8A: isolateSpirals.py

1. # EIE Investigation: "Which Hand?"
2. # Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
3. # ELEN4012A NOVEMBER 2022
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. # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
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
26. # Read in the PDFs and convert to .jpg- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
27. image\_array = []
28. image\_names = []
29. image\_path = []
31. counter = 0
32. rawCount = 0
33. sizeErrorCountA = 0
34. sizeErrorCountB = 0
35. sizeErrorCountC = 0
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)
54. def convertPDFtoJGP(originalImage,finalImage,dpi=200):
55. pages = convert\_from\_path(originalImage, dpi)
56. for page in pages:
57. page.save(finalImage, 'JPEG')
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
65. cropToleranceA = 0.9
66. cropToleranceB = 0.95
67. cropToleranceC = 0.9
69. MidPlaneFractionX = 0.4
70. MidPlaneFractionY = 0.5
72. # Desired dimensions of Drawings A and B (square) or Drawing C (rectangle)
73. squareDimension = 300
74. rectangleDimension = 600
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)
93. for counter,image in enumerate(image\_array):
94. print("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -")
95. print("Starting image ", image\_names[counter])
96. copied = image.copy()
98. orientedImage = copied.copy()
99. finalA = orientedImage.copy()
100. finalB = orientedImage.copy()
101. finalC = orientedImage.copy()
102. image = orientedImage.copy()
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)
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)
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]
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. # Load the pre-trained EAST text detector
129. net = cv2.dnn.readNet("Cropping/frozen\_east\_text\_detection.pb")
131. # Construct a blob from image and then perform a forward pass of the model to obtain the two output layer sets
132. blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
133. (123.68, 116.78, 103.94), swapRB=True, crop=False)
135. net.setInput(blob)
136. (scores, geometry) = net.forward(layerNames)
138. # 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
139. (numRows, numCols) = scores.shape[2:4]
140. rects = []
141. confidences = []
143. for y in range(0, numRows):
144. # Extract scores (probabilities), followed by geometrical data to derive bounding box coordinates
145. scoresData = scores[0, 0, y]
146. xData0 = geometry[0, 0, y]
147. xData1 = geometry[0, 1, y]
148. xData2 = geometry[0, 2, y]
149. xData3 = geometry[0, 3, y]
150. anglesData = geometry[0, 4, y]
152. # Loop over the number of columns
153. for x in range(0, numCols):
154. # If score does not have sufficient probability, ignore it
155. if scoresData[x] < minConf:
156. Continue
157. # Compute the offset factor as resulting feature maps will be 4x smaller than the input image
158. (offsetX, offsetY) = (x \* 4.0, y \* 4.0)
160. # Extract the rotation angle for the prediction and then compute the sin and cosine
161. angle = anglesData[x]
162. cos = np.cos(angle)
163. sin = np.sin(angle)
165. # Use the geometry volume to derive the width and height of the bounding box
166. h = xData0[x] + xData2[x]
167. w = xData1[x] + xData3[x]
169. # Compute both the starting and ending (x, y)-coordinates for the text prediction bounding box
170. endX = int(offsetX + (cos \* xData1[x]) + (sin \* xData2[x]))
171. endY = int(offsetY - (sin \* xData1[x]) + (cos \* xData2[x]))
172. startX = int(endX - w)
173. startY = int(endY - h)
175. # Add the bounding box coordinates and probability score to respective lists
176. rects.append((startX, startY, endX, endY))
177. confidences.append(scoresData[x])
179. # Apply non-maxima suppression to suppress weak, overlapping bounding boxes
180. boxes = non\_max\_suppression(np.array(rects), probs=confidences)
182. xStart = []
183. xEnd = []
184. yStart = []
185. yEnd = []
187. # Loop over the bounding boxes
188. for (startX, startY, endX, endY) in boxes:
189. # Scale the bounding box coordinates based on the respective ratios
190. startX = int(startX \* rW)
191. startY = int(startY \* rH)
192. endX = int(endX \* rW)
193. endY = int(endY \* rH)
195. # Draw the bounding box on the image
196. xStart.append(startX) # vector of x coords
197. yStart.append(startY)
198. xEnd.append(endX)
199. yEnd.append(endY)
201. (height,width) = finalA.shape[:2]
202. orientation = 0
204. # Set array position of the Drawing's to be -1 (i.e. not found)
205. A\_position = -1
206. B\_position = -1
207. C\_position = -1
209. # If the coordinates match where a drawing should be, save that position to the corresponding drawing's X\_position variable
210. for i in range(0, len(xStart)):
211. if xStart[i] < 400:
212. if yEnd[i] < total\_h/2:
213. if A\_position == -1:
214. A\_position = i
215. else:
216. if xEnd[i] - xStart[i] < 100:
217. A\_position = i
218. elif yEnd[i] < 1300 and xEnd[i] < 500:
219. C\_position = i
220. elif yStart[i] < total\_h/2:
221. if B\_position == -1:
222. B\_position = i
223. else:
224. if xEnd[i] - xStart[i] < 100:
225. B\_position = i
227. # In case a drawing is not found, use the average values instead
228. width = np.mean(array\_width)
229. xStart\_A = np.mean(array\_xStartA)
230. xStart\_B = np.mean(array\_xStartB)
231. xStart\_C = np.mean(array\_xStartC)
232. yEnd\_A = np.mean(array\_yEndA)
233. yEnd\_B = np.mean(array\_yEndB)
234. yEnd\_C = np.mean(array\_yEndC)

237. # Check if the drawing was found, and save the coordinates respectively
238. if A\_position != -1:
239. xStart\_A = xStart[A\_position]
240. yEnd\_A = yEnd[A\_position]
241. array\_xStartA.append(xStart\_A)
242. array\_yEndA.append(yEnd\_A)
243. else:
244. print("THERE IS NO A TEXT DETECTED - USING AVERAGE VALUES")
246. if B\_position != -1:
247. xStart\_B = xStart[B\_position]
248. yEnd\_B = yEnd[B\_position]
249. array\_xStartB.append(xStart\_B)
250. array\_yEndB.append(yEnd\_B)
251. else:
252. print("THERE IS NO B TEXT DETECTED - USING AVERAGE VALUES")
254. if C\_position != -1:
255. xStart\_C = xStart[C\_position]
256. yEnd\_C = yEnd[C\_position]
257. array\_xStartC.append(xStart\_C)
258. array\_yEndC.append(yEnd\_C)
259. else:
260. print("THERE IS NO C TEXT DETECTED - USING AVERAGE VALUES")
262. # Using other detected Drawings to find better coordinates for missing Drawings
263. if A\_position == -1 and B\_position != -1:
264. xStart\_A = xStart\_B-total\_w/2
265. yEnd\_A = yEnd[B\_position]
267. if B\_position == -1 and A\_position != -1:
268. xStart\_B = xStart\_A\*0.5 + total\_w/2
269. yEnd\_B = yEnd[A\_position]
271. if A\_position == -1 and B\_position == -1:
272. xStart\_A = xStart\_C
273. xStart\_B = xStart\_A\*0.5 + total\_w/2
275. yEnd\_A = yStart[C\_position] - total\_h/3
276. yEnd\_B = yStart[C\_position] - total\_h/3
278. # Calculate the width of Drawing A, either with average or coordinate values
279. width = (xStart\_B\*cropToleranceA-xStart\_A)
280. array\_width.append(width)
282. # CROPPING OUT SPIRAL A - - - - - - - - - - - #
283. try:
284. finalA = cv2.cvtColor(finalA, cv2.COLOR\_BGR2GRAY)
285. finalA = finalA[int(yEnd\_A):int(yEnd\_A+width), int(xStart\_A):int(xStart\_A + width)]
286. finalA = imutils.resize(finalA, width=squareDimension)
288. new\_image\_path = str(image\_path[counter])+'/DrawingA/'+str(image\_names[counter])+"\_A"+".jpg"
289. cv2.imwrite(new\_image\_path, finalA)
290. except Exception as e:
291. print("Unable to save to file A as image size is empty: ",str(e))
293. # CROPPING OUT SPIRAL B - - - - - - - - - - - #
294. try:
295. finalB = cv2.cvtColor(finalB, cv2.COLOR\_BGR2GRAY)
296. finalB = finalB[ int(yEnd\_B) : int((yEnd\_B+width)), int(xStart\_B\*cropToleranceB): int((xStart\_B+width)\*cropToleranceB)]
297. finalB = imutils.resize(finalB, width=squareDimension)
299. new\_image\_path = str(image\_path[counter])+'/DrawingB/'+str(image\_names[counter])+"\_B"+".jpg"
300. cv2.imwrite(new\_image\_path, finalB)
301. except Exception as e:
302. print("Unable to save to file B as image size is empty: ",str(e))
304. # CROPPING SPIRAL C - - - - - - - - - - - #
305. try:
306. finalC = cv2.cvtColor(finalC, cv2.COLOR\_BGR2GRAY)
307. finalC = finalC[ int(yEnd\_C) : int(yEnd\_C+0.75\*width), int(xStart\_C): int(xStart\_B+width)]
308. finalC = imutils.resize(finalC, width=rectangleDimension)
310. new\_image\_path = str(image\_path[counter])+'/DrawingC/'+str(image\_names[counter])+"\_C"+".jpg"
311. cv2.imwrite(new\_image\_path, finalC)
312. except Exception as e:
313. print("Unable to save to file C as image size is empty: ",str(e))
314. print("Finished image ", image\_names[counter])
316. # 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
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. # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
16. # IMPORTS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
17. import imutils
18. import cv2
19. import glob
20. import os
21. from PIL import Image
22. import numpy as np
24. # Read in the .jpgs and save into an image array- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
25. image\_array = []
26. image\_names = []
27. image\_path = []
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)
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)
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[0])
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]
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 = []
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)
70. c = c.astype("float")
71. c \*= ratio # multiply the contour (x, y)-coordinates by the resize ratio
72. c = c.astype("int")
74. rect = cv2.minAreaRect(c)
75. box = cv2.boxPoints(rect)
76. box = np.int0(box)
78. b\_counter = 0
79. b\_left\_1 = -1
80. b\_left\_2 = -1
81. b\_right\_1 = -1
82. b\_right\_2 = -1
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
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])
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)
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)
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")
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")
141. M = cv2.getPerspectiveTransform(new\_box\_float, straightened\_array) # the perspective transformation matrix
143. # directly warp the rotated rectangle to get the straightened rectangle
144. warped = cv2.warpPerspective(image.copy(), M, (new\_width, new\_height))
146. (warped\_H,warped\_W) = warped.shape[:2]
147. if warped\_H > warped\_W: warped = cv2.rotate(warped, cv2.ROTATE\_90\_CLOCKWISE)
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]
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")
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
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
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
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
45. def get\_time\_period(image\_name):
46. image\_name = str.lower(image\_name)
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()]
52. if period.find("one") != -1:
53. output = "1W"
54. else:
55. output = str(time[0]) + "W"
56. return output
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()]
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")
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()]
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))
87. if image\_name.find("before") != -1:
88. return "before"
89. return -1
91. def is\_treated\_hand(image\_name, patient\_number):
92. image\_name = str.lower(image\_name)
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
115. def is\_dominant\_hand(image\_name, patient\_number):
116. image\_name = str.lower(image\_name)
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
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
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("#", "")
150. for filename in glob.glob(foldername + "\DrawingC\Rectangles\\*.jpg"):
151. arrayName = os.path.basename(filename)
152. arrayName = arrayName.replace(".jpg","")
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)
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
173. results\_array = [[-1 for x in range(len(Results))] for y in range(file\_count)]
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", "")
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]
188. coordinates = np.argwhere(image < 0.9)
189. # fig, (axs\_original, axs\_fft, axs\_ifft, axs\_area) = plt.subplots(4)
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)
195. x = np.array(x\_tuple).tolist()
196. y = np.array(y\_tuple).tolist()
197. z = np.array(z\_tuple).tolist()
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
207. temp\_y = y[j]
208. y[j] = y[j+1]
209. y[j+1] = temp\_y
211. # GRAPH 1: ORIGINAL
212. # axs\_original.set\_title("ORIGINAL GRAPH: " + str(temp\_title))
213. # axs\_original.plot(y, x)
215. average\_x = np.mean(x)
216. new\_array\_x = []
217. new\_array\_y = []
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)
224. data\_step = 0.1
225. n = len(new\_array\_y)
226. yf = rfft(new\_array\_x)
227. xf = rfftfreq(n, data\_step)
229. yf\_abs = np.abs(yf)
230. yf\_max = np.max(yf\_abs)
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)
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))
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))
252. multiplier = multiplier+5
254. # # GRAPH 3: IFFT
255. # axs\_ifft.set\_title("IFFT: " + str(temp\_title))
256. # axs\_ifft.plot(new\_array\_y, new\_array\_x)
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)
265. # axs\_ifft.set\_xlim(0, 600)
266. # axs\_ifft.set\_ylim(-20, 20)
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)
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
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])
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 = []
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])
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]))
305. average\_peakmin\_distance = np.mean(peakmin\_distance\_array)
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

312. # # GRAPH 4: AREA GRAPH
313. # axs\_area.set\_title("AREA UNDER CURVE: " + str(temp\_title))
314. # axs\_area.fill\_between(new\_array\_y, new\_array\_x\_ABS, color="grey")
315. # axs\_area.plot(new\_array\_y, new\_array\_x\_ABS)
316. # axs\_area.set\_xlim(0, 600)
317. # axs\_area.set\_ylim(0, 20)
319. results\_array[image\_counter][Results.AREA\_TRAPZ.value] = total\_area\_trapz\_x
320. results\_array[image\_counter][Results.MAX.value] = max
321. results\_array[image\_counter][Results.STDDEV.value] = std
323. average\_areas = []
324. temp\_counter = 1
325. max\_new\_array\_y = np.max(new\_array\_y)
326. percent\_of\_xaxis = int(5/100\*max\_new\_array\_y)
327. array\_iterator = 0
329. for i in range(percent\_of\_xaxis, max\_new\_array\_y, percent\_of\_xaxis):
330. temp\_array\_x = []
331. temp\_array\_y = []
332. before\_array\_iterator = array\_iterator
334. while new\_array\_y[array\_iterator] < i:
335. temp\_array\_x.append(new\_array\_x[array\_iterator])
336. temp\_array\_y.append(new\_array\_y[array\_iterator])
337. array\_iterator = array\_iterator + 1
339. if array\_iterator == before\_array\_iterator:
340. temp\_array\_x.append(new\_array\_x[array\_iterator-1])
341. temp\_array\_y.append(new\_array\_y[array\_iterator-1])
342. temp\_array\_x.append(new\_array\_x[array\_iterator])
343. temp\_array\_y.append(new\_array\_y[array\_iterator])
345. temp\_area\_x = trapz(temp\_array\_x, temp\_array\_y)
346. average\_areas.append(temp\_area\_x)
347. temp\_counter = temp\_counter + 1
349. # Quantile values of the data
350. avg\_avg\_area = np.mean(average\_areas)
351. avg\_min, avg\_q1, avg\_q2, avg\_q3, avg\_max = np.quantile(average\_areas, [0, 0.25, 0.5, 0.75, 1])
352. avg\_iqr = avg\_q3-avg\_q1
353. avg\_std = np.std(average\_areas)
355. results\_array[image\_counter][Results.AVG\_AREA\_TRAPZ.value] = avg\_avg\_area
356. results\_array[image\_counter][Results.AVG\_AREA\_MAX.value] = avg\_max
357. results\_array[image\_counter][Results.AVG\_AREA\_STDDEV.value] = avg\_std
359. results\_array[image\_counter][Results.NUM\_PEAKS.value] = num\_peaks
360. results\_array[image\_counter][Results.AVG\_PEAK\_DIST.value] = average\_peakmin\_distance
361. except:
362. print("ERROR: \t" + str(image\_patient\_number[image\_counter]) + "\t -\t " + str(image\_names[image\_counter]))
364. # plt.tight\_layout()
365. # plt.show()
367. # SAVING THE DATA:
368. table\_columns = []
369. for r in Results:
370. table\_columns.append(str(r.name))
371. df = pd.DataFrame(np.array(results\_array), columns=table\_columns)
372. print("- - - - - RESULTS ARRAY - - - - -")
373. display(df)
374. df.to\_csv('Method2/Results.csv', index=False)
376. # END OF CODE - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

8D: analyseResults.py

1. # EIE Investigation: "Which Hand?"
2. # Jesse van der Merwe (1829172) and Robyn Gebbie (2127777)
3. # ELEN4012A NOVEMBER 2022
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. # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
16. # IMPORTS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
17. import csv
18. from matplotlib import pyplot as plt
19. import numpy as np
20. import pandas as pd
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 = []
28. patient\_area\_trapz = []
29. patient\_avg\_area\_trapz = []
30. patient\_avg\_std\_dev\_array = []
32. patient\_num\_peaks = []
33. patient\_avg\_peak\_dist = []
34. patient\_determinator\_1 = []
35. patient\_determinator\_2 = []
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'])
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'])
49. num\_peaks = line['NUM\_PEAKS']
50. avg\_peak\_dist = line['AVG\_PEAK\_DIST']
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))
57. n = len(patient\_number\_array)
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
66. temp = patient\_dominant\_hand[j]
67. patient\_dominant\_hand[j] = patient\_dominant\_hand[j+1]
68. patient\_dominant\_hand[j+1] = temp
70. temp = patient\_treated\_hand[j]
71. patient\_treated\_hand[j] = patient\_treated\_hand[j+1]
72. patient\_treated\_hand[j+1] = temp
74. temp = patient\_time\_array[j]
75. patient\_time\_array[j] = patient\_time\_array[j+1]
76. patient\_time\_array[j+1] = temp
78. temp = patient\_area\_trapz[j]
79. patient\_area\_trapz[j] = patient\_area\_trapz[j+1]
80. patient\_area\_trapz[j+1] = temp
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
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
90. temp = patient\_num\_peaks[j]
91. patient\_num\_peaks[j] = patient\_num\_peaks[j+1]
92. patient\_num\_peaks[j+1] = temp
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
98. temp = patient\_determinator\_1[j]
99. patient\_determinator\_1[j] = patient\_determinator\_1[j+1]
100. patient\_determinator\_1[j+1] = temp
102. temp = patient\_determinator\_2[j]
103. patient\_determinator\_2[j] = patient\_determinator\_2[j+1]
104. patient\_determinator\_2[j+1] = temp
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
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 = []
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)
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
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 = []
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)
154. max\_std\_dev = 0.0
155. min\_std\_dev = 100.0



160. for i in range(0, n):
161. if float(patient\_avg\_std\_dev\_array[i]) > float(max\_std\_dev):
162. max\_std\_dev = patient\_avg\_std\_dev\_array[i]
164. if float(patient\_avg\_std\_dev\_array[i])<float(min\_std\_dev) and float(patient\_avg\_std\_dev\_array[i])>=float(0.0):
165. min\_std\_dev = patient\_avg\_std\_dev\_array[i]
167. if patient\_treated\_hand[i] == 'True':
168. if int(patient\_number\_array[i]) != int(D\_patient\_number\_array\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev]):
169. D\_patient\_number\_array\_avg\_std\_dev.append(patient\_number\_array[i])
170. D\_patient\_counter\_avg\_std\_dev = D\_patient\_counter\_avg\_std\_dev + 1
171. D\_time\_before\_avg\_std\_dev.append(0)
172. D\_time\_1W\_avg\_std\_dev.append(0)
173. D\_time\_1M\_avg\_std\_dev.append(0)
174. D\_time\_3M\_avg\_std\_dev.append(0)
175. D\_time\_6M\_avg\_std\_dev.append(0)
176. D\_time\_1Y\_avg\_std\_dev.append(0)
177. D\_time\_2Y\_avg\_std\_dev.append(0)
178. D\_time\_3Y\_avg\_std\_dev.append(0)
179. D\_time\_4Y\_avg\_std\_dev.append(0)
181. if patient\_time\_array[i] == "before":
182. D\_time\_before\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
183. elif patient\_time\_array[i] == "1W":
184. D\_time\_1W\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
185. elif patient\_time\_array[i] == "1M":
186. D\_time\_1M\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
187. elif patient\_time\_array[i] == "3M":
188. D\_time\_3M\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
189. elif patient\_time\_array[i] == "6M":
190. D\_time\_6M\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
191. elif patient\_time\_array[i] == "1Y":
192. D\_time\_1Y\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
193. elif patient\_time\_array[i] == "2Y":
194. D\_time\_2Y\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
195. elif patient\_time\_array[i] == "3Y":
196. D\_time\_3Y\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
197. elif patient\_time\_array[i] == "4Y":
198. D\_time\_4Y\_avg\_std\_dev[D\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
200. else:
201. if int(patient\_number\_array[i])!=int(ND\_patient\_number\_array\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev]):
202. ND\_patient\_number\_array\_avg\_std\_dev.append(patient\_number\_array[i])
203. ND\_patient\_counter\_avg\_std\_dev = ND\_patient\_counter\_avg\_std\_dev + 1
204. ND\_time\_before\_avg\_std\_dev.append(0)
205. ND\_time\_1W\_avg\_std\_dev.append(0)
206. ND\_time\_1M\_avg\_std\_dev.append(0)
207. ND\_time\_3M\_avg\_std\_dev.append(0)
208. ND\_time\_6M\_avg\_std\_dev.append(0)
209. ND\_time\_1Y\_avg\_std\_dev.append(0)
210. ND\_time\_2Y\_avg\_std\_dev.append(0)
211. ND\_time\_3Y\_avg\_std\_dev.append(0)
212. ND\_time\_4Y\_avg\_std\_dev.append(0)
214. if patient\_time\_array[i] == "before":
215. ND\_time\_before\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
216. elif patient\_time\_array[i] == "1W":
217. ND\_time\_1W\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
218. elif patient\_time\_array[i] == "1M":
219. ND\_time\_1M\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
220. elif patient\_time\_array[i] == "3M":
221. ND\_time\_3M\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
222. elif patient\_time\_array[i] == "6M":
223. ND\_time\_6M\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
224. elif patient\_time\_array[i] == "1Y":
225. ND\_time\_1Y\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
226. elif patient\_time\_array[i] == "2Y":
227. ND\_time\_2Y\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
228. elif patient\_time\_array[i] == "3Y":
229. ND\_time\_3Y\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
230. elif patient\_time\_array[i] == "4Y":
231. ND\_time\_4Y\_avg\_std\_dev[ND\_patient\_counter\_avg\_std\_dev] = patient\_avg\_std\_dev\_array[i]
233. 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 }
234. D\_df\_avg\_std\_dev = pd.DataFrame(D\_data\_avg\_std\_dev)
235. D\_df\_avg\_std\_dev.to\_csv('RESULTS/D\_AvgStdDev.csv', index=False)
237. 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 }
238. ND\_df\_avg\_std\_dev = pd.DataFrame(ND\_data\_avg\_std\_dev)
239. ND\_df\_avg\_std\_dev.to\_csv('RESULTS/ND\_AvgStdDev.csv', index=False)
241. #------------------------------------
243. D\_patient\_number\_array\_avg\_area\_trapz = []
244. D\_patient\_number\_array\_avg\_area\_trapz.append(1)
245. D\_patient\_counter\_avg\_area\_trapz = 0
247. D\_time\_before\_avg\_area\_trapz = []
248. D\_time\_1W\_avg\_area\_trapz = []
249. D\_time\_1M\_avg\_area\_trapz = []
250. D\_time\_3M\_avg\_area\_trapz = []
251. D\_time\_6M\_avg\_area\_trapz = []
252. D\_time\_1Y\_avg\_area\_trapz = []
253. D\_time\_2Y\_avg\_area\_trapz = []
254. D\_time\_3Y\_avg\_area\_trapz = []
255. D\_time\_4Y\_avg\_area\_trapz = []
257. D\_time\_before\_avg\_area\_trapz.append(0)
258. D\_time\_1W\_avg\_area\_trapz.append(0)
259. D\_time\_1M\_avg\_area\_trapz.append(0)
260. D\_time\_3M\_avg\_area\_trapz.append(0)
261. D\_time\_6M\_avg\_area\_trapz.append(0)
262. D\_time\_1Y\_avg\_area\_trapz.append(0)
263. D\_time\_2Y\_avg\_area\_trapz.append(0)
264. D\_time\_3Y\_avg\_area\_trapz.append(0)
265. D\_time\_4Y\_avg\_area\_trapz.append(0)
267. ND\_patient\_number\_array\_avg\_area\_trapz = []
268. ND\_patient\_number\_array\_avg\_area\_trapz.append(1)
269. ND\_patient\_counter\_avg\_area\_trapz = 0
271. ND\_time\_before\_avg\_area\_trapz = []
272. ND\_time\_1W\_avg\_area\_trapz = []
273. ND\_time\_1M\_avg\_area\_trapz = []
274. ND\_time\_3M\_avg\_area\_trapz = []
275. ND\_time\_6M\_avg\_area\_trapz = []
276. ND\_time\_1Y\_avg\_area\_trapz = []
277. ND\_time\_2Y\_avg\_area\_trapz = []
278. ND\_time\_3Y\_avg\_area\_trapz = []
279. ND\_time\_4Y\_avg\_area\_trapz = []
281. ND\_time\_before\_avg\_area\_trapz.append(0)
282. ND\_time\_1W\_avg\_area\_trapz.append(0)
283. ND\_time\_1M\_avg\_area\_trapz.append(0)
284. ND\_time\_3M\_avg\_area\_trapz.append(0)
285. ND\_time\_6M\_avg\_area\_trapz.append(0)
286. ND\_time\_1Y\_avg\_area\_trapz.append(0)
287. ND\_time\_2Y\_avg\_area\_trapz.append(0)
288. ND\_time\_3Y\_avg\_area\_trapz.append(0)
289. ND\_time\_4Y\_avg\_area\_trapz.append(0)
291. max\_area = 0.0
292. min\_area = 100.0
294. for i in range(0, n):
295. if float(patient\_avg\_area\_trapz[i]) > float(max\_area):
296. max\_area = patient\_avg\_area\_trapz[i]
298. if float(patient\_avg\_area\_trapz[i]) < float(min\_area) and float(patient\_avg\_area\_trapz[i]) >= float(0.0):
299. min\_area = patient\_avg\_area\_trapz[i]
301. if patient\_treated\_hand[i] == 'True':
302. if int(patient\_number\_array[i]) != int(D\_patient\_number\_array\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz]):
303. D\_patient\_number\_array\_avg\_area\_trapz.append(patient\_number\_array[i])
304. D\_patient\_counter\_avg\_area\_trapz = D\_patient\_counter\_avg\_area\_trapz + 1
305. D\_time\_before\_avg\_area\_trapz.append(0)
306. D\_time\_1W\_avg\_area\_trapz.append(0)
307. D\_time\_1M\_avg\_area\_trapz.append(0)
308. D\_time\_3M\_avg\_area\_trapz.append(0)
309. D\_time\_6M\_avg\_area\_trapz.append(0)
310. D\_time\_1Y\_avg\_area\_trapz.append(0)
311. D\_time\_2Y\_avg\_area\_trapz.append(0)
312. D\_time\_3Y\_avg\_area\_trapz.append(0)
313. D\_time\_4Y\_avg\_area\_trapz.append(0)

316. if patient\_time\_array[i] == "before":
317. D\_time\_before\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
318. elif patient\_time\_array[i] == "1W":
319. D\_time\_1W\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
320. elif patient\_time\_array[i] == "1M":
321. D\_time\_1M\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
322. elif patient\_time\_array[i] == "3M":
323. D\_time\_3M\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
324. elif patient\_time\_array[i] == "6M":
325. D\_time\_6M\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
326. elif patient\_time\_array[i] == "1Y":
327. D\_time\_1Y\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
328. elif patient\_time\_array[i] == "2Y":
329. D\_time\_2Y\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
330. elif patient\_time\_array[i] == "3Y":
331. D\_time\_3Y\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
332. elif patient\_time\_array[i] == "4Y":
333. D\_time\_4Y\_avg\_area\_trapz[D\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
335. else:
336. if int(patient\_number\_array[i]) != int(ND\_patient\_number\_array\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz]):
337. ND\_patient\_number\_array\_avg\_area\_trapz.append(patient\_number\_array[i])
338. ND\_patient\_counter\_avg\_area\_trapz = ND\_patient\_counter\_avg\_area\_trapz + 1
339. ND\_time\_before\_avg\_area\_trapz.append(0)
340. ND\_time\_1W\_avg\_area\_trapz.append(0)
341. ND\_time\_1M\_avg\_area\_trapz.append(0)
342. ND\_time\_3M\_avg\_area\_trapz.append(0)
343. ND\_time\_6M\_avg\_area\_trapz.append(0)
344. ND\_time\_1Y\_avg\_area\_trapz.append(0)
345. ND\_time\_2Y\_avg\_area\_trapz.append(0)
346. ND\_time\_3Y\_avg\_area\_trapz.append(0)
347. ND\_time\_4Y\_avg\_area\_trapz.append(0)
349. if patient\_time\_array[i] == "before":
350. ND\_time\_before\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
351. elif patient\_time\_array[i] == "1W":
352. ND\_time\_1W\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
353. elif patient\_time\_array[i] == "1M":
354. ND\_time\_1M\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
355. elif patient\_time\_array[i] == "3M":
356. ND\_time\_3M\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
357. elif patient\_time\_array[i] == "6M":
358. ND\_time\_6M\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
359. elif patient\_time\_array[i] == "1Y":
360. ND\_time\_1Y\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
361. elif patient\_time\_array[i] == "2Y":
362. ND\_time\_2Y\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
363. elif patient\_time\_array[i] == "3Y":
364. ND\_time\_3Y\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
365. elif patient\_time\_array[i] == "4Y":
366. ND\_time\_4Y\_avg\_area\_trapz[ND\_patient\_counter\_avg\_area\_trapz] = patient\_avg\_area\_trapz[i]
368. D\_data\_avg\_area\_trapz = {'Patient': D\_patient\_number\_array\_avg\_area\_trapz, 'Before' : D\_time\_before\_avg\_area\_trapz, '1 Week' : D\_time\_1W\_avg\_area\_trapz, '1 Month' : D\_time\_1M\_avg\_area\_trapz, '3 Months': D\_time\_3M\_avg\_area\_trapz, '6 Months': D\_time\_6M\_avg\_area\_trapz, '1 Year': D\_time\_1Y\_avg\_area\_trapz, '2 Years': D\_time\_2Y\_avg\_area\_trapz, '3 Years':D\_time\_3Y\_avg\_area\_trapz, '4 Years':D\_time\_4Y\_avg\_area\_trapz }
369. D\_df\_avg\_area\_trapz = pd.DataFrame(D\_data\_avg\_area\_trapz)
370. D\_df\_avg\_area\_trapz.to\_csv('RESULTS/D\_AreaTrapz.csv', index=False)
372. ND\_data\_avg\_area\_trapz = {'Patient': ND\_patient\_number\_array\_avg\_area\_trapz, 'Before' : ND\_time\_before\_avg\_area\_trapz, '1 Week' : ND\_time\_1W\_avg\_area\_trapz, '1 Month' : ND\_time\_1M\_avg\_area\_trapz, '3 Months': ND\_time\_3M\_avg\_area\_trapz, '6 Months': ND\_time\_6M\_avg\_area\_trapz, '1 Year': ND\_time\_1Y\_avg\_area\_trapz, '2 Years': ND\_time\_2Y\_avg\_area\_trapz, '3 Years':ND\_time\_3Y\_avg\_area\_trapz, '4 Years':ND\_time\_4Y\_avg\_area\_trapz }
373. ND\_df\_avg\_area\_trapz = pd.DataFrame(ND\_data\_avg\_area\_trapz)
374. ND\_df\_avg\_area\_trapz.to\_csv('RESULTS/ND\_AreaTrapz.csv', index=False)
376. #------------------------------------------------
378. D\_patient\_number\_array\_det\_2 = []
379. D\_patient\_number\_array\_det\_2.append(1)
380. D\_patient\_counter\_det\_2 = 0
382. D\_time\_before\_det\_2 = []
383. D\_time\_1W\_det\_2 = []
384. D\_time\_1M\_det\_2 = []
385. D\_time\_3M\_det\_2 = []
386. D\_time\_6M\_det\_2 = []
387. D\_time\_1Y\_det\_2 = []
388. D\_time\_2Y\_det\_2 = []
389. D\_time\_3Y\_det\_2 = []
390. D\_time\_4Y\_det\_2 = []
392. D\_time\_before\_det\_2.append(0)
393. D\_time\_1W\_det\_2.append(0)
394. D\_time\_1M\_det\_2.append(0)
395. D\_time\_3M\_det\_2.append(0)
396. D\_time\_6M\_det\_2.append(0)
397. D\_time\_1Y\_det\_2.append(0)
398. D\_time\_2Y\_det\_2.append(0)
399. D\_time\_3Y\_det\_2.append(0)
400. D\_time\_4Y\_det\_2.append(0)
402. ND\_patient\_number\_array\_det\_2 = []
403. ND\_patient\_number\_array\_det\_2.append(1)
404. ND\_patient\_counter\_det\_2 = 0
406. ND\_time\_before\_det\_2 = []
407. ND\_time\_1W\_det\_2 = []
408. ND\_time\_1M\_det\_2 = []
409. ND\_time\_3M\_det\_2 = []
410. ND\_time\_6M\_det\_2 = []
411. ND\_time\_1Y\_det\_2 = []
412. ND\_time\_2Y\_det\_2 = []
413. ND\_time\_3Y\_det\_2 = []
414. ND\_time\_4Y\_det\_2 = []
416. ND\_time\_before\_det\_2.append(0)
417. ND\_time\_1W\_det\_2.append(0)
418. ND\_time\_1M\_det\_2.append(0)
419. ND\_time\_3M\_det\_2.append(0)
420. ND\_time\_6M\_det\_2.append(0)
421. ND\_time\_1Y\_det\_2.append(0)
422. ND\_time\_2Y\_det\_2.append(0)
423. ND\_time\_3Y\_det\_2.append(0)
424. ND\_time\_4Y\_det\_2.append(0)
426. D\_patient\_hand\_array\_det\_2 = []
427. ND\_patient\_hand\_array\_det\_2 = []
429. max\_det\_2 = 0.0
430. min\_det\_2 = 100.0
432. for i in range(0, n):
433. if float(patient\_determinator\_2[i]) > float(max\_det\_2):
434. max\_det\_2 = patient\_determinator\_2[i]
436. if float(patient\_determinator\_2[i]) < float(min\_det\_2) and float(patient\_determinator\_2[i]) >= float(0.0):
437. min\_det\_2 = patient\_determinator\_2[i]
439. if patient\_treated\_hand[i] == 'True':
440. if int(patient\_number\_array[i]) != int(D\_patient\_number\_array\_det\_2[D\_patient\_counter\_det\_2]):
441. D\_patient\_hand\_array\_det\_2.append(patient\_dominant\_hand[i])
442. D\_patient\_number\_array\_det\_2.append(patient\_number\_array[i])
443. D\_patient\_counter\_det\_2 = D\_patient\_counter\_det\_2 + 1
444. D\_time\_before\_det\_2.append(0)
445. D\_time\_1W\_det\_2.append(0)
446. D\_time\_1M\_det\_2.append(0)
447. D\_time\_3M\_det\_2.append(0)
448. D\_time\_6M\_det\_2.append(0)
449. D\_time\_1Y\_det\_2.append(0)
450. D\_time\_2Y\_det\_2.append(0)
451. D\_time\_3Y\_det\_2.append(0)
452. D\_time\_4Y\_det\_2.append(0)
454. if patient\_time\_array[i] == "before":
455. D\_time\_before\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
456. elif patient\_time\_array[i] == "1W":
457. D\_time\_1W\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
458. elif patient\_time\_array[i] == "1M":
459. D\_time\_1M\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
460. elif patient\_time\_array[i] == "3M":
461. D\_time\_3M\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
462. elif patient\_time\_array[i] == "6M":
463. D\_time\_6M\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
464. elif patient\_time\_array[i] == "1Y":
465. D\_time\_1Y\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
466. elif patient\_time\_array[i] == "2Y":
467. D\_time\_2Y\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
468. elif patient\_time\_array[i] == "3Y":
469. D\_time\_3Y\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
470. elif patient\_time\_array[i] == "4Y":
471. D\_time\_4Y\_det\_2[D\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
473. else:
474. if int(patient\_number\_array[i]) != int(ND\_patient\_number\_array\_det\_2[ND\_patient\_counter\_det\_2]):
475. ND\_patient\_hand\_array\_det\_2.append(patient\_dominant\_hand[i])
476. ND\_patient\_number\_array\_det\_2.append(patient\_number\_array[i])
477. ND\_patient\_counter\_det\_2 = ND\_patient\_counter\_det\_2 + 1
478. ND\_time\_before\_det\_2.append(0)
479. ND\_time\_1W\_det\_2.append(0)
480. ND\_time\_1M\_det\_2.append(0)
481. ND\_time\_3M\_det\_2.append(0)
482. ND\_time\_6M\_det\_2.append(0)
483. ND\_time\_1Y\_det\_2.append(0)
484. ND\_time\_2Y\_det\_2.append(0)
485. ND\_time\_3Y\_det\_2.append(0)
486. ND\_time\_4Y\_det\_2.append(0)
488. if patient\_time\_array[i] == "before":
489. ND\_time\_before\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
490. elif patient\_time\_array[i] == "1W":
491. ND\_time\_1W\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
492. elif patient\_time\_array[i] == "1M":
493. ND\_time\_1M\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
494. elif patient\_time\_array[i] == "3M":
495. ND\_time\_3M\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
496. elif patient\_time\_array[i] == "6M":
497. ND\_time\_6M\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
498. elif patient\_time\_array[i] == "1Y":
499. ND\_time\_1Y\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
500. elif patient\_time\_array[i] == "2Y":
501. ND\_time\_2Y\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
502. elif patient\_time\_array[i] == "3Y":
503. ND\_time\_3Y\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
504. elif patient\_time\_array[i] == "4Y":
505. ND\_time\_4Y\_det\_2[ND\_patient\_counter\_det\_2] = patient\_determinator\_2[i]
507. D\_data\_det\_2 = {'Patient': D\_patient\_number\_array\_det\_2, 'Before' : D\_time\_before\_det\_2, '1 Week' : D\_time\_1W\_det\_2, '1 Month' : D\_time\_1M\_det\_2, '3 Months': D\_time\_3M\_det\_2, '6 Months': D\_time\_6M\_det\_2, '1 Year': D\_time\_1Y\_det\_2, '2 Years': D\_time\_2Y\_det\_2, '3 Years':D\_time\_3Y\_det\_2, '4 Years':D\_time\_4Y\_det\_2 }
508. D\_df\_det\_2 = pd.DataFrame(D\_data\_det\_2)
509. D\_df\_det\_2.to\_csv('RESULTS/D\_Determinant2.csv', index=False)
511. ND\_data\_det\_2 = {'Patient': ND\_patient\_number\_array\_det\_2, 'Before' : ND\_time\_before\_det\_2, '1 Week' : ND\_time\_1W\_det\_2, '1 Month' : ND\_time\_1M\_det\_2, '3 Months': ND\_time\_3M\_det\_2, '6 Months': ND\_time\_6M\_det\_2, '1 Year': ND\_time\_1Y\_det\_2, '2 Years': ND\_time\_2Y\_det\_2, '3 Years':ND\_time\_3Y\_det\_2, '4 Years':ND\_time\_4Y\_det\_2 }
512. ND\_df\_det\_2 = pd.DataFrame(ND\_data\_det\_2)
513. ND\_df\_det\_2.to\_csv('RESULTS/ND\_Determinant2.csv', index=False)
515. # GRAPHS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
516. def checkImproved(before, arr):
517. improved = 0
518. both = 0
519. avg\_amount = 0
520. avg\_diff = 0
521. avg\_diff\_improved = 0
523. for i in range(len(before)):
524. abs\_before = abs(float(before[i]))
525. abs\_arr = abs(float(arr[i]))
526. if (abs\_before != 0) and (abs\_arr != 0) and (abs\_before != -1) and (abs\_arr != -1):
527. both += 1
528. avg\_diff += float(abs\_before) - float(abs\_arr)
530. if float(abs\_arr) < float(abs\_before):
531. improved += 1
532. avg\_amount += float(abs\_arr)
533. avg\_diff\_improved += float(abs\_before) - float(abs\_arr)
535. avg\_diff = avg\_diff/both
536. avg\_diff\_improved = avg\_diff\_improved/improved
537. avg\_amount = avg\_amount/improved
538. return both, improved, avg\_amount, avg\_diff, avg\_diff\_improved
540. D\_both\_1W, D\_improved\_1W, D\_avg\_amount\_1W, D\_avg\_diff\_1W, D\_avg\_diff\_improved\_1W = checkImproved(D\_time\_before\_det\_2, D\_time\_1W\_det\_2)
541. D\_both\_1M, D\_improved\_1M, D\_avg\_amount\_1M, D\_avg\_diff\_1M, D\_avg\_diff\_improved\_1M = checkImproved(D\_time\_before\_det\_2, D\_time\_1M\_det\_2)
542. D\_both\_3M, D\_improved\_3M, D\_avg\_amount\_3M, D\_avg\_diff\_3M, D\_avg\_diff\_improved\_3M = checkImproved(D\_time\_before\_det\_2, D\_time\_3M\_det\_2)
543. D\_both\_6M, D\_improved\_6M, D\_avg\_amount\_6M, D\_avg\_diff\_6M, D\_avg\_diff\_improved\_6M = checkImproved(D\_time\_before\_det\_2, D\_time\_6M\_det\_2)
544. D\_both\_1Y, D\_improved\_1Y, D\_avg\_amount\_1Y, D\_avg\_diff\_1Y, D\_avg\_diff\_improved\_1Y = checkImproved(D\_time\_before\_det\_2, D\_time\_1Y\_det\_2)
545. 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)
546. 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)
547. 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)
549. 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]
550. 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]
551. 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]
552. 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]
553. 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]
554. 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]
556. 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)
557. 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)
558. 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)
559. 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)
560. 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)
561. 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)
562. 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)
563. 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)
565. 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]
566. 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]
567. 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]
568. 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]
569. 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]
570. 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]
572. 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]
574. 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)
575. 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)
576. 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)
577. 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)
578. 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)
579. 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)
580. 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)
581. 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)
583. 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)
584. 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)
585. 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)
586. 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)
587. 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)
588. 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)
589. 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)
590. 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)
592. 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]
593. 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]
595. ## GENERAL GRAPH SETTINGS
596. plt.rcParams["font.family"] = "Times New Roman"
597. plt.rc('axes', axisbelow=True)
598. plt.rcParams.update({'font.size': 16})
600. ## GRAPH 1: PERCENTAGE OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES
601. x = ['1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y']
602. x\_axis = np.arange(len(x))
603. fig, ax1 = plt.subplots(figsize = (8,4))
604. plt.title('Percentage of Patients with Tremor Before Treatment that Improved\nAfter Various Treatment Times - PEAK DISTANCE (METHOD 2B)')
605. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
606. plt.tight\_layout()
608. ax1.bar(x\_axis - 0.15, D\_improved\_percentage, 0.3, label = 'Treated Hand', color = 'darkblue')
609. ax1.bar(x\_axis + 0.15, ND\_improved\_percentage, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
610. ax1.set\_ylabel('Percentage Improved')
611. ax1.set\_xlabel('Time')
612. ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
613. ax1.set\_ylim(0, 100)
614. ax2 = ax1.twinx()
615. ax2.plot(x, avg\_total\_num, color = 'red')
616. ax2.set\_ylabel('Number of Patients')
617. ax2.legend(['Number of\npatients'], loc="upper center", prop={'size': 14})
618. ax2.set\_ylim(0, 100)
619. print("AVERAGE PEAK DISTANCE DOMINANT: " + str(np.average(D\_improved\_percentage)))
620. print("AVERAGE PEAK DISTANCE NON-DOMINANT: " + str(np.average(ND\_improved\_percentage)))
621. plt.savefig('RESULTS\GRAPHS\PercentageOfPatients\_Det2.png', bbox\_inches='tight', dpi=150)
623. ## GRAPH 2: PERCENTAGE OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES
624. fig, ax1 = plt.subplots(figsize = (8,4))
625. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
626. plt.title('Difference Between Tremor Severity Before Treatment\nand After Various Treatment Periods')
628. ax1.bar(x\_axis - 0.15, D\_avg\_diff, 0.3, label = 'Treated Hand', color = 'darkblue')
629. ax1.bar(x\_axis + 0.15, ND\_avg\_diff, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
630. ax1.set\_ylabel('Difference in Tremor')
631. ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
632. ax1.set\_xlabel('Time')
634. plt.xticks(x\_axis, x)
635. plt.tight\_layout()
636. plt.show()
638. ## GRAPH 3: PERCENT OF PATIENTS WITH TREMOR BEFORE TREATMENT THAT IMPROVED AFTER VARIOUS TREATMENT TIMES -AVG AREA
639. x = ['1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y']
640. x\_axis = np.arange(len(x))
641. fig, ax1 = plt.subplots(figsize = (8,4))
642. plt.title('Percentage of Patients with Tremor Before Treatment that Improved\nAfter Various Treatment Times - AVERAGE AREA (METHOD 2A)')
643. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
644. plt.tight\_layout()
646. ax1.bar(x\_axis - 0.15, D\_improved\_percentage\_AVG\_AREA, 0.3, label = 'Treated Hand', color = 'darkblue')
647. ax1.bar(x\_axis + 0.15, ND\_improved\_percentage\_AVG\_AREA, 0.3, label = 'Treated Hand', color = 'cornflowerblue')
648. ax1.set\_ylabel('Percentage Improved')
649. ax1.set\_xlabel('Time')
650. ax1.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
651. ax1.set\_ylim(0, 100)
652. ax2 = ax1.twinx()
653. ax2.plot(x, avg\_total\_num, color = 'red')
654. ax2.set\_ylabel('Number of Patients')
655. ax2.legend(['Number of\npatients'], loc="upper center", prop={'size': 14})
656. ax2.set\_ylim(0, 100)
657. plt.rcParams["figure.figsize"] = (8,4)
658. print("AVERAGE AREA METHOD DOM: " + str(np.average(D\_improved\_percentage\_AVG\_AREA)))
659. print("AVERAGE AREA METHOD NON-DOM: " + str(np.average(ND\_improved\_percentage\_AVG\_AREA)))
660. plt.savefig('RESULTS\GRAPHS\PercentageOfPatients\_AvgArea.png', bbox\_inches='tight', dpi=150)
662. # START OF "WHICH HAND?" GRAPH ONE
663. combined\_improved\_array = []
664. which\_hand = []
665. combined\_patient\_number\_array = []
666. dominant\_hand = []
667. longest\_array = 0
669. if len(D\_patient\_number\_array\_det\_2) > len(ND\_patient\_number\_array\_det\_2):
670. longest\_array = len(D\_patient\_number\_array\_det\_2)
671. else:
672. longest\_array = len(ND\_patient\_number\_array\_det\_2)
674. D\_counter = -1
675. ND\_counter = -1
677. \_before = 1
678. \_1W = 7
679. \_1M = 30
680. \_3M = 91
681. \_6M = 183
682. \_1Y = 365
683. \_2Y = 730
684. \_3Y = 1095
685. \_4Y = 1460
687. \_before\_linear = 1
688. \_1W\_linear = 2
689. \_1M\_linear = 3
690. \_3M\_linear = 4
691. \_6M\_linear = 5
692. \_1Y\_linear = 6
693. \_2Y\_linear = 7
694. \_3Y\_linear = 8
695. \_4Y\_linear = 9
697. fig\_ALLPATIENTS, ax1\_ALLPATIENTS = plt.subplots(figsize = (8,4))
699. for i in range(0, longest\_array):
700. D\_counter += 1
701. ND\_counter += 1
703. if D\_counter >= len(D\_patient\_number\_array\_det\_2) or ND\_counter >= len(ND\_patient\_number\_array\_det\_2):
704. break
705. else:
706. while D\_patient\_number\_array\_det\_2[D\_counter] != ND\_patient\_number\_array\_det\_2[ND\_counter]:
707. if D\_patient\_number\_array\_det\_2[D\_counter+1] == ND\_patient\_number\_array\_det\_2[ND\_counter]:
708. D\_counter += 1
709. elif D\_patient\_number\_array\_det\_2[D\_counter] == ND\_patient\_number\_array\_det\_2[ND\_counter + 1]:
710. ND\_counter += 1
711. else:
712. D\_counter += 1
713. ND\_counter += 1
715. D\_x\_array = []
716. D\_x\_array\_linear = []
717. D\_y\_array = []
719. if D\_time\_before\_det\_2[D\_counter] != 0 and D\_time\_before\_det\_2[D\_counter] != -1:
720. D\_y\_array.append(D\_time\_before\_det\_2[D\_counter])
721. D\_x\_array.append(\_before)
722. D\_x\_array\_linear.append(\_before\_linear)
723. if D\_time\_1W\_det\_2[D\_counter] != 0 and D\_time\_1W\_det\_2[D\_counter] != -1:
724. D\_y\_array.append(D\_time\_1W\_det\_2[D\_counter])
725. D\_x\_array.append(\_1W)
726. D\_x\_array\_linear.append(\_1W\_linear)
727. if D\_time\_1M\_det\_2[D\_counter] != 0 and D\_time\_1M\_det\_2[D\_counter] != -1:
728. D\_y\_array.append(D\_time\_1M\_det\_2[D\_counter])
729. D\_x\_array.append(\_1M)
730. D\_x\_array\_linear.append(\_1M\_linear)
731. if D\_time\_3M\_det\_2[D\_counter] != 0 and D\_time\_3M\_det\_2[D\_counter] != -1:
732. D\_y\_array.append(D\_time\_3M\_det\_2[D\_counter])
733. D\_x\_array.append(\_3M)
734. D\_x\_array\_linear.append(\_3M\_linear)
735. if D\_time\_6M\_det\_2[D\_counter] != 0 and D\_time\_6M\_det\_2[D\_counter] != -1:
736. D\_y\_array.append(D\_time\_6M\_det\_2[D\_counter])
737. D\_x\_array.append(\_6M)
738. D\_x\_array\_linear.append(\_6M\_linear)
739. if D\_time\_1Y\_det\_2[D\_counter] != 0 and D\_time\_1Y\_det\_2[D\_counter] != -1:
740. D\_y\_array.append(D\_time\_1Y\_det\_2[D\_counter])
741. D\_x\_array.append(\_1Y)
742. D\_x\_array\_linear.append(\_1Y\_linear)
743. if D\_time\_2Y\_det\_2[D\_counter] != 0 and D\_time\_2Y\_det\_2[D\_counter] != -1:
744. D\_y\_array.append(D\_time\_2Y\_det\_2[D\_counter])
745. D\_x\_array.append(\_2Y)
746. D\_x\_array\_linear.append(\_2Y\_linear)
747. if D\_time\_3Y\_det\_2[D\_counter] != 0 and D\_time\_3Y\_det\_2[D\_counter] != -1:
748. D\_y\_array.append(D\_time\_3Y\_det\_2[D\_counter])
749. D\_x\_array.append(\_3Y)
750. D\_x\_array\_linear.append(\_3Y\_linear)
751. if D\_time\_4Y\_det\_2[D\_counter] != 0 and D\_time\_4Y\_det\_2[D\_counter] != -1:
752. D\_y\_array.append(D\_time\_4Y\_det\_2[D\_counter])
753. D\_x\_array.append(\_4Y)
754. D\_x\_array\_linear.append(\_4Y\_linear)
756. ND\_x\_array = []
757. ND\_x\_array\_linear = []
758. ND\_y\_array = []
759. if ND\_time\_before\_det\_2[ND\_counter] != 0 and ND\_time\_before\_det\_2[ND\_counter] != -1:
760. ND\_y\_array.append(ND\_time\_before\_det\_2[ND\_counter])
761. ND\_x\_array.append(\_before)
762. ND\_x\_array\_linear.append(\_before\_linear)
763. if ND\_time\_1W\_det\_2[ND\_counter] != 0 and ND\_time\_1W\_det\_2[ND\_counter] != -1:
764. ND\_y\_array.append(ND\_time\_1W\_det\_2[ND\_counter])
765. ND\_x\_array.append(\_1W)
766. ND\_x\_array\_linear.append(\_1W\_linear)
767. if ND\_time\_1M\_det\_2[ND\_counter] != 0 and ND\_time\_1M\_det\_2[ND\_counter] != -1:
768. ND\_y\_array.append(ND\_time\_1M\_det\_2[ND\_counter])
769. ND\_x\_array.append(\_1M)
770. ND\_x\_array\_linear.append(\_1M\_linear)
771. if ND\_time\_3M\_det\_2[ND\_counter] != 0 and ND\_time\_3M\_det\_2[ND\_counter] != -1:
772. ND\_y\_array.append(ND\_time\_3M\_det\_2[ND\_counter])
773. ND\_x\_array.append(\_3M)
774. ND\_x\_array\_linear.append(\_3M\_linear)
775. if ND\_time\_6M\_det\_2[ND\_counter] != 0 and ND\_time\_6M\_det\_2[ND\_counter] != -1:
776. ND\_y\_array.append(ND\_time\_6M\_det\_2[ND\_counter])
777. ND\_x\_array.append(\_6M)
778. ND\_x\_array\_linear.append(\_6M\_linear)
779. if ND\_time\_1Y\_det\_2[ND\_counter] != 0 and ND\_time\_1Y\_det\_2[ND\_counter] != -1:
780. ND\_y\_array.append(ND\_time\_1Y\_det\_2[ND\_counter])
781. ND\_x\_array.append(\_1Y)
782. ND\_x\_array\_linear.append(\_1Y\_linear)
783. if ND\_time\_2Y\_det\_2[ND\_counter] != 0 and ND\_time\_2Y\_det\_2[ND\_counter] != -1:
784. ND\_y\_array.append(ND\_time\_2Y\_det\_2[ND\_counter])
785. ND\_x\_array.append(\_2Y)
786. ND\_x\_array\_linear.append(\_2Y\_linear)
787. if ND\_time\_3Y\_det\_2[ND\_counter] != 0 and ND\_time\_3Y\_det\_2[ND\_counter] != -1:
788. ND\_y\_array.append(ND\_time\_3Y\_det\_2[ND\_counter])
789. ND\_x\_array.append(\_3Y)
790. ND\_x\_array\_linear.append(\_3Y\_linear)
791. if ND\_time\_4Y\_det\_2[ND\_counter] != 0 and ND\_time\_4Y\_det\_2[ND\_counter] != -1:
792. ND\_y\_array.append(ND\_time\_4Y\_det\_2[ND\_counter])
793. ND\_x\_array.append(\_4Y)
794. ND\_x\_array\_linear.append(\_4Y\_linear)
796. if len(D\_x\_array) > 4 and len(D\_y\_array) > 4 and len(ND\_x\_array) > 4 and len(ND\_y\_array) > 4:
798. combined\_patient\_number\_array.append(D\_patient\_number\_array\_det\_2[D\_counter])
800. D\_gradient, D\_intercept = np.polyfit(D\_x\_array, D\_y\_array, 1)
801. ND\_gradient, ND\_intercept = np.polyfit(ND\_x\_array, ND\_y\_array, 1)
803. plt.title('Results of Patients\' Most Improved Hand')
804. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
805. plt.xticks([1,2,3,4,5,6,7,8,9], ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
807. temp\_D\_y\_array = np.array(D\_y\_array.copy())
808. D\_y\_array\_max = max(temp\_D\_y\_array)
809. temp\_D\_y\_array = temp\_D\_y\_array/D\_y\_array\_max\*100
810. temp\_ND\_y\_array = np.array(ND\_y\_array.copy())
811. ND\_y\_array\_max = max(temp\_ND\_y\_array)
812. temp\_ND\_y\_array = temp\_ND\_y\_array/ND\_y\_array\_max\*100
814. if D\_gradient < 0 and ND\_gradient < 0:
815. combined\_improved\_array.append('BOTH')
816. if D\_gradient < ND\_gradient:
817. which\_hand.append('TREATED')
818. ax1\_ALLPATIENTS.plot(D\_x\_array\_linear, temp\_D\_y\_array)
820. else:
821. which\_hand.append('NON-TREATED')
822. ax1\_ALLPATIENTS.plot(ND\_x\_array\_linear, temp\_ND\_y\_array)

825. elif D\_gradient < 0:
826. combined\_improved\_array.append('TREATED')
827. which\_hand.append('TREATED')
828. ax1\_ALLPATIENTS.plot(D\_x\_array\_linear, temp\_D\_y\_array)
830. elif ND\_gradient < 0:
831. combined\_improved\_array.append('NON-TREATED')
832. which\_hand.append('NON-TREATED')
833. ax1\_ALLPATIENTS.plot(ND\_x\_array\_linear, temp\_ND\_y\_array)
835. else:
836. combined\_improved\_array.append('NEITHER')
837. if D\_gradient < ND\_gradient:
838. which\_hand.append('TREATED')
839. else:
840. which\_hand.append('NON-TREATED')
842. plt.tight\_layout()
843. plt.show()
845. # START OF "WHICH HAND?" TWO
846. combined\_improved\_array = []
847. which\_hand = []
848. combined\_patient\_number\_array = []
849. dominant\_hand = []
851. longest\_array = 0
853. if len(D\_patient\_number\_array\_det\_2) > len(ND\_patient\_number\_array\_det\_2):
854. longest\_array = len(D\_patient\_number\_array\_det\_2)
855. else:
856. longest\_array = len(ND\_patient\_number\_array\_det\_2)
858. D\_counter = -1
859. ND\_counter = -1
861. \_before = 1
862. \_1W = 7
863. \_1M = 30
864. \_3M = 91
865. \_6M = 183
866. \_1Y = 365
867. \_2Y = 730
868. \_3Y = 1095
869. \_4Y = 1460
871. \_before\_linear = 1
872. \_1W\_linear = 2
873. \_1M\_linear = 3
874. \_3M\_linear = 4
875. \_6M\_linear = 5
876. \_1Y\_linear = 6
877. \_2Y\_linear = 7
878. \_3Y\_linear = 8
879. \_4Y\_linear = 9
881. fig\_ALLPATIENTS2, ax1\_ALLPATIENTS2 = plt.subplots(figsize = (8,4))
882. plt.title('Results of Patients Whose \'Before\' Drawings are the Most Severe')
883. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
884. plt.xticks([1,2,3,4,5,6,7,8,9], ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
886. for i in range(0, longest\_array):
887. D\_counter += 1
888. ND\_counter += 1
890. if D\_counter >= len(D\_patient\_number\_array\_det\_2) or ND\_counter >= len(ND\_patient\_number\_array\_det\_2):
891. break
892. else:
893. while D\_patient\_number\_array\_det\_2[D\_counter] != ND\_patient\_number\_array\_det\_2[ND\_counter]:
894. if D\_patient\_number\_array\_det\_2[D\_counter+1] == ND\_patient\_number\_array\_det\_2[ND\_counter]:
895. D\_counter += 1
896. elif D\_patient\_number\_array\_det\_2[D\_counter] == ND\_patient\_number\_array\_det\_2[ND\_counter + 1]:
897. ND\_counter += 1
898. else:
899. D\_counter += 1
900. ND\_counter += 1
902. D\_x\_array = []
903. D\_x\_array\_linear = []
904. D\_y\_array = []
906. if D\_time\_before\_det\_2[D\_counter] != 0 and D\_time\_before\_det\_2[D\_counter] != -1:
907. D\_y\_array.append(D\_time\_before\_det\_2[D\_counter])
908. D\_x\_array.append(\_before)
909. D\_x\_array\_linear.append(\_before\_linear)
910. if D\_time\_1W\_det\_2[D\_counter] != 0 and D\_time\_1W\_det\_2[D\_counter] != -1:
911. D\_y\_array.append(D\_time\_1W\_det\_2[D\_counter])
912. D\_x\_array.append(\_1W)
913. D\_x\_array\_linear.append(\_1W\_linear)
914. if D\_time\_1M\_det\_2[D\_counter] != 0 and D\_time\_1M\_det\_2[D\_counter] != -1:
915. D\_y\_array.append(D\_time\_1M\_det\_2[D\_counter])
916. D\_x\_array.append(\_1M)
917. D\_x\_array\_linear.append(\_1M\_linear)
918. if D\_time\_3M\_det\_2[D\_counter] != 0 and D\_time\_3M\_det\_2[D\_counter] != -1:
919. D\_y\_array.append(D\_time\_3M\_det\_2[D\_counter])
920. D\_x\_array.append(\_3M)
921. D\_x\_array\_linear.append(\_3M\_linear)
922. if D\_time\_6M\_det\_2[D\_counter] != 0 and D\_time\_6M\_det\_2[D\_counter] != -1:
923. D\_y\_array.append(D\_time\_6M\_det\_2[D\_counter])
924. D\_x\_array.append(\_6M)
925. D\_x\_array\_linear.append(\_6M\_linear)
926. if D\_time\_1Y\_det\_2[D\_counter] != 0 and D\_time\_1Y\_det\_2[D\_counter] != -1:
927. D\_y\_array.append(D\_time\_1Y\_det\_2[D\_counter])
928. D\_x\_array.append(\_1Y)
929. D\_x\_array\_linear.append(\_1Y\_linear)
930. if D\_time\_2Y\_det\_2[D\_counter] != 0 and D\_time\_2Y\_det\_2[D\_counter] != -1:
931. D\_y\_array.append(D\_time\_2Y\_det\_2[D\_counter])
932. D\_x\_array.append(\_2Y)
933. D\_x\_array\_linear.append(\_2Y\_linear)
934. if D\_time\_3Y\_det\_2[D\_counter] != 0 and D\_time\_3Y\_det\_2[D\_counter] != -1:
935. D\_y\_array.append(D\_time\_3Y\_det\_2[D\_counter])
936. D\_x\_array.append(\_3Y)
937. D\_x\_array\_linear.append(\_3Y\_linear)
938. if D\_time\_4Y\_det\_2[D\_counter] != 0 and D\_time\_4Y\_det\_2[D\_counter] != -1:
939. D\_y\_array.append(D\_time\_4Y\_det\_2[D\_counter])
940. D\_x\_array.append(\_4Y)
941. D\_x\_array\_linear.append(\_4Y\_linear)
943. ND\_x\_array = []
944. ND\_x\_array\_linear = []
945. ND\_y\_array = []
946. if ND\_time\_before\_det\_2[ND\_counter] != 0 and ND\_time\_before\_det\_2[ND\_counter] != -1:
947. ND\_y\_array.append(ND\_time\_before\_det\_2[ND\_counter])
948. ND\_x\_array.append(\_before)
949. ND\_x\_array\_linear.append(\_before\_linear)
951. if ND\_time\_1W\_det\_2[ND\_counter] != 0 and ND\_time\_1W\_det\_2[ND\_counter] != -1:
952. ND\_y\_array.append(ND\_time\_1W\_det\_2[ND\_counter])
953. ND\_x\_array.append(\_1W)
954. ND\_x\_array\_linear.append(\_1W\_linear)
955. if ND\_time\_1M\_det\_2[ND\_counter] != 0 and ND\_time\_1M\_det\_2[ND\_counter] != -1:
956. ND\_y\_array.append(ND\_time\_1M\_det\_2[ND\_counter])
957. ND\_x\_array.append(\_1M)
958. ND\_x\_array\_linear.append(\_1M\_linear)
959. if ND\_time\_3M\_det\_2[ND\_counter] != 0 and ND\_time\_3M\_det\_2[ND\_counter] != -1:
960. ND\_y\_array.append(ND\_time\_3M\_det\_2[ND\_counter])
961. ND\_x\_array.append(\_3M)
962. ND\_x\_array\_linear.append(\_3M\_linear)
963. if ND\_time\_6M\_det\_2[ND\_counter] != 0 and ND\_time\_6M\_det\_2[ND\_counter] != -1:
964. ND\_y\_array.append(ND\_time\_6M\_det\_2[ND\_counter])
965. ND\_x\_array.append(\_6M)
966. ND\_x\_array\_linear.append(\_6M\_linear)
967. if ND\_time\_1Y\_det\_2[ND\_counter] != 0 and ND\_time\_1Y\_det\_2[ND\_counter] != -1:
968. ND\_y\_array.append(ND\_time\_1Y\_det\_2[ND\_counter])
969. ND\_x\_array.append(\_1Y)
970. ND\_x\_array\_linear.append(\_1Y\_linear)
971. if ND\_time\_2Y\_det\_2[ND\_counter] != 0 and ND\_time\_2Y\_det\_2[ND\_counter] != -1:
972. ND\_y\_array.append(ND\_time\_2Y\_det\_2[ND\_counter])
973. ND\_x\_array.append(\_2Y)
974. ND\_x\_array\_linear.append(\_2Y\_linear)
975. if ND\_time\_3Y\_det\_2[ND\_counter] != 0 and ND\_time\_3Y\_det\_2[ND\_counter] != -1:
976. ND\_y\_array.append(ND\_time\_3Y\_det\_2[ND\_counter])
977. ND\_x\_array.append(\_3Y)
978. ND\_x\_array\_linear.append(\_3Y\_linear)
979. if ND\_time\_4Y\_det\_2[ND\_counter] != 0 and ND\_time\_4Y\_det\_2[ND\_counter] != -1:
980. ND\_y\_array.append(ND\_time\_4Y\_det\_2[ND\_counter])
981. ND\_x\_array.append(\_4Y)
982. ND\_x\_array\_linear.append(\_4Y\_linear)
984. if len(D\_x\_array) > 4 and len(D\_y\_array) > 4 and len(ND\_x\_array) > 4 and len(ND\_y\_array) > 4:
986. combined\_patient\_number\_array.append(D\_patient\_number\_array\_det\_2[D\_counter])
988. D\_gradient, D\_intercept = np.polyfit(D\_x\_array, D\_y\_array, 1)
989. ND\_gradient, ND\_intercept = np.polyfit(ND\_x\_array, ND\_y\_array, 1)
991. temp\_D\_y\_array = np.array(D\_y\_array.copy())
992. D\_y\_array\_max = max(temp\_D\_y\_array)
993. temp\_D\_y\_array = temp\_D\_y\_array/D\_y\_array\_max\*100
994. temp\_ND\_y\_array = np.array(ND\_y\_array.copy())
995. ND\_y\_array\_max = max(temp\_ND\_y\_array)
996. temp\_ND\_y\_array = temp\_ND\_y\_array/ND\_y\_array\_max\*100
998. if D\_time\_before\_det\_2[D\_counter] < 600 and ND\_time\_before\_det\_2[ND\_counter] < 600: # remove outlier
999. if D\_gradient < 0 and ND\_gradient < 0:
1000. combined\_improved\_array.append('BOTH')
1001. if D\_gradient < ND\_gradient:
1002. which\_hand.append('TREATED')
1003. if D\_y\_array[0] == D\_y\_array\_max:
1004. ax1\_ALLPATIENTS2.plot(D\_x\_array\_linear, D\_y\_array)
1005. else:
1006. which\_hand.append('NON-TREATED')
1007. if ND\_y\_array[0] == ND\_y\_array\_max:
1008. ax1\_ALLPATIENTS2.plot(ND\_x\_array\_linear, ND\_y\_array)
1010. elif D\_gradient < 0:
1011. combined\_improved\_array.append('TREATED')
1012. which\_hand.append('TREATED')
1013. if D\_y\_array[0] == D\_y\_array\_max:
1014. ax1\_ALLPATIENTS2.plot(D\_x\_array\_linear, D\_y\_array)
1016. elif ND\_gradient < 0:
1017. combined\_improved\_array.append('NON-TREATED')
1018. which\_hand.append('NON-TREATED')
1019. if ND\_y\_array[0] == ND\_y\_array\_max:
1020. ax1\_ALLPATIENTS2.plot(ND\_x\_array\_linear, ND\_y\_array)
1022. else:
1023. combined\_improved\_array.append('NEITHER')
1024. if D\_gradient < ND\_gradient:
1025. which\_hand.append('TREATED')
1026. else:
1027. which\_hand.append('NON-TREATED')
1028. plt.tight\_layout()
1029. plt.show()
1031. # START OF "WHICH HAND?" THREE
1032. x\_array = [1,2,3,4,5,6,7,8,9]
1033. fig\_ALLPATIENTS3, ax1\_ALLPATIENTS3 = plt.subplots(figsize = (8,4))
1034. plt.title('Average Tremor Severities for Each Hand\nPEAK DISTANCE (METHOD 2B)')
1035. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
1036. plt.xticks(x\_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
1038. D\_average\_before\_det\_2 = np.mean(D\_time\_before\_det\_2)
1039. D\_average\_1W\_det\_2 = np.mean(D\_time\_1W\_det\_2)
1040. D\_average\_1M\_det\_2 = np.mean(D\_time\_1M\_det\_2)
1041. D\_average\_3M\_det\_2 = np.mean(D\_time\_3M\_det\_2)
1042. D\_average\_6M\_det\_2 = np.mean(D\_time\_6M\_det\_2)
1043. D\_average\_1Y\_det\_2 = np.mean(D\_time\_1Y\_det\_2)
1044. D\_average\_2Y\_det\_2 = np.mean(D\_time\_2Y\_det\_2)
1045. D\_average\_3Y\_det\_2 = np.mean(D\_time\_3Y\_det\_2)
1046. D\_average\_4Y\_det\_2 = np.mean(D\_time\_4Y\_det\_2)
1048. ND\_average\_before\_det\_2 = np.mean(ND\_time\_before\_det\_2)
1049. ND\_average\_1W\_det\_2 = np.mean(ND\_time\_1W\_det\_2)
1050. ND\_average\_1M\_det\_2 = np.mean(ND\_time\_1M\_det\_2)
1051. ND\_average\_3M\_det\_2 = np.mean(ND\_time\_3M\_det\_2)
1052. ND\_average\_6M\_det\_2 = np.mean(ND\_time\_6M\_det\_2)
1053. ND\_average\_1Y\_det\_2 = np.mean(ND\_time\_1Y\_det\_2)
1054. ND\_average\_2Y\_det\_2 = np.mean(ND\_time\_2Y\_det\_2)
1055. ND\_average\_3Y\_det\_2 = np.mean(ND\_time\_3Y\_det\_2)
1056. ND\_average\_4Y\_det\_2 = np.mean(ND\_time\_4Y\_det\_2)
1058. 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]
1059. 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]
1061. plt.legend()
1062. ax1\_ALLPATIENTS3.plot(x\_array, D\_averages, color = 'darkblue')
1063. ax1\_ALLPATIENTS3.plot(x\_array, ND\_averages, color = 'cornflowerblue')
1064. ax1\_ALLPATIENTS3.set\_ylabel('Tremor Severity')
1065. ax1\_ALLPATIENTS3.set\_xlabel('Time')
1066. ax1\_ALLPATIENTS3.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
1067. plt.tight\_layout()
1068. plt.show()
1070. # AVERAGE TREMOR SEVERITY USING AVG AREA TRAPZ METHOD
1071. x\_array = [1,2,3,4,5,6,7,8,9]
1072. fig\_ALLPATIENTS3B, ax1\_ALLPATIENTS3B = plt.subplots(figsize = (8,4))
1073. plt.title('Average Tremor Severities for Each Hand\nAVERAGE AREA (METHOD 2A)')
1074. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
1075. plt.xticks(x\_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
1077. D\_average\_before\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_before\_avg\_area\_trapz])
1078. D\_average\_1W\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_1W\_avg\_area\_trapz])
1079. D\_average\_1M\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_1M\_avg\_area\_trapz])
1080. D\_average\_3M\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_3M\_avg\_area\_trapz])
1081. D\_average\_6M\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_6M\_avg\_area\_trapz])
1082. D\_average\_1Y\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_1Y\_avg\_area\_trapz])
1083. D\_average\_2Y\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_2Y\_avg\_area\_trapz])
1084. D\_average\_3Y\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_3Y\_avg\_area\_trapz])
1085. D\_average\_4Y\_avg\_area\_trapz = np.mean([abs(float(x)) for x in D\_time\_4Y\_avg\_area\_trapz])
1087. ND\_average\_before\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_before\_avg\_area\_trapz])
1088. ND\_average\_1W\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_1W\_avg\_area\_trapz])
1089. ND\_average\_1M\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_1M\_avg\_area\_trapz])
1090. ND\_average\_3M\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_3M\_avg\_area\_trapz])
1091. ND\_average\_6M\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_6M\_avg\_area\_trapz])
1092. ND\_average\_1Y\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_1Y\_avg\_area\_trapz])
1093. ND\_average\_2Y\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_2Y\_avg\_area\_trapz])
1094. ND\_average\_3Y\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_3Y\_avg\_area\_trapz])
1095. ND\_average\_4Y\_avg\_area\_trapz = np.mean([abs(float(x)) for x in ND\_time\_4Y\_avg\_area\_trapz])
1097. 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]
1098. 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]
1100. plt.legend()
1101. ax1\_ALLPATIENTS3B.plot(x\_array, D\_averages, color = 'darkblue')
1102. ax1\_ALLPATIENTS3B.plot(x\_array, ND\_averages, color = 'cornflowerblue')
1103. ax1\_ALLPATIENTS3B.set\_ylabel('Tremor Severity')
1104. ax1\_ALLPATIENTS3B.set\_xlabel('Time')
1105. ax1\_ALLPATIENTS3B.legend(['Treated Hand', 'Untreated Hand'], loc="upper right")
1106. plt.savefig('RESULTS\GRAPHS\AverageTremSev\_AvgArea.png', bbox\_inches='tight', dpi=150)
1108. plt.tight\_layout()
1109. plt.show()
1111. # START OF "WHICH HAND?" FOUR
1113. x\_array = [1,2,3,4,5,6,7,8,9]
1114. fig\_ALLPATIENTS4, ax1\_ALLPATIENTS4 = plt.subplots(figsize = (8,4))
1115. plt.title('Normalised Average Tremor Severities for Each Hand\nPEAK DISTANCE (METHOD 2B)')
1116. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
1117. plt.xticks(x\_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])
1119. 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]
1120. 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])
1121. denom = q95 - all\_min
1123. D\_time\_before\_normalised = [(x - all\_min)/denom for x in D\_time\_before\_det\_2]
1124. D\_time\_1W\_normalised = [(x - all\_min)/denom for x in D\_time\_1W\_det\_2]
1125. D\_time\_1M\_normalised = [(x - all\_min)/denom for x in D\_time\_1M\_det\_2]
1126. D\_time\_3M\_normalised = [(x - all\_min)/denom for x in D\_time\_3M\_det\_2]
1127. D\_time\_6M\_normalised = [(x - all\_min)/denom for x in D\_time\_6M\_det\_2]
1128. D\_time\_1Y\_normalised = [(x - all\_min)/denom for x in D\_time\_1Y\_det\_2]
1129. D\_time\_2Y\_normalised = [(x - all\_min)/denom for x in D\_time\_2Y\_det\_2]
1130. D\_time\_3Y\_normalised = [(x - all\_min)/denom for x in D\_time\_3Y\_det\_2]
1131. D\_time\_4Y\_normalised = [(x - all\_min)/denom for x in D\_time\_4Y\_det\_2]
1133. 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]
1135. ND\_time\_before\_normalised = [(x - all\_min)/denom for x in ND\_time\_before\_det\_2]
1136. ND\_time\_1W\_normalised = [(x - all\_min)/denom for x in ND\_time\_1W\_det\_2]
1137. ND\_time\_1M\_normalised = [(x - all\_min)/denom for x in ND\_time\_1M\_det\_2]
1138. ND\_time\_3M\_normalised = [(x - all\_min)/denom for x in ND\_time\_3M\_det\_2]
1139. ND\_time\_6M\_normalised = [(x - all\_min)/denom for x in ND\_time\_6M\_det\_2]
1140. ND\_time\_1Y\_normalised = [(x - all\_min)/denom for x in ND\_time\_1Y\_det\_2]
1141. ND\_time\_2Y\_normalised = [(x - all\_min)/denom for x in ND\_time\_2Y\_det\_2]
1142. ND\_time\_3Y\_normalised = [(x - all\_min)/denom for x in ND\_time\_3Y\_det\_2]
1143. ND\_time\_4Y\_normalised = [(x - all\_min)/denom for x in ND\_time\_4Y\_det\_2]
1145. D\_average\_before\_det\_2 = np.mean(D\_time\_before\_normalised)
1146. D\_average\_1W\_det\_2 = np.mean(D\_time\_1W\_normalised)
1147. D\_average\_1M\_det\_2 = np.mean(D\_time\_1M\_normalised)
1148. D\_average\_3M\_det\_2 = np.mean(D\_time\_3M\_normalised)
1149. D\_average\_6M\_det\_2 = np.mean(D\_time\_6M\_normalised)
1150. D\_average\_1Y\_det\_2 = np.mean(D\_time\_1Y\_normalised)
1151. D\_average\_2Y\_det\_2 = np.mean(D\_time\_2Y\_normalised)
1152. D\_average\_3Y\_det\_2 = np.mean(D\_time\_3Y\_normalised)
1153. D\_average\_4Y\_det\_2 = np.mean(D\_time\_4Y\_normalised)
1155. ND\_average\_before\_det\_2 = np.mean(ND\_time\_before\_normalised)
1156. ND\_average\_1W\_det\_2 = np.mean(ND\_time\_1W\_normalised)
1157. ND\_average\_1M\_det\_2 = np.mean(ND\_time\_1M\_normalised)
1158. ND\_average\_3M\_det\_2 = np.mean(ND\_time\_3M\_normalised)
1159. ND\_average\_6M\_det\_2 = np.mean(ND\_time\_6M\_normalised)
1160. ND\_average\_1Y\_det\_2 = np.mean(ND\_time\_1Y\_normalised)
1161. ND\_average\_2Y\_det\_2 = np.mean(ND\_time\_2Y\_normalised)
1162. ND\_average\_3Y\_det\_2 = np.mean(ND\_time\_3Y\_normalised)
1163. ND\_average\_4Y\_det\_2 = np.mean(ND\_time\_4Y\_normalised)
1165. 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]
1166. 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]
1168. plt.legend()
1169. ax1\_ALLPATIENTS4.plot(x\_array, D\_averages, color = 'darkblue')
1170. ax1\_ALLPATIENTS4.plot(x\_array, ND\_averages, color = 'cornflowerblue')
1171. ax1\_ALLPATIENTS4.set\_ylabel('Tremor Severity')
1172. ax1\_ALLPATIENTS4.set\_xlabel('Time')
1174. ax1\_ALLPATIENTS4.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
1175. plt.savefig('RESULTS\GRAPHS\AverageTremSev\_Det2.png', bbox\_inches='tight', dpi=150)
1176. plt.show()
1178. # START OF "WHICH HAND?" FOUR B
1179. x\_array = [1,2,3,4,5,6,7,8,9]
1180. fig\_ALLPATIENTS4B, ax1\_ALLPATIENTS4B = plt.subplots(figsize = (8,4))
1181. plt.title('Normalised Average Tremor Severities for Each Hand\nAVERAGE AREA (METHOD 2A)')
1182. plt.grid(linestyle = '-', linewidth=0.5, axis='y')
1183. plt.xticks(x\_array, ['Before', '1W', '1M', '3M', '6M', '1Y', '2Y', '3Y', '4Y'])

1186. copy\_D\_time\_before\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_before\_avg\_area\_trapz]
1187. for c in copy\_D\_time\_before\_avg\_area\_trapz:
1188. if c == 0.0:
1189. copy\_D\_time\_before\_avg\_area\_trapz.remove(c)
1190. copy\_D\_time\_1W\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_1W\_avg\_area\_trapz]
1191. for c in copy\_D\_time\_1W\_avg\_area\_trapz:
1192. if c == 0.0:
1193. copy\_D\_time\_1W\_avg\_area\_trapz.remove(c)
1194. copy\_D\_time\_1M\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_1M\_avg\_area\_trapz]
1195. for c in copy\_D\_time\_1M\_avg\_area\_trapz:
1196. if c == 0.0:
1197. copy\_D\_time\_1M\_avg\_area\_trapz.remove(c)
1198. copy\_D\_time\_3M\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_3M\_avg\_area\_trapz]
1199. for c in copy\_D\_time\_3M\_avg\_area\_trapz:
1200. if c == 0.0:
1201. copy\_D\_time\_3M\_avg\_area\_trapz.remove(c)
1202. copy\_D\_time\_6M\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_6M\_avg\_area\_trapz]
1203. for c in copy\_D\_time\_6M\_avg\_area\_trapz:
1204. if c == 0.0:
1205. copy\_D\_time\_6M\_avg\_area\_trapz.remove(c)
1206. copy\_D\_time\_1Y\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_1Y\_avg\_area\_trapz]
1207. for c in copy\_D\_time\_1Y\_avg\_area\_trapz:
1208. if c == 0.0:
1209. copy\_D\_time\_1Y\_avg\_area\_trapz.remove(c)
1210. copy\_D\_time\_2Y\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_2Y\_avg\_area\_trapz]
1211. for c in copy\_D\_time\_2Y\_avg\_area\_trapz:
1212. if c == 0.0:
1213. copy\_D\_time\_2Y\_avg\_area\_trapz.remove(c)
1214. copy\_D\_time\_3Y\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_3Y\_avg\_area\_trapz]
1215. for c in copy\_D\_time\_3Y\_avg\_area\_trapz:
1216. if c == 0.0:
1217. copy\_D\_time\_3Y\_avg\_area\_trapz.remove(c)
1218. copy\_D\_time\_4Y\_avg\_area\_trapz = [abs(float(x)) for x in D\_time\_4Y\_avg\_area\_trapz]
1219. for c in copy\_D\_time\_4Y\_avg\_area\_trapz:
1220. if c == 0.0:
1221. copy\_D\_time\_4Y\_avg\_area\_trapz.remove(c)
1223. copy\_ND\_time\_before\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_before\_avg\_area\_trapz]
1224. for c in copy\_ND\_time\_before\_avg\_area\_trapz:
1225. if c == 0.0:
1226. copy\_ND\_time\_before\_avg\_area\_trapz.remove(c)
1227. copy\_ND\_time\_1W\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_1W\_avg\_area\_trapz]
1228. for c in copy\_ND\_time\_1W\_avg\_area\_trapz:
1229. if c == 0.0:
1230. copy\_ND\_time\_1W\_avg\_area\_trapz.remove(c)
1231. copy\_ND\_time\_1M\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_1M\_avg\_area\_trapz]
1232. for c in copy\_ND\_time\_1M\_avg\_area\_trapz:
1233. if c == 0.0:
1234. copy\_ND\_time\_1M\_avg\_area\_trapz.remove(c)
1235. copy\_ND\_time\_3M\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_3M\_avg\_area\_trapz]
1236. for c in copy\_ND\_time\_3M\_avg\_area\_trapz:
1237. if c == 0.0:
1238. copy\_ND\_time\_3M\_avg\_area\_trapz.remove(c)
1239. copy\_ND\_time\_6M\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_6M\_avg\_area\_trapz]
1240. for c in copy\_ND\_time\_6M\_avg\_area\_trapz:
1241. if c == 0.0:
1242. copy\_ND\_time\_6M\_avg\_area\_trapz.remove(c)
1243. copy\_ND\_time\_1Y\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_1Y\_avg\_area\_trapz]
1244. for c in copy\_ND\_time\_1Y\_avg\_area\_trapz:
1245. if c == 0.0:
1246. copy\_ND\_time\_1Y\_avg\_area\_trapz.remove(c)
1247. copy\_ND\_time\_2Y\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_2Y\_avg\_area\_trapz]
1248. for c in copy\_ND\_time\_2Y\_avg\_area\_trapz:
1249. if c == 0.0:
1250. copy\_ND\_time\_2Y\_avg\_area\_trapz.remove(c)
1251. copy\_ND\_time\_3Y\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_3Y\_avg\_area\_trapz]
1252. for c in copy\_ND\_time\_3Y\_avg\_area\_trapz:
1253. if c == 0.0:
1254. copy\_ND\_time\_3Y\_avg\_area\_trapz.remove(c)
1255. copy\_ND\_time\_4Y\_avg\_area\_trapz = [abs(float(x)) for x in ND\_time\_4Y\_avg\_area\_trapz]
1256. for c in copy\_ND\_time\_4Y\_avg\_area\_trapz:
1257. if c == 0.0:
1258. copy\_ND\_time\_4Y\_avg\_area\_trapz.remove(c)
1260. print(copy\_D\_time\_before\_avg\_area\_trapz)
1261. print(copy\_ND\_time\_before\_avg\_area\_trapz)
1262. m1 = max(copy\_D\_time\_before\_avg\_area\_trapz)
1263. m2 = max(copy\_D\_time\_1W\_avg\_area\_trapz)
1264. m3 = max(copy\_D\_time\_1M\_avg\_area\_trapz)
1265. m4 = max(copy\_D\_time\_3M\_avg\_area\_trapz)
1266. m5 = max(copy\_D\_time\_6M\_avg\_area\_trapz)
1267. m6 = max(copy\_D\_time\_1Y\_avg\_area\_trapz )
1268. m7 = max(copy\_D\_time\_2Y\_avg\_area\_trapz )
1269. m8 = max(copy\_D\_time\_3Y\_avg\_area\_trapz )
1270. m9 = max(copy\_D\_time\_4Y\_avg\_area\_trapz )
1272. m10 = max(copy\_ND\_time\_before\_avg\_area\_trapz)
1273. m11 = max(copy\_ND\_time\_1W\_avg\_area\_trapz)
1274. m12 = max(copy\_ND\_time\_1M\_avg\_area\_trapz)
1275. m13 = max(copy\_ND\_time\_3M\_avg\_area\_trapz)
1276. m14 = max(copy\_ND\_time\_6M\_avg\_area\_trapz)
1277. m15 = max(copy\_ND\_time\_1Y\_avg\_area\_trapz)
1278. m16 = max(copy\_ND\_time\_2Y\_avg\_area\_trapz)
1279. m17 = max(copy\_ND\_time\_3Y\_avg\_area\_trapz)
1280. m18 = max(copy\_ND\_time\_4Y\_avg\_area\_trapz)
1282. m1 = min(copy\_D\_time\_before\_avg\_area\_trapz)
1283. m2 = min(copy\_D\_time\_1W\_avg\_area\_trapz)
1284. m3 = min(copy\_D\_time\_1M\_avg\_area\_trapz)
1285. m4 = min(copy\_D\_time\_3M\_avg\_area\_trapz)
1286. m5 = min(copy\_D\_time\_6M\_avg\_area\_trapz)
1287. m6 = min(copy\_D\_time\_1Y\_avg\_area\_trapz )
1288. m7 = min(copy\_D\_time\_2Y\_avg\_area\_trapz )
1289. m8 = min(copy\_D\_time\_3Y\_avg\_area\_trapz )
1290. m9 = min(copy\_D\_time\_4Y\_avg\_area\_trapz )
1292. m10 = min(copy\_ND\_time\_before\_avg\_area\_trapz)
1293. m11 = min(copy\_ND\_time\_1W\_avg\_area\_trapz)
1294. m12 = min(copy\_ND\_time\_1M\_avg\_area\_trapz)
1295. m13 = min(copy\_ND\_time\_3M\_avg\_area\_trapz)
1296. m14 = min(copy\_ND\_time\_6M\_avg\_area\_trapz)
1297. m15 = min(copy\_ND\_time\_1Y\_avg\_area\_trapz)
1298. m16 = min(copy\_ND\_time\_2Y\_avg\_area\_trapz)
1299. m17 = min(copy\_ND\_time\_3Y\_avg\_area\_trapz)
1300. m18 = min(copy\_ND\_time\_4Y\_avg\_area\_trapz)
1302. actual\_max = max(m1, m2, m3, m4, m5, m6, m7, m8, m9, m11, m12, m13, m14, m15, m16, m17, m18)
1303. all\_min = min(m1, m2, m3, m4, m5, m6, m7, m8, m9, m11, m12, m13, m14, m15, m16, m17, m18)
1305. # NB: BUG WARNING HERE - WHY IS MAX 0?
1306. denom = 50 # actual\_max - all\_min
1308. D\_time\_before\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_before\_avg\_area\_trapz]]
1309. D\_time\_1W\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_1W\_avg\_area\_trapz]]
1310. D\_time\_1M\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_1M\_avg\_area\_trapz]]
1311. D\_time\_3M\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_3M\_avg\_area\_trapz]]
1312. D\_time\_6M\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_6M\_avg\_area\_trapz]]
1313. D\_time\_1Y\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_1Y\_avg\_area\_trapz]]
1314. D\_time\_2Y\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_2Y\_avg\_area\_trapz]]
1315. D\_time\_3Y\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_3Y\_avg\_area\_trapz]]
1316. D\_time\_4Y\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_D\_time\_4Y\_avg\_area\_trapz]]
1318. 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]
1320. ND\_time\_before\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_before\_avg\_area\_trapz]]
1321. ND\_time\_1W\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_1W\_avg\_area\_trapz]]
1322. ND\_time\_1M\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_1M\_avg\_area\_trapz]]
1323. ND\_time\_3M\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_3M\_avg\_area\_trapz]]
1324. ND\_time\_6M\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_6M\_avg\_area\_trapz]]
1325. ND\_time\_1Y\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_1Y\_avg\_area\_trapz]]
1326. ND\_time\_2Y\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_2Y\_avg\_area\_trapz]]
1327. ND\_time\_3Y\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_3Y\_avg\_area\_trapz]]
1328. ND\_time\_4Y\_normalised = [(x - all\_min)/denom for x in [abs(float(x)) for x in copy\_ND\_time\_4Y\_avg\_area\_trapz]]
1330. D\_average\_before\_det\_2 = np.mean(D\_time\_before\_normalised)
1331. D\_average\_1W\_det\_2 = np.mean(D\_time\_1W\_normalised)
1332. D\_average\_1M\_det\_2 = np.mean(D\_time\_1M\_normalised)
1333. D\_average\_3M\_det\_2 = np.mean(D\_time\_3M\_normalised)
1334. D\_average\_6M\_det\_2 = np.mean(D\_time\_6M\_normalised)
1335. D\_average\_1Y\_det\_2 = np.mean(D\_time\_1Y\_normalised)
1336. D\_average\_2Y\_det\_2 = np.mean(D\_time\_2Y\_normalised)
1337. D\_average\_3Y\_det\_2 = np.mean(D\_time\_3Y\_normalised)
1338. D\_average\_4Y\_det\_2 = np.mean(D\_time\_4Y\_normalised)
1340. ND\_average\_before\_det\_2 = np.mean(ND\_time\_before\_normalised)
1341. ND\_average\_1W\_det\_2 = np.mean(ND\_time\_1W\_normalised)
1342. ND\_average\_1M\_det\_2 = np.mean(ND\_time\_1M\_normalised)
1343. ND\_average\_3M\_det\_2 = np.mean(ND\_time\_3M\_normalised)
1344. ND\_average\_6M\_det\_2 = np.mean(ND\_time\_6M\_normalised)
1345. ND\_average\_1Y\_det\_2 = np.mean(ND\_time\_1Y\_normalised)
1346. ND\_average\_2Y\_det\_2 = np.mean(ND\_time\_2Y\_normalised)
1347. ND\_average\_3Y\_det\_2 = np.mean(ND\_time\_3Y\_normalised)
1348. ND\_average\_4Y\_det\_2 = np.mean(ND\_time\_4Y\_normalised)
1350. 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]
1351. 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]
1353. plt.legend()
1354. ax1\_ALLPATIENTS4B.plot(x\_array, D\_averages, color = 'darkblue')
1355. ax1\_ALLPATIENTS4B.plot(x\_array, ND\_averages, color = 'cornflowerblue')
1356. ax1\_ALLPATIENTS4B.set\_ylabel('Tremor Severity')
1357. ax1\_ALLPATIENTS4B.set\_xlabel('Time')
1358. ax1\_ALLPATIENTS4B.legend(['Treated Hand', 'Untreated Hand'], loc="upper right", prop={'size': 14})
1359. plt.savefig('RESULTS\GRAPHS\AverageTremSev\_AvgArea.png', bbox\_inches='tight', dpi=150)
1360. plt.show()
1362. # END OF CODE - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #