

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*

This code is used to accurately extract and save the spirals and line-blocks from each patient's scanned PDF of the filled in template as JPEG images.

8B. *extractRectangle.py*

This code is used to further extract and save the top-most line drawing from each of the line-blocks as a JPEG image.

8C. *method2.py*

This code is used to analyse the JPEG line drawing image and convert it into a useable Python function to calculate various values including average area and number of peaks and save them all in a 2D Python list for further analysis.

8D. *analyseResults.py*

This code takes the list mentioned above and compares, computes, and graphically demonstrates various values, combinations of values as well as combinations between various patients, etc. All to answer the investigation question and provide results in order to determine whether method 2 is of any use.

8A: isolateSpirals.py

```
1. # EIE Investigation: "Which Hand?"
2. # Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
3. # ELEN4012A NOVEMBER 2022
4.
5. # - - - - - #
6. # COPYRIGHT NOTICE:
7. # This code is taken from Kelvin da Silva's Master's project which involved the classification of tremors.
8. # This can be found at: https://github.com/kdasilva835842/tremor\_classification
9. #
10. # Kelvin's code is based on the OpenCV Text Detection (EAST text detector) article by Adrian Rosebrock
    (20/08/2021).
11. # This can be found at: https://pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/
12. #
13. # The code has been further modified, with permission from Kelvin, to suit the needs of this project.
14. # - - - - - #
15.
16. # IMPORTS - - - - - #
17. import numpy as np
18. import cv2
19. import imutils
20. from imutils.object_detection import non_max_suppression
21. from pdf2image import convert_from_path
22. import glob
23. import os
24. import tempfile
25.
26. # Read in the PDFs and convert to .jpg- - - - - #
27. image_array = []
28. image_names = []
29. image_path = []
30.
31. counter = 0
32. rawCount = 0
33. sizeErrorCountA = 0
34. sizeErrorCountB = 0
35. sizeErrorCountC = 0
36.
37. with tempfile.TemporaryDirectory() as tempDir:
38.     print('created temporary directory', tempDir)
39.     for outer_foldername in glob.glob('Data/Original/*'):
40.         for foldername in glob.glob(outer_foldername + '/*'):
41.             newFolderName = foldername.replace("Original", "Cropped")
42.             os.makedirs(newFolderName, exist_ok = True)
43.             os.makedirs(newFolderName+'/DrawingA', exist_ok = True)
44.             os.makedirs(newFolderName+'/DrawingB', exist_ok = True)
45.             os.makedirs(newFolderName+'/DrawingC', exist_ok = True)
46.             for filename in glob.glob(foldername + '/*.pdf'):
47.                 arrayName = os.path.basename(filename)
48.                 arrayName = arrayName.replace(".pdf", "")
49.                 newName = filename.replace(".pdf", "")
50.                 newName = str(tempDir) + "/" + str(arrayName)
51.                 newPath = foldername.replace("Original", "Cropped")
52.                 image_path.append(newPath)
53.
54.                 def convertPDFtoJGP(originalImage,finalImage,dpi=200):
55.                     pages = convert_from_path(originalImage, dpi)
56.                     for page in pages:
57.                         page.save(finalImage, 'JPEG')
58.
59.                 convertPDFtoJGP(filename,newName+'.jpg',150)
60.                 image = cv2.imread(newName+'.jpg')
61.                 image_array.append(image)
62.                 image_names.append(arrayName)
63.                 rawCount = rawCount + 1
64.
65. cropToleranceA = 0.9
66. cropToleranceB = 0.95
67. cropToleranceC = 0.9
68.
69. MidPlaneFractionX = 0.4
70. MidPlaneFractionY = 0.5
71.
72. # Desired dimensions of Drawings A and B (square) or Drawing C (rectangle)
73. squareDimension = 300
74. rectangleDimension = 600
75.
76. # Array used to save the various widths of Drawing A, to better calculate the average width
77. array_width = []
78. array_width.append(475)
```

```

79. # Arrays used to save the various x and y coordinates, to better calculate the average coordinates
80. array_xStartA = []
81. array_xStartA.append(140)
82. array_xStartB = []
83. array_xStartB.append(710)
84. array_xStartC = []
85. array_xStartC.append(140)
86. array_yEndA = []
87. array_yEndA.append(490)
88. array_yEndB = []
89. array_yEndB.append(490)
90. array_yEndC = []
91. array_yEndC.append(1032)
92.
93. for counter,image in enumerate(image_array):
94.     print("- - - - -")
95.     print("Starting image ", image_names[counter])
96.     copied = image.copy()
97.
98.     orientedImage = copied.copy()
99.     finalA = orientedImage.copy()
100.    finalB = orientedImage.copy()
101.    finalC = orientedImage.copy()
102.    image = orientedImage.copy()
103.
104.    (total_h,total_w) = image.shape[:2]
105.    mask = np.zeros((total_h,total_w), np.uint8)
106.    cv2.rectangle (mask, (0, int(0.245*total_h)), (total_w, int(total_h*0.8)), 255, -1)
107.
108.    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
109.    image = cv2.bitwise_and(image,mask)
110.    kernel = np.ones((3,3),np.uint8)
111.    image = cv2.erode(image,kernel,iterations = 1)
112.    image = cv2.cvtColor(image, cv2.COLOR_BAYER_BG2BGR)
113.
114.    # Important: The EAST text requires that your input image dimensions be multiples of 32
115.    newW = 704
116.    newH = 704
117.    minConf = 0.8
118.    (H,W) = image.shape[:2]
119.    rW = W / float(newW)
120.    rH = H / float(newH)
121.    image = cv2.resize(image, (newW, newH))
122.    (H, W) = image.shape[:2]
123.
124.    # 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
125.    layerNames = [
126.        "feature_fusion/Conv_7/Sigmoid",
127.        "feature_fusion/concat_3"]
128.
129.    # Load the pre-trained EAST text detector
130.    net = cv2.dnn.readNet("Cropping/frozen_east_text_detection.pb")
131.
132.    # Construct a blob from image and then perform a forward pass of the model to obtain the two output layer sets
133.    blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
134.        (123.68, 116.78, 103.94), swapRB=True, crop=False)
135.
136.    net.setInput(blob)
137.    (scores, geometry) = net.forward(layerNames)
138.
139.    # 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
140.    (numRows, numCols) = scores.shape[2:4]
141.    rects = []
142.    confidences = []
143.
144.    for y in range(0, numRows):
145.        # Extract scores (probabilities), followed by geometrical data to derive bounding box coordinates
146.        scoresData = scores[0, 0, y]
147.        xData0 = geometry[0, 0, y]
148.        xData1 = geometry[0, 1, y]
149.        xData2 = geometry[0, 2, y]
150.        xData3 = geometry[0, 3, y]
151.        anglesData = geometry[0, 4, y]
152.
153.    # Loop over the number of columns
154.    for x in range(0, numCols):
155.        # If score does not have sufficient probability, ignore it
156.        if scoresData[x] < minConf:
157.            Continue
158.        # Compute the offset factor as resulting feature maps will be 4x smaller than the input image

```

```

159.         (offsetX, offsetY) = (x * 4.0, y * 4.0)
160.
161.         # Extract the rotation angle for the prediction and then compute the sin and cosine
162.         angle = anglesData[x]
163.         cos = np.cos(angle)
164.         sin = np.sin(angle)
165.
166.         # Use the geometry volume to derive the width and height of the bounding box
167.         h = xData0[x] + xData2[x]
168.         w = xData1[x] + xData3[x]
169.
170.         # Compute both the starting and ending (x, y)-coordinates for the text prediction bounding box
171.         endX = int(offsetX + (cos * xData1[x]) + (sin * xData2[x]))
172.         endY = int(offsetY - (sin * xData1[x]) + (cos * xData2[x]))
173.         startX = int(endX - w)
174.         startY = int(endY - h)
175.
176.         # Add the bounding box coordinates and probability score to respective lists
177.         rects.append((startX, startY, endX, endY))
178.         confidences.append(scoresData[x])
179.
180.     # Apply non-maxima suppression to suppress weak, overlapping bounding boxes
181.     boxes = non_max_suppression(np.array(rects), probs=confidences)
182.
183.     xStart = []
184.     xEnd = []
185.     yStart = []
186.     yEnd = []
187.
188.     # Loop over the bounding boxes
189.     for (startX, startY, endX, endY) in boxes:
190.         # Scale the bounding box coordinates based on the respective ratios
191.         startX = int(startX * rW)
192.         startY = int(startY * rH)
193.         endX = int(endX * rW)
194.         endY = int(endY * rH)
195.
196.         # Draw the bounding box on the image
197.         xStart.append(startX) # vector of x coords
198.         yStart.append(startY)
199.         xEnd.append(endX)
200.         yEnd.append(endY)
201.
202.     (height,width) = finalA.shape[:2]
203.     orientation = 0
204.
205.     # Set array position of the Drawing's to be -1 (i.e. not found)
206.     A_position = -1
207.     B_position = -1
208.     C_position = -1
209.
210.     # If the coordinates match where a drawing should be, save that position to the corresponding drawing's
211.     # X_position variable
212.     for i in range(0, len(xStart)):
213.         if xStart[i] < 400:
214.             if yEnd[i] < total_h/2:
215.                 if A_position == -1:
216.                     A_position = i
217.             else:
218.                 if xEnd[i] - xStart[i] < 100:
219.                     A_position = i
220.             elif yEnd[i] < 1300 and xEnd[i] < 500:
221.                 C_position = i
222.             elif yStart[i] < total_h/2:
223.                 if B_position == -1:
224.                     B_position = i
225.             else:
226.                 if xEnd[i] - xStart[i] < 100:
227.                     B_position = i
228.
229.     # In case a drawing is not found, use the average values instead
230.     width = np.mean(array_width)
231.     xStart_A = np.mean(array_xStartA)
232.     xStart_B = np.mean(array_xStartB)
233.     xStart_C = np.mean(array_xStartC)
234.     yEnd_A = np.mean(array_yEndA)
235.     yEnd_B = np.mean(array_yEndB)
236.     yEnd_C = np.mean(array_yEndC)
237.
238.
239.     # Check if the drawing was found, and save the coordinates respectively

```

```

240. if A_position != -1:
241.     xStart_A = xStart[A_position]
242.     yEnd_A = yEnd[A_position]
243.     array_xStartA.append(xStart_A)
244.     array_yEndA.append(yEnd_A)
245. else:
246.     print("THERE IS NO A TEXT DETECTED - USING AVERAGE VALUES")
247.
248. if B_position != -1:
249.     xStart_B = xStart[B_position]
250.     yEnd_B = yEnd[B_position]
251.     array_xStartB.append(xStart_B)
252.     array_yEndB.append(yEnd_B)
253. else:
254.     print("THERE IS NO B TEXT DETECTED - USING AVERAGE VALUES")
255.
256. if C_position != -1:
257.     xStart_C = xStart[C_position]
258.     yEnd_C = yEnd[C_position]
259.     array_xStartC.append(xStart_C)
260.     array_yEndC.append(yEnd_C)
261. else:
262.     print("THERE IS NO C TEXT DETECTED - USING AVERAGE VALUES")
263.
264. # Using other detected Drawings to find better coordinates for missing Drawings
265. if A_position == -1 and B_position != -1:
266.     xStart_A = xStart_B-total_w/2
267.     yEnd_A = yEnd[B_position]
268.
269. if B_position == -1 and A_position != -1:
270.     xStart_B = xStart_A*0.5 + total_w/2
271.     yEnd_B = yEnd[A_position]
272.
273. if A_position == -1 and B_position == -1:
274.     xStart_A = xStart_C
275.     xStart_B = xStart_A*0.5 + total_w/2
276.
277.     yEnd_A = yStart[C_position] - total_h/3
278.     yEnd_B = yStart[C_position] - total_h/3
279.
280. # Calculate the width of Drawing A, either with average or coordinate values
281. width = (xStart_B*cropToleranceA-xStart_A)
282. array_width.append(width)
283.
284. # CROPPING OUT SPIRAL A - - - - - #
285. try:
286.     finalA = cv2.cvtColor(finalA, cv2.COLOR_BGR2GRAY)
287.     finalA = finalA[int(yEnd_A):int(yEnd_A+width), int(xStart_A):int(xStart_A + width)]
288.     finalA = imutils.resize(finalA, width=squareDimension)
289.
290.     new_image_path = str(image_path[counter])+'/DrawingA/'+str(image_names[counter])+"_A".jpg"
291.     cv2.imwrite(new_image_path, finalA)
292. except Exception as e:
293.     print("Unable to save to file A as image size is empty: ",str(e))
294.
295. # CROPPING OUT SPIRAL B - - - - - #
296. try:
297.     finalB = cv2.cvtColor(finalB, cv2.COLOR_BGR2GRAY)
298.     finalB = finalB[ int(yEnd_B) : int((yEnd_B+width)), int(xStart_B*cropToleranceB):
int((xStart_B+width)*cropToleranceB)]
299.     finalB = imutils.resize(finalB, width=squareDimension)
300.
301.     new_image_path = str(image_path[counter])+'/DrawingB/'+str(image_names[counter])+"_B".jpg"
302.     cv2.imwrite(new_image_path, finalB)
303. except Exception as e:
304.     print("Unable to save to file B as image size is empty: ",str(e))
305.
306. # CROPPING SPIRAL C - - - - - #
307. try:
308.     finalC = cv2.cvtColor(finalC, cv2.COLOR_BGR2GRAY)
309.     finalC = finalC[ int(yEnd_C) : int(yEnd_C+0.75*width), int(xStart_C): int(xStart_B+width)]
310.     finalC = imutils.resize(finalC, width=rectangleDimension)
311.
312.     new_image_path = str(image_path[counter])+'/DrawingC/'+str(image_names[counter])+"_C".jpg"
313.     cv2.imwrite(new_image_path, finalC)
314. except Exception as e:
315.     print("Unable to save to file C as image size is empty: ",str(e))
316. print("Finished image ", image_names[counter])
317.
318. # END OF CODE - - - - - #

```

8B: extractRectangle.py

```
1. # EIE Investigation: "Which Hand?"
2. # Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
3. # ELEN4012A NOVEMBER 2022
4.
5. # - - - - - #
6. # COPYRIGHT NOTICE:
7. # This code contains snippets from Adrian Rosebrock's article "OpenCV shape detection" (08/02/2016)
8. # Which can be found at: https://pyimagesearch.com/2016/02/08/opencv-shape-detection/
9. #
10. # This code also contains snippets from jdhao's article "Cropping Rotated Rectangles from Image with OpenCV"
    (23/02/2019)
11. # Which can be found at: https://jdhao.github.io/2019/02/23/crop_rotated_rectangle_opencv/
12. # This code is under the license: CC BY-NC-ND 4.0 (https://creativecommons.org/licenses/by-nc-nd/4.0/)
13. # It has further been modified and combined to suit the needs of this project, but will not be further
    distributed.
14. # - - - - - #
15.
16. # IMPORTS - - - - - #
17. import imutils
18. import cv2
19. import glob
20. import os
21. from PIL import Image
22. import numpy as np
23.
24. # Read in the .jpgs and save into an image array- - - - - #
25. image_array = []
26. image_names = []
27. image_path = []
28.
29. for outer_foldername in glob.glob('Data\Cropped*'):
30.     for foldername in glob.glob(outer_foldername + '*'):
31.         for filename in glob.glob(foldername + "\DrawingC*.jpg"):
32.             im = Image.open(filename).convert('L') # convert image to grayscale
33.             res = im.point((lambda p: 256 if p>=200 else 0)) # convert each pixel into either black or white
34.             res.save(filename)
35.
36.             arrayName = os.path.basename(filename)
37.             arrayName = arrayName.replace(".jpg","")
38.             newPath = foldername + '\DrawingC\Rectangles'
39.             os.makedirs(newPath, exist_ok = True)
40.             image_path.append(newPath)
41.             image = cv2.imread(filename)
42.             image = cv2.bitwise_not(image) # invert the colors of the image
43.             image_array.append(image)
44.             image_names.append(arrayName)
45.
46. # Loop through the image array and extract the top-most hand drawn rectangle- - - - - #
47. for image_counter,images in enumerate(image_array):
48.     image = images.copy()
49.     ratio = image.shape[0] / float(image.shape[1])
50.     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
51.     blurred = cv2.GaussianBlur(gray, (5, 5), 0)
52.     thresh = cv2.threshold(blurred, 200, 255, cv2.THRESH_BINARY)[1]
53.
54.     contours = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
55.     contours = imutils.grab_contours(contours)
56.     coordinate_1 = []
57.     coordinate_2 = []
58.     coordinate_3 = []
59.     coordinate_4 = []
60.
61.     for c in contours:
62.         if cv2.contourArea(c) > 50:
63.             try:
64.                 x1, y1, w, h = cv2.boundingRect(c)
65.                 if w > 350:
66.                     M = cv2.moments(c)
67.                     cX = int((M["m10"] / M["m00"]) * ratio)
68.                     cY = int((M["m01"] / M["m00"]) * ratio)
69.
70.                     c = c.astype("float")
71.                     c *= ratio # multiply the contour (x, y)-coordinates by the resize ratio
72.                     c = c.astype("int")
73.
74.                     rect = cv2.minAreaRect(c)
75.                     box = cv2.boxPoints(rect)
76.                     box = np.int0(box)
77.
```

```

78.         b_counter = 0
79.         b_left_1 = -1
80.         b_left_2 = -1
81.         b_right_1 = -1
82.         b_right_2 = -1
83.
84.         # Sort the boxes so that the coordinates are in the correct order
85.         for b in box:
86.             if b[0] < 100:
87.                 if b_left_1 == -1:
88.                     b_left_1 = b_counter
89.                 else:
90.                     b_left_2 = b_counter
91.             else:
92.                 if b_right_1 == -1:
93.                     b_right_1 = b_counter
94.                 else:
95.                     b_right_2 = b_counter
96.             b_counter = b_counter + 1
97.
98.             if box[b_left_1][1] < box[b_left_2][1]:
99.                 coordinate_1.append(box[b_left_1])
100.                coordinate_4.append(box[b_left_2])
101.            else:
102.                coordinate_1.append(box[b_left_2])
103.                coordinate_4.append(box[b_left_1])
104.
105.            if box[b_right_1][1] < box[b_right_2][1]:
106.                coordinate_2.append(box[b_right_1])
107.                coordinate_3.append(box[b_right_2])
108.            else:
109.                coordinate_2.append(box[b_right_2])
110.                coordinate_3.append(box[b_right_1])
111.        except:
112.            print("ERROR")
113.    try:
114.        counter = 0
115.        number_boxes = len(coordinate_1)
116.
117.        # Since we detected the black surrounding boxes, we can now work out the coordinates of inner drawing
118.        new_coordinate_1 = coordinate_4[number_boxes-1]
119.        new_coordinate_2 = coordinate_3[number_boxes-1]
120.        new_coordinate_3 = coordinate_2[number_boxes-2]
121.        new_coordinate_4 = coordinate_1[number_boxes-2]
122.        new_coordinate_array = np.array([
123.            [new_coordinate_1],
124.            [new_coordinate_2],
125.            [new_coordinate_3],
126.            [new_coordinate_4]
127.        ])
128.        rect = cv2.minAreaRect(new_coordinate_array)
129.        new_box = cv2.boxPoints(rect)
130.        new_box = np.int0(new_box)
131.
132.        # get width and height of the detected rectangle
133.        new_width = int(rect[1][0])
134.        new_height = int(rect[1][1])
135.        new_box_float = new_box.astype("float32")
136.
137.        # coordinate of the points in box points after the rectangle has been straightened
138.        straightened_array = np.array([[0, new_height-1], [0, 0], [new_width-1, 0],
139.            [new_width-1, new_height-1]], dtype="float32")
140.
141.        M = cv2.getPerspectiveTransform(new_box_float, straightened_array) # the perspective transformation matrix
142.
143.        # directly warp the rotated rectangle to get the straightened rectangle
144.        warped = cv2.warpPerspective(image.copy(), M, (new_width, new_height))
145.
146.        (warped_H,warped_W) = warped.shape[:2]
147.        if warped_H > warped_W: warped = cv2.rotate(warped, cv2.ROTATE_90_CLOCKWISE)
148.
149.        warped = cv2.bitwise_not(imutils.resize(warped, width=600))
150.        (warped_H,warped_W) = warped.shape[:2]
151.        warped = warped[5 : warped_H - 5, 5: warped_W-5]
152.
153.        cv2.imwrite(str(image_path[image_counter]) + "/" + str(image_names[image_counter]) + "_RECT.jpg", warped)
154.        cv2.waitKey(0)
155.    except:
156.        print("ERROR")
157.
158.    # END OF CODE - - - - -

```

8C: method2.py

```
1. # EIE Investigation: "Which Hand?"
2. # Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
3. # ELEN4012A NOVEMBER 2022
4.
5. # IMPORTS - - - - - #
6. import cv2
7. import glob
8. import os
9. import numpy as np
10. import matplotlib.pyplot as plt
11. from PIL import Image
12. from numpy import trapz
13. from enum import Enum
14. import pandas as pd
15. from IPython.display import display
16. from scipy import signal
17. from scipy.fft import rfft, rfftfreq, irfft
18.
19. class Results(Enum):
20.     FILENAME = 0
21.     PATIENT_NUMBER = 1
22.     DOMINANT_HAND = 2
23.     TREATED_HAND = 3
24.     TIME_PERIOD = 4
25.     PD_HAND = 5
26.     AREA_TRAPZ = 6
27.     MAX = 7
28.     STDDEV = 8
29.     AVG_AREA_TRAPZ = 9
30.     AVG_AREA_MAX = 10
31.     AVG_AREA_STDDEV = 11
32.     NUM_PEAKS = 12
33.     AVG_PEAK_DIST = 13
34.
35. # Read in the .jpgs and save into an image array, ensuring grayscale- - - - - #
36. image_array = []
37. image_names = []
38. image_patient_number = []
39. image_time_frame = []
40. image_dominant_hand = []
41. image_treated_hand = []
42. image_PD_hand = []
43. file_count = 0
44.
45. def get_time_period(image_name):
46.     image_name = str.lower(image_name)
47.
48.     if image_name.find("w") != -1:
49.         period = image_name[0:image_name.find("w")]
50.         time = [int(s) for s in period.split() if s.isdigit()]
51.
52.         if period.find("one") != -1:
53.             output = "1W"
54.         else:
55.             output = str(time[0]) + "W"
56.         return output
57.
58.     if image_name.find("m") != -1:
59.         period = image_name[0:image_name.find("m")]
60.         time = [int(s) for s in period.split() if s.isdigit()]
61.
62.         if period.find("one") != -1:
63.             output = "1M"
64.             return output
65.         else:
66.             try:
67.                 output = str(time[0]) + "M"
68.                 return output
69.             except:
70.                 pass # Presentation purposes, delete later
71.                 # print("ERROR - m but no time")
72.
73.     if image_name.find("y") != -1:
74.         period = image_name[0:image_name.find("y")]
75.         time = [int(s) for s in period.split() if s.isdigit()]
76.
77.         if period.find("one") != -1:
78.             output = "1Y"
79.             return output
```



```

80.         else:
81.             try:
82.                 output = str(time[0]) + "Y"
83.                 return output
84.             except:
85.                 print("ERROR - y but no time - " + str(image_name))
86.
87.         if image_name.find("before") != -1:
88.             return "before"
89.         return -1
90.
91. def is_treated_hand(image_name, patient_number):
92.     image_name = str.lower(image_name)
93.
94.     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:
95.         left_position = image_name.find("lt")
96.         if left_position == -1:
97.             left_position = image_name.find("left")
98.             if left_position == -1:
99.                 return False
100.            else:
101.                return True
102.        else:
103.            return True
104.    else:
105.        right_position = image_name.find("rt")
106.        if right_position == -1:
107.            right_position = image_name.find("right")
108.            if right_position == -1:
109.                return False
110.            else:
111.                return True
112.        else:
113.            return True
114.
115. def is_dominant_hand(image_name, patient_number):
116.     image_name = str.lower(image_name)
117.
118.     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:
119.         left_position = image_name.find("lt")
120.         if left_position == -1:
121.             left_position = image_name.find("left")
122.             if left_position == -1:
123.                 return False
124.            else:
125.                return True
126.        else:
127.            return True
128.    else:
129.        right_position = image_name.find("rt")
130.        if right_position == -1:
131.            right_position = image_name.find("right")
132.            if right_position == -1:
133.                return False
134.            else:
135.                return True
136.        else:
137.            return True
138.
139. def is_PD_hand(patient_number):
140.     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:
141.         return True
142.    else:
143.        return False
144.
145. for outer_foldername in glob.glob('Data\Cropped*'):
146.     for foldername in glob.glob(outer_foldername + '\*'):
147.         patient_number = os.path.basename(foldername)
148.         patient_number = patient_number.replace("#", "")

```

```

149.
150.     for filename in glob.glob(foldername + "\\DrawingC\\Rectangles\\*.jpg"):
151.         arrayName = os.path.basename(filename)
152.         arrayName = arrayName.replace(".jpg", "")
153.
154.         time_period = get_time_period(arrayName)
155.         is_dominant = is_dominant_hand(arrayName, patient_number)
156.         is_treated = is_treated_hand(arrayName, patient_number)
157.         is_PD = is_PD_hand(patient_number)
158.         if time_period != -1:
159.             im = Image.open(filename).convert('L') # convert image to grayscale
160.             res = im.point((lambda p: 256 if p>=200 else 0)) # convert each pixel into either black or white
161.             res.save(filename)
162.
163.             image = cv2.imread(filename)
164.             image_array.append(image)
165.             image_names.append(arrayName)
166.             image_patient_number.append(patient_number)
167.             image_time_frame.append(time_period)
168.             image_dominant_hand.append(is_dominant)
169.             image_treated_hand.append(is_treated)
170.             image_PD_hand.append(is_PD)
171.             file_count = file_count + 1
172.
173. results_array = [[-1 for x in range(len(Results))] for y in range(file_count)]
174.
175. temp_counter_patient_5 = -1
176. # Loop through the image array to perform image processing- - - - - #
177. for image_counter, image in enumerate(image_array):
178.     temp_counter_patient_5 += 1
179.     temp_title = image_names[image_counter].replace("(5)_C_RECT", "")
180.
181.     results_array[image_counter][Results.FILENAME.value] = image_names[image_counter]
182.     results_array[image_counter][Results.PATIENT_NUMBER.value] = image_patient_number[image_counter]
183.     results_array[image_counter][Results.TIME_PERIOD.value] = image_time_frame[image_counter]
184.     results_array[image_counter][Results.DOMINANT_HAND.value] = image_dominant_hand[image_counter]
185.     results_array[image_counter][Results.TREATED_HAND.value] = image_treated_hand[image_counter]
186.     results_array[image_counter][Results.PD_HAND.value] = image_PD_hand[image_counter]
187.
188.     coordinates = np.argwhere(image < 0.9)
189.     # fig, (axs_original, axs_fft, axs_ifft, axs_area) = plt.subplots(4)
190.
191.     try:
192.         # NOTE: The horizontal axis is denoted by y, and the vertical axis is denoted by x (ACCIDENTAL MISTAKE)
193.         x_tuple, y_tuple, z_tuple = zip(*coordinates)
194.
195.         x = np.array(x_tuple).tolist()
196.         y = np.array(y_tuple).tolist()
197.         z = np.array(z_tuple).tolist()
198.
199.         # Sort the array of pixels into an ordered array according to the horizontal x-axis
200.         for i in range(0, len(y)-1):
201.             for j in range(0, len(y)-i-1):
202.                 if y[j] > y[j+1]:
203.                     temp_x = x[j]
204.                     x[j] = x[j+1]
205.                     x[j+1] = temp_x
206.
207.                     temp_y = y[j]
208.                     y[j] = y[j+1]
209.                     y[j+1] = temp_y
210.
211.         # GRAPH 1: ORIGINAL
212.         # axs_original.set_title("ORIGINAL GRAPH: " + str(temp_title))
213.         # axs_original.plot(y, x)
214.
215.         average_x = np.mean(x)
216.         new_array_x = []
217.         new_array_y = []
218.
219.         # Take the absolute value and shift the points down to be centered around the horizontal axis
220.         for i in range(0, len(x)):
221.             new_array_y.append(y[i])
222.             new_array_x.append(x[i]-average_x)
223.
224.         data_step = 0.1
225.         n = len(new_array_y)
226.         yf = rfft(new_array_x)
227.         xf = rfftfreq(n, data_step)
228.
229.         yf_abs = np.abs(yf)
230.         yf_max = np.max(yf_abs)

```

```

231.
232.     ## GRAPH 2: FFT
233.     # axs_fft.set_title("FFT: " + str(temp_title))
234.     # axs_fft.plot(xf, yf_abs)
235.     # axs_fft.set_xlim(0, 0.5)
236.
237.     multiplier = 5
238.     MIN_multiplier = 5
239.     indices = yf_abs > (multiplier/100*yf_max)
240.     yf_clean = indices*yf
241.     new_f_clean = irfft(yf_clean)
242.     x_peaks = signal.find_peaks(np.array(new_f_clean))
243.     MIN_x_peaks = signal.find_peaks(np.array(-new_f_clean))
244.
245.     while len(x_peaks[0]) > 50 or len(MIN_x_peaks[0]) > 50:
246.         indices = yf_abs > (multiplier/100*yf_max)
247.         yf_clean = indices*yf
248.         new_f_clean = irfft(yf_clean)
249.         x_peaks = signal.find_peaks(np.array(new_f_clean))
250.         MIN_x_peaks = signal.find_peaks(np.array(-new_f_clean))
251.
252.         multiplier = multiplier+5
253.
254.     ## GRAPH 3: IFFT
255.     # axs_ifft.set_title("IFFT: " + str(temp_title))
256.     # axs_ifft.plot(new_array_y, new_array_x)
257.
258.     # if len(new_array_y) > len(new_f_clean):
259.     #     new_new_array_y = new_array_y
260.     #     new_new_array_y.pop(0)
261.     #     axs_ifft.plot(new_new_array_y, new_f_clean)
262.     # else:
263.     #     axs_ifft.plot(new_array_y, new_f_clean)
264.
265.     # axs_ifft.set_xlim(0, 600)
266.     # axs_ifft.set_ylim(-20, 20)
267.
268.     # Quantile values of the data
269.     min, q1, q2, q3, q90, max = np.quantile(new_array_x, [0, 0.25, 0.5, 0.75, 0.9, 1])
270.     iqr = q3-q1
271.     std = np.std(new_array_x)
272.
273.     x_peaks = signal.find_peaks(np.array(new_f_clean))
274.     num_peaks = len(x_peaks[0])
275.     y_peaks_points = []
276.     x_peaks_points = []
277.     sum_peaks = 0
278.
279.     for p in x_peaks[0]:
280.         x_peaks_points.append(new_f_clean[p])
281.         y_peaks_points.append(new_array_y[p])
282.
283.     MIN_x_peaks = signal.find_peaks(np.array(-new_f_clean))
284.     MIN_num_peaks = len(MIN_x_peaks[0])
285.     MIN_y_peaks_points = []
286.     MIN_x_peaks_points = []
287.
288.     for p in MIN_x_peaks[0]:
289.         MIN_x_peaks_points.append(new_f_clean[p])
290.         MIN_y_peaks_points.append(new_array_y[p])
291.
292.     peakmin_distance_array = []
293.     for i in range(len(x_peaks_points)):
294.         if y_peaks_points[i] < MIN_y_peaks_points[i]:
295.             if i == 0:
296.                 peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i]))
297.             else:
298.                 peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i-1]))
299.                 peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i]))
300.         else:
301.             peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i]))
302.             if i < len(x_peaks_points)-1:
303.                 peakmin_distance_array.append(abs(x_peaks_points[i] - MIN_x_peaks_points[i+1]))
304.
305.     average_peakmin_distance = np.mean(peakmin_distance_array)
306.
307.     # METHOD: TRAPZ FORMULA (NUMPY)
308.     new_array_x_ABS = [abs(x) for x in new_f_clean]
309.     total_area_trapz_x = trapz(new_array_x_ABS) # Area under the curve, using numpy's trapz formula
310.
311.
312.

```

```

313.     ## GRAPH 4: AREA GRAPH
314.     # axs_area.set_title("AREA UNDER CURVE: " + str(temp_title))
315.     # axs_area.fill_between(new_array_y, new_array_x_ABS, color="grey")
316.     # axs_area.plot(new_array_y, new_array_x_ABS)
317.     # axs_area.set_xlim(0, 600)
318.     # axs_area.set_ylim(0, 20)
319.
320.     results_array[image_counter][Results.AREA_TRAPZ.value] = total_area_trapz_x
321.     results_array[image_counter][Results.MAX.value] = max
322.     results_array[image_counter][Results.STDDEV.value] = std
323.
324.     average_areas = []
325.     temp_counter = 1
326.     max_new_array_y = np.max(new_array_y)
327.     percent_of_xaxis = int(5/100*max_new_array_y)
328.     array_iterator = 0
329.
330.     for i in range(percent_of_xaxis, max_new_array_y, percent_of_xaxis):
331.         temp_array_x = []
332.         temp_array_y = []
333.         before_array_iterator = array_iterator
334.
335.         while new_array_y[array_iterator] < i:
336.             temp_array_x.append(new_array_x[array_iterator])
337.             temp_array_y.append(new_array_y[array_iterator])
338.             array_iterator = array_iterator + 1
339.
340.         if array_iterator == before_array_iterator:
341.             temp_array_x.append(new_array_x[array_iterator-1])
342.             temp_array_y.append(new_array_y[array_iterator-1])
343.             temp_array_x.append(new_array_x[array_iterator])
344.             temp_array_y.append(new_array_y[array_iterator])
345.
346.             temp_area_x = trapz(temp_array_x, temp_array_y)
347.             average_areas.append(temp_area_x)
348.             temp_counter = temp_counter + 1
349.
350.     # Quantile values of the data
351.     avg_avg_area = np.mean(average_areas)
352.     avg_min, avg_q1, avg_q2, avg_q3, avg_max = np.quantile(average_areas, [0, 0.25, 0.5, 0.75, 1])
353.     avg_iqr = avg_q3-avg_q1
354.     avg_std = np.std(average_areas)
355.
356.     results_array[image_counter][Results.AVG_AREA_TRAPZ.value] = avg_avg_area
357.     results_array[image_counter][Results.AVG_AREA_MAX.value] = avg_max
358.     results_array[image_counter][Results.AVG_AREA_STDDEV.value] = avg_std
359.
360.     results_array[image_counter][Results.NUM_PEAKS.value] = num_peaks
361.     results_array[image_counter][Results.AVG_PEAK_DIST.value] = average_peakmin_distance
362. except:
363.     print("ERROR: \t" + str(image_patient_number[image_counter]) + "\t -\t " +
364.           str(image_names[image_counter]))
365.
366.     # plt.tight_layout()
367.     # plt.show()
368.
369. # SAVING THE DATA:
370. table_columns = []
371. for r in Results:
372.     table_columns.append(str(r.name))
373. df = pd.DataFrame(np.array(results_array), columns=table_columns)
374. print("- - - - RESULTS ARRAY - - - -")
375. display(df)
376. df.to_csv('Method2/Results.csv', index=False)
377. # END OF CODE - - - - - #
378.

```

8D: analyseResults.py

```
1. # EIE Investigation: "Which Hand?"
2. # Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
3. # ELEN4012A NOVEMBER 2022
4.
5. # - - - - - #
6. # COPYRIGHT NOTICE:
7. # This code is taken from Kelvin da Silva's Master's project which involved the classification of tremors.
8. # This can be found at: https://github.com/kdasilva835842/tremor\_classification
9. #
10. # Kelvin's code is based on the OpenCV Text Detection (EAST text detector) article by Adrian Rosebrock
    (20/08/2021).
11. # This can be found at: https://pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/
12. #
13. # The code has been further modified, with permission from Kelvin, to suit the needs of this project.
14. # - - - - - #
15.
16. # IMPORTS - - - - - #
17. import csv
18. from matplotlib import pyplot as plt
19. import numpy as np
20. import pandas as pd
21.
22. # IT SHOULD BE NOTED THAT "DOMINANT" AND "NON-DOMINANT" HAND REFERS TO "TREATED" AND "NON-TREATED" HANDS.
23. patient_number_array = []
24. patient_dominant_hand = []
25. patient_treated_hand = []
26. patient_time_array = []
27.
28. patient_area_trapz = []
29. patient_avg_area_trapz = []
30. patient_avg_std_dev_array = []
31.
32. patient_num_peaks = []
33. patient_avg_peak_dist = []
34. patient_determinator_1 = []
35. patient_determinator_2 = []
36.
37. with open('Method2/Results.csv', 'r') as theFile:
38.     reader = csv.DictReader(theFile)
39.     for line in reader:
40.         patient_number_array.append(line['PATIENT_NUMBER'])
41.         patient_dominant_hand.append(line['DOMINANT_HAND'])
42.         patient_treated_hand.append(line['TREATED_HAND'])
43.         patient_time_array.append(line['TIME_PERIOD'])
44.
45.         patient_area_trapz.append(line['AREA_TRAPZ'])
46.         patient_avg_area_trapz.append(line['AVG_AREA_TRAPZ'])
47.         patient_avg_std_dev_array.append(line['AVG_AREA_STDDEV'])
48.
49.         num_peaks = line['NUM_PEAKS']
50.         avg_peak_dist = line['AVG_PEAK_DIST']
51.
52.         patient_num_peaks.append(num_peaks)
53.         patient_avg_peak_dist.append(avg_peak_dist)
54.         patient_determinator_1.append(float(avg_peak_dist)/float(num_peaks))
55.         patient_determinator_2.append(float(avg_peak_dist)*float(num_peaks))
56.
57. n = len(patient_number_array)
58.
59. for i in range(0, n):
60.     for j in range(0, n-i-1):
61.         if patient_number_array[j] > patient_number_array[j+1]:
62.             temp = patient_number_array[j]
63.             patient_number_array[j] = patient_number_array[j+1]
64.             patient_number_array[j+1] = temp
65.
66.             temp = patient_dominant_hand[j]
67.             patient_dominant_hand[j] = patient_dominant_hand[j+1]
68.             patient_dominant_hand[j+1] = temp
69.
70.             temp = patient_treated_hand[j]
71.             patient_treated_hand[j] = patient_treated_hand[j+1]
72.             patient_treated_hand[j+1] = temp
73.
74.             temp = patient_time_array[j]
75.             patient_time_array[j] = patient_time_array[j+1]
76.             patient_time_array[j+1] = temp
77.
78.             temp = patient_area_trapz[j]
```

```

79.         patient_area_trapz[j] = patient_area_trapz[j+1]
80.         patient_area_trapz[j+1] = temp
81.
82.         temp = patient_avg_area_trapz[j]
83.         patient_avg_area_trapz[j] = patient_avg_area_trapz[j+1]
84.         patient_avg_area_trapz[j+1] = temp
85.
86.         temp = patient_avg_std_dev_array[j]
87.         patient_avg_std_dev_array[j] = patient_avg_std_dev_array[j+1]
88.         patient_avg_std_dev_array[j+1] = temp
89.
90.         temp = patient_num_peaks[j]
91.         patient_num_peaks[j] = patient_num_peaks[j+1]
92.         patient_num_peaks[j+1] = temp
93.
94.         temp = patient_avg_peak_dist[j]
95.         patient_avg_peak_dist[j] = patient_avg_peak_dist[j+1]
96.         patient_avg_peak_dist[j+1] = temp
97.
98.         temp = patient_determinator_1[j]
99.         patient_determinator_1[j] = patient_determinator_1[j+1]
100.        patient_determinator_1[j+1] = temp
101.
102.        temp = patient_determinator_2[j]
103.        patient_determinator_2[j] = patient_determinator_2[j+1]
104.        patient_determinator_2[j+1] = temp
105.
106.D_patient_number_array_avg_std_dev = []
107.D_patient_number_array_avg_std_dev.append(1)
108.D_patient_counter_avg_std_dev = 0
109.
110.D_time_before_avg_std_dev = []
111.D_time_1W_avg_std_dev = []
112.D_time_1M_avg_std_dev = []
113.D_time_3M_avg_std_dev = []
114.D_time_6M_avg_std_dev = []
115.D_time_1Y_avg_std_dev = []
116.D_time_2Y_avg_std_dev = []
117.D_time_3Y_avg_std_dev = []
118.D_time_4Y_avg_std_dev = []
119.
120.D_time_before_avg_std_dev.append(0)
121.D_time_1W_avg_std_dev.append(0)
122.D_time_1M_avg_std_dev.append(0)
123.D_time_3M_avg_std_dev.append(0)
124.D_time_6M_avg_std_dev.append(0)
125.D_time_1Y_avg_std_dev.append(0)
126.D_time_2Y_avg_std_dev.append(0)
127.D_time_3Y_avg_std_dev.append(0)
128.D_time_4Y_avg_std_dev.append(0)
129.
130.ND_patient_number_array_avg_std_dev = []
131.ND_patient_number_array_avg_std_dev.append(1)
132.ND_patient_counter_avg_std_dev = 0
133.
134.ND_time_before_avg_std_dev = []
135.ND_time_1W_avg_std_dev = []
136.ND_time_1M_avg_std_dev = []
137.ND_time_3M_avg_std_dev = []
138.ND_time_6M_avg_std_dev = []
139.ND_time_1Y_avg_std_dev = []
140.ND_time_2Y_avg_std_dev = []
141.ND_time_3Y_avg_std_dev = []
142.ND_time_4Y_avg_std_dev = []
143.
144.ND_time_before_avg_std_dev.append(0)
145.ND_time_1W_avg_std_dev.append(0)
146.ND_time_1M_avg_std_dev.append(0)
147.ND_time_3M_avg_std_dev.append(0)
148.ND_time_6M_avg_std_dev.append(0)
149.ND_time_1Y_avg_std_dev.append(0)
150.ND_time_2Y_avg_std_dev.append(0)
151.ND_time_3Y_avg_std_dev.append(0)
152.ND_time_4Y_avg_std_dev.append(0)
153.
154.max_std_dev = 0.0
155.min_std_dev = 100.0
156.
157.
158.
159.
160.

```

```

161. for i in range(0, n):
162.     if float(patient_avg_std_dev_array[i]) > float(max_std_dev):
163.         max_std_dev = patient_avg_std_dev_array[i]
164.
165.     if float(patient_avg_std_dev_array[i]) < float(min_std_dev) and float(patient_avg_std_dev_array[i]) >= float(0.0):
166.         min_std_dev = patient_avg_std_dev_array[i]
167.
168.     if patient_treated_hand[i] == 'True':
169.         if int(patient_number_array[i]) != int(D_patient_number_array_avg_std_dev[D_patient_counter_avg_std_dev]):
170.             D_patient_number_array_avg_std_dev.append(patient_number_array[i])
171.             D_patient_counter_avg_std_dev = D_patient_counter_avg_std_dev + 1
172.             D_time_before_avg_std_dev.append(0)
173.             D_time_1W_avg_std_dev.append(0)
174.             D_time_1M_avg_std_dev.append(0)
175.             D_time_3M_avg_std_dev.append(0)
176.             D_time_6M_avg_std_dev.append(0)
177.             D_time_1Y_avg_std_dev.append(0)
178.             D_time_2Y_avg_std_dev.append(0)
179.             D_time_3Y_avg_std_dev.append(0)
180.             D_time_4Y_avg_std_dev.append(0)
181.
182.             if patient_time_array[i] == "before":
183.                 D_time_before_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
184.             elif patient_time_array[i] == "1W":
185.                 D_time_1W_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
186.             elif patient_time_array[i] == "1M":
187.                 D_time_1M_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
188.             elif patient_time_array[i] == "3M":
189.                 D_time_3M_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
190.             elif patient_time_array[i] == "6M":
191.                 D_time_6M_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
192.             elif patient_time_array[i] == "1Y":
193.                 D_time_1Y_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
194.             elif patient_time_array[i] == "2Y":
195.                 D_time_2Y_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
196.             elif patient_time_array[i] == "3Y":
197.                 D_time_3Y_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
198.             elif patient_time_array[i] == "4Y":
199.                 D_time_4Y_avg_std_dev[D_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
200.
201.         else:
202.             if int(patient_number_array[i]) != int(ND_patient_number_array_avg_std_dev[ND_patient_counter_avg_std_dev]):
203.                 ND_patient_number_array_avg_std_dev.append(patient_number_array[i])
204.                 ND_patient_counter_avg_std_dev = ND_patient_counter_avg_std_dev + 1
205.                 ND_time_before_avg_std_dev.append(0)
206.                 ND_time_1W_avg_std_dev.append(0)
207.                 ND_time_1M_avg_std_dev.append(0)
208.                 ND_time_3M_avg_std_dev.append(0)
209.                 ND_time_6M_avg_std_dev.append(0)
210.                 ND_time_1Y_avg_std_dev.append(0)
211.                 ND_time_2Y_avg_std_dev.append(0)
212.                 ND_time_3Y_avg_std_dev.append(0)
213.                 ND_time_4Y_avg_std_dev.append(0)
214.
215.                 if patient_time_array[i] == "before":
216.                     ND_time_before_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
217.                 elif patient_time_array[i] == "1W":
218.                     ND_time_1W_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
219.                 elif patient_time_array[i] == "1M":
220.                     ND_time_1M_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
221.                 elif patient_time_array[i] == "3M":
222.                     ND_time_3M_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
223.                 elif patient_time_array[i] == "6M":
224.                     ND_time_6M_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
225.                 elif patient_time_array[i] == "1Y":
226.                     ND_time_1Y_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
227.                 elif patient_time_array[i] == "2Y":
228.                     ND_time_2Y_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
229.                 elif patient_time_array[i] == "3Y":
230.                     ND_time_3Y_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
231.                 elif patient_time_array[i] == "4Y":
232.                     ND_time_4Y_avg_std_dev[ND_patient_counter_avg_std_dev] = patient_avg_std_dev_array[i]
233.
234. 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 }
235. D_df_avg_std_dev = pd.DataFrame(D_data_avg_std_dev)
236. D_df_avg_std_dev.to_csv('RESULTS/D_AvgStdDev.csv', index=False)
237.
238. 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 }
239. ND_df_avg_std_dev = pd.DataFrame(ND_data_avg_std_dev)
240. ND_df_avg_std_dev.to_csv('RESULTS/ND_AvgStdDev.csv', index=False)
241.
242. #------
243.
244. D_patient_number_array_avg_area_trapz = []
245. D_patient_number_array_avg_area_trapz.append(1)
246. D_patient_counter_avg_area_trapz = 0
247.
248. D_time_before_avg_area_trapz = []
249. D_time_1W_avg_area_trapz = []
250. D_time_1M_avg_area_trapz = []
251. D_time_3M_avg_area_trapz = []
252. D_time_6M_avg_area_trapz = []
253. D_time_1Y_avg_area_trapz = []
254. D_time_2Y_avg_area_trapz = []
255. D_time_3Y_avg_area_trapz = []
256. D_time_4Y_avg_area_trapz = []
257.
258. D_time_before_avg_area_trapz.append(0)
259. D_time_1W_avg_area_trapz.append(0)
260. D_time_1M_avg_area_trapz.append(0)
261. D_time_3M_avg_area_trapz.append(0)
262. D_time_6M_avg_area_trapz.append(0)
263. D_time_1Y_avg_area_trapz.append(0)
264. D_time_2Y_avg_area_trapz.append(0)
265. D_time_3Y_avg_area_trapz.append(0)
266. D_time_4Y_avg_area_trapz.append(0)
267.
268. ND_patient_number_array_avg_area_trapz = []
269. ND_patient_number_array_avg_area_trapz.append(1)
270. ND_patient_counter_avg_area_trapz = 0
271.
272. ND_time_before_avg_area_trapz = []
273. ND_time_1W_avg_area_trapz = []
274. ND_time_1M_avg_area_trapz = []
275. ND_time_3M_avg_area_trapz = []
276. ND_time_6M_avg_area_trapz = []
277. ND_time_1Y_avg_area_trapz = []
278. ND_time_2Y_avg_area_trapz = []
279. ND_time_3Y_avg_area_trapz = []
280. ND_time_4Y_avg_area_trapz = []
281.
282. ND_time_before_avg_area_trapz.append(0)
283. ND_time_1W_avg_area_trapz.append(0)
284. ND_time_1M_avg_area_trapz.append(0)
285. ND_time_3M_avg_area_trapz.append(0)
286. ND_time_6M_avg_area_trapz.append(0)
287. ND_time_1Y_avg_area_trapz.append(0)
288. ND_time_2Y_avg_area_trapz.append(0)
289. ND_time_3Y_avg_area_trapz.append(0)
290. ND_time_4Y_avg_area_trapz.append(0)
291.
292. max_area = 0.0
293. min_area = 100.0
294.
295. for i in range(0, n):
296.     if float(patient_avg_area_trapz[i]) > float(max_area):
297.         max_area = patient_avg_area_trapz[i]
298.
299.     if float(patient_avg_area_trapz[i]) < float(min_area) and float(patient_avg_area_trapz[i]) >= float(0.0):
300.         min_area = patient_avg_area_trapz[i]
301.
302.     if patient_treated_hand[i] == 'True':
303.         if int(patient_number_array[i]) !=
int(D_patient_number_array_avg_area_trapz[D_patient_counter_avg_area_trapz]):
304.             D_patient_number_array_avg_area_trapz.append(patient_number_array[i])
305.             D_patient_counter_avg_area_trapz = D_patient_counter_avg_area_trapz + 1
306.             D_time_before_avg_area_trapz.append(0)
307.             D_time_1W_avg_area_trapz.append(0)
308.             D_time_1M_avg_area_trapz.append(0)
309.             D_time_3M_avg_area_trapz.append(0)
310.             D_time_6M_avg_area_trapz.append(0)
311.             D_time_1Y_avg_area_trapz.append(0)
312.             D_time_2Y_avg_area_trapz.append(0)
313.             D_time_3Y_avg_area_trapz.append(0)
314.             D_time_4Y_avg_area_trapz.append(0)
315.
316.
317.

```



```

318.     if patient_time_array[i] == "before":
319.         D_time_before_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
320.     elif patient_time_array[i] == "1W":
321.         D_time_1W_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
322.     elif patient_time_array[i] == "1M":
323.         D_time_1M_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
324.     elif patient_time_array[i] == "3M":
325.         D_time_3M_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
326.     elif patient_time_array[i] == "6M":
327.         D_time_6M_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
328.     elif patient_time_array[i] == "1Y":
329.         D_time_1Y_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
330.     elif patient_time_array[i] == "2Y":
331.         D_time_2Y_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
332.     elif patient_time_array[i] == "3Y":
333.         D_time_3Y_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
334.     elif patient_time_array[i] == "4Y":
335.         D_time_4Y_avg_area_trapz[D_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
336.
337.     else:
338.         if int(patient_number_array[i]) !=
339.         int(ND_patient_number_array_avg_area_trapz[ND_patient_counter_avg_area_trapz]):
340.             ND_patient_number_array_avg_area_trapz.append(patient_number_array[i])
341.             ND_patient_counter_avg_area_trapz = ND_patient_counter_avg_area_trapz + 1
342.             ND_time_before_avg_area_trapz.append(0)
343.             ND_time_1W_avg_area_trapz.append(0)
344.             ND_time_1M_avg_area_trapz.append(0)
345.             ND_time_3M_avg_area_trapz.append(0)
346.             ND_time_6M_avg_area_trapz.append(0)
347.             ND_time_1Y_avg_area_trapz.append(0)
348.             ND_time_2Y_avg_area_trapz.append(0)
349.             ND_time_3Y_avg_area_trapz.append(0)
350.             ND_time_4Y_avg_area_trapz.append(0)
351.
352.         if patient_time_array[i] == "before":
353.             ND_time_before_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
354.         elif patient_time_array[i] == "1W":
355.             ND_time_1W_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
356.         elif patient_time_array[i] == "1M":
357.             ND_time_1M_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
358.         elif patient_time_array[i] == "3M":
359.             ND_time_3M_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
360.         elif patient_time_array[i] == "6M":
361.             ND_time_6M_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
362.         elif patient_time_array[i] == "1Y":
363.             ND_time_1Y_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
364.         elif patient_time_array[i] == "2Y":
365.             ND_time_2Y_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
366.         elif patient_time_array[i] == "3Y":
367.             ND_time_3Y_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
368.         elif patient_time_array[i] == "4Y":
369.             ND_time_4Y_avg_area_trapz[ND_patient_counter_avg_area_trapz] = patient_avg_area_trapz[i]
370.
371. D_data_avg_area_trapz = {'Patient': D_patient_number_array_avg_area_trapz, 'Before' :
372.     D_time_before_avg_area_trapz, '1 Week' : D_time_1W_avg_area_trapz, '1 Month' : D_time_1M_avg_area_trapz, '3
373.     Months': D_time_3M_avg_area_trapz, '6 Months': D_time_6M_avg_area_trapz, '1 Year': D_time_1Y_avg_area_trapz, '2
374.     Years': D_time_2Y_avg_area_trapz, '3 Years':D_time_3Y_avg_area_trapz, '4 Years':D_time_4Y_avg_area_trapz }
375. D_df_avg_area_trapz = pd.DataFrame(D_data_avg_area_trapz)
376. D_df_avg_area_trapz.to_csv('RESULTS/D_AreaTrapz.csv', index=False)
377.
378. #-----
379.
380. ND_patient_number_array_det_2 = []
381. ND_patient_number_array_det_2.append(1)
382. ND_patient_counter_det_2 = 0
383.
384. ND_time_before_det_2 = []
385. ND_time_1W_det_2 = []
386. ND_time_1M_det_2 = []
387. ND_time_3M_det_2 = []
388. ND_time_6M_det_2 = []
389. ND_time_1Y_det_2 = []
390. ND_time_2Y_det_2 = []
391. ND_time_3Y_det_2 = []
392. ND_time_4Y_det_2 = []

```

```

393.
394.D_time_before_det_2.append(0)
395.D_time_1W_det_2.append(0)
396.D_time_1M_det_2.append(0)
397.D_time_3M_det_2.append(0)
398.D_time_6M_det_2.append(0)
399.D_time_1Y_det_2.append(0)
400.D_time_2Y_det_2.append(0)
401.D_time_3Y_det_2.append(0)
402.D_time_4Y_det_2.append(0)
403.
404.ND_patient_number_array_det_2 = []
405.ND_patient_number_array_det_2.append(1)
406.ND_patient_counter_det_2 = 0
407.
408.ND_time_before_det_2 = []
409.ND_time_1W_det_2 = []
410.ND_time_1M_det_2 = []
411.ND_time_3M_det_2 = []
412.ND_time_6M_det_2 = []
413.ND_time_1Y_det_2 = []
414.ND_time_2Y_det_2 = []
415.ND_time_3Y_det_2 = []
416.ND_time_4Y_det_2 = []
417.
418.ND_time_before_det_2.append(0)
419.ND_time_1W_det_2.append(0)
420.ND_time_1M_det_2.append(0)
421.ND_time_3M_det_2.append(0)
422.ND_time_6M_det_2.append(0)
423.ND_time_1Y_det_2.append(0)
424.ND_time_2Y_det_2.append(0)
425.ND_time_3Y_det_2.append(0)
426.ND_time_4Y_det_2.append(0)
427.
428.D_patient_hand_array_det_2 = []
429.ND_patient_hand_array_det_2 = []
430.
431.max_det_2 = 0.0
432.min_det_2 = 100.0
433.
434.for i in range(0, n):
435.    if float(patient_determinator_2[i]) > float(max_det_2):
436.        max_det_2 = patient_determinator_2[i]
437.
438.    if float(patient_determinator_2[i]) < float(min_det_2) and float(patient_determinator_2[i]) >= float(0.0):
439.        min_det_2 = patient_determinator_2[i]
440.
441.    if patient_treated_hand[i] == 'True':
442.        if int(patient_number_array[i]) != int(D_patient_number_array_det_2[D_patient_counter_det_2]):
443.            D_patient_hand_array_det_2.append(patient_dominant_hand[i])
444.            D_patient_number_array_det_2.append(patient_number_array[i])
445.            D_patient_counter_det_2 = D_patient_counter_det_2 + 1
446.            D_time_before_det_2.append(0)
447.            D_time_1W_det_2.append(0)
448.            D_time_1M_det_2.append(0)
449.            D_time_3M_det_2.append(0)
450.            D_time_6M_det_2.append(0)
451.            D_time_1Y_det_2.append(0)
452.            D_time_2Y_det_2.append(0)
453.            D_time_3Y_det_2.append(0)
454.            D_time_4Y_det_2.append(0)
455.
456.            if patient_time_array[i] == "before":
457.                D_time_before_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
458.            elif patient_time_array[i] == "1W":
459.                D_time_1W_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
460.            elif patient_time_array[i] == "1M":
461.                D_time_1M_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
462.            elif patient_time_array[i] == "3M":
463.                D_time_3M_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
464.            elif patient_time_array[i] == "6M":
465.                D_time_6M_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
466.            elif patient_time_array[i] == "1Y":
467.                D_time_1Y_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
468.            elif patient_time_array[i] == "2Y":
469.                D_time_2Y_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
470.            elif patient_time_array[i] == "3Y":
471.                D_time_3Y_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
472.            elif patient_time_array[i] == "4Y":
473.                D_time_4Y_det_2[D_patient_counter_det_2] = patient_determinator_2[i]
474.

```

```

475.     else:
476.         if int(patient_number_array[i]) != int(ND_patient_number_array_det_2[ND_patient_counter_det_2]):
477.             ND_patient_hand_array_det_2.append(patient_dominant_hand[i])
478.             ND_patient_number_array_det_2.append(patient_number_array[i])
479.             ND_patient_counter_det_2 = ND_patient_counter_det_2 + 1
480.             ND_time_before_det_2.append(0)
481.             ND_time_1W_det_2.append(0)
482.             ND_time_1M_det_2.append(0)
483.             ND_time_3M_det_2.append(0)
484.             ND_time_6M_det_2.append(0)
485.             ND_time_1Y_det_2.append(0)
486.             ND_time_2Y_det_2.append(0)
487.             ND_time_3Y_det_2.append(0)
488.             ND_time_4Y_det_2.append(0)
489.
490.             if patient_time_array[i] == "before":
491.                 ND_time_before_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
492.             elif patient_time_array[i] == "1W":
493.                 ND_time_1W_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
494.             elif patient_time_array[i] == "1M":
495.                 ND_time_1M_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
496.             elif patient_time_array[i] == "3M":
497.                 ND_time_3M_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
498.             elif patient_time_array[i] == "6M":
499.                 ND_time_6M_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
500.             elif patient_time_array[i] == "1Y":
501.                 ND_time_1Y_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
502.             elif patient_time_array[i] == "2Y":
503.                 ND_time_2Y_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
504.             elif patient_time_array[i] == "3Y":
505.                 ND_time_3Y_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
506.             elif patient_time_array[i] == "4Y":
507.                 ND_time_4Y_det_2[ND_patient_counter_det_2] = patient_determinator_2[i]
508.
509. D_data_det_2 = {'Patient': D_patient_number_array_det_2, 'Before' : D_time_before_det_2, '1 Week' :
510.               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':
511.               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 }
512. D_df_det_2 = pd.DataFrame(D_data_det_2)
513. D_df_det_2.to_csv('RESULTS/D_Determinant2.csv', index=False)
514.
515. ND_data_det_2 = {'Patient': ND_patient_number_array_det_2, 'Before' : ND_time_before_det_2, '1 Week' :
516.               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
517.               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 }
518. ND_df_det_2 = pd.DataFrame(ND_data_det_2)
519. ND_df_det_2.to_csv('RESULTS/ND_Determinant2.csv', index=False)
520.
521. # GRAPHS - - - - - #
522. def checkImproved(before, arr):
523.     improved = 0
524.     both = 0
525.     avg_amount = 0
526.     avg_diff = 0
527.     avg_diff_improved = 0
528.
529.     for i in range(len(before)):
530.         abs_before = abs(float(before[i]))
531.         abs_arr = abs(float(arr[i]))
532.         if (abs_before != 0) and (abs_arr != 0) and (abs_before != -1) and (abs_arr != -1):
533.             both += 1
534.             avg_diff += float(abs_before) - float(abs_arr)
535.
536.             if float(abs_arr) < float(abs_before):
537.                 improved += 1
538.                 avg_amount += float(abs_arr)
539.                 avg_diff_improved += float(abs_before) - float(abs_arr)
540.
541.     avg_diff = avg_diff/both
542.     avg_diff_improved = avg_diff_improved/improved
543.     avg_amount = avg_amount/improved
544.     return both, improved, avg_amount, avg_diff, avg_diff_improved
545.
546. D_both_1W, D_improved_1W, D_avg_amount_1W, D_avg_diff_1W, D_avg_diff_improved_1W =
547.     checkImproved(D_time_before_det_2, D_time_1W_det_2)
548. D_both_1M, D_improved_1M, D_avg_amount_1M, D_avg_diff_1M, D_avg_diff_improved_1M =
549.     checkImproved(D_time_before_det_2, D_time_1M_det_2)
550. D_both_3M, D_improved_3M, D_avg_amount_3M, D_avg_diff_3M, D_avg_diff_improved_3M =
551.     checkImproved(D_time_before_det_2, D_time_3M_det_2)
552. D_both_6M, D_improved_6M, D_avg_amount_6M, D_avg_diff_6M, D_avg_diff_improved_6M =
553.     checkImproved(D_time_before_det_2, D_time_6M_det_2)
554. D_both_1Y, D_improved_1Y, D_avg_amount_1Y, D_avg_diff_1Y, D_avg_diff_improved_1Y =
555.     checkImproved(D_time_before_det_2, D_time_1Y_det_2)

```

```

547.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)
548.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)
549.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)
550.
551.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]
552.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]
553.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]
554.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]
555.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]
556.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]
557.
558.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)
559.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)
560.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)
561.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)
562.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)
563.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)
564.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)
565.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)
566.
567.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]
568.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]
569.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]
570.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]
571.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]
572.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]
573.
574.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]
575.
576.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)
577.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)
578.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)
579.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)
580.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)
581.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)
582.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)
583.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)
584.
585.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)
586.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)
587.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)
588.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)

```

```

589.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)
590.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)
591.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)
592.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)
593.
594.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]
595.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]
596.
597.## GENERAL GRAPH SETTINGS
598.plt.rcParams["font.family"] = "Times New Roman"
599.plt.rc('axes', axisbelow=True)
600.plt.rcParams.update({'font.size': 16})
601.
602.## GRAPH 1: PERCENTAGE OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES
603.x = ['1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y']
604.x_axis = np.arange(len(x))
605.fig, ax1 = plt.subplots(figsize = (8,4))
606.plt.title('Percentage of Patients with Tremor Before Treatment that Improved\nAfter Various Treatment Times - PEAK
    DISTANCE (METHOD 2B)')
607.plt.grid(linestyle = '-', linewidth=0.5, axis='y')
608.plt.tight_layout()
609.
610.ax1.bar(x_axis - 0.15, D_improved_percentage, 0.3, label = 'Treated Hand', color = 'darkblue')
611.ax1.bar(x_axis + 0.15, ND_improved_percentage, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
612.ax1.set_ylabel('Percentage Improved')
613.ax1.set_xlabel('Time')
614.ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
615.ax1.set_ylim(0, 100)
616.ax2 = ax1.twinx()
617.ax2.plot(x, avg_total_num, color = 'red')
618.ax2.set_ylabel('Number of Patients')
619.ax2.legend(['Number of\npatients'], loc="upper center", prop={'size': 14})
620.ax2.set_ylim(0, 100)
621.print("AVERAGE PEAK DISTANCE DOMINANT: " + str(np.average(D_improved_percentage)))
622.print("AVERAGE PEAK DISTANCE NON-DOMINANT: " + str(np.average(ND_improved_percentage)))
623.plt.savefig('RESULTS\GRAPHS\PercentageOfPatients_Det2.png', bbox_inches='tight', dpi=150)
624.
625.## GRAPH 2: PERCENTAGE OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES
626.fig, ax1 = plt.subplots(figsize = (8,4))
627.plt.grid(linestyle = '-', linewidth=0.5, axis='y')
628.plt.title('Difference Between Tremor Severity Before Treatment\nand After Various Treatment Periods')
629.
630.ax1.bar(x_axis - 0.15, D_avg_diff, 0.3, label = 'Treated Hand', color = 'darkblue')
631.ax1.bar(x_axis + 0.15, ND_avg_diff, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
632.ax1.set_ylabel('Difference in Tremor')
633.ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
634.ax1.set_xlabel('Time')
635.
636.plt.xticks(x_axis, x)
637.plt.tight_layout()
638.plt.show()
639.
640.## GRAPH 3: PERCENT OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES -AVG AREA
641.x = ['1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y']
642.x_axis = np.arange(len(x))
643.fig, ax1 = plt.subplots(figsize = (8,4))
644.plt.title('Percentage of Patients with Tremor Before Treatment that Improved\nAfter Various Treatment Times -
    AVERAGE AREA (METHOD 2A)')
645.plt.grid(linestyle = '-', linewidth=0.5, axis='y')
646.plt.tight_layout()
647.
648.ax1.bar(x_axis - 0.15, D_improved_percentage_AVG_AREA, 0.3, label = 'Treated Hand', color = 'darkblue')
649.ax1.bar(x_axis + 0.15, ND_improved_percentage_AVG_AREA, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
650.ax1.set_ylabel('Percentage Improved')
651.ax1.set_xlabel('Time')
652.ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
653.ax1.set_ylim(0, 100)
654.ax2 = ax1.twinx()
655.ax2.plot(x, avg_total_num, color = 'red')
656.ax2.set_ylabel('Number of Patients')

```



```

657.ax2.legend(['Number of\npatients'], loc="upper center", prop={'size': 14})
658.ax2.set_ylim(0, 100)
659.plt.rcParams["figure.figsize"] = (8,4)
660.print("AVERAGE AREA METHOD DOM: " + str(np.average(D_improved_percentage_AVG_AREA)))
661.print("AVERAGE AREA METHOD NON-DOM: " + str(np.average(ND_improved_percentage_AVG_AREA)))
662.plt.savefig('RESULTS\GRAPHS\PercentageOfPatients_AvgArea.png', bbox_inches='tight', dpi=150)
663.
664.# START OF "WHICH HAND?" GRAPH ONE
665.combined_improved_array = []
666.which_hand = []
667.combined_patient_number_array = []
668.dominant_hand = []
669.longest_array = 0
670.
671.if len(D_patient_number_array_det_2) > len(ND_patient_number_array_det_2):
672.    longest_array = len(D_patient_number_array_det_2)
673.else:
674.    longest_array = len(ND_patient_number_array_det_2)
675.
676.D_counter = -1
677.ND_counter = -1
678.
679._before = 1
680._1W = 7
681._1M = 30
682._3M = 91
683._6M = 183
684._1Y = 365
685._2Y = 730
686._3Y = 1095
687._4Y = 1460
688.
689._before_linear = 1
690._1W_linear = 2
691._1M_linear = 3
692._3M_linear = 4
693._6M_linear = 5
694._1Y_linear = 6
695._2Y_linear = 7
696._3Y_linear = 8
697._4Y_linear = 9
698.
699.fig_ALLPATIENTS, ax1_ALLPATIENTS = plt.subplots(figsize = (8,4))
700.
701.for i in range(0, longest_array):
702.    D_counter += 1
703.    ND_counter += 1
704.
705.    if D_counter >= len(D_patient_number_array_det_2) or ND_counter >= len(ND_patient_number_array_det_2):
706.        break
707.    else:
708.        while D_patient_number_array_det_2[D_counter] != ND_patient_number_array_det_2[ND_counter]:
709.            if D_patient_number_array_det_2[D_counter+1] == ND_patient_number_array_det_2[ND_counter]:
710.                D_counter += 1
711.            elif D_patient_number_array_det_2[D_counter] == ND_patient_number_array_det_2[ND_counter + 1]:
712.                ND_counter += 1
713.            else:
714.                D_counter += 1
715.                ND_counter += 1
716.
717.    D_x_array = []
718.    D_x_array_linear = []
719.    D_y_array = []
720.
721.    if D_time_before_det_2[D_counter] != 0 and D_time_before_det_2[D_counter] != -1:
722.        D_y_array.append(D_time_before_det_2[D_counter])
723.        D_x_array.append(_before)
724.        D_x_array_linear.append(_before_linear)
725.    if D_time_1W_det_2[D_counter] != 0 and D_time_1W_det_2[D_counter] != -1:
726.        D_y_array.append(D_time_1W_det_2[D_counter])
727.        D_x_array.append(_1W)
728.        D_x_array_linear.append(_1W_linear)
729.    if D_time_1M_det_2[D_counter] != 0 and D_time_1M_det_2[D_counter] != -1:
730.        D_y_array.append(D_time_1M_det_2[D_counter])
731.        D_x_array.append(_1M)
732.        D_x_array_linear.append(_1M_linear)
733.    if D_time_3M_det_2[D_counter] != 0 and D_time_3M_det_2[D_counter] != -1:
734.        D_y_array.append(D_time_3M_det_2[D_counter])
735.        D_x_array.append(_3M)
736.        D_x_array_linear.append(_3M_linear)
737.    if D_time_6M_det_2[D_counter] != 0 and D_time_6M_det_2[D_counter] != -1:
738.        D_y_array.append(D_time_6M_det_2[D_counter])

```

```

739.     D_x_array.append(_6M)
740.     D_x_array_linear.append(_6M_linear)
741. if D_time_1Y_det_2[D_counter] != 0 and D_time_1Y_det_2[D_counter] != -1:
742.     D_y_array.append(D_time_1Y_det_2[D_counter])
743.     D_x_array.append(_1Y)
744.     D_x_array_linear.append(_1Y_linear)
745. if D_time_2Y_det_2[D_counter] != 0 and D_time_2Y_det_2[D_counter] != -1:
746.     D_y_array.append(D_time_2Y_det_2[D_counter])
747.     D_x_array.append(_2Y)
748.     D_x_array_linear.append(_2Y_linear)
749. if D_time_3Y_det_2[D_counter] != 0 and D_time_3Y_det_2[D_counter] != -1:
750.     D_y_array.append(D_time_3Y_det_2[D_counter])
751.     D_x_array.append(_3Y)
752.     D_x_array_linear.append(_3Y_linear)
753. if D_time_4Y_det_2[D_counter] != 0 and D_time_4Y_det_2[D_counter] != -1:
754.     D_y_array.append(D_time_4Y_det_2[D_counter])
755.     D_x_array.append(_4Y)
756.     D_x_array_linear.append(_4Y_linear)
757.
758. ND_x_array = []
759. ND_x_array_linear = []
760. ND_y_array = []
761. if ND_time_before_det_2[ND_counter] != 0 and ND_time_before_det_2[ND_counter] != -1:
762.     ND_y_array.append(ND_time_before_det_2[ND_counter])
763.     ND_x_array.append(_before)
764.     ND_x_array_linear.append(_before_linear)
765. if ND_time_1W_det_2[ND_counter] != 0 and ND_time_1W_det_2[ND_counter] != -1:
766.     ND_y_array.append(ND_time_1W_det_2[ND_counter])
767.     ND_x_array.append(_1W)
768.     ND_x_array_linear.append(_1W_linear)
769. if ND_time_1M_det_2[ND_counter] != 0 and ND_time_1M_det_2[ND_counter] != -1:
770.     ND_y_array.append(ND_time_1M_det_2[ND_counter])
771.     ND_x_array.append(_1M)
772.     ND_x_array_linear.append(_1M_linear)
773. if ND_time_3M_det_2[ND_counter] != 0 and ND_time_3M_det_2[ND_counter] != -1:
774.     ND_y_array.append(ND_time_3M_det_2[ND_counter])
775.     ND_x_array.append(_3M)
776.     ND_x_array_linear.append(_3M_linear)
777. if ND_time_6M_det_2[ND_counter] != 0 and ND_time_6M_det_2[ND_counter] != -1:
778.     ND_y_array.append(ND_time_6M_det_2[ND_counter])
779.     ND_x_array.append(_6M)
780.     ND_x_array_linear.append(_6M_linear)
781. if ND_time_1Y_det_2[ND_counter] != 0 and ND_time_1Y_det_2[ND_counter] != -1:
782.     ND_y_array.append(ND_time_1Y_det_2[ND_counter])
783.     ND_x_array.append(_1Y)
784.     ND_x_array_linear.append(_1Y_linear)
785. if ND_time_2Y_det_2[ND_counter] != 0 and ND_time_2Y_det_2[ND_counter] != -1:
786.     ND_y_array.append(ND_time_2Y_det_2[ND_counter])
787.     ND_x_array.append(_2Y)
788.     ND_x_array_linear.append(_2Y_linear)
789. if ND_time_3Y_det_2[ND_counter] != 0 and ND_time_3Y_det_2[ND_counter] != -1:
790.     ND_y_array.append(ND_time_3Y_det_2[ND_counter])
791.     ND_x_array.append(_3Y)
792.     ND_x_array_linear.append(_3Y_linear)
793. if ND_time_4Y_det_2[ND_counter] != 0 and ND_time_4Y_det_2[ND_counter] != -1:
794.     ND_y_array.append(ND_time_4Y_det_2[ND_counter])
795.     ND_x_array.append(_4Y)
796.     ND_x_array_linear.append(_4Y_linear)
797.
798. if len(D_x_array) > 4 and len(D_y_array) > 4 and len(ND_x_array) > 4 and len(ND_y_array) > 4:
799.
800.     combined_patient_number_array.append(D_patient_number_array_det_2[D_counter])
801.
802.     D_gradient, D_intercept = np.polyfit(D_x_array, D_y_array, 1)
803.     ND_gradient, ND_intercept = np.polyfit(ND_x_array, ND_y_array, 1)
804.
805.     plt.title('Results of Patients\' Most Improved Hand')
806.     plt.grid(linestyle = '-', linewidth=0.5, axis='y')
807.     plt.xticks([1,2,3,4,5,6,7,8,9], ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
808.
809.     temp_D_y_array = np.array(D_y_array.copy())
810.     D_y_array_max = max(temp_D_y_array)
811.     temp_D_y_array = temp_D_y_array/D_y_array_max*100
812.     temp_ND_y_array = np.array(ND_y_array.copy())
813.     ND_y_array_max = max(temp_ND_y_array)
814.     temp_ND_y_array = temp_ND_y_array/ND_y_array_max*100
815.
816.     if D_gradient < 0 and ND_gradient < 0:
817.         combined_improved_array.append('BOTH')
818.         if D_gradient < ND_gradient:
819.             which_hand.append('TREATED')
820.             ax1_ALLPATIENTS.plot(D_x_array_linear, temp_D_y_array)

```

```

821.
822.         else:
823.             which_hand.append('NON-TREATED')
824.             ax1_ALLPATIENTS.plot(ND_x_array_linear, temp_ND_y_array)
825.
826.
827.         elif D_gradient < 0:
828.             combined_improved_array.append('TREATED')
829.             which_hand.append('TREATED')
830.             ax1_ALLPATIENTS.plot(D_x_array_linear, temp_D_y_array)
831.
832.         elif ND_gradient < 0:
833.             combined_improved_array.append('NON-TREATED')
834.             which_hand.append('NON-TREATED')
835.             ax1_ALLPATIENTS.plot(ND_x_array_linear, temp_ND_y_array)
836.
837.         else:
838.             combined_improved_array.append('NEITHER')
839.             if D_gradient < ND_gradient:
840.                 which_hand.append('TREATED')
841.             else:
842.                 which_hand.append('NON-TREATED')
843.
844. plt.tight_layout()
845. plt.show()
846.
847. # START OF "WHICH HAND?" TWO
848. combined_improved_array = []
849. which_hand = []
850. combined_patient_number_array = []
851. dominant_hand = []
852.
853. longest_array = 0
854.
855. if len(D_patient_number_array_det_2) > len(ND_patient_number_array_det_2):
856.     longest_array = len(D_patient_number_array_det_2)
857. else:
858.     longest_array = len(ND_patient_number_array_det_2)
859.
860. D_counter = -1
861. ND_counter = -1
862.
863. _before = 1
864. _1W = 7
865. _1M = 30
866. _3M = 91
867. _6M = 183
868. _1Y = 365
869. _2Y = 730
870. _3Y = 1095
871. _4Y = 1460
872.
873. _before_linear = 1
874. _1W_linear = 2
875. _1M_linear = 3
876. _3M_linear = 4
877. _6M_linear = 5
878. _1Y_linear = 6
879. _2Y_linear = 7
880. _3Y_linear = 8
881. _4Y_linear = 9
882.
883. fig_ALLPATIENTS2, ax1_ALLPATIENTS2 = plt.subplots(figsize = (8,4))
884. plt.title('Results of Patients Whose \'Before\' Drawings are the Most Severe')
885. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
886. plt.xticks([1,2,3,4,5,6,7,8,9], ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
887.
888. for i in range(0, longest_array):
889.     D_counter += 1
890.     ND_counter += 1
891.
892.     if D_counter >= len(D_patient_number_array_det_2) or ND_counter >= len(ND_patient_number_array_det_2):
893.         break
894.     else:
895.         while D_patient_number_array_det_2[D_counter] != ND_patient_number_array_det_2[ND_counter]:
896.             if D_patient_number_array_det_2[D_counter+1] == ND_patient_number_array_det_2[ND_counter]:
897.                 D_counter += 1
898.             elif D_patient_number_array_det_2[D_counter] == ND_patient_number_array_det_2[ND_counter + 1]:
899.                 ND_counter += 1
900.             else:
901.                 D_counter += 1
902.                 ND_counter += 1

```



```

903.
904. D_x_array = []
905. D_x_array_linear = []
906. D_y_array = []
907.
908. if D_time_before_det_2[D_counter] != 0 and D_time_before_det_2[D_counter] != -1:
909.     D_y_array.append(D_time_before_det_2[D_counter])
910.     D_x_array.append(_before)
911.     D_x_array_linear.append(_before_linear)
912.     if D_time_1W_det_2[D_counter] != 0 and D_time_1W_det_2[D_counter] != -1:
913.         D_y_array.append(D_time_1W_det_2[D_counter])
914.         D_x_array.append(_1W)
915.         D_x_array_linear.append(_1W_linear)
916.     if D_time_1M_det_2[D_counter] != 0 and D_time_1M_det_2[D_counter] != -1:
917.         D_y_array.append(D_time_1M_det_2[D_counter])
918.         D_x_array.append(_1M)
919.         D_x_array_linear.append(_1M_linear)
920.     if D_time_3M_det_2[D_counter] != 0 and D_time_3M_det_2[D_counter] != -1:
921.         D_y_array.append(D_time_3M_det_2[D_counter])
922.         D_x_array.append(_3M)
923.         D_x_array_linear.append(_3M_linear)
924.     if D_time_6M_det_2[D_counter] != 0 and D_time_6M_det_2[D_counter] != -1:
925.         D_y_array.append(D_time_6M_det_2[D_counter])
926.         D_x_array.append(_6M)
927.         D_x_array_linear.append(_6M_linear)
928.     if D_time_1Y_det_2[D_counter] != 0 and D_time_1Y_det_2[D_counter] != -1:
929.         D_y_array.append(D_time_1Y_det_2[D_counter])
930.         D_x_array.append(_1Y)
931.         D_x_array_linear.append(_1Y_linear)
932.     if D_time_2Y_det_2[D_counter] != 0 and D_time_2Y_det_2[D_counter] != -1:
933.         D_y_array.append(D_time_2Y_det_2[D_counter])
934.         D_x_array.append(_2Y)
935.         D_x_array_linear.append(_2Y_linear)
936.     if D_time_3Y_det_2[D_counter] != 0 and D_time_3Y_det_2[D_counter] != -1:
937.         D_y_array.append(D_time_3Y_det_2[D_counter])
938.         D_x_array.append(_3Y)
939.         D_x_array_linear.append(_3Y_linear)
940.     if D_time_4Y_det_2[D_counter] != 0 and D_time_4Y_det_2[D_counter] != -1:
941.         D_y_array.append(D_time_4Y_det_2[D_counter])
942.         D_x_array.append(_4Y)
943.         D_x_array_linear.append(_4Y_linear)
944.
945. ND_x_array = []
946. ND_x_array_linear = []
947. ND_y_array = []
948. if ND_time_before_det_2[ND_counter] != 0 and ND_time_before_det_2[ND_counter] != -1:
949.     ND_y_array.append(ND_time_before_det_2[ND_counter])
950.     ND_x_array.append(_before)
951.     ND_x_array_linear.append(_before_linear)
952.
953.     if ND_time_1W_det_2[ND_counter] != 0 and ND_time_1W_det_2[ND_counter] != -1:
954.         ND_y_array.append(ND_time_1W_det_2[ND_counter])
955.         ND_x_array.append(_1W)
956.         ND_x_array_linear.append(_1W_linear)
957.     if ND_time_1M_det_2[ND_counter] != 0 and ND_time_1M_det_2[ND_counter] != -1:
958.         ND_y_array.append(ND_time_1M_det_2[ND_counter])
959.         ND_x_array.append(_1M)
960.         ND_x_array_linear.append(_1M_linear)
961.     if ND_time_3M_det_2[ND_counter] != 0 and ND_time_3M_det_2[ND_counter] != -1:
962.         ND_y_array.append(ND_time_3M_det_2[ND_counter])
963.         ND_x_array.append(_3M)
964.         ND_x_array_linear.append(_3M_linear)
965.     if ND_time_6M_det_2[ND_counter] != 0 and ND_time_6M_det_2[ND_counter] != -1:
966.         ND_y_array.append(ND_time_6M_det_2[ND_counter])
967.         ND_x_array.append(_6M)
968.         ND_x_array_linear.append(_6M_linear)
969.     if ND_time_1Y_det_2[ND_counter] != 0 and ND_time_1Y_det_2[ND_counter] != -1:
970.         ND_y_array.append(ND_time_1Y_det_2[ND_counter])
971.         ND_x_array.append(_1Y)
972.         ND_x_array_linear.append(_1Y_linear)
973.     if ND_time_2Y_det_2[ND_counter] != 0 and ND_time_2Y_det_2[ND_counter] != -1:
974.         ND_y_array.append(ND_time_2Y_det_2[ND_counter])
975.         ND_x_array.append(_2Y)
976.         ND_x_array_linear.append(_2Y_linear)
977.     if ND_time_3Y_det_2[ND_counter] != 0 and ND_time_3Y_det_2[ND_counter] != -1:
978.         ND_y_array.append(ND_time_3Y_det_2[ND_counter])
979.         ND_x_array.append(_3Y)
980.         ND_x_array_linear.append(_3Y_linear)
981.     if ND_time_4Y_det_2[ND_counter] != 0 and ND_time_4Y_det_2[ND_counter] != -1:
982.         ND_y_array.append(ND_time_4Y_det_2[ND_counter])
983.         ND_x_array.append(_4Y)
984.         ND_x_array_linear.append(_4Y_linear)

```

```

985.
986. if len(D_x_array) > 4 and len(D_y_array) > 4 and len(ND_x_array) > 4 and len(ND_y_array) > 4:
987.
988.     combined_patient_number_array.append(D_patient_number_array_det_2[D_counter])
989.
990.     D_gradient, D_intercept = np.polyfit(D_x_array, D_y_array, 1)
991.     ND_gradient, ND_intercept = np.polyfit(ND_x_array, ND_y_array, 1)
992.
993.     temp_D_y_array = np.array(D_y_array.copy())
994.     D_y_array_max = max(temp_D_y_array)
995.     temp_D_y_array = temp_D_y_array/D_y_array_max*100
996.     temp_ND_y_array = np.array(ND_y_array.copy())
997.     ND_y_array_max = max(temp_ND_y_array)
998.     temp_ND_y_array = temp_ND_y_array/ND_y_array_max*100
999.
1000.     if D_time_before_det_2[D_counter] < 600 and ND_time_before_det_2[ND_counter] < 600: # remove outlier
1001.         if D_gradient < 0 and ND_gradient < 0:
1002.             combined_improved_array.append('BOTH')
1003.             if D_gradient < ND_gradient:
1004.                 which_hand.append('TREATED')
1005.                 if D_y_array[0] == D_y_array_max:
1006.                     ax1_ALLPATIENTS2.plot(D_x_array_linear, D_y_array)
1007.             else:
1008.                 which_hand.append('NON-TREATED')
1009.                 if ND_y_array[0] == ND_y_array_max:
1010.                     ax1_ALLPATIENTS2.plot(ND_x_array_linear, ND_y_array)
1011.
1012.         elif D_gradient < 0:
1013.             combined_improved_array.append('TREATED')
1014.             which_hand.append('TREATED')
1015.             if D_y_array[0] == D_y_array_max:
1016.                 ax1_ALLPATIENTS2.plot(D_x_array_linear, D_y_array)
1017.
1018.         elif ND_gradient < 0:
1019.             combined_improved_array.append('NON-TREATED')
1020.             which_hand.append('NON-TREATED')
1021.             if ND_y_array[0] == ND_y_array_max:
1022.                 ax1_ALLPATIENTS2.plot(ND_x_array_linear, ND_y_array)
1023.
1024.         else:
1025.             combined_improved_array.append('NEITHER')
1026.             if D_gradient < ND_gradient:
1027.                 which_hand.append('TREATED')
1028.             else:
1029.                 which_hand.append('NON-TREATED')
1030.
1031. plt.tight_layout()
1032. plt.show()
1033.
1034. # START OF "WHICH HAND?" THREE
1035. x_array = [1,2,3,4,5,6,7,8,9]
1036. fig_ALLPATIENTS3, ax1_ALLPATIENTS3 = plt.subplots(figsize = (8,4))
1037. plt.title('Average Tremor Severities for Each Hand\ndPEAK DISTANCE (METHOD 2B)')
1038. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
1039. plt.xticks(x_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
1040.
1041. D_average_before_det_2 = np.mean(D_time_before_det_2)
1042. D_average_1W_det_2 = np.mean(D_time_1W_det_2)
1043. D_average_1M_det_2 = np.mean(D_time_1M_det_2)
1044. D_average_3M_det_2 = np.mean(D_time_3M_det_2)
1045. D_average_6M_det_2 = np.mean(D_time_6M_det_2)
1046. D_average_1Y_det_2 = np.mean(D_time_1Y_det_2)
1047. D_average_2Y_det_2 = np.mean(D_time_2Y_det_2)
1048. D_average_3Y_det_2 = np.mean(D_time_3Y_det_2)
1049. D_average_4Y_det_2 = np.mean(D_time_4Y_det_2)
1050.
1051. ND_average_before_det_2 = np.mean(ND_time_before_det_2)
1052. ND_average_1W_det_2 = np.mean(ND_time_1W_det_2)
1053. ND_average_1M_det_2 = np.mean(ND_time_1M_det_2)
1054. ND_average_3M_det_2 = np.mean(ND_time_3M_det_2)
1055. ND_average_6M_det_2 = np.mean(ND_time_6M_det_2)
1056. ND_average_1Y_det_2 = np.mean(ND_time_1Y_det_2)
1057. ND_average_2Y_det_2 = np.mean(ND_time_2Y_det_2)
1058. ND_average_3Y_det_2 = np.mean(ND_time_3Y_det_2)
1059. ND_average_4Y_det_2 = np.mean(ND_time_4Y_det_2)
1060.
1061. 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]
1062. 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]
1063.
1064. plt.legend()
ax1_ALLPATIENTS3.plot(x_array, D_averages, color = 'darkblue')

```

```

1065. ax1_ALLPATIENTS3.plot(x_array, ND_averages, color = 'cornflowerblue')
1066. ax1_ALLPATIENTS3.set_ylabel('Tremor Severity')
1067. ax1_ALLPATIENTS3.set_xlabel('Time')
1068. ax1_ALLPATIENTS3.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
1069. plt.tight_layout()
1070. plt.show()
1071.
1072. # AVERAGE TREMOR SEVERITY USING AVG AREA TRAPZ METHOD
1073. x_array = [1,2,3,4,5,6,7,8,9]
1074. fig_ALLPATIENTS3B, ax1_ALLPATIENTS3B = plt.subplots(figsize = (8,4))
1075. plt.title('Average Tremor Severities for Each Hand\nAVERAGE AREA (METHOD 2A)')
1076. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
1077. plt.xticks(x_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
1078.
1079. D_average_before_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_before_avg_area_trapz])
1080. D_average_1W_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_1W_avg_area_trapz])
1081. D_average_1M_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_1M_avg_area_trapz])
1082. D_average_3M_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_3M_avg_area_trapz])
1083. D_average_6M_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_6M_avg_area_trapz])
1084. D_average_1Y_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_1Y_avg_area_trapz])
1085. D_average_2Y_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_2Y_avg_area_trapz])
1086. D_average_3Y_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_3Y_avg_area_trapz])
1087. D_average_4Y_avg_area_trapz = np.mean([abs(float(x)) for x in D_time_4Y_avg_area_trapz])
1088.
1089. ND_average_before_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_before_avg_area_trapz])
1090. ND_average_1W_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_1W_avg_area_trapz])
1091. ND_average_1M_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_1M_avg_area_trapz])
1092. ND_average_3M_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_3M_avg_area_trapz])
1093. ND_average_6M_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_6M_avg_area_trapz])
1094. ND_average_1Y_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_1Y_avg_area_trapz])
1095. ND_average_2Y_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_2Y_avg_area_trapz])
1096. ND_average_3Y_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_3Y_avg_area_trapz])
1097. ND_average_4Y_avg_area_trapz = np.mean([abs(float(x)) for x in ND_time_4Y_avg_area_trapz])
1098.
1099. 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]
1100. 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]
1101.
1102. plt.legend()
1103. ax1_ALLPATIENTS3B.plot(x_array, D_averages, color = 'darkblue')
1104. ax1_ALLPATIENTS3B.plot(x_array, ND_averages, color = 'cornflowerblue')
1105. ax1_ALLPATIENTS3B.set_ylabel('Tremor Severity')
1106. ax1_ALLPATIENTS3B.set_xlabel('Time')
1107. ax1_ALLPATIENTS3B.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
1108. plt.savefig('RESULTS\GRAPHS\AverageTremSev_AvgArea.png', bbox_inches='tight', dpi=150)
1109.
1110. plt.tight_layout()
1111. plt.show()
1112.
1113. # START OF "WHICH HAND?" FOUR
1114.
1115. x_array = [1,2,3,4,5,6,7,8,9]
1116. fig_ALLPATIENTS4, ax1_ALLPATIENTS4 = plt.subplots(figsize = (8,4))
1117. plt.title('Normalised Average Tremor Severities for Each Hand\nPEAK DISTANCE (METHOD 2B)')
1118. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
1119. plt.xticks(x_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
1120.
1121. 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]
1122. 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])
1123. denom = q95 - all_min
1124.
1125. D_time_before_normalised = [(x - all_min)/denom for x in D_time_before_det_2]
1126. D_time_1W_normalised = [(x - all_min)/denom for x in D_time_1W_det_2]
1127. D_time_1M_normalised = [(x - all_min)/denom for x in D_time_1M_det_2]
1128. D_time_3M_normalised = [(x - all_min)/denom for x in D_time_3M_det_2]
1129. D_time_6M_normalised = [(x - all_min)/denom for x in D_time_6M_det_2]
1130. D_time_1Y_normalised = [(x - all_min)/denom for x in D_time_1Y_det_2]
1131. D_time_2Y_normalised = [(x - all_min)/denom for x in D_time_2Y_det_2]
1132. D_time_3Y_normalised = [(x - all_min)/denom for x in D_time_3Y_det_2]
1133. D_time_4Y_normalised = [(x - all_min)/denom for x in D_time_4Y_det_2]
1134.
1135. 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]
1136.
1137. ND_time_before_normalised = [(x - all_min)/denom for x in ND_time_before_det_2]

```

```

1138. ND_time_1W_normalised = [(x - all_min)/denom for x in ND_time_1W_det_2]
1139. ND_time_1M_normalised = [(x - all_min)/denom for x in ND_time_1M_det_2]
1140. ND_time_3M_normalised = [(x - all_min)/denom for x in ND_time_3M_det_2]
1141. ND_time_6M_normalised = [(x - all_min)/denom for x in ND_time_6M_det_2]
1142. ND_time_1Y_normalised = [(x - all_min)/denom for x in ND_time_1Y_det_2]
1143. ND_time_2Y_normalised = [(x - all_min)/denom for x in ND_time_2Y_det_2]
1144. ND_time_3Y_normalised = [(x - all_min)/denom for x in ND_time_3Y_det_2]
1145. ND_time_4Y_normalised = [(x - all_min)/denom for x in ND_time_4Y_det_2]
1146.
1147. D_average_before_det_2 = np.mean(D_time_before_normalised)
1148. D_average_1W_det_2 = np.mean(D_time_1W_normalised)
1149. D_average_1M_det_2 = np.mean(D_time_1M_normalised)
1150. D_average_3M_det_2 = np.mean(D_time_3M_normalised)
1151. D_average_6M_det_2 = np.mean(D_time_6M_normalised)
1152. D_average_1Y_det_2 = np.mean(D_time_1Y_normalised)
1153. D_average_2Y_det_2 = np.mean(D_time_2Y_normalised)
1154. D_average_3Y_det_2 = np.mean(D_time_3Y_normalised)
1155. D_average_4Y_det_2 = np.mean(D_time_4Y_normalised)
1156.
1157. ND_average_before_det_2 = np.mean(ND_time_before_normalised)
1158. ND_average_1W_det_2 = np.mean(ND_time_1W_normalised)
1159. ND_average_1M_det_2 = np.mean(ND_time_1M_normalised)
1160. ND_average_3M_det_2 = np.mean(ND_time_3M_normalised)
1161. ND_average_6M_det_2 = np.mean(ND_time_6M_normalised)
1162. ND_average_1Y_det_2 = np.mean(ND_time_1Y_normalised)
1163. ND_average_2Y_det_2 = np.mean(ND_time_2Y_normalised)
1164. ND_average_3Y_det_2 = np.mean(ND_time_3Y_normalised)
1165. ND_average_4Y_det_2 = np.mean(ND_time_4Y_normalised)
1166.
1167. 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]
1168. 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]
1169.
1170. plt.legend()
1171. ax1_ALLPATIENTS4.plot(x_array, D_averages, color = 'darkblue')
1172. ax1_ALLPATIENTS4.plot(x_array, ND_averages, color = 'cornflowerblue')
1173. ax1_ALLPATIENTS4.set_ylabel('Tremor Severity')
1174. ax1_ALLPATIENTS4.set_xlabel('Time')
1175.
1176. ax1_ALLPATIENTS4.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
1177. plt.savefig('RESULTS\GRAPHS\AverageTremSev_Det2.png', bbox_inches='tight', dpi=150)
1178. plt.show()
1179.
1180. # START OF "WHICH HAND?" FOUR B
1181. x_array = [1,2,3,4,5,6,7,8,9]
1182. fig_ALLPATIENTS4B, ax1_ALLPATIENTS4B = plt.subplots(figsize = (8,4))
1183. plt.title('Normalised Average Tremor Severities for Each Hand\nAVERAGE AREA (METHOD 2A)')
1184. plt.grid(linestyle = '--', linewidth=0.5, axis='y')
1185. plt.xticks(x_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
1186.
1187.
1188. copy_D_time_before_avg_area_trapz = [abs(float(x)) for x in D_time_before_avg_area_trapz]
1189. for c in copy_D_time_before_avg_area_trapz:
1190.     if c == 0.0:
1191.         copy_D_time_before_avg_area_trapz.remove(c)
1192. copy_D_time_1W_avg_area_trapz = [abs(float(x)) for x in D_time_1W_avg_area_trapz]
1193. for c in copy_D_time_1W_avg_area_trapz:
1194.     if c == 0.0:
1195.         copy_D_time_1W_avg_area_trapz.remove(c)
1196. copy_D_time_1M_avg_area_trapz = [abs(float(x)) for x in D_time_1M_avg_area_trapz]
1197. for c in copy_D_time_1M_avg_area_trapz:
1198.     if c == 0.0:
1199.         copy_D_time_1M_avg_area_trapz.remove(c)
1200. copy_D_time_3M_avg_area_trapz = [abs(float(x)) for x in D_time_3M_avg_area_trapz]
1201. for c in copy_D_time_3M_avg_area_trapz:
1202.     if c == 0.0:
1203.         copy_D_time_3M_avg_area_trapz.remove(c)
1204. copy_D_time_6M_avg_area_trapz = [abs(float(x)) for x in D_time_6M_avg_area_trapz]
1205. for c in copy_D_time_6M_avg_area_trapz:
1206.     if c == 0.0:
1207.         copy_D_time_6M_avg_area_trapz.remove(c)
1208. copy_D_time_1Y_avg_area_trapz = [abs(float(x)) for x in D_time_1Y_avg_area_trapz]
1209. for c in copy_D_time_1Y_avg_area_trapz:
1210.     if c == 0.0:
1211.         copy_D_time_1Y_avg_area_trapz.remove(c)
1212. copy_D_time_2Y_avg_area_trapz = [abs(float(x)) for x in D_time_2Y_avg_area_trapz]
1213. for c in copy_D_time_2Y_avg_area_trapz:
1214.     if c == 0.0:
1215.         copy_D_time_2Y_avg_area_trapz.remove(c)
1216. copy_D_time_3Y_avg_area_trapz = [abs(float(x)) for x in D_time_3Y_avg_area_trapz]
1217. for c in copy_D_time_3Y_avg_area_trapz:

```

```

1218.         if c == 0.0:
1219.             copy_D_time_3Y_avg_area_trapz.remove(c)
1220. copy_D_time_4Y_avg_area_trapz = [abs(float(x)) for x in D_time_4Y_avg_area_trapz]
1221. for c in copy_D_time_4Y_avg_area_trapz:
1222.     if c == 0.0:
1223.         copy_D_time_4Y_avg_area_trapz.remove(c)
1224.
1225. copy_ND_time_before_avg_area_trapz = [abs(float(x)) for x in ND_time_before_avg_area_trapz]
1226. for c in copy_ND_time_before_avg_area_trapz:
1227.     if c == 0.0:
1228.         copy_ND_time_before_avg_area_trapz.remove(c)
1229. copy_ND_time_1W_avg_area_trapz = [abs(float(x)) for x in ND_time_1W_avg_area_trapz]
1230. for c in copy_ND_time_1W_avg_area_trapz:
1231.     if c == 0.0:
1232.         copy_ND_time_1W_avg_area_trapz.remove(c)
1233. copy_ND_time_1M_avg_area_trapz = [abs(float(x)) for x in ND_time_1M_avg_area_trapz]
1234. for c in copy_ND_time_1M_avg_area_trapz:
1235.     if c == 0.0:
1236.         copy_ND_time_1M_avg_area_trapz.remove(c)
1237. copy_ND_time_3M_avg_area_trapz = [abs(float(x)) for x in ND_time_3M_avg_area_trapz]
1238. for c in copy_ND_time_3M_avg_area_trapz:
1239.     if c == 0.0:
1240.         copy_ND_time_3M_avg_area_trapz.remove(c)
1241. copy_ND_time_6M_avg_area_trapz = [abs(float(x)) for x in ND_time_6M_avg_area_trapz]
1242. for c in copy_ND_time_6M_avg_area_trapz:
1243.     if c == 0.0:
1244.         copy_ND_time_6M_avg_area_trapz.remove(c)
1245. copy_ND_time_1Y_avg_area_trapz = [abs(float(x)) for x in ND_time_1Y_avg_area_trapz]
1246. for c in copy_ND_time_1Y_avg_area_trapz:
1247.     if c == 0.0:
1248.         copy_ND_time_1Y_avg_area_trapz.remove(c)
1249. copy_ND_time_2Y_avg_area_trapz = [abs(float(x)) for x in ND_time_2Y_avg_area_trapz]
1250. for c in copy_ND_time_2Y_avg_area_trapz:
1251.     if c == 0.0:
1252.         copy_ND_time_2Y_avg_area_trapz.remove(c)
1253. copy_ND_time_3Y_avg_area_trapz = [abs(float(x)) for x in ND_time_3Y_avg_area_trapz]
1254. for c in copy_ND_time_3Y_avg_area_trapz:
1255.     if c == 0.0:
1256.         copy_ND_time_3Y_avg_area_trapz.remove(c)
1257. copy_ND_time_4Y_avg_area_trapz = [abs(float(x)) for x in ND_time_4Y_avg_area_trapz]
1258. for c in copy_ND_time_4Y_avg_area_trapz:
1259.     if c == 0.0:
1260.         copy_ND_time_4Y_avg_area_trapz.remove(c)
1261.
1262. print(copy_D_time_before_avg_area_trapz)
1263. print(copy_ND_time_before_avg_area_trapz)
1264. m1 = max(copy_D_time_before_avg_area_trapz)
1265. m2 = max(copy_D_time_1W_avg_area_trapz)
1266. m3 = max(copy_D_time_1M_avg_area_trapz)
1267. m4 = max(copy_D_time_3M_avg_area_trapz)
1268. m5 = max(copy_D_time_6M_avg_area_trapz)
1269. m6 = max(copy_D_time_1Y_avg_area_trapz )
1270. m7 = max(copy_D_time_2Y_avg_area_trapz )
1271. m8 = max(copy_D_time_3Y_avg_area_trapz )
1272. m9 = max(copy_D_time_4Y_avg_area_trapz )
1273.
1274. m10 = max(copy_ND_time_before_avg_area_trapz)
1275. m11 = max(copy_ND_time_1W_avg_area_trapz)
1276. m12 = max(copy_ND_time_1M_avg_area_trapz)
1277. m13 = max(copy_ND_time_3M_avg_area_trapz)
1278. m14 = max(copy_ND_time_6M_avg_area_trapz)
1279. m15 = max(copy_ND_time_1Y_avg_area_trapz)
1280. m16 = max(copy_ND_time_2Y_avg_area_trapz)
1281. m17 = max(copy_ND_time_3Y_avg_area_trapz)
1282. m18 = max(copy_ND_time_4Y_avg_area_trapz)
1283.
1284. m1 = min(copy_D_time_before_avg_area_trapz)
1285. m2 = min(copy_D_time_1W_avg_area_trapz)
1286. m3 = min(copy_D_time_1M_avg_area_trapz)
1287. m4 = min(copy_D_time_3M_avg_area_trapz)
1288. m5 = min(copy_D_time_6M_avg_area_trapz)
1289. m6 = min(copy_D_time_1Y_avg_area_trapz )
1290. m7 = min(copy_D_time_2Y_avg_area_trapz )
1291. m8 = min(copy_D_time_3Y_avg_area_trapz )
1292. m9 = min(copy_D_time_4Y_avg_area_trapz )
1293.
1294. m10 = min(copy_ND_time_before_avg_area_trapz)
1295. m11 = min(copy_ND_time_1W_avg_area_trapz)
1296. m12 = min(copy_ND_time_1M_avg_area_trapz)
1297. m13 = min(copy_ND_time_3M_avg_area_trapz)
1298. m14 = min(copy_ND_time_6M_avg_area_trapz)
1299. m15 = min(copy_ND_time_1Y_avg_area_trapz)

```



```

1300.     m16 = min(copy_ND_time_2Y_avg_area_trapz)
1301.     m17 = min(copy_ND_time_3Y_avg_area_trapz)
1302.     m18 = min(copy_ND_time_4Y_avg_area_trapz)
1303.
1304.     actual_max = max(m1, m2, m3, m4, m5, m6, m7, m8, m9, m11, m12, m13, m14, m15, m16, m17, m18)
1305.     all_min = min(m1, m2, m3, m4, m5, m6, m7, m8, m9, m11, m12, m13, m14, m15, m16, m17, m18)
1306.
1307.     # NB: BUG WARNING HERE - WHY IS MAX 0?
1308.     denom = 50 # actual_max - all_min
1309.
1310.     D_time_before_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_D_time_before_avg_area_trapz]]
1311.     D_time_1W_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_D_time_1W_avg_area_trapz]]
1312.     D_time_1M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_1M_avg_area_trapz]]
1313.     D_time_3M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_3M_avg_area_trapz]]
1314.     D_time_6M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_6M_avg_area_trapz]]
1315.     D_time_1Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_1Y_avg_area_trapz]]
1316.     D_time_2Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_2Y_avg_area_trapz]]
1317.     D_time_3Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_3Y_avg_area_trapz]]
1318.     D_time_4Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in copy_D_time_4Y_avg_area_trapz]]
1319.
1320.     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]
1321.
1322.     ND_time_before_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_before_avg_area_trapz]]
1323.     ND_time_1W_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_1W_avg_area_trapz]]
1324.     ND_time_1M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_1M_avg_area_trapz]]
1325.     ND_time_3M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_3M_avg_area_trapz]]
1326.     ND_time_6M_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_6M_avg_area_trapz]]
1327.     ND_time_1Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_1Y_avg_area_trapz]]
1328.     ND_time_2Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_2Y_avg_area_trapz]]
1329.     ND_time_3Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_3Y_avg_area_trapz]]
1330.     ND_time_4Y_normalised = [(x - all_min)/denom for x in [abs(float(x)) for x in
copy_ND_time_4Y_avg_area_trapz]]
1331.
1332.     D_average_before_det_2 = np.mean(D_time_before_normalised)
1333.     D_average_1W_det_2 = np.mean(D_time_1W_normalised)
1334.     D_average_1M_det_2 = np.mean(D_time_1M_normalised)
1335.     D_average_3M_det_2 = np.mean(D_time_3M_normalised)
1336.     D_average_6M_det_2 = np.mean(D_time_6M_normalised)
1337.     D_average_1Y_det_2 = np.mean(D_time_1Y_normalised)
1338.     D_average_2Y_det_2 = np.mean(D_time_2Y_normalised)
1339.     D_average_3Y_det_2 = np.mean(D_time_3Y_normalised)
1340.     D_average_4Y_det_2 = np.mean(D_time_4Y_normalised)
1341.
1342.     ND_average_before_det_2 = np.mean(ND_time_before_normalised)
1343.     ND_average_1W_det_2 = np.mean(ND_time_1W_normalised)
1344.     ND_average_1M_det_2 = np.mean(ND_time_1M_normalised)
1345.     ND_average_3M_det_2 = np.mean(ND_time_3M_normalised)
1346.     ND_average_6M_det_2 = np.mean(ND_time_6M_normalised)
1347.     ND_average_1Y_det_2 = np.mean(ND_time_1Y_normalised)
1348.     ND_average_2Y_det_2 = np.mean(ND_time_2Y_normalised)
1349.     ND_average_3Y_det_2 = np.mean(ND_time_3Y_normalised)
1350.     ND_average_4Y_det_2 = np.mean(ND_time_4Y_normalised)
1351.
1352.     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]
1353.     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]
1354.
1355.     plt.legend()
1356.     ax1_ALLPATIENTS4B.plot(x_array, D_averages, color = 'darkblue')
1357.     ax1_ALLPATIENTS4B.plot(x_array, ND_averages, color = 'cornflowerblue')
1358.     ax1_ALLPATIENTS4B.set_ylabel('Tremor Severity')
1359.     ax1_ALLPATIENTS4B.set_xlabel('Time')
1360.     ax1_ALLPATIENTS4B.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
1361.     plt.savefig('RESULTS\GRAPHS\AverageTremSev_AvgArea.png', bbox_inches='tight', dpi=150)
1362.     plt.show()
1363.
1364.     # END OF CODE - - - - -

```