 63 or patient_number == 68 or patient_number == 70 or patient_number == 71 or patient_number == 76 or patient_number == 77 or patient_number == 92 or patient_number == 100 or patient_number == 112 or patient_number == 113 or patient_number == 114 or patient_number == 120 or patient_number == 124:
        return True
    else:
        return False

for outer_foldername in glob.glob('Data\Cropped\*'):
    for foldername in glob.glob(outer_foldername + '\*'):
        patient_number = os.path.basename(foldername)
        patient_number = patient_number.replace("#", "")

        for filename in glob.glob(foldername + "\DrawingC\Rectangles\*.jpg"):
            arrayName = os.path.basename(filename)
            arrayName = arrayName.replace(".jpg","")

            time_period = get_time_period(arrayName)
            is_dominant = is_dominant_hand(arrayName, patient_number)
            is_treated = is_treated_hand(arrayName, patient_number)
            is_PD = is_PD_hand(patient_number)
            if time_period != -1:
                im = Image.open(filename).convert('L') # convert image to grayscale
                res = im.point((lambda p: 256 if p>=200 else 0)) # convert each pixel into either black or white
                res.save(filename)

                image = cv2.imread(filename)
                image_array.append(image)
                image_names.append(arrayName)
                image_patient_number.append(patient_number)
                image_time_frame.append(time_period)
                image_dominant_hand.append(is_dominant)
                image_treated_hand.append(is_treated)
                image_PD_hand.append(is_PD)
                file_count = file_count + 1

results_array = [[-1 for x in range(len(Results))] for y in range(file_count)]

temp_counter_patient_5 = -1
# Loop through the image array to perform image processing- - - - - - - - - - - - - - - - - - - - - - - - - - - #
for image_counter, image in enumerate(image_array):
    temp_counter_patient_5 += 1
    temp_title = image_names[image_counter].replace("(5)_C__RECT", "")

    results_array[image_counter][Results.FILENAME.value] = image_names[image_counter]
    results_array[image_counter][Results.PATIENT_NUMBER.value] = image_patient_number[image_counter]
    results_array[image_counter][Results.TIME_PERIOD.value] = image_time_frame[image_counter]
    results_array[image_counter][Results.DOMINANT_HAND.value] = image_dominant_hand[image_counter]
    results_array[image_counter][Results.TREATED_HAND.value] = image_treated_hand[image_counter]
    results_array[image_counter][Results.PD_HAND.value] = image_PD_hand[image_counter]

    coordinates = np.argwhere(image < 0.9)
    # fig, (axs_original, axs_fft, axs_ifft, axs_area) = plt.subplots(4)

    try:
        # NOTE: The horizontal axis is denoted by y, and the vertical axis is denoted by x (ACCIDENTAL MISTAKE)
        x_tuple, y_tuple, z_tuple = zip(*coordinates)

        x = np.array(x_tuple).tolist()
        y = np.array(y_tuple).tolist()
        z = np.array(z_tuple).tolist()

        # Sort the array of pixels into an ordered array according to the horizontal x-axis
        for i in range(0, len(y)-1):
            for j in range(0, len(y)-i-1):
                if y[j] > y[j+1]:
                    temp_x = x[j]
                    x[j] = x[j+1]
                    x[j+1] = temp_x

                    temp_y = y[j]
                    y[j] = y[j+1]
                    y[j+1] = temp_y

        # GRAPH 1: ORIGINAL
        # axs_original.set_title("ORIGINAL GRAPH: " + str(temp_title))
        # axs_original.plot(y, x)

        average_x = np.mean(x)
        new_array_x = []
        new_array_y = []

        # Take the absolute value and shift the points down to be centered around the horizontal axis
        for i in range(0, len(x)):
            new_array_y.append(y[i])
            new_array_x.append(x[i]-average_x)

        data_step = 0.1
        n = len(new_array_y)
        yf = rfft(new_array_x)
        xf = rfftfreq(n, data_step)

        yf_abs = np.abs(yf)
        yf_max = np.max(yf_abs)

        # # GRAPH 2: FFT
        # axs_fft.set_title("FFT: " + str(temp_title))
        # axs_fft.plot(xf, yf_abs)
        # axs_fft.set_xlim(0, 0.5)

        multiplier = 5
        MIN_multiplier = 5
        indices = yf_abs > (multiplier/100*yf_max)
        yf_clean = indices*yf
        new_f_clean = irfft(yf_clean)
        x_peaks = signal.find_peaks(np.array(new_f_clean))
        MIN_x_peaks = signal.find_peaks(np.array(-new_f_clean))

        while len(x_peaks[0]) > 50 or len(MIN_x_peaks[0]) > 50:
            indices = yf_abs > (multiplier/100*yf_max)
            yf_clean = indices*yf
            new_f_clean = irfft(yf_clean)
            x_peaks = signal.find_peaks(np.array(new_f_clean))
            MIN_x_peaks = signal.find_peaks(np.array(-new_f_clean))

            multiplier = multiplier+5

        # # GRAPH 3: IFFT
        # axs_ifft.set_title("IFFT: " + str(temp_title))
        # axs_ifft.plot(new_array_y, new_array_x)

        # if len(new_array_y) > len(new_f_clean):
        #     new_new_array_y = new_array_y
        #     new_new_array_y.pop(0)
        #     axs_ifft.plot(new_new_array_y, new_f_clean)
        # else:
        #     axs_ifft.plot(new_array_y, new_f_clean)

        # axs_ifft.set_xlim(0, 600)
        # axs_ifft.set_ylim(-20, 20)

        # Quantile values of the data
        min, q1, q2, q3, q90, max = np.quantile(new_array_x, [0, 0.25, 0.5, 0.75, 0.9, 1])
        iqr = q3-q1
        std = np.std(new_array_x)

        x_peaks = signal.find_peaks(np.array(new_f_clean))
        num_peaks = len(x_peaks[0])
        y_peaks_points = []
        x_peaks_points = []
        sum_peaks = 0

        for p in x_peaks[0]:
            x_peaks_points.append(new_f_clean[p])
            y_peaks_points.append(new_array_y[p])

        MIN_x_peaks = signal.find_peaks(np.array(-new_f_clean))
        MIN_num_peaks = len(MIN_x_peaks[0])
        MIN_y_peaks_points = []
        MIN_x_peaks_points = []

        for p in MIN_x_peaks[0]:
            MIN_x_peaks_points.append(new_f_clean[p])
            MIN_y_peaks_points.append(new_array_y[p])

        peakmin_distance_array = []
        for i in range(len(x_peaks_points)):
            if y_peaks_points[i] < MIN_y_peaks_points[i]:
                if i == 0:
                    peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i]))
                else:
                    peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i-1]))
                    peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i]))
            else:
                peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i]))
                if i < len(x_peaks_points)-1:
                    peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i+1]))

        average_peakmin_distance = np.mean(peakmin_distance_array)

        # METHOD: TRAPZ FORMULA (NUMPY)
        new_array_x_ABS = [abs(x) for x in new_f_clean]
        total_area_trapz_x = trapz(new_array_x_ABS) # Area under the curve, using numpy's trapz formula



        # # GRAPH 4: AREA GRAPH
        # axs_area.set_title("AREA UNDER CURVE: " + str(temp_title))
        # axs_area.fill_between(new_array_y, new_array_x_ABS, color="grey")
        # axs_area.plot(new_array_y, new_array_x_ABS)
        # axs_area.set_xlim(0, 600)
        # axs_area.set_ylim(0, 20)

        results_array[image_counter][Results.AREA_TRAPZ.value] = total_area_trapz_x
        results_array[image_counter][Results.MAX.value] = max
        results_array[image_counter][Results.STDDEV.value] = std

        average_areas = []
        temp_counter = 1
        max_new_array_y = np.max(new_array_y)
        percent_of_xaxis = int(5/100*max_new_array_y)
        array_iterator = 0

        for i in range(percent_of_xaxis, max_new_array_y, percent_of_xaxis):
            temp_array_x = []
            temp_array_y = []
            before_array_iterator = array_iterator

            while new_array_y[array_iterator] < i:
                temp_array_x.append(new_array_x[array_iterator])
                temp_array_y.append(new_array_y[array_iterator])
                array_iterator = array_iterator + 1

            if array_iterator == before_array_iterator:
                temp_array_x.append(new_array_x[array_iterator-1])
                temp_array_y.append(new_array_y[array_iterator-1])
                temp_array_x.append(new_array_x[array_iterator])
                temp_array_y.append(new_array_y[array_iterator])

            temp_area_x = trapz(temp_array_x, temp_array_y)
            average_areas.append(temp_area_x)
            temp_counter = temp_counter + 1

        # Quantile values of the data
        avg_avg_area = np.mean(average_areas)
        avg_min, avg_q1, avg_q2, avg_q3, avg_max = np.quantile(average_areas, [0, 0.25, 0.5, 0.75, 1])
        avg_iqr = avg_q3-avg_q1
        avg_std = np.std(average_areas)

        results_array[image_counter][Results.AVG_AREA_TRAPZ.value] = avg_avg_area
        results_array[image_counter][Results.AVG_AREA_MAX.value] = avg_max
        results_array[image_counter][Results.AVG_AREA_STDDEV.value] = avg_std

        results_array[image_counter][Results.NUM_PEAKS.value] = num_peaks
        results_array[image_counter][Results.AVG_PEAK_DIST.value] = average_peakmin_distance
    except:
        print("ERROR: \t" + str(image_patient_number[image_counter]) + "\t -\t " + 
str(image_names[image_counter]))

    # plt.tight_layout()
    # plt.show()

# SAVING THE DATA:
table_columns = []
for r in Results:
    table_columns.append(str(r.name))
df = pd.DataFrame(np.array(results_array), columns=table_columns)
print("- - - - - RESULTS ARRAY - - - - -")
display(df)
df.to_csv('Method2/Results.csv', index=False)

# END OF CODE - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

```

8D: analyseResults.py

```
# EIE Investigation: "Which Hand?"
# Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
# ELEN4012A NOVEMBER 2022

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# COPYRIGHT NOTICE:
# This code is taken from Kelvin da Silva's Master's project which involved the classification of tremors.
# This can be found at: https://github.com/kdasilva835842/tremor_classification
#
# Kelvin's code is based on the OpenCV Text Detection (EAST text detector) article by Adrian Rosebrock
(20/08/2021).
# This can be found at: https://pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/
#
# The code has been further modified, with permission from Kelvin, to suit the needs of this project.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

# IMPORTS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
import csv
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd

# IT SHOULD BE NOTED THAT "DOMINANT" AND "NON-DOMINANT" HAND REFERS TO "TREATED" AND "NON-TREATED" HANDS.
patient_number_array = []
patient_dominant_hand = []
patient_treated_hand = []
patient_time_array = []

patient_area_trapz = []
patient_avg_area_trapz = []
patient_avg_std_dev_array = []

patient_num_peaks = []
patient_avg_peak_dist = []
patient_determinator_1 = []
patient_determinator_2 = []

with open('Method2/Results.csv', 'r') as theFile:
    reader = csv.DictReader(theFile)
    for line in reader:
        patient_number_array.append(line['PATIENT_NUMBER'])
        patient_dominant_hand.append(line['DOMINANT_HAND'])
        patient_treated_hand.append(line['TREATED_HAND'])
        patient_time_array.append(line['TIME_PERIOD'])

        patient_area_trapz.append(line['AREA_TRAPZ'])
        patient_avg_area_trapz.append(line['AVG_AREA_TRAPZ'])
        patient_avg_std_dev_array.append(line['AVG_AREA_STDDEV'])

        num_peaks = line['NUM_PEAKS']
        avg_peak_dist = line['AVG_PEAK_DIST']

        patient_num_peaks.append(num_peaks)
        patient_avg_peak_dist.append(avg_peak_dist)
        patient_determinator_1.append(float(avg_peak_dist)/float(num_peaks))
        patient_determinator_2.append(float(avg_peak_dist)*float(num_peaks))

n = len(patient_number_array)

for i in range(0, n):
    for j in range(0, n-i-1):
        if patient_number_array[j] > patient_number_array[j+1]:
            temp = patient_number_array[j]
            patient_number_array[j] = patient_number_array[j+1]
            patient_number_array[j+1] = temp

            temp = patient_dominant_hand[j]
            patient_dominant_hand[j] = patient_dominant_hand[j+1]
            patient_dominant_hand[j+1] = temp

            temp = patient_treated_hand[j]
            patient_treated_hand[j] = patient_treated_hand[j+1]
            patient_treated_hand[j+1] = temp

            temp = patient_time_array[j]
            patient_time_array[j] = patient_time_array[j+1]
            patient_time_array[j+1] = temp

            temp = patient_area_trapz[j]
```

```
                patient_area_trapz[j] = patient_area_trapz[j+1]
                patient_area_trapz[j+1] = temp

                temp = patient_avg_area_trapz[j]
                patient_avg_area_trapz[j] = patient_avg_area_trapz[j+1]
                patient_avg_area_trapz[j+1] = temp

                temp = patient_avg_std_dev_array[j]
                patient_avg_std_dev_array[j] = patient_avg_std_dev_array[j+1]
                patient_avg_std_dev_array[j+1] = temp

                temp = patient_num_peaks[j]
                patient_num_peaks[j] = patient_num_peaks[j+1]
                patient_num_peaks[j+1] = temp

                temp = patient_avg_peak_dist[j]
                patient_avg_peak_dist[j] = patient_avg_peak_dist[j+1]
                patient_avg_peak_dist[j+1] = temp

                temp = patient_determinator_1[j]
                patient_determinator_1[j] = patient_determinator_1[j+1]
                patient_determinator_1[j+1] = temp

                temp = patient_determinator_2[j]
                patient_determinator_2[j] = patient_determinator_2[j+1]
                patient_determinator_2[j+1] = temp

D_patient_number_array_avg_std_dev = []
D_patient_number_array_avg_std_dev.append(1)
D_patient_counter_avg_std_dev = 0

D_time_before_avg_std_dev = []
D_time_1W_avg_std_dev = []
D_time_1M_avg_std_dev = []
D_time_3M_avg_std_dev = []
D_time_6M_avg_std_dev = []
D_time_1Y_avg_std_dev = []
D_time_2Y_avg_std_dev = []
D_time_3Y_avg_std_dev = []
D_time_4Y_avg_std_dev = []

D_time_before_avg_std_dev.append(0)
D_time_1W_avg_std_dev.append(0)
D_time_1M_avg_std_dev.append(0)
D_time_3M_avg_std_dev.append(0)
D_time_6M_avg_std_dev.append(0)
D_time_1Y_avg_std_dev.append(0)
D_time_2Y_avg_std_dev.append(0)
D_time_3Y_avg_std_dev.append(0)
D_time_4Y_avg_std_dev.append(0)

ND_patient_number_array_avg_std_dev = []
ND_patient_number_array_avg_std_dev.append(1)
ND_patient_counter_avg_std_dev = 0

ND_time_before_avg_std_dev = []
ND_time_1W_avg_std_dev = []
ND_time_1M_avg_std_dev = []
ND_time_3M_avg_std_dev = []
ND_time_6M_avg_std_dev = []
ND_time_1Y_avg_std_dev = []
ND_time_2Y_avg_std_dev = []
ND_time_3Y_avg_std_dev = []
ND_time_4Y_avg_std_dev = []

ND_time_before_avg_std_dev.append(0)
ND_time_1W_avg_std_dev.append(0)
ND_time_1M_avg_std_dev.append(0)
ND_time_3M_avg_std_dev.append(0)
ND_time_6M_avg_std_dev.append(0)
ND_time_1Y_avg_std_dev.append(0)
ND_time_2Y_avg_std_dev.append(0)
ND_time_3Y_avg_std_dev.append(0)
ND_time_4Y_avg_std_dev.append(0)

max_std_dev = 0.0
min_std_dev = 100.0





```

```python
for i in range(0, n):
    if float(patient_avg_std_dev_array[i]) > float(max_std_dev):
        max_std_dev = patient_avg_std_dev_array[i]

    if float(patient_avg_std_dev_array[i])<float(min_std_dev) and float(patient_avg_std_dev_array[i])>=float(0.0):
        min_std_dev = patient_avg_std_dev_array[i]

    if patient_treated_hand[i] == 'True':
        if int(patient_number_array[i]) != int(D_patient_number_array_avg_std_dev[D_patient_counter_avg_std_dev]):
            D_patient_number_array_avg_std_dev.append(patient_number_array[i])
            D_patient_counter_avg_std_dev = D_patient_counter_avg_std_dev + 1
            D_time_before_avg_std_dev.append(0)
            D_time_1W_avg_std_dev.append(0)
            D_time_1M_avg_std_dev.append(0)
            D_time_3M_avg_std_dev.append(0)
            D_time_6M_avg_std_dev.append(0)
            D_time_1Y_avg_std_dev.append(0)
            D_time_2Y_avg_std_dev.append(0)
            D_time_3Y_avg_std_dev.append(0)
            D_time_4Y_avg_std_dev.append(0)

        if patient_time_array[i] == "before":
            D_time_before_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "1W":
            D_time_1W_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "1M":
            D_time_1M_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "3M":
            D_time_3M_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "6M":
            D_time_6M_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "1Y":
            D_time_1Y_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "2Y":
            D_time_2Y_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "3Y":
            D_time_3Y_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "4Y":
            D_time_4Y_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]

    else:
        if int(patient_number_array[i])!=int(ND_patient_number_array_avg_std_dev[ND_patient_counter_avg_std_dev]):
            ND_patient_number_array_avg_std_dev.append(patient_number_array[i])
            ND_patient_counter_avg_std_dev = ND_patient_counter_avg_std_dev + 1
            ND_time_before_avg_std_dev.append(0)
            ND_time_1W_avg_std_dev.append(0)
            ND_time_1M_avg_std_dev.append(0)
            ND_time_3M_avg_std_dev.append(0)
            ND_time_6M_avg_std_dev.append(0)
            ND_time_1Y_avg_std_dev.append(0)
            ND_time_2Y_avg_std_dev.append(0)
            ND_time_3Y_avg_std_dev.append(0)
            ND_time_4Y_avg_std_dev.append(0)

        if patient_time_array[i] == "before":
            ND_time_before_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "1W":
            ND_time_1W_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "1M":
            ND_time_1M_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "3M":
            ND_time_3M_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "6M":
            ND_time_6M_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "1Y":
            ND_time_1Y_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "2Y":
            ND_time_2Y_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "3Y":
            ND_time_3Y_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
        elif patient_time_array[i] == "4Y":
            ND_time_4Y_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]

D_data_avg_std_dev = {'Patient': D_patient_number_array_avg_std_dev, 'Before' : D_time_before_avg_std_dev, '1
Week' : D_time_1W_avg_std_dev, '1 Month' : D_time_1M_avg_std_dev, '3 Months': D_time_3M_avg_std_dev, '6 Months':
D_time_6M_avg_std_dev, '1 Year': D_time_1Y_avg_std_dev, '2 Years': D_time_2Y_avg_std_dev, '3
Years':D_time_3Y_avg_std_dev, '4 Years':D_time_4Y_avg_std_dev }
D_df_avg_std_dev = pd.DataFrame(D_data_avg_std_dev)
D_df_avg_std_dev.to_csv('RESULTS/D_AvgStdDev.csv', index=False)

ND_data_avg_std_dev = {'Patient': ND_patient_number_array_avg_std_dev, 'Before' : ND_time_before_avg_std_dev, '1
Week' : ND_time_1W_avg_std_dev, '1 Month' : ND_time_1M_avg_std_dev, '3 Months': ND_time_3M_avg_std_dev, '6
Months': ND_time_6M_avg_std_dev, '1 Year': ND_time_1Y_avg_std_dev, '2 Years': ND_time_2Y_avg_std_dev, '3 Years':ND_time_3Y_avg_std_dev, '4 Years':ND_time_4Y_avg_std_dev }
ND_df_avg_std_dev = pd.DataFrame(ND_data_avg_std_dev)
ND_df_avg_std_dev.to_csv('RESULTS/ND_AvgStdDev.csv', index=False)

#------------------------------------

D_patient_number_array_avg_area_trapz = []
            D_patient_number_array_avg_area_trapz.append(1)
            D_patient_counter_avg_area_trapz = 0

            D_time_before_avg_area_trapz = []
            D_time_1W_avg_area_trapz = []
            D_time_1M_avg_area_trapz = []
            D_time_3M_avg_area_trapz = []
            D_time_6M_avg_area_trapz = []
            D_time_1Y_avg_area_trapz = []
            D_time_2Y_avg_area_trapz = []
            D_time_3Y_avg_area_trapz = []
            D_time_4Y_avg_area_trapz = []

            D_time_before_avg_area_trapz.append(0)
            D_time_1W_avg_area_trapz.append(0)
            D_time_1M_avg_area_trapz.append(0)
            D_time_3M_avg_area_trapz.append(0)
            D_time_6M_avg_area_trapz.append(0)
            D_time_1Y_avg_area_trapz.append(0)
            D_time_2Y_avg_area_trapz.append(0)
            D_time_3Y_avg_area_trapz.append(0)
            D_time_4Y_avg_area_trapz.append(0)

            ND_patient_number_array_avg_area_trapz = []
            ND_patient_number_array_avg_area_trapz.append(1)
            ND_patient_counter_avg_area_trapz = 0

            ND_time_before_avg_area_trapz = []
            ND_time_1W_avg_area_trapz = []
            ND_time_1M_avg_area_trapz = []
            ND_time_3M_avg_area_trapz = []
            ND_time_6M_avg_area_trapz = []
            ND_time_1Y_avg_area_trapz = []
            ND_time_2Y_avg_area_trapz = []
            ND_time_3Y_avg_area_trapz = []
            ND_time_4Y_avg_area_trapz = []

            ND_time_before_avg_area_trapz.append(0)
            ND_time_1W_avg_area_trapz.append(0)
            ND_time_1M_avg_area_trapz.append(0)
            ND_time_3M_avg_area_trapz.append(0)
            ND_time_6M_avg_area_trapz.append(0)
            ND_time_1Y_avg_area_trapz.append(0)
            ND_time_2Y_avg_area_trapz.append(0)
            ND_time_3Y_avg_area_trapz.append(0)
            ND_time_4Y_avg_area_trapz.append(0)

            max_area = 0.0
            min_area = 100.0

            for i in range(0, n):
                if float(patient_avg_area_trapz[i]) > float(max_area):
                    max_area = patient_avg_area_trapz[i]

                if float(patient_avg_area_trapz[i]) < float(min_area) and float(patient_avg_area_trapz[i]) >= float(0.0):
                    min_area = patient_avg_area_trapz[i]

                if patient_treated_hand[i] == 'True':
                    if int(patient_number_array[i]) != int(D_patient_number_array_avg_area_trapz[D_patient_counter_avg_area_trapz]):
                        D_patient_number_array_avg_area_trapz.append(patient_number_array[i])
                        D_patient_counter_avg_area_trapz = D_patient_counter_avg_area_trapz + 1
                        D_time_before_avg_area_trapz.append(0)
                        D_time_1W_avg_area_trapz.append(0)
                        D_time_1M_avg_area_trapz.append(0)
                        D_time_3M_avg_area_trapz.append(0)
                        D_time_6M_avg_area_trapz.append(0)
                        D_time_1Y_avg_area_trapz.append(0)
                        D_time_2Y_avg_area_trapz.append(0)
                        D_time_3Y_avg_area_trapz.append(0)
                        D_time_4Y_avg_area_trapz.append(0)



        if patient_time_array[i] == "before":
            D_time_before_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "1W":
            D_time_1W_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "1M":
            D_time_1M_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "3M":
            D_time_3M_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "6M":
            D_time_6M_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "1Y":
            D_time_1Y_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "2Y":
            D_time_2Y_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "3Y":
            D_time_3Y_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "4Y":
            D_time_4Y_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]

    else:
        if int(patient_number_array[i]) != int(ND_patient_number_array_avg_area_trapz[ND_patient_counter_avg_area_trapz]):
            ND_patient_number_array_avg_area_trapz.append(patient_number_array[i])
            ND_patient_counter_avg_area_trapz = ND_patient_counter_avg_area_trapz + 1
            ND_time_before_avg_area_trapz.append(0)
            ND_time_1W_avg_area_trapz.append(0)
            ND_time_1M_avg_area_trapz.append(0)
            ND_time_3M_avg_area_trapz.append(0)
            ND_time_6M_avg_area_trapz.append(0)
            ND_time_1Y_avg_area_trapz.append(0)
            ND_time_2Y_avg_area_trapz.append(0)
            ND_time_3Y_avg_area_trapz.append(0)
            ND_time_4Y_avg_area_trapz.append(0)

        if patient_time_array[i] == "before":
            ND_time_before_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "1W":
            ND_time_1W_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "1M":
            ND_time_1M_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "3M":
            ND_time_3M_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "6M":
            ND_time_6M_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "1Y":
            ND_time_1Y_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "2Y":
            ND_time_2Y_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "3Y":
            ND_time_3Y_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
        elif patient_time_array[i] == "4Y":
            ND_time_4Y_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]

D_data_avg_area_trapz = {'Patient': D_patient_number_array_avg_area_trapz, 'Before' : D_time_before_avg_area_trapz, '1 Week' : D_time_1W_avg_area_trapz, '1 Month' : D_time_1M_avg_area_trapz, '3 Months': D_time_3M_avg_area_trapz, '6 Months': D_time_6M_avg_area_trapz, '1 Year': D_time_1Y_avg_area_trapz, '2 Years': D_time_2Y_avg_area_trapz, '3 Years':D_time_3Y_avg_area_trapz, '4 Years':D_time_4Y_avg_area_trapz }
D_df_avg_area_trapz = pd.DataFrame(D_data_avg_area_trapz)
D_df_avg_area_trapz.to_csv('RESULTS/D_AreaTrapz.csv', index=False)

ND_data_avg_area_trapz = {'Patient': ND_patient_number_array_avg_area_trapz, 'Before' : ND_time_before_avg_area_trapz, '1 Week' : ND_time_1W_avg_area_trapz, '1 Month' : ND_time_1M_avg_area_trapz, '3 Months': ND_time_3M_avg_area_trapz, '6 Months': ND_time_6M_avg_area_trapz, '1 Year': ND_time_1Y_avg_area_trapz, '2 Years': ND_time_2Y_avg_area_trapz, '3 Years':ND_time_3Y_avg_area_trapz, '4 Years':ND_time_4Y_avg_area_trapz }
ND_df_avg_area_trapz = pd.DataFrame(ND_data_avg_area_trapz)
ND_df_avg_area_trapz.to_csv('RESULTS/ND_AreaTrapz.csv', index=False)

#------------------------------------------------

D_patient_number_array_det_2 = []
D_patient_number_array_det_2.append(1)
D_patient_counter_det_2 = 0

D_time_before_det_2 = []
D_time_1W_det_2 = []
D_time_1M_det_2 = []
D_time_3M_det_2 = []
D_time_6M_det_2 = []
D_time_1Y_det_2 = []
D_time_2Y_det_2 = []
D_time_3Y_det_2 = []
D_time_4Y_det_2 = []
```

```

D_time_before_det_2.append(0)
D_time_1W_det_2.append(0)
D_time_1M_det_2.append(0)
D_time_3M_det_2.append(0)
D_time_6M_det_2.append(0)
D_time_1Y_det_2.append(0)
D_time_2Y_det_2.append(0)
D_time_3Y_det_2.append(0)
D_time_4Y_det_2.append(0)

ND_patient_number_array_det_2 = []
ND_patient_number_array_det_2.append(1)
ND_patient_counter_det_2 = 0

ND_time_before_det_2 = []
ND_time_1W_det_2 = []
ND_time_1M_det_2 = []
ND_time_3M_det_2 = []
ND_time_6M_det_2 = []
ND_time_1Y_det_2 = []
ND_time_2Y_det_2 = []
ND_time_3Y_det_2 = []
ND_time_4Y_det_2 = []

ND_time_before_det_2.append(0)
ND_time_1W_det_2.append(0)
ND_time_1M_det_2.append(0)
ND_time_3M_det_2.append(0)
ND_time_6M_det_2.append(0)
ND_time_1Y_det_2.append(0)
ND_time_2Y_det_2.append(0)
ND_time_3Y_det_2.append(0)
ND_time_4Y_det_2.append(0)

D_patient_hand_array_det_2 = []
ND_patient_hand_array_det_2 = []

max_det_2 = 0.0
min_det_2 = 100.0

for i in range(0, n):
    if float(patient_determinator_2[i]) > float(max_det_2):
        max_det_2 = patient_determinator_2[i]

    if float(patient_determinator_2[i]) < float(min_det_2) and float(patient_determinator_2[i]) >= float(0.0):
        min_det_2 = patient_determinator_2[i]

    if patient_treated_hand[i] == 'True':
        if int(patient_number_array[i]) != int(D_patient_number_array_det_2[D_patient_counter_det_2]):
            D_patient_hand_array_det_2.append(patient_dominant_hand[i])
            D_patient_number_array_det_2.append(patient_number_array[i])
            D_patient_counter_det_2 = D_patient_counter_det_2 + 1
            D_time_before_det_2.append(0)
            D_time_1W_det_2.append(0)
            D_time_1M_det_2.append(0)
            D_time_3M_det_2.append(0)
            D_time_6M_det_2.append(0)
            D_time_1Y_det_2.append(0)
            D_time_2Y_det_2.append(0)
            D_time_3Y_det_2.append(0)
            D_time_4Y_det_2.append(0)

        if patient_time_array[i] == "before":
            D_time_before_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "1W":
            D_time_1W_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "1M":
            D_time_1M_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "3M":
            D_time_3M_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "6M":
            D_time_6M_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "1Y":
            D_time_1Y_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "2Y":
            D_time_2Y_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "3Y":
            D_time_3Y_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "4Y":
            D_time_4Y_det_2[D_patient_counter_det_2] = patient_determinator_2[i]

    else:
        if int(patient_number_array[i]) != int(ND_patient_number_array_det_2[ND_patient_counter_det_2]):
            ND_patient_hand_array_det_2.append(patient_dominant_hand[i])
            ND_patient_number_array_det_2.append(patient_number_array[i])
            ND_patient_counter_det_2 = ND_patient_counter_det_2 + 1
            ND_time_before_det_2.append(0)
            ND_time_1W_det_2.append(0)
            ND_time_1M_det_2.append(0)
            ND_time_3M_det_2.append(0)
            ND_time_6M_det_2.append(0)
            ND_time_1Y_det_2.append(0)
            ND_time_2Y_det_2.append(0)
            ND_time_3Y_det_2.append(0)
            ND_time_4Y_det_2.append(0)

        if patient_time_array[i] == "before":
            ND_time_before_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "1W":
            ND_time_1W_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "1M":
            ND_time_1M_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "3M":
            ND_time_3M_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "6M":
            ND_time_6M_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "1Y":
            ND_time_1Y_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "2Y":
            ND_time_2Y_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "3Y":
            ND_time_3Y_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
        elif patient_time_array[i] == "4Y":
            ND_time_4Y_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]

D_data_det_2 = {'Patient': D_patient_number_array_det_2, 'Before' : D_time_before_det_2, '1 Week' : D_time_1W_det_2, '1 Month' : D_time_1M_det_2, '3 Months': D_time_3M_det_2, '6 Months': D_time_6M_det_2, '1 Year': D_time_1Y_det_2, '2 Years': D_time_2Y_det_2, '3 Years':D_time_3Y_det_2, '4 Years':D_time_4Y_det_2 }
D_df_det_2 = pd.DataFrame(D_data_det_2)
D_df_det_2.to_csv('RESULTS/D_Determinant2.csv', index=False)

ND_data_det_2 = {'Patient': ND_patient_number_array_det_2, 'Before' : ND_time_before_det_2, '1 Week' : ND_time_1W_det_2, '1 Month' : ND_time_1M_det_2, '3 Months': ND_time_3M_det_2, '6 Months': ND_time_6M_det_2, '1 Year': ND_time_1Y_det_2, '2 Years': ND_time_2Y_det_2, '3 Years':ND_time_3Y_det_2, '4 Years':ND_time_4Y_det_2 }
ND_df_det_2 = pd.DataFrame(ND_data_det_2)
ND_df_det_2.to_csv('RESULTS/ND_Determinant2.csv', index=False)

# GRAPHS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
def checkImproved(before, arr):
    improved = 0
    both = 0
    avg_amount = 0
    avg_diff = 0
    avg_diff_improved = 0

    for i in range(len(before)):
        abs_before = abs(float(before[i]))
        abs_arr = abs(float(arr[i]))
        if (abs_before != 0) and (abs_arr != 0) and (abs_before != -1) and (abs_arr != -1):
            both += 1
            avg_diff += float(abs_before) - float(abs_arr)

            if float(abs_arr) < float(abs_before):
                improved += 1
                avg_amount += float(abs_arr)
                avg_diff_improved += float(abs_before) - float(abs_arr)

    avg_diff = avg_diff/both
    avg_diff_improved = avg_diff_improved/improved
    avg_amount = avg_amount/improved
    return both, improved, avg_amount, avg_diff, avg_diff_improved

D_both_1W, D_improved_1W, D_avg_amount_1W, D_avg_diff_1W, D_avg_diff_improved_1W = checkImproved(D_time_before_det_2, D_time_1W_det_2)
D_both_1M, D_improved_1M, D_avg_amount_1M, D_avg_diff_1M, D_avg_diff_improved_1M = checkImproved(D_time_before_det_2, D_time_1M_det_2)
D_both_3M, D_improved_3M, D_avg_amount_3M, D_avg_diff_3M, D_avg_diff_improved_3M = checkImproved(D_time_before_det_2, D_time_3M_det_2)
D_both_6M, D_improved_6M, D_avg_amount_6M, D_avg_diff_6M, D_avg_diff_improved_6M = checkImproved(D_time_before_det_2, D_time_6M_det_2)
D_both_1Y, D_improved_1Y, D_avg_amount_1Y, D_avg_diff_1Y, D_avg_diff_improved_1Y = checkImproved(D_time_before_det_2, D_time_1Y_det_2)
```

```python
D_both_2Y, D_improved_2Y, D_avg_amount_2Y, D_avg_diff_2Y, D_avg_diff_improved_2Y = checkImproved(D_time_before_det_2, D_time_2Y_det_2)
D_both_3Y, D_improved_3Y, D_avg_amount_3Y, D_avg_diff_3Y, D_avg_diff_improved_3Y = checkImproved(D_time_before_det_2, D_time_3Y_det_2)
D_both_4Y, D_improved_4Y, D_avg_amount_4Y, D_avg_diff_4Y, D_avg_diff_improved_4Y = checkImproved(D_time_before_det_2, D_time_4Y_det_2)

D_improved_percentage = [D_improved_1W/D_both_1W*100, D_improved_1M/D_both_1M*100, D_improved_3M/D_both_3M*100, D_improved_6M/D_both_6M*100, D_improved_1Y/D_both_1Y*100, D_improved_2Y/D_both_2Y*100, D_improved_3Y/D_both_3Y*100, D_improved_4Y/D_both_4Y*100]
D_avg_improved_amount = [D_avg_amount_1W, D_avg_amount_1M, D_avg_amount_3M, D_avg_amount_6M, D_avg_amount_1Y, D_avg_amount_2Y, D_avg_amount_3Y, D_avg_amount_4Y]
D_num_improved = [D_improved_1W, D_improved_1M, D_improved_3M, D_improved_6M, D_improved_1Y, D_improved_2Y, D_improved_3Y, D_improved_4Y]
D_total_num = [D_both_1W, D_both_1M, D_both_3M, D_both_6M, D_both_1Y, D_both_2Y, D_both_3Y, D_both_4Y]
D_avg_diff = [D_avg_diff_1W, D_avg_diff_1M, D_avg_diff_3M, D_avg_diff_6M, D_avg_diff_1Y, D_avg_diff_2Y, D_avg_diff_3Y, D_avg_diff_4Y]
D_avg_diff_improved = [D_avg_diff_improved_1W, D_avg_diff_improved_1M, D_avg_diff_improved_3M, D_avg_diff_improved_6M, D_avg_diff_improved_1Y, D_avg_diff_improved_2Y, D_avg_diff_improved_3Y, D_avg_diff_improved_4Y]

ND_both_1W, ND_improved_1W, ND_avg_amount_1W, ND_avg_diff_1W, ND_avg_diff_improved_1W = checkImproved(ND_time_before_det_2, ND_time_1W_det_2)
ND_both_1M, ND_improved_1M, ND_avg_amount_1M, ND_avg_diff_1M, ND_avg_diff_improved_1M = checkImproved(ND_time_before_det_2, ND_time_1M_det_2)
ND_both_3M, ND_improved_3M, ND_avg_amount_3M, ND_avg_diff_3M, ND_avg_diff_improved_3M = checkImproved(ND_time_before_det_2, ND_time_3M_det_2)
ND_both_6M, ND_improved_6M, ND_avg_amount_6M, ND_avg_diff_6M, ND_avg_diff_improved_6M = checkImproved(ND_time_before_det_2, ND_time_6M_det_2)
ND_both_1Y, ND_improved_1Y, ND_avg_amount_1Y, ND_avg_diff_1Y, ND_avg_diff_improved_1Y = checkImproved(ND_time_before_det_2, ND_time_1Y_det_2)
ND_both_2Y, ND_improved_2Y, ND_avg_amount_2Y, ND_avg_diff_2Y, ND_avg_diff_improved_2Y = checkImproved(ND_time_before_det_2, ND_time_2Y_det_2)
ND_both_3Y, ND_improved_3Y, ND_avg_amount_3Y, ND_avg_diff_3Y, ND_avg_diff_improved_3Y = checkImproved(ND_time_before_det_2, ND_time_3Y_det_2)
ND_both_4Y, ND_improved_4Y, ND_avg_amount_4Y, ND_avg_diff_4Y, ND_avg_diff_improved_4Y = checkImproved(ND_time_before_det_2, ND_time_4Y_det_2)

ND_improved_percentage = [ND_improved_1W/ND_both_1W*100, ND_improved_1M/ND_both_1M*100, ND_improved_3M/ND_both_3M*100, ND_improved_6M/ND_both_6M*100, ND_improved_1Y/ND_both_1Y*100, ND_improved_2Y/ND_both_2Y*100, ND_improved_3Y/ND_both_3Y*100, ND_improved_4Y/ND_both_4Y*100]
ND_avg_improved_amount = [ND_avg_amount_1W, ND_avg_amount_1M, ND_avg_amount_3M, ND_avg_amount_6M, ND_avg_amount_1Y, ND_avg_amount_2Y, ND_avg_amount_3Y, ND_avg_amount_4Y]
ND_num_improved = [ND_improved_1W, ND_improved_1M, ND_improved_3M, ND_improved_6M, ND_improved_1Y, ND_improved_2Y, ND_improved_3Y, ND_improved_4Y]
ND_total_num = [ND_both_1W, ND_both_1M, ND_both_3M, ND_both_6M, ND_both_1Y, ND_both_2Y, ND_both_3Y, ND_both_4Y]
ND_avg_diff = [ND_avg_diff_1W, ND_avg_diff_1M, ND_avg_diff_3M, ND_avg_diff_6M, ND_avg_diff_1Y, ND_avg_diff_2Y, ND_avg_diff_3Y, ND_avg_diff_4Y]
ND_avg_diff_improved = [ND_avg_diff_improved_1W, ND_avg_diff_improved_1M, ND_avg_diff_improved_3M, ND_avg_diff_improved_6M, ND_avg_diff_improved_1Y, ND_avg_diff_improved_2Y, ND_avg_diff_improved_3Y, ND_avg_diff_improved_4Y]

avg_total_num = [(D_both_1W+ND_both_1W)/2, (D_both_1M+ND_both_1M)/2, (D_both_3M+ND_both_3M)/2, (D_both_6M+ND_both_6M)/2, (D_both_1Y+ND_both_1Y)/2, (D_both_2Y+ND_both_2Y)/2, (D_both_3Y+ND_both_3Y)/2, (D_both_4Y+ND_both_4Y)/2]

D_both_1W_AVG_AREA, D_improved_1W_AVG_AREA, D_avg_amount_1W_AVG_AREA, D_avg_diff_1W_AVG_AREA, D_avg_diff_improved_1W_AVG_AREA = checkImproved(D_time_before_avg_area_trapz, D_time_1W_avg_area_trapz)
D_both_1M_AVG_AREA, D_improved_1M_AVG_AREA, D_avg_amount_1M_AVG_AREA, D_avg_diff_1M_AVG_AREA, D_avg_diff_improved_1M_AVG_AREA = checkImproved(D_time_before_avg_area_trapz, D_time_1M_avg_area_trapz)
D_both_3M_AVG_AREA, D_improved_3M_AVG_AREA, D_avg_amount_3M_AVG_AREA, D_avg_diff_3M_AVG_AREA, D_avg_diff_improved_3M_AVG_AREA = checkImproved(D_time_before_avg_area_trapz, D_time_3M_avg_area_trapz)
D_both_6M_AVG_AREA, D_improved_6M_AVG_AREA, D_avg_amount_6M_AVG_AREA, D_avg_diff_6M_AVG_AREA, D_avg_diff_improved_6M_AVG_AREA = checkImproved(D_time_before_avg_area_trapz, D_time_6M_avg_area_trapz)
D_both_1Y_AVG_AREA, D_improved_1Y_AVG_AREA, D_avg_amount_1Y_AVG_AREA, D_avg_diff_1Y_AVG_AREA, D_avg_diff_improved_1Y_AVG_AREA = checkImproved(D_time_before_avg_area_trapz, D_time_1Y_avg_area_trapz)
D_both_2Y_AVG_AREA, D_improved_2Y_AVG_AREA, D_avg_amount_2Y_AVG_AREA, D_avg_diff_2Y_AVG_AREA, D_avg_diff_improved_2Y_AVG_AREA = checkImproved(D_time_before_avg_area_trapz, D_time_2Y_avg_area_trapz)
D_both_3Y_AVG_AREA, D_improved_3Y_AVG_AREA, D_avg_amount_3Y_AVG_AREA, D_avg_diff_3Y_AVG_AREA, D_avg_diff_improved_3Y_AVG_AREA = checkImproved(D_time_before_avg_area_trapz, D_time_3Y_avg_area_trapz)
D_both_4Y_AVG_AREA, D_improved_4Y_AVG_AREA, D_avg_amount_4Y_AVG_AREA, D_avg_diff_4Y_AVG_AREA, D_avg_diff_improved_4Y_AVG_AREA = checkImproved(D_time_before_avg_area_trapz, D_time_4Y_avg_area_trapz)

ND_both_1W_AVG_AREA, ND_improved_1W_AVG_AREA, ND_avg_amount_1W_AVG_AREA, ND_avg_diff_1W_AVG_AREA, ND_avg_diff_improved_1W_AVG_AREA = checkImproved(ND_time_before_avg_area_trapz, ND_time_1W_avg_area_trapz)
ND_both_1M_AVG_AREA, ND_improved_1M_AVG_AREA, ND_avg_amount_1M_AVG_AREA, ND_avg_diff_1M_AVG_AREA, ND_avg_diff_improved_1M_AVG_AREA = checkImproved(ND_time_before_avg_area_trapz, ND_time_1M_avg_area_trapz)
ND_both_3M_AVG_AREA, ND_improved_3M_AVG_AREA, ND_avg_amount_3M_AVG_AREA, ND_avg_diff_3M_AVG_AREA, ND_avg_diff_improved_3M_AVG_AREA = checkImproved(ND_time_before_avg_area_trapz, ND_time_3M_avg_area_trapz)
ND_both_6M_AVG_AREA, ND_improved_6M_AVG_AREA, ND_avg_amount_6M_AVG_AREA, ND_avg_diff_6M_AVG_AREA, ND_avg_diff_improved_6M_AVG_AREA = checkImproved(ND_time_before_avg_area_trapz, ND_time_6M_avg_area_trapz)
```

```python
ND_both_1Y_AVG_AREA, ND_improved_1Y_AVG_AREA, ND_avg_amount_1Y_AVG_AREA, ND_avg_diff_1Y_AVG_AREA, ND_avg_diff_improved_1Y_AVG_AREA = checkImproved(ND_time_before_avg_area_trapz, ND_time_1Y_avg_area_trapz)
ND_both_2Y_AVG_AREA, ND_improved_2Y_AVG_AREA, ND_avg_amount_2Y_AVG_AREA, ND_avg_diff_2Y_AVG_AREA, ND_avg_diff_improved_2Y_AVG_AREA = checkImproved(ND_time_before_avg_area_trapz, ND_time_2Y_avg_area_trapz)
ND_both_3Y_AVG_AREA, ND_improved_3Y_AVG_AREA, ND_avg_amount_3Y_AVG_AREA, ND_avg_diff_3Y_AVG_AREA, ND_avg_diff_improved_3Y_AVG_AREA = checkImproved(ND_time_before_avg_area_trapz, ND_time_3Y_avg_area_trapz)
ND_both_4Y_AVG_AREA, ND_improved_4Y_AVG_AREA, ND_avg_amount_4Y_AVG_AREA, ND_avg_diff_4Y_AVG_AREA, ND_avg_diff_improved_4Y_AVG_AREA = checkImproved(ND_time_before_avg_area_trapz, ND_time_4Y_avg_area_trapz)

D_improved_percentage_AVG_AREA = [D_improved_1W_AVG_AREA/D_both_1W_AVG_AREA*100, D_improved_1M_AVG_AREA/D_both_1M_AVG_AREA*100, D_improved_3M_AVG_AREA/D_both_3M_AVG_AREA*100, D_improved_6M_AVG_AREA/D_both_6M_AVG_AREA*100, D_improved_1Y_AVG_AREA/D_both_1Y_AVG_AREA*100, D_improved_2Y_AVG_AREA/D_both_2Y_AVG_AREA*100, D_improved_3Y_AVG_AREA/D_both_3Y_AVG_AREA*100, D_improved_4Y_AVG_AREA/D_both_4Y_AVG_AREA*100]
ND_improved_percentage_AVG_AREA = [ND_improved_1W_AVG_AREA/ND_both_1W_AVG_AREA*100, ND_improved_1M_AVG_AREA/ND_both_1M_AVG_AREA*100, ND_improved_3M_AVG_AREA/ND_both_3M_AVG_AREA*100, ND_improved_6M_AVG_AREA/ND_both_6M_AVG_AREA*100, ND_improved_1Y_AVG_AREA/ND_both_1Y_AVG_AREA*100, ND_improved_2Y_AVG_AREA/ND_both_2Y_AVG_AREA*100, ND_improved_3Y_AVG_AREA/ND_both_3Y_AVG_AREA*100, ND_improved_4Y_AVG_AREA/ND_both_4Y_AVG_AREA*100]

## GENERAL GRAPH SETTINGS
plt.rcParams["font.family"] = "Times New Roman"
plt.rc('axes', axisbelow=True)
plt.rcParams.update({'font.size': 16})

## GRAPH 1: PERCENTAGE OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES
x = ['1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y']
x_axis = np.arange(len(x))
fig, ax1 = plt.subplots(figsize = (8,4))
plt.title('Percentage of Patients with Tremor Before Treatment that Improved\nAfter Various Treatment Times - PEAK DISTANCE (METHOD 2B)')
plt.grid(linestyle = '-', linewidth=0.5, axis='y')
plt.tight_layout()

ax1.bar(x_axis - 0.15, D_improved_percentage, 0.3, label = 'Treated Hand', color = 'darkblue')
ax1.bar(x_axis + 0.15, ND_improved_percentage, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
ax1.set_ylabel('Percentage Improved')
ax1.set_xlabel('Time')
ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
ax1.set_ylim(0, 100)
ax2 = ax1.twinx()
ax2.plot(x, avg_total_num, color = 'red')
ax2.set_ylabel('Number of Patients')
ax2.legend(['Number of\npatients'], loc="upper center", prop={'size': 14})
ax2.set_ylim(0, 100)
print("AVERAGE PEAK DISTANCE DOMINANT: " + str(np.average(D_improved_percentage)))
print("AVERAGE PEAK DISTANCE NON-DOMINANT: " + str(np.average(ND_improved_percentage)))
plt.savefig('RESULTS\GRAPHS\PercentageOfPatients_Det2.png', bbox_inches='tight', dpi=150)

## GRAPH 2: PERCENTAGE OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES
fig, ax1 = plt.subplots(figsize = (8,4))
plt.grid(linestyle = '-', linewidth=0.5, axis='y')
plt.title('Difference Between Tremor Severity Before Treatment\nand After Various Treatment Periods')

ax1.bar(x_axis - 0.15, D_avg_diff, 0.3, label = 'Treated Hand', color = 'darkblue')
ax1.bar(x_axis + 0.15, ND_avg_diff, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
ax1.set_ylabel('Difference in Tremor')
ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
ax1.set_xlabel('Time')

plt.xticks(x_axis, x)
plt.tight_layout()
plt.show()

## GRAPH 3: PERCENT OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES -AVG AREA
x = ['1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y']
x_axis = np.arange(len(x))
fig, ax1 = plt.subplots(figsize = (8,4))
plt.title('Percentage of Patients with Tremor Before Treatment that Improved\nAfter Various Treatment Times - AVERAGE AREA (METHOD 2A)')
plt.grid(linestyle = '-', linewidth=0.5, axis='y')
plt.tight_layout()

ax1.bar(x_axis - 0.15, D_improved_percentage_AVG_AREA, 0.3, label = 'Treated Hand', color = 'darkblue')
ax1.bar(x_axis + 0.15, ND_improved_percentage_AVG_AREA, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
ax1.set_ylabel('Percentage Improved')
ax1.set_xlabel('Time')
ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
ax1.set_ylim(0, 100)
ax2 = ax1.twinx()
ax2.plot(x, avg_total_num, color = 'red')
ax2.set_ylabel('Number of Patients')
```

```python
ax2.legend(['Number of\npatients'], loc="upper center", prop={'size': 14})
ax2.set_ylim(0, 100)
plt.rcParams["figure.figsize"] = (8,4)
print("AVERAGE AREA METHOD DOM: " + str(np.average(D_improved_percentage_AVG_AREA)))
print("AVERAGE AREA METHOD NON-DOM: " + str(np.average(ND_improved_percentage_AVG_AREA)))
plt.savefig('RESULTS\GRAPHS\PercentageOfPatients_AvgArea.png', bbox_inches='tight', dpi=150)

# START OF "WHICH HAND?" GRAPH ONE
combined_improved_array = []
which_hand = []
combined_patient_number_array = []
dominant_hand = []
longest_array = 0

if len(D_patient_number_array_det_2) > len(ND_patient_number_array_det_2):
    longest_array = len(D_patient_number_array_det_2)
else:
    longest_array = len(ND_patient_number_array_det_2)

D_counter = -1
ND_counter = -1

_before = 1
_1W = 7
_1M = 30
_3M = 91
_6M = 183
_1Y = 365
_2Y = 730
_3Y = 1095
_4Y = 1460

_before_linear = 1
_1W_linear = 2
_1M_linear = 3
_3M_linear = 4
_6M_linear = 5
_1Y_linear = 6
_2Y_linear = 7
_3Y_linear = 8
_4Y_linear = 9

fig_ALLPATIENTS, ax1_ALLPATIENTS = plt.subplots(figsize = (8,4))

for i in range(0, longest_array):
    D_counter += 1
    ND_counter += 1

    if D_counter >= len(D_patient_number_array_det_2) or ND_counter >= len(ND_patient_number_array_det_2):
        break
    else:
        while D_patient_number_array_det_2[D_counter] != ND_patient_number_array_det_2[ND_counter]:
            if D_patient_number_array_det_2[D_counter+1] == ND_patient_number_array_det_2[ND_counter]:
                D_counter += 1
            elif D_patient_number_array_det_2[D_counter] == ND_patient_number_array_det_2[ND_counter + 1]:
                ND_counter += 1
            else:
                D_counter += 1
                ND_counter += 1

    D_x_array = []
    D_x_array_linear = []
    D_y_array = []

    if D_time_before_det_2[D_counter] != 0 and D_time_before_det_2[D_counter] != -1:
        D_y_array.append(D_time_before_det_2[D_counter])
        D_x_array.append(_before)
        D_x_array_linear.append(_before_linear)
    if D_time_1W_det_2[D_counter] != 0 and D_time_1W_det_2[D_counter] != -1:
        D_y_array.append(D_time_1W_det_2[D_counter])
        D_x_array.append(_1W)
        D_x_array_linear.append(_1W_linear)
    if D_time_1M_det_2[D_counter] != 0 and D_time_1M_det_2[D_counter] != -1:
        D_y_array.append(D_time_1M_det_2[D_counter])
        D_x_array.append(_1M)
        D_x_array_linear.append(_1M_linear)
    if D_time_3M_det_2[D_counter] != 0 and D_time_3M_det_2[D_counter] != -1:
        D_y_array.append(D_time_3M_det_2[D_counter])
        D_x_array.append(_3M)
        D_x_array_linear.append(_3M_linear)
    if D_time_6M_det_2[D_counter] != 0 and D_time_6M_det_2[D_counter] != -1:
        D_y_array.append(D_time_6M_det_2[D_counter])
        D_x_array.append(_6M)
        D_x_array_linear.append(_6M_linear)
    if D_time_1Y_det_2[D_counter] != 0 and D_time_1Y_det_2[D_counter] != -1:
        D_y_array.append(D_time_1Y_det_2[D_counter])
        D_x_array.append(_1Y)
        D_x_array_linear.append(_1Y_linear)
    if D_time_2Y_det_2[D_counter] != 0 and D_time_2Y_det_2[D_counter] != -1:
        D_y_array.append(D_time_2Y_det_2[D_counter])
        D_x_array.append(_2Y)
        D_x_array_linear.append(_2Y_linear)
    if D_time_3Y_det_2[D_counter] != 0 and D_time_3Y_det_2[D_counter] != -1:
        D_y_array.append(D_time_3Y_det_2[D_counter])
        D_x_array.append(_3Y)
        D_x_array_linear.append(_3Y_linear)
    if D_time_4Y_det_2[D_counter] != 0 and D_time_4Y_det_2[D_counter] != -1:
        D_y_array.append(D_time_4Y_det_2[D_counter])
        D_x_array.append(_4Y)
        D_x_array_linear.append(_4Y_linear)

    ND_x_array = []
    ND_x_array_linear = []
    ND_y_array = []
    if ND_time_before_det_2[ND_counter] != 0 and ND_time_before_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_before_det_2[ND_counter])
        ND_x_array.append(_before)
        ND_x_array_linear.append(_before_linear)
    if ND_time_1W_det_2[ND_counter] != 0 and ND_time_1W_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_1W_det_2[ND_counter])
        ND_x_array.append(_1W)
        ND_x_array_linear.append(_1W_linear)
    if ND_time_1M_det_2[ND_counter] != 0 and ND_time_1M_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_1M_det_2[ND_counter])
        ND_x_array.append(_1M)
        ND_x_array_linear.append(_1M_linear)
    if ND_time_3M_det_2[ND_counter] != 0 and ND_time_3M_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_3M_det_2[ND_counter])
        ND_x_array.append(_3M)
        ND_x_array_linear.append(_3M_linear)
    if ND_time_6M_det_2[ND_counter] != 0 and ND_time_6M_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_6M_det_2[ND_counter])
        ND_x_array.append(_6M)
        ND_x_array_linear.append(_6M_linear)
    if ND_time_1Y_det_2[ND_counter] != 0 and ND_time_1Y_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_1Y_det_2[ND_counter])
        ND_x_array.append(_1Y)
        ND_x_array_linear.append(_1Y_linear)
    if ND_time_2Y_det_2[ND_counter] != 0 and ND_time_2Y_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_2Y_det_2[ND_counter])
        ND_x_array.append(_2Y)
        ND_x_array_linear.append(_2Y_linear)
    if ND_time_3Y_det_2[ND_counter] != 0 and ND_time_3Y_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_3Y_det_2[ND_counter])
        ND_x_array.append(_3Y)
        ND_x_array_linear.append(_3Y_linear)
    if ND_time_4Y_det_2[ND_counter] != 0 and ND_time_4Y_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_4Y_det_2[ND_counter])
        ND_x_array.append(_4Y)
        ND_x_array_linear.append(_4Y_linear)

    if len(D_x_array) > 4 and len(D_y_array) > 4 and len(ND_x_array) > 4 and len(ND_y_array) > 4:

        combined_patient_number_array.append(D_patient_number_array_det_2[D_counter])

        D_gradient, D_intercept = np.polyfit(D_x_array, D_y_array, 1)
        ND_gradient, ND_intercept = np.polyfit(ND_x_array, ND_y_array, 1)

        plt.title('Results of Patients\' Most Improved Hand')
        plt.grid(linestyle = '-', linewidth=0.5, axis='y')
        plt.xticks([1,2,3,4,5,6,7,8,9], ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])

        temp_D_y_array = np.array(D_y_array.copy())
        D_y_array_max = max(temp_D_y_array)
        temp_D_y_array = temp_D_y_array/D_y_array_max*100
        temp_ND_y_array = np.array(ND_y_array.copy())
        ND_y_array_max = max(temp_ND_y_array)
        temp_ND_y_array = temp_ND_y_array/ND_y_array_max*100

        if D_gradient < 0 and ND_gradient < 0:
            combined_improved_array.append('BOTH')
            if D_gradient < ND_gradient:
                which_hand.append('TREATED')
                ax1_ALLPATIENTS.plot(D_x_array_linear, temp_D_y_array)

            else:
                which_hand.append('NON-TREATED')
                ax1_ALLPATIENTS.plot(ND_x_array_linear, temp_ND_y_array)


        elif D_gradient < 0:
            combined_improved_array.append('TREATED')
            which_hand.append('TREATED')
            ax1_ALLPATIENTS.plot(D_x_array_linear, temp_D_y_array)

        elif ND_gradient < 0:
            combined_improved_array.append('NON-TREATED')
            which_hand.append('NON-TREATED')
            ax1_ALLPATIENTS.plot(ND_x_array_linear, temp_ND_y_array)

        else:
            combined_improved_array.append('NEITHER')
            if D_gradient < ND_gradient:
                which_hand.append('TREATED')
            else:
                which_hand.append('NON-TREATED')

plt.tight_layout()
plt.show()

# START OF "WHICH HAND?" TWO
combined_improved_array = []
which_hand = []
combined_patient_number_array = []
dominant_hand = []

longest_array = 0

if len(D_patient_number_array_det_2) > len(ND_patient_number_array_det_2):
    longest_array = len(D_patient_number_array_det_2)
else:
    longest_array = len(ND_patient_number_array_det_2)

D_counter = -1
ND_counter = -1

_before = 1
_1W = 7
_1M = 30
_3M = 91
_6M = 183
_1Y = 365
_2Y = 730
_3Y = 1095
_4Y = 1460

_before_linear = 1
_1W_linear = 2
_1M_linear = 3
_3M_linear = 4
_6M_linear = 5
_1Y_linear = 6
_2Y_linear = 7
_3Y_linear = 8
_4Y_linear = 9

fig_ALLPATIENTS2, ax1_ALLPATIENTS2 = plt.subplots(figsize = (8,4))
plt.title('Results of Patients Whose \'Before\' Drawings are the Most Severe')
plt.grid(linestyle = '-', linewidth=0.5, axis='y')
plt.xticks([1,2,3,4,5,6,7,8,9], ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])

for i in range(0, longest_array):
    D_counter += 1
    ND_counter += 1

    if D_counter >= len(D_patient_number_array_det_2) or ND_counter >= len(ND_patient_number_array_det_2):
        break
    else:
        while D_patient_number_array_det_2[D_counter] != ND_patient_number_array_det_2[ND_counter]:
            if D_patient_number_array_det_2[D_counter+1] == ND_patient_number_array_det_2[ND_counter]:
                D_counter += 1
            elif D_patient_number_array_det_2[D_counter] == ND_patient_number_array_det_2[ND_counter + 1]:
                ND_counter += 1
            else:
                D_counter += 1
                ND_counter += 1

    D_x_array = []
    D_x_array_linear = []
    D_y_array = []

    if D_time_before_det_2[D_counter] != 0 and D_time_before_det_2[D_counter] != -1:
        D_y_array.append(D_time_before_det_2[D_counter])
        D_x_array.append(_before)
        D_x_array_linear.append(_before_linear)
        if D_time_1W_det_2[D_counter] != 0 and D_time_1W_det_2[D_counter] != -1:
            D_y_array.append(D_time_1W_det_2[D_counter])
            D_x_array.append(_1W)
            D_x_array_linear.append(_1W_linear)
        if D_time_1M_det_2[D_counter] != 0 and D_time_1M_det_2[D_counter] != -1:
            D_y_array.append(D_time_1M_det_2[D_counter])
            D_x_array.append(_1M)
            D_x_array_linear.append(_1M_linear)
        if D_time_3M_det_2[D_counter] != 0 and D_time_3M_det_2[D_counter] != -1:
            D_y_array.append(D_time_3M_det_2[D_counter])
            D_x_array.append(_3M)
            D_x_array_linear.append(_3M_linear)
        if D_time_6M_det_2[D_counter] != 0 and D_time_6M_det_2[D_counter] != -1:
            D_y_array.append(D_time_6M_det_2[D_counter])
            D_x_array.append(_6M)
            D_x_array_linear.append(_6M_linear)
        if D_time_1Y_det_2[D_counter] != 0 and D_time_1Y_det_2[D_counter] != -1:
            D_y_array.append(D_time_1Y_det_2[D_counter])
            D_x_array.append(_1Y)
            D_x_array_linear.append(_1Y_linear)
        if D_time_2Y_det_2[D_counter] != 0 and D_time_2Y_det_2[D_counter] != -1:
            D_y_array.append(D_time_2Y_det_2[D_counter])
            D_x_array.append(_2Y)
            D_x_array_linear.append(_2Y_linear)
        if D_time_3Y_det_2[D_counter] != 0 and D_time_3Y_det_2[D_counter] != -1:
            D_y_array.append(D_time_3Y_det_2[D_counter])
            D_x_array.append(_3Y)
            D_x_array_linear.append(_3Y_linear)
        if D_time_4Y_det_2[D_counter] != 0 and D_time_4Y_det_2[D_counter] != -1:
            D_y_array.append(D_time_4Y_det_2[D_counter])
            D_x_array.append(_4Y)
            D_x_array_linear.append(_4Y_linear)

    ND_x_array = []
    ND_x_array_linear = []
    ND_y_array = []
    if ND_time_before_det_2[ND_counter] != 0 and ND_time_before_det_2[ND_counter] != -1:
        ND_y_array.append(ND_time_before_det_2[ND_counter])
        ND_x_array.append(_before)
        ND_x_array_linear.append(_before_linear)

        if ND_time_1W_det_2[ND_counter] != 0 and ND_time_1W_det_2[ND_counter] != -1:
            ND_y_array.append(ND_time_1W_det_2[ND_counter])
            ND_x_array.append(_1W)
            ND_x_array_linear.append(_1W_linear)
        if ND_time_1M_det_2[ND_counter] != 0 and ND_time_1M_det_2[ND_counter] != -1:
            ND_y_array.append(ND_time_1M_det_2[ND_counter])
            ND_x_array.append(_1M)
            ND_x_array_linear.append(_1M_linear)
        if ND_time_3M_det_2[ND_counter] != 0 and ND_time_3M_det_2[ND_counter] != -1:
            ND_y_array.append(ND_time_3M_det_2[ND_counter])
            ND_x_array.append(_3M)
            ND_x_array_linear.append(_3M_linear)
        if ND_time_6M_det_2[ND_counter] != 0 and ND_time_6M_det_2[ND_counter] != -1:
            ND_y_array.append(ND_time_6M_det_2[ND_counter])
            ND_x_array.append(_6M)
            ND_x_array_linear.append(_6M_linear)
        if ND_time_1Y_det_2[ND_counter] != 0 and ND_time_1Y_det_2[ND_counter] != -1:
            ND_y_array.append(ND_time_1Y_det_2[ND_counter])
            ND_x_array.append(_1Y)
            ND_x_array_linear.append(_1Y_linear)
        if ND_time_2Y_det_2[ND_counter] != 0 and ND_time_2Y_det_2[ND_counter] != -1:
            ND_y_array.append(ND_time_2Y_det_2[ND_counter])
            ND_x_array.append(_2Y)
            ND_x_array_linear.append(_2Y_linear)
        if ND_time_3Y_det_2[ND_counter] != 0 and ND_time_3Y_det_2[ND_counter] != -1:
            ND_y_array.append(ND_time_3Y_det_2[ND_counter])
            ND_x_array.append(_3Y)
            ND_x_array_linear.append(_3Y_linear)
        if ND_time_4Y_det_2[ND_counter] != 0 and ND_time_4Y_det_2[ND_counter] != -1:
            ND_y_array.append(ND_time_4Y_det_2[ND_counter])
            ND_x_array.append(_4Y)
            ND_x_array_linear.append(_4Y_linear)

    if len(D_x_array) > 4 and len(D_y_array) > 4 and len(ND_x_array) > 4 and len(ND_y_array) > 4:

        combined_patient_number_array.append(D_patient_number_array_det_2[D_counter])

        D_gradient, D_intercept = np.polyfit(D_x_array, D_y_array, 1)
        ND_gradient, ND_intercept = np.polyfit(ND_x_array, ND_y_array, 1)

        temp_D_y_array = np.array(D_y_array.copy())
        D_y_array_max = max(temp_D_y_array)
        temp_D_y_array = temp_D_y_array/D_y_array_max*100
        temp_ND_y_array = np.array(ND_y_array.copy())
        ND_y_array_max = max(temp_ND_y_array)
        temp_ND_y_array = temp_ND_y_array/ND_y_array_max*100

            if D_time_before_det_2[D_counter] < 600 and ND_time_before_det_2[ND_counter] < 600: # remove outlier
                 if D_gradient < 0 and ND_gradient < 0:
                    combined_improved_array.append('BOTH')
                    if D_gradient < ND_gradient:
                        which_hand.append('TREATED')
                        if D_y_array[0] == D_y_array_max:
                            ax1_ALLPATIENTS2.plot(D_x_array_linear, D_y_array)
                     else:
                        which_hand.append('NON-TREATED')
                        if ND_y_array[0] == ND_y_array_max:
                            ax1_ALLPATIENTS2.plot(ND_x_array_linear, ND_y_array)

                elif D_gradient < 0:
                    combined_improved_array.append('TREATED')
                    which_hand.append('TREATED')
                    if D_y_array[0] == D_y_array_max:
                        ax1_ALLPATIENTS2.plot(D_x_array_linear, D_y_array)

                elif ND_gradient < 0:
                    combined_improved_array.append('NON-TREATED')
                    which_hand.append('NON-TREATED')
                    if ND_y_array[0] == ND_y_array_max:
                        ax1_ALLPATIENTS2.plot(ND_x_array_linear, ND_y_array)

                else:
                    combined_improved_array.append('NEITHER')
                    if D_gradient < ND_gradient:
                        which_hand.append('TREATED')
                    else:
                        which_hand.append('NON-TREATED')
    plt.tight_layout()
    plt.show()

    # START OF "WHICH HAND?" THREE
    x_array = [1,2,3,4,5,6,7,8,9]
    fig_ALLPATIENTS3, ax1_ALLPATIENTS3 = plt.subplots(figsize = (8,4))
    plt.title('Average Tremor Severities for Each Hand\nPEAK DISTANCE (METHOD 2B)')
    plt.grid(linestyle = '-', linewidth=0.5, axis='y')
    plt.xticks(x_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])

    D_average_before_det_2 = np.mean(D_time_before_det_2)
    D_average_1W_det_2 = np.mean(D_time_1W_det_2)
    D_average_1M_det_2 = np.mean(D_time_1M_det_2)
    D_average_3M_det_2 = np.mean(D_time_3M_det_2)
    D_average_6M_det_2 = np.mean(D_time_6M_det_2)
    D_average_1Y_det_2 = np.mean(D_time_1Y_det_2)
    D_average_2Y_det_2 = np.mean(D_time_2Y_det_2)
    D_average_3Y_det_2 = np.mean(D_time_3Y_det_2)
    D_average_4Y_det_2 = np.mean(D_time_4Y_det_2)

    ND_average_before_det_2 = np.mean(ND_time_before_det_2)
    ND_average_1W_det_2 = np.mean(ND_time_1W_det_2)
    ND_average_1M_det_2 = np.mean(ND_time_1M_det_2)
    ND_average_3M_det_2 = np.mean(ND_time_3M_det_2)
    ND_average_6M_det_2 = np.mean(ND_time_6M_det_2)
    ND_average_1Y_det_2 = np.mean(ND_time_1Y_det_2)
    ND_average_2Y_det_2 = np.mean(ND_time_2Y_det_2)
    ND_average_3Y_det_2 = np.mean(ND_time_3Y_det_2)
    ND_average_4Y_det_2 = np.mean(ND_time_4Y_det_2)

    D_averages = [D_average_before_det_2, D_average_1W_det_2, D_average_1M_det_2, D_average_3M_det_2, D_average_6M_det_2, D_average_1Y_det_2, D_average_2Y_det_2, D_average_3Y_det_2, D_average_4Y_det_2]
    ND_averages = [ND_average_before_det_2, ND_average_1W_det_2, ND_average_1M_det_2, ND_average_3M_det_2, ND_average_6M_det_2, ND_average_1Y_det_2, ND_average_2Y_det_2, ND_average_3Y_det_2, ND_average_4Y_det_2]

    plt.legend()
    ax1_ALLPATIENTS3.plot(x_array, D_averages, color = 'darkblue')
```

```python
    ax1_ALLPATIENTS3.plot(x_array, ND_averages, color = 'cornflowerblue')
    ax1_ALLPATIENTS3.set_ylabel('Tremor Severity')
    ax1_ALLPATIENTS3.set_xlabel('Time')
    ax1_ALLPATIENTS3.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
    plt.tight_layout()
    plt.show()

    # AVERAGE TREMOR SEVERITY USING AVG AREA TRAPZ METHOD
    x_array = [1,2,3,4,5,6,7,8,9]
    fig_ALLPATIENTS3B, ax1_ALLPATIENTS3B = plt.subplots(figsize = (8,4))
    plt.title('Average Tremor Severities for Each Hand\nAVERAGE AREA (METHOD 2A)')
    plt.grid(linestyle = '-', linewidth=0.5, axis='y')
    plt.xticks(x_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])

    D_average_before_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_before_avg_area_trapz])
    D_average_1W_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_1W_avg_area_trapz])
    D_average_1M_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_1M_avg_area_trapz])
    D_average_3M_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_3M_avg_area_trapz])
    D_average_6M_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_6M_avg_area_trapz])
    D_average_1Y_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_1Y_avg_area_trapz])
    D_average_2Y_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_2Y_avg_area_trapz])
    D_average_3Y_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_3Y_avg_area_trapz])
    D_average_4Y_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_4Y_avg_area_trapz])

    ND_average_before_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_before_avg_area_trapz])
    ND_average_1W_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_1W_avg_area_trapz])
    ND_average_1M_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_1M_avg_area_trapz])
    ND_average_3M_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_3M_avg_area_trapz])
    ND_average_6M_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_6M_avg_area_trapz])
    ND_average_1Y_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_1Y_avg_area_trapz])
    ND_average_2Y_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_2Y_avg_area_trapz])
    ND_average_3Y_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_3Y_avg_area_trapz])
    ND_average_4Y_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_4Y_avg_area_trapz])

    D_averages = [D_average_before_avg_area_trapz, D_average_1W_avg_area_trapz, D_average_1M_avg_area_trapz,
D_average_3M_avg_area_trapz, D_average_6M_avg_area_trapz, D_average_1Y_avg_area_trapz,
D_average_2Y_avg_area_trapz, D_average_3Y_avg_area_trapz, D_average_4Y_avg_area_trapz]
    ND_averages = [ND_average_before_avg_area_trapz, ND_average_1W_avg_area_trapz, ND_average_1M_avg_area_trapz,
ND_average_3M_avg_area_trapz, ND_average_6M_avg_area_trapz, ND_average_1Y_avg_area_trapz,
ND_average_2Y_avg_area_trapz, ND_average_3Y_avg_area_trapz, ND_average_4Y_avg_area_trapz]

    plt.legend()
    ax1_ALLPATIENTS3B.plot(x_array, D_averages, color = 'darkblue')
    ax1_ALLPATIENTS3B.plot(x_array, ND_averages, color = 'cornflowerblue')
    ax1_ALLPATIENTS3B.set_ylabel('Tremor Severity')
    ax1_ALLPATIENTS3B.set_xlabel('Time')
    ax1_ALLPATIENTS3B.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
    plt.savefig('RESULTS\GRAPHS\AverageTremSev_AvgArea.png', bbox_inches='tight', dpi=150)

    plt.tight_layout()
    plt.show()

    # START OF "WHICH HAND?" FOUR

    x_array = [1,2,3,4,5,6,7,8,9]
    fig_ALLPATIENTS4, ax1_ALLPATIENTS4 = plt.subplots(figsize = (8,4))
    plt.title('Normalised Average Tremor Severities for Each Hand\nPEAK DISTANCE (METHOD 2B)')
    plt.grid(linestyle = '-', linewidth=0.5, axis='y')
    plt.xticks(x_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])

    all_arrays = [*D_time_before_det_2, *D_time_1W_det_2, *D_time_1M_det_2, *D_time_3M_det_2, *D_time_6M_det_2,
*D_time_1Y_det_2, *D_time_2Y_det_2, *D_time_3Y_det_2, *D_time_4Y_det_2, *ND_time_before_det_2, *ND_time_1W_det_2,
*ND_time_1M_det_2, *ND_time_3M_det_2, *ND_time_6M_det_2, *ND_time_1Y_det_2, *ND_time_2Y_det_2, *ND_time_3Y_det_2,
*ND_time_4Y_det_2]
    all_min, q10, q90, q95, q98, q999, all_max = np.quantile(all_arrays, [0, 0.1, 0.9, 0.95, 0.98, 0.99, 1])
    denom = q95 - all_min

    D_time_before_normalised = [(x - all_min)/denom for x in D_time_before_det_2]
    D_time_1W_normalised = [(x - all_min)/denom for x in D_time_1W_det_2]
    D_time_1M_normalised = [(x - all_min)/denom for x in D_time_1M_det_2]
    D_time_3M_normalised = [(x - all_min)/denom for x in D_time_3M_det_2]
    D_time_6M_normalised = [(x - all_min)/denom for x in D_time_6M_det_2]
    D_time_1Y_normalised = [(x - all_min)/denom for x in D_time_1Y_det_2]
    D_time_2Y_normalised = [(x - all_min)/denom for x in D_time_2Y_det_2]
    D_time_3Y_normalised = [(x - all_min)/denom for x in D_time_3Y_det_2]
    D_time_4Y_normalised = [(x - all_min)/denom for x in D_time_4Y_det_2]

    D_times_normalised = [D_time_before_normalised, D_time_1W_normalised, D_time_1M_normalised,
D_time_3M_normalised, D_time_6M_normalised,  D_time_1Y_normalised, D_time_2Y_normalised, D_time_3Y_normalised,
D_time_4Y_normalised]

    ND_time_before_normalised = [(x - all_min)/denom for x in ND_time_before_det_2]
    ND_time_1W_normalised = [(x - all_min)/denom for x in ND_time_1W_det_2]
    ND_time_1M_normalised = [(x - all_min)/denom for x in ND_time_1M_det_2]
    ND_time_3M_normalised = [(x - all_min)/denom for x in ND_time_3M_det_2]
    ND_time_6M_normalised = [(x - all_min)/denom for x in ND_time_6M_det_2]
    ND_time_1Y_normalised = [(x - all_min)/denom for x in ND_time_1Y_det_2]
    ND_time_2Y_normalised = [(x - all_min)/denom for x in ND_time_2Y_det_2]
    ND_time_3Y_normalised = [(x - all_min)/denom for x in ND_time_3Y_det_2]
    ND_time_4Y_normalised = [(x - all_min)/denom for x in ND_time_4Y_det_2]

    D_average_before_det_2 = np.mean(D_time_before_normalised)
    D_average_1W_det_2 = np.mean(D_time_1W_normalised)
    D_average_1M_det_2 = np.mean(D_time_1M_normalised)
    D_average_3M_det_2 = np.mean(D_time_3M_normalised)
    D_average_6M_det_2 = np.mean(D_time_6M_normalised)
    D_average_1Y_det_2 = np.mean(D_time_1Y_normalised)
    D_average_2Y_det_2 = np.mean(D_time_2Y_normalised)
    D_average_3Y_det_2 = np.mean(D_time_3Y_normalised)
    D_average_4Y_det_2 = np.mean(D_time_4Y_normalised)

    ND_average_before_det_2 = np.mean(ND_time_before_normalised)
    ND_average_1W_det_2 = np.mean(ND_time_1W_normalised)
    ND_average_1M_det_2 = np.mean(ND_time_1M_normalised)
    ND_average_3M_det_2 = np.mean(ND_time_3M_normalised)
    ND_average_6M_det_2 = np.mean(ND_time_6M_normalised)
    ND_average_1Y_det_2 = np.mean(ND_time_1Y_normalised)
    ND_average_2Y_det_2 = np.mean(ND_time_2Y_normalised)
    ND_average_3Y_det_2 = np.mean(ND_time_3Y_normalised)
    ND_average_4Y_det_2 = np.mean(ND_time_4Y_normalised)

    D_averages = [D_average_before_det_2, D_average_1W_det_2, D_average_1M_det_2, D_average_3M_det_2,
D_average_6M_det_2, D_average_1Y_det_2, D_average_2Y_det_2, D_average_3Y_det_2, D_average_4Y_det_2]
    ND_averages = [ND_average_before_det_2, ND_average_1W_det_2, ND_average_1M_det_2, ND_average_3M_det_2,
ND_average_6M_det_2, ND_average_1Y_det_2, ND_average_2Y_det_2, ND_average_3Y_det_2, ND_average_4Y_det_2]

    plt.legend()
    ax1_ALLPATIENTS4.plot(x_array, D_averages, color = 'darkblue')
    ax1_ALLPATIENTS4.plot(x_array, ND_averages, color = 'cornflowerblue')
    ax1_ALLPATIENTS4.set_ylabel('Tremor Severity')
    ax1_ALLPATIENTS4.set_xlabel('Time')

    ax1_ALLPATIENTS4.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
    plt.savefig('RESULTS\GRAPHS\AverageTremSev_Det2.png', bbox_inches='tight', dpi=150)
    plt.show()

    # START OF "WHICH HAND?" FOUR B
    x_array = [1,2,3,4,5,6,7,8,9]
    fig_ALLPATIENTS4B, ax1_ALLPATIENTS4B = plt.subplots(figsize = (8,4))
    plt.title('Normalised Average Tremor Severities for Each Hand\nAVERAGE AREA (METHOD 2A)')
    plt.grid(linestyle = '-', linewidth=0.5, axis='y')
    plt.xticks(x_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])


    copy_D_time_before_avg_area_trapz = [abs(float(x)) for x in D_time_before_avg_area_trapz]
    for c in copy_D_time_before_avg_area_trapz:
        if c == 0.0:
            copy_D_time_before_avg_area_trapz.remove(c)
    copy_D_time_1W_avg_area_trapz = [abs(float(x)) for x in D_time_1W_avg_area_trapz]
    for c in copy_D_time_1W_avg_area_trapz:
        if c == 0.0:
            copy_D_time_1W_avg_area_trapz.remove(c)
    copy_D_time_1M_avg_area_trapz = [abs(float(x)) for x in D_time_1M_avg_area_trapz]
    for c in copy_D_time_1M_avg_area_trapz:
        if c == 0.0:
            copy_D_time_1M_avg_area_trapz.remove(c)
    copy_D_time_3M_avg_area_trapz = [abs(float(x)) for x in D_time_3M_avg_area_trapz]
    for c in copy_D_time_3M_avg_area_trapz:
        if c == 0.0:
            copy_D_time_3M_avg_area_trapz.remove(c)
    copy_D_time_6M_avg_area_trapz = [abs(float(x)) for x in D_time_6M_avg_area_trapz]
    for c in copy_D_time_6M_avg_area_trapz:
        if c == 0.0:
            copy_D_time_6M_avg_area_trapz.remove(c)
    copy_D_time_1Y_avg_area_trapz = [abs(float(x)) for x in D_time_1Y_avg_area_trapz]
    for c in copy_D_time_1Y_avg_area_trapz:
        if c == 0.0:
            copy_D_time_1Y_avg_area_trapz.remove(c)
    copy_D_time_2Y_avg_area_trapz = [abs(float(x)) for x in D_time_2Y_avg_area_trapz]
    for c in copy_D_time_2Y_avg_area_trapz:
        if c == 0.0:
            copy_D_time_2Y_avg_area_trapz.remove(c)
    copy_D_time_3Y_avg_area_trapz = [abs(float(x)) for x in D_time_3Y_avg_area_trapz]
    for c in copy_D_time_3Y_avg_area_trapz:
        if c == 0.0:
            copy_D_time_3Y_avg_area_trapz.remove(c)
    copy_D_time_4Y_avg_area_trapz = [abs(float(x)) for x in D_time_4Y_avg_area_trapz]
    for c in copy_D_time_4Y_avg_area_trapz:
        if c == 0.0:
            copy_D_time_4Y_avg_area_trapz.remove(c)

    copy_ND_time_before_avg_area_trapz = [abs(float(x)) for x in ND_time_before_avg_area_trapz]
    for c in copy_ND_time_before_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_before_avg_area_trapz.remove(c)
    copy_ND_time_1W_avg_area_trapz = [abs(float(x)) for x in ND_time_1W_avg_area_trapz]
    for c in copy_ND_time_1W_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_1W_avg_area_trapz.remove(c)
    copy_ND_time_1M_avg_area_trapz = [abs(float(x)) for x in ND_time_1M_avg_area_trapz]
    for c in copy_ND_time_1M_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_1M_avg_area_trapz.remove(c)
    copy_ND_time_3M_avg_area_trapz = [abs(float(x)) for x in ND_time_3M_avg_area_trapz]
    for c in copy_ND_time_3M_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_3M_avg_area_trapz.remove(c)
    copy_ND_time_6M_avg_area_trapz = [abs(float(x)) for x in ND_time_6M_avg_area_trapz]
    for c in copy_ND_time_6M_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_6M_avg_area_trapz.remove(c)
    copy_ND_time_1Y_avg_area_trapz = [abs(float(x)) for x in ND_time_1Y_avg_area_trapz]
    for c in copy_ND_time_1Y_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_1Y_avg_area_trapz.remove(c)
    copy_ND_time_2Y_avg_area_trapz = [abs(float(x)) for x in ND_time_2Y_avg_area_trapz]
    for c in copy_ND_time_2Y_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_2Y_avg_area_trapz.remove(c)
    copy_ND_time_3Y_avg_area_trapz = [abs(float(x)) for x in ND_time_3Y_avg_area_trapz]
    for c in copy_ND_time_3Y_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_3Y_avg_area_trapz.remove(c)
    copy_ND_time_4Y_avg_area_trapz = [abs(float(x)) for x in ND_time_4Y_avg_area_trapz]
    for c in copy_ND_time_4Y_avg_area_trapz:
        if c == 0.0:
            copy_ND_time_4Y_avg_area_trapz.remove(c)

    print(copy_D_time_before_avg_area_trapz)
    print(copy_ND_time_before_avg_area_trapz)
    m1 = max(copy_D_time_before_avg_area_trapz)
    m2 = max(copy_D_time_1W_avg_area_trapz)
    m3 = max(copy_D_time_1M_avg_area_trapz)
    m4 = max(copy_D_time_3M_avg_area_trapz)
    m5 = max(copy_D_time_6M_avg_area_trapz)
    m6 = max(copy_D_time_1Y_avg_area_trapz )
    m7 = max(copy_D_time_2Y_avg_area_trapz )
    m8 = max(copy_D_time_3Y_avg_area_trapz )
    m9 = max(copy_D_time_4Y_avg_area_trapz )

    m10 = max(copy_ND_time_before_avg_area_trapz)
    m11 = max(copy_ND_time_1W_avg_area_trapz)
    m12 = max(copy_ND_time_1M_avg_area_trapz)
    m13 = max(copy_ND_time_3M_avg_area_trapz)
    m14 = max(copy_ND_time_6M_avg_area_trapz)
    m15 = max(copy_ND_time_1Y_avg_area_trapz)
    m16 = max(copy_ND_time_2Y_avg_area_trapz)
    m17 = max(copy_ND_time_3Y_avg_area_trapz)
    m18 = max(copy_ND_time_4Y_avg_area_trapz)

    m1 = min(copy_D_time_before_avg_area_trapz)
    m2 = min(copy_D_time_1W_avg_area_trapz)
    m3 = min(copy_D_time_1M_avg_area_trapz)
    m4 = min(copy_D_time_3M_avg_area_trapz)
    m5 = min(copy_D_time_6M_avg_area_trapz)
    m6 = min(copy_D_time_1Y_avg_area_trapz )
    m7 = min(copy_D_time_2Y_avg_area_trapz )
    m8 = min(copy_D_time_3Y_avg_area_trapz )
    m9 = min(copy_D_time_4Y_avg_area_trapz )

    m10 = min(copy_ND_time_before_avg_area_trapz)
    m11 = min(copy_ND_time_1W_avg_area_trapz)
    m12 = min(copy_ND_time_1M_avg_area_trapz)
    m13 = min(copy_ND_time_3M_avg_area_trapz)
    m14 = min(copy_ND_time_6M_avg_area_trapz)
    m15 = min(copy_ND_time_1Y_avg_area_trapz)
```

```python
    m16 = min(copy_ND_time_2Y_avg_area_trapz)
    m17 = min(copy_ND_time_3Y_avg_area_trapz)
    m18 = min(copy_ND_time_4Y_avg_area_trapz)

    actual_max = max(m1, m2, m3, m4, m5, m6, m7, m8, m9, m11, m12, m13, m14, m15, m16, m17, m18)
    all_min = min(m1, m2, m3, m4, m5, m6, m7, m8, m9, m11, m12, m13, m14, m15, m16, m17, m18)

    # NB: BUG WARNING HERE - WHY IS MAX 0?
    denom = 50 # actual_max - all_min

    D_time_before_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_before_avg_area_trapz]]
    D_time_1W_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_1W_avg_area_trapz]]
    D_time_1M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_1M_avg_area_trapz]]
    D_time_3M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_3M_avg_area_trapz]]
    D_time_6M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_6M_avg_area_trapz]]
    D_time_1Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_1Y_avg_area_trapz]]
    D_time_2Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_2Y_avg_area_trapz]]
    D_time_3Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_3Y_avg_area_trapz]]
    D_time_4Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_4Y_avg_area_trapz]]

    D_times_normalised = [D_time_before_normalised, D_time_1W_normalised, D_time_1M_normalised, D_time_3M_normalised, D_time_6M_normalised,  D_time_1Y_normalised, D_time_2Y_normalised, D_time_3Y_normalised, D_time_4Y_normalised]

    ND_time_before_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_before_avg_area_trapz]]
    ND_time_1W_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_1W_avg_area_trapz]]
    ND_time_1M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_1M_avg_area_trapz]]
    ND_time_3M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_3M_avg_area_trapz]]
    ND_time_6M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_6M_avg_area_trapz]]
    ND_time_1Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_1Y_avg_area_trapz]]
    ND_time_2Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_2Y_avg_area_trapz]]
    ND_time_3Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_3Y_avg_area_trapz]]
    ND_time_4Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_ND_time_4Y_avg_area_trapz]]

    D_average_before_det_2 = np.mean(D_time_before_normalised)
    D_average_1W_det_2 = np.mean(D_time_1W_normalised)
    D_average_1M_det_2 = np.mean(D_time_1M_normalised)
    D_average_3M_det_2 = np.mean(D_time_3M_normalised)
    D_average_6M_det_2 = np.mean(D_time_6M_normalised)
    D_average_1Y_det_2 = np.mean(D_time_1Y_normalised)
    D_average_2Y_det_2 = np.mean(D_time_2Y_normalised)
    D_average_3Y_det_2 = np.mean(D_time_3Y_normalised)
    D_average_4Y_det_2 = np.mean(D_time_4Y_normalised)

    ND_average_before_det_2 = np.mean(ND_time_before_normalised)
    ND_average_1W_det_2 = np.mean(ND_time_1W_normalised)
    ND_average_1M_det_2 = np.mean(ND_time_1M_normalised)
    ND_average_3M_det_2 = np.mean(ND_time_3M_normalised)
    ND_average_6M_det_2 = np.mean(ND_time_6M_normalised)
    ND_average_1Y_det_2 = np.mean(ND_time_1Y_normalised)
    ND_average_2Y_det_2 = np.mean(ND_time_2Y_normalised)
    ND_average_3Y_det_2 = np.mean(ND_time_3Y_normalised)
    ND_average_4Y_det_2 = np.mean(ND_time_4Y_normalised)

    D_averages = [D_average_before_det_2, D_average_1W_det_2, D_average_1M_det_2, D_average_3M_det_2, D_average_6M_det_2, D_average_1Y_det_2, D_average_2Y_det_2, D_average_3Y_det_2, D_average_4Y_det_2]
    ND_averages = [ND_average_before_det_2, ND_average_1W_det_2, ND_average_1M_det_2, ND_average_3M_det_2, ND_average_6M_det_2, ND_average_1Y_det_2, ND_average_2Y_det_2, ND_average_3Y_det_2, ND_average_4Y_det_2]

    plt.legend()
    ax1_ALLPATIENTS4B.plot(x_array, D_averages, color = 'darkblue')
    ax1_ALLPATIENTS4B.plot(x_array, ND_averages, color = 'cornflowerblue')
    ax1_ALLPATIENTS4B.set_ylabel('Tremor Severity')
    ax1_ALLPATIENTS4B.set_xlabel('Time')
    ax1_ALLPATIENTS4B.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
    plt.savefig('RESULTS\GRAPHS\AverageTremSev_AvgArea.png', bbox_inches='tight', dpi=150)
    plt.show()

    # END OF CODE - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
```