COMP 204
Programming Studio

CHARACTER RECOGNITION USING MOMENTS (February 2020)

Eyüp Yasuntimur, MEF University, Istanbul, TURKEY
E-mail: yasuntimure@mef.edu.tr

Abstract

This project report explains what character recognition is, how it is implemented, how it is encoded and tested on python. The project explains (blob_coloring_8_connected method) how to distinguish the characters in an image (with coloring), how to calculate the coordinates of these separated characters and get a black square background from these coordinates and explains how to get the moments of the characters in (hu, r, zernike). After these steps, the moments are added to a dataset and gui interface that we have designed, and individually train characters and tests for recognition quality.

Keywords: image recognition, r moment, zernike moment, hu moment

CONTENT:

CHARACTER RECOGNITION USING MOMENTS

Introduction:

Character Recognition Using Moments is a project to recognize characters on an image. Then, by taking certain moments of these characters, a more comprehensive scan of the characters can be made.

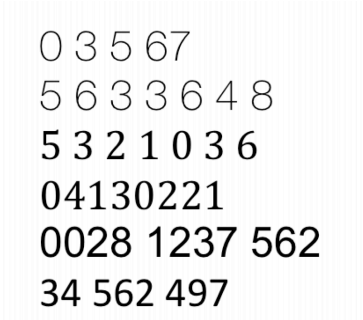
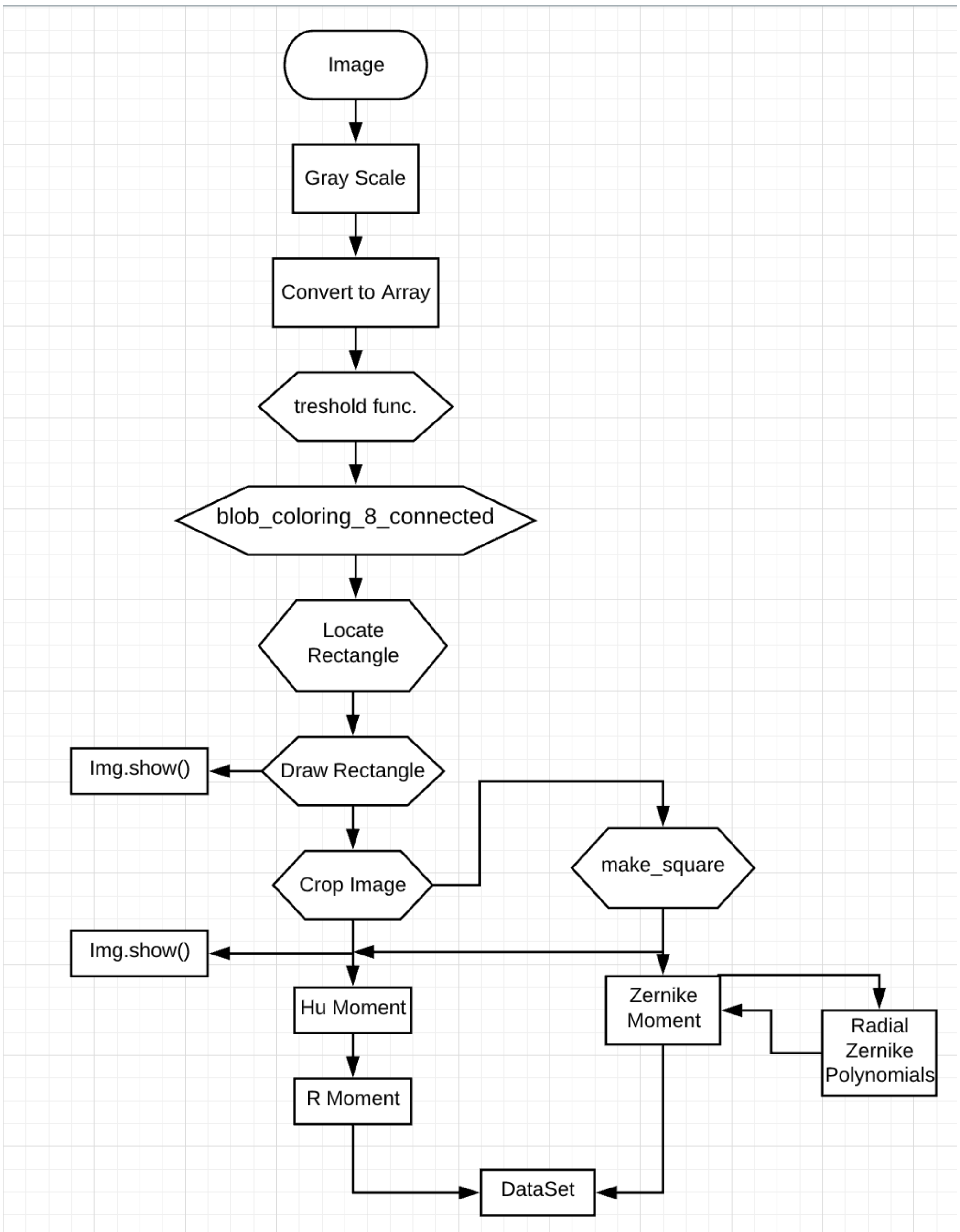


Figure 1: Example Image



UML Diagram of project

```
img = Image.open('image.png') #reads image
img_gray = img.convert('L')  # converts the image to grayscale image
# img_bin = img.convert('1') #converts to a binary image, T=128, LOW=0, HIGH=255
img_gray.show()
ONE = 150
a = np.asarray(img_gray)  # from PIL to np array
a_bin = threshold(a,100,ONE,0)
im = Image.fromarray(a_bin)  # from np array to PIL format
im.show()
# a_bin = binary_image(100,100,ONE)  # creates a binary image
```

Figure 2: Opening image and convert it to array

We got the code you see above ready at the beginning of the lesson and briefly you see the steps like opening the image file, making it grayscale and converting it into an array as a format. There are many different methods used in the project, there is a comprehensive explanation of the methods below.

8 Connected Blob Coloring Function:

```
label,img = blob_coloring_8_connected(a_bin,ONE) #obtains coloured and k valued array
```

Figure 3: The code in Main () where the function is called.

Blob Coloring function was also available in the sample file we have 4 connected. So I only made certain adjustments to make 8 connected. In the case of 4 Connected functions of the function, the left and top pixels are controlled except for the center pixel and labeling concept is established over them.

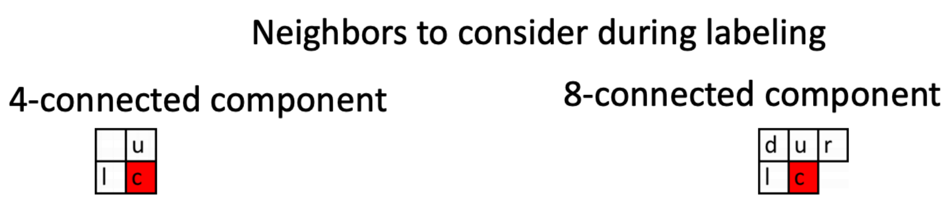


Figure 4: Pixel checks for 4 and 8 Neighbors

As you can see in Figure 4, we need to assign the variables "left up" and "right up" in order to have a 8 Connected check. In this way, we can make a more efficient and quality analysis. In Figure 5 (blob_coloring_8_connected (a_bin, ONE)) you see a part of the function. To make the above distinction, I have added the variables "lu" and "ru" in addition to the "l" and "u" variables in this "for" loop. Then I applied the same process for the "label" variables and added the conditions in the "if, else" conditions to make the scan correctly, respect to the 4Connected algorithm. Then I got the "color_im" and "im" variables in the function with "return".

```
for i in range(1,nrow - 1):
    for j in range(1,ncol - 1):
        c = bim[i][j]
        l = bim[i][j - 1]
        u = bim[i - 1][j]
        lu = bim[i - 1][j - 1]
        ru = bim[i - 1][j + 1]
        label_u = im[i - 1][j]
        label_l = im[i][j - 1]
        label_lu = im[i - 1][j - 1]  # for the 8 connected system we need to add left up and right up.
        label_ru = im[i - 1][j + 1]
        im[i][j] = max_label
        if c == ONE:
            min_label = min(label_u,label_l,label_lu,label_ru)
            if min_label == max_label:
                k += 1
                im[i][j] = k
            else:
                im[i][j] = min_label
                if min_label != label_u and label_u != max_label:
                    update_array(a,min_label,label_u)
                if min_label != label_l and label_l != max_label:
                    update_array(a,min_label,label_l)
                if min_label != label_lu and label_lu != max_label:
                    update_array(a,min_label,label_lu)
                if min_label != label_ru and label_ru != max_label:
                    update_array(a,min_label,label_ru)
        else:
            im[i][j] = max_label
```

Figure 5: For loop of (blob_coloring_8_connected)

I took the two variables I extracted from the Blob Coloring function as label = color_im and im = img. Then, img locateRectangle from these variables entered function with 'numpy.ndarray' type.

Locate Rectangle Function:

The Locate Rectangle function finds the minimum and maximum values of the individually (labeled) characters in the numpy array I bought. First think first, it creates a k_values array. After that while we were checking the array of course we will need the row and column number. So I defined it with length of array and length of horizontal array. The labels array is stores my k values so I put a maximal number first, then after the first check it will append my k values pixel by pixel.

```
def locateRectangle(im): #defines the minimum and maximum side of characters and saves it into k_values
    k_values = []
    nrow = len(im)
    ncol = len(im[0])
    labels = [10000]
    for i in range(nrow):
        for j in range(ncol):
            if im[i][j] not in labels:
                mini = nrow
                minj = ncol
                maxi = 0
                maxj = 0
                k = im[i][j]
                for i in range(nrow):
                    for j in range(ncol):
                        if k == im[i][j]:
                            if mini > i:
                                mini = i
                            if minj > j:
                                minj = j
                            if maxi < i:
                                maxi = i
                            if maxj < j:
                                maxj = j
                k_values.append([k,mini,minj,maxi,maxj])
                labels.append(k)
    return k_values
```

Figure 6: Locate Rectangle Function

This 4 “for” loop, checks for labels and after the check append my k values to labels and therefore there is no confusion between the min-max values of the objects.

Draw Rectangle Function:

The draw rectangle function is basically draws red (255,0,0,"RGB") rectangle on the image by the kValues which are minimum and maximum values of the objects for both axis. I made it by ImageDraw.Draw and draw.rectangle methods.

```
def drawRectengle(k_values,new_img):
    global drawRectengle
    # importing image object from PIL
    from PIL import Image,ImageDraw
    for i in range(len(k_values)):
        mini = k_values[i][1]
        minj = k_values[i][2]
        maxi = k_values[i][3]
        maxj = k_values[i][4]
        print("k values"," mini:",mini," minj:",minj," maxi:",maxi," maxj:",maxj)
        draw = ImageDraw.Draw(new_img)
        draw.rectangle((minj,mini,maxj,maxi),outline=(255,0,0))
        new_img.show()
```

Figure 7: Draw Rectangle Function

As you see the above in the figure 7 I created a for loop to repeat the draw operation until the number of the objects all done.

After finishing this operation I created another "for" loop in the main function.

```
for i in range(len(kValues)): # loop continues k variable times to calculate all figures in the image
    sqrImg = cropImage(kValues[i][2],kValues[i][1],kValues[i][4],kValues[i][3],new_img2) #cropes image to square sized
    H_Moments = huMoments(sqrImg)
    R_Moments = rMoments(H_Moments)
    Z_Moments = zernikeMomennts(sqrImg)
```

Figure 8: Main part of the code

So I cropped image, squared it and took the moments by functions that I wrote separately.

Crop Image Function:

Crop Image Function crops the given rectangles from the image. Then it determines which of the x, y axes of this rectangular image is smaller with the if-else condition and equals the result to the min_size variable. Sends the cropped image and the min_size variable to the make_square function. Here, min_size rectangle determines which side of the rectangle is short, so fills it in black and sends it back.

```
def make_square(im,min_size,fill_color=(0,0,0,0)): # fills with black colour the empty side og rectengles
    x,y = im.size
    size = max(min_size,x,y)
    new_im = Image.new('RGB',(size,size),fill_color)
    new_im.paste(im,(int((size - x) / 2),int((size - y) / 2)))
    return new_im
```

Figure 9: make_square func.

At the same time, after rotating the square image, the picture is converted to gray scale and array format. In this way, we get an array with numerical values from the image again.

```
def cropImage(minj,mini,maxj,maxi,new_img):
    croppedImages = []
    try:
        area = (minj,mini,maxj,maxi)
        img = new_img.crop(area)
        croppedImages.append([img])
        img.show()
        # img.save("cropped_picture.jpg")
    except IOError:
        pass
    x,y = img.size  # defines x or y greater than that and defines it min_size. So make_square func. fills the true are with black
    if x < y:
        min_size = x
    else:
        min_size = y
    img = make_square(img,min_size)
    img.show()
    img_gray = img.convert('L')  # converts the image to grayscale image
    ONE = 150
    a = np.asarray(img_gray)  # from PIL to np array
    return a
```

Figure 10: cropImage func.

Moments and Moment Functions (huMoment, rMoment, zernikeMoment):

This part of the code was mainly on math. I applied the formulas Hu moment, R moment and Zernike moment respectively. I frequently used nested for loops and math library while applying. When calculating the Zernike Moment and Hu Moment, I opened extra functions (normalizedMoments, radialZernike) and called them inside the function to make my job easier.

```
def huMoments(sqrImg): # calculates Hu moments of cropped square img
    moments = []
    m00 = 0
    m01 = 0
    m10 = 0
    m11 = 0
    for i in range(len(sqrImg)):
        for j in range(len(sqrImg[0])):
            m00 = m00 + pow(i,0) * pow(j,0) * sqrImg[i][j]
    for i in range(len(sqrImg)):
        for j in range(len(sqrImg[0])):
            m01 = m01 + pow(i,0) * pow(j,1) * sqrImg[i][j]
    for i in range(len(sqrImg)):
        for j in range(len(sqrImg[0])):
            m10 = m10 + pow(i,1) * pow(j,0) * sqrImg[i][j]
    for i in range(len(sqrImg)):
        for j in range(len(sqrImg[0])):
            m11 = m11 + pow(i,1) * pow(j,1) * sqrImg[i][j]
    uMoments = []
    u = 0
    for p in range(4):
        for q in range(4):
            for i in range(len(sqrImg)):
                for j in range(len(sqrImg[0])):
                    u = u + pow((i - (m10 / m00)),p) * pow((j - (m01 / m00)),q) * sqrImg[i][j]
            uMoments.append([p,q,u])
    n20 = normalizedMoments(2,0,uMoments[0][2],uMoments[8][2])
    n02 = normalizedMoments(0,2,uMoments[0][2],uMoments[2][2])
    n11 = normalizedMoments(1,1,uMoments[0][2],uMoments[5][2])
    n30 = normalizedMoments(3,0,uMoments[0][2],uMoments[12][2])
    n03 = normalizedMoments(0,3,uMoments[0][2],uMoments[3][2])
    n12 = normalizedMoments(1,2,uMoments[0][2],uMoments[6][2])
    n21 = normalizedMoments(2,1,uMoments[0][2],uMoments[9][2])
    huMoments = []
    H1 = n20 + n02
    H2 = (n20 - n02) ** 2 + 4 * (n11 ** 2)
    H3 = (n30 - (3 * n12)) ** 2 + (3 * n21 - n03) ** 2
    H4 = (n30 + n12) ** 2 + (n21 + n03) ** 2
    H5 = (n30 - 3 * n12) * (n30 + n12) * ((n30 + n12) ** 2 - 3 * ((n21 + n03) ** 2)) + (3 * n21 - n03) * (
            n21 + n03) * (3 * ((n30 + n12) ** 2) - ((n21 + n03) ** 2))
    H6 = (n20 - n02) * ((n30 + n12) ** 2 - (n21 + n03) ** 2) + 4 * n11 * (n30 + n12) * (n21 + n03)
    H7 = (3 * n21 - n03) * (n30 + n12) * ((n30 + n12) ** 2 - 3 * ((n21 + n03) ** 2)) + (3 * n12 - n30) * (
            n21 + n03) * (3 * ((n30 + n12) ** 2) - (n21 + n03) ** 2)
    huMoments.append([H1,H2,H3,H4,H5,H6,H7])
    return huMoments
```

Figure 11: Hu Moment function

```python
def rMoments(array): # calculates R moments of cropped img by hu moments
    R_Moments=[]
    R1= math.sqrt(array[0][1]) / array[0][0]
    R2 = (array[0][0] + math.sqrt(array[0][1])) / (array[0][0] - (math.sqrt(array[0][1])))
    R3 = (math.sqrt(array[0][2])) / (math.sqrt(array[0][3]))
    R4 = (math.sqrt(array[0][2])) / math.sqrt(abs(array[0][4]))
    R5 = (math.sqrt(array[0][3])) / math.sqrt(abs(array[0][4]))
    R6 = abs(array[0][5]) / array[0][0] * array[0][2]
    R7 = abs(array[0][5]) / array[0][0] * math.sqrt(abs(array[0][4]))
    R8 = abs(array[0][5]) / array[0][2] * math.sqrt(array[0][1])
    R9 = abs(array[0][5]) / math.sqrt(array[0][1]*abs(array[0][4]))
    R10= abs(array[0][4]) / array[0][2]*array[0][3]
    R_Moments.append([R1,R2,R3,R4,R5,R6,R7,R8,R9,R10])
    return  R_Moments
```

Figure 12: R Moment function

```python
def radialZernike(p,n1,m1): # calculates Radial Zernike Polynomials formula
    global r1
    r1=0
    a = int((n1 - abs(m1) / 2))
    for s in range(a):
        r1 = (pow(-1,s) * pow(p,(n1-2*s)) * math.factorial(abs(int(n1-2*s)))) / math.factor
    return r1

def zernikeMomennts(img): #calculates zernike moment of cropped square img
    N = len(img)
    Z_nm = []
    ZRnm=0
    ZInm=0
    Rnm=0
    Znm=0
    for n in range(6):
        for m in range(6):
            for i in range(N-1):
                for j in range(N-1):
                    xi = (math.sqrt(2)/N-1)*i - 1/math.sqrt(2)
                    yj = (math.sqrt(2)/N-1)*j - 1/math.sqrt(2)
                    dxi = 2 / N * math.sqrt(2)
                    dyj = dxi
                    Qij = math.atan(yj/xi)
                    pij = math.sqrt(pow(xi,2)+pow(yj,2))
                    Rnm = radialZernike(pij,n,m)
                    ZRnm = ZRnm + (img[i][j] * Rnm * math.cos(m*Qij) * dxi * dyj)
                    ZInm = ZInm + (img[i][j] * Rnm * math.sin(m*Qij) * dxi * dyj)
            ZRnm = ZRnm * (n+1)/math.pi
            ZInm = ZInm * -((n+1)/math.pi)
            Znm = math.sqrt(pow(ZRnm,2) + pow(ZInm,2))
            Z_nm.append(Znm)
    return Z_nm
```

Figure 13: Zernike Moment function

Dataset and Spliting Testset-Trainset:

In this section I could not write a working kNN and create a dataset as I wanted. For this reason, I created a dataset with dataset = np.zeros ((len (kValues), 36)), then I added the Zernike Moment values, which have 36 columns, to each row for k values. Later, I divided the dataset into 80% trainset and 20% testset.

```
dataset = np.random.rand(len(kValues),36)
np.random.shuffle(dataset)
percent80 = int(len(kValues)*80/100) # percent for the divide 80% of data
training,test = dataset[:percent80,:],dataset[percent80:,:]
print("Train Set",training)
print("\n\nTest Set ",test)
```

Figure 14: Train set - Test set

In doing so, I used the commands np.random.rand and np.random.shuffle, which you see in figure 14 above. In this way, each time there will be a random distinction.

Achievements:

In this part, I listed my achievements from the project. First, I had a lot of experience with syntax regarding python language use and it was a very good exercise project. However, I learned how to use the Numpy library. I learned how to open and gray scale an image and convert it into an array.

Secondly, the project that I frequently use Image and math libraries has contributed to me in this regard. While taking the moment values, I noticed that I accelerated in applying a formula to the code. I understood how to separate the dataset I obtained. Apart from all these, although I could not complete the whole project, it was a tiring but very productive process for me. Briefly, the libraries I have used and experienced;

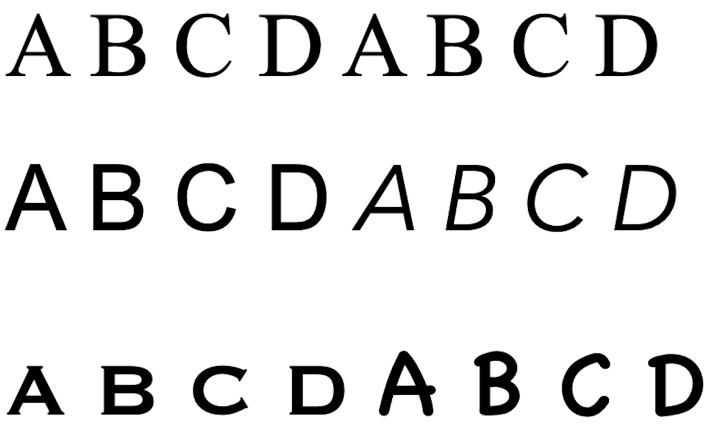
> Image, ImageDraw

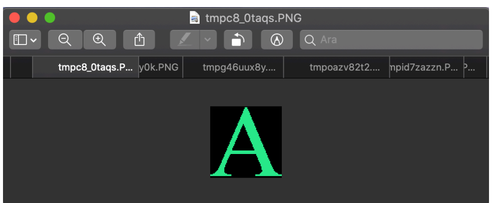
> Numpy

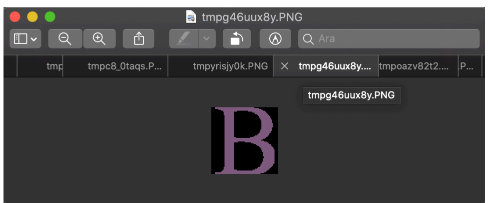
> Math

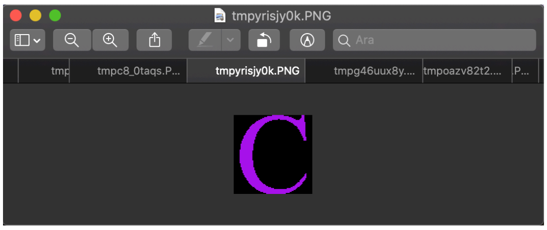
> tkinter

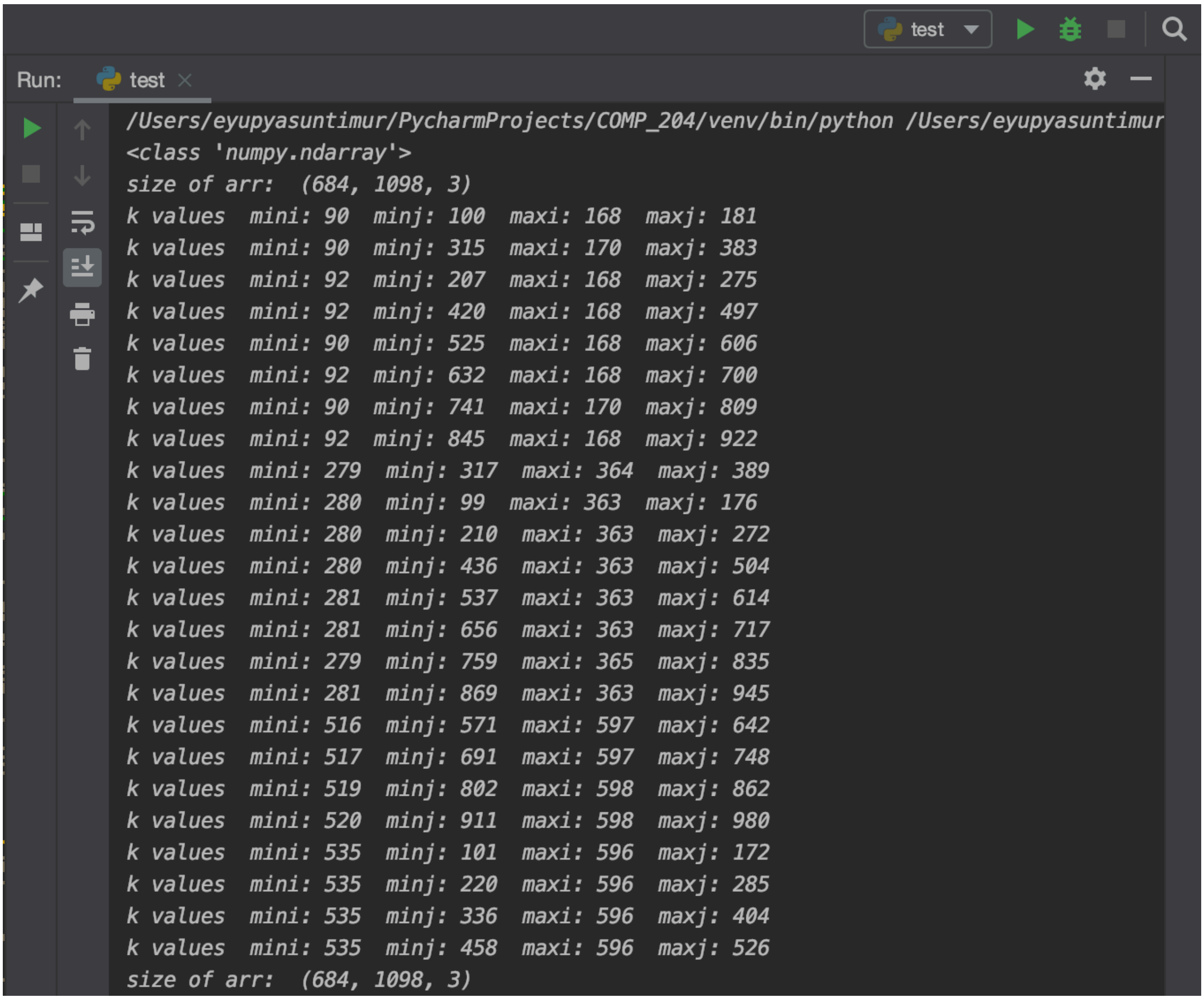
Inputs and Outputs:











```
/Users/eyupyasuntimur/PycharmProjects/COMP_204/venv/bin/python /Users/eyupyasuntimur
<class 'numpy.ndarray'>
size of arr:  (684, 1098, 3)
k values  mini: 90  minj: 100  maxi: 168  maxj: 181
k values  mini: 90  minj: 315  maxi: 170  maxj: 383
k values  mini: 92  minj: 207  maxi: 168  maxj: 275
k values  mini: 92  minj: 420  maxi: 168  maxj: 497
k values  mini: 90  minj: 525  maxi: 168  maxj: 606
k values  mini: 92  minj: 632  maxi: 168  maxj: 700
k values  mini: 90  minj: 741  maxi: 170  maxj: 809
k values  mini: 92  minj: 845  maxi: 168  maxj: 922
k values  mini: 279  minj: 317  maxi: 364  maxj: 389
k values  mini: 280  minj: 99  maxi: 363  maxj: 176
k values  mini: 280  minj: 210  maxi: 363  maxj: 272
k values  mini: 280  minj: 436  maxi: 363  maxj: 504
k values  mini: 281  minj: 537  maxi: 363  maxj: 614
k values  mini: 281  minj: 656  maxi: 363  maxj: 717
k values  mini: 279  minj: 759  maxi: 365  maxj: 835
k values  mini: 281  minj: 869  maxi: 363  maxj: 945
k values  mini: 516  minj: 571  maxi: 597  maxj: 642
k values  mini: 517  minj: 691  maxi: 597  maxj: 748
k values  mini: 519  minj: 802  maxi: 598  maxj: 862
k values  mini: 520  minj: 911  maxi: 598  maxj: 980
k values  mini: 535  minj: 101  maxi: 596  maxj: 172
k values  mini: 535  minj: 220  maxi: 596  maxj: 285
k values  mini: 535  minj: 336  maxi: 596  maxj: 404
k values  mini: 535  minj: 458  maxi: 596  maxj: 526
size of arr:  (684, 1098, 3)
```


```
Train Set [[0.33789893 0.72056997 0.99237592 0.16061651 0.91137137 0.04354742
  0.11971779 0.19255473 0.19825349 0.67623244 0.17360745 0.5059938
  0.92836372 0.58230101 0.80424944 0.96323446 0.18539037 0.25995468
  0.11211679 0.56444861 0.82404079 0.95717947 0.39478585 0.6759059
  0.34328943 0.75149285 0.4850589  0.5648246  0.25186519 0.77044667
  0.95185166 0.50134632 0.08966476 0.96158514 0.37226369 0.52803109]
 [0.74851056 0.13536772 0.75275444 0.45436976 0.35580855 0.86441148
  0.66947761 0.28511584 0.82940756 0.64645319 0.60016319 0.71124865
  0.31619165 0.07488889 0.67401502 0.94746435 0.22389493 0.35152287
  0.26615216 0.4653527  0.08001754 0.72121787 0.71121665 0.97670541
  0.1853321  0.33578572 0.43299961 0.16159062 0.11700816 0.52708043
  0.60605187 0.48421085 0.91072527 0.52317553 0.40484962 0.1584771 ]
 [0.31345785 0.68419343 0.67632878 0.59433598 0.40694443 0.374871
  0.31636869 0.13414874 0.49049622 0.25101229 0.79599771 0.36570284
  0.80702927 0.79583227 0.20638991 0.86691119 0.24722357 0.03575448
  0.84540335 0.26715101 0.09641584 0.30135332 0.55704316 0.87912286
  0.72464248 0.70409173 0.42401517 0.11098123 0.98316557 0.88730554
  0.55044041 0.75928604 0.95167601 0.8842307  0.22605426 0.95541061]
 [0.24964212 0.28832227 0.54656984 0.99475435 0.7432731  0.47663017
  0.6627934  0.23288011 0.07066064 0.22003067 0.73346509 0.74463682
  0.04301636 0.0821399  0.45494698 0.44852901 0.49396035 0.83718348
  0.73964839 0.7897624  0.89067539 0.99089953 0.27815253 0.55424881
  0.20948364 0.9108219  0.51040691 0.79226364 0.18384554 0.079806
  0.76042942 0.47679347 0.69889456 0.271872   0.08930049 0.29092336]
 [0.49578438 0.22096784 0.72311585 0.20341229 0.66734042 0.60464724
  0.7242294  0.75051713 0.23625082 0.76096115 0.59332187 0.94728156
  0.51492841 0.81092718 0.64359472 0.06362947 0.21411265 0.52961027
  0.53458165 0.2776775  0.2892661  0.89211584 0.96250854 0.85891258
  0.7358562  0.2392911  0.44924171 0.56644538 0.33496924 0.92931662
  0.42120629 0.53105326 0.53738376 0.28502488 0.94957901 0.72978935]
 [0.92980536 0.89504768 0.42573744 0.27090944 0.26683641 0.91432194
  0.64003047 0.77856519 0.85464556 0.3575033  0.48812506 0.61468602
  0.81271594 0.00601121 0.63649912 0.27176071 0.83223753 0.55046032
  0.26790073 0.46861646 0.17855446 0.5260488  0.4706381  0.40226797
  0.7925413  0.64094944 0.45945101 0.42439641 0.57813259 0.99011364
  0.28602442 0.60100543 0.21697629 0.68270243 0.39122832 0.21371438]
 [0.33108466 0.66868004 0.5429246  0.31117134 0.38332773 0.96628672
  0.60872011 0.90824524 0.71010361 0.93157946 0.51069562 0.66913986
  0.72370569 0.73813392 0.42225798 0.63733372 0.31406686 0.8400199
  0.14120934 0.81580664 0.04279347 0.46898338 0.27532615 0.71084194
  0.11001655 0.58181325 0.62825221 0.46641431 0.76870291 0.64144443
  0.35474414 0.56127254 0.08258224 0.27086518 0.64903324 0.70992274]
 [0.23570163 0.18469635 0.05847389 0.83524731 0.34398182 0.96442315
  0.08428519 0.59166106 0.6706439  0.95899822 0.27457884 0.59420638
  0.6223691  0.22795918 0.92145973 0.70958389 0.04252229 0.7349639
  0.52674689 0.67377935 0.27036079 0.80773681 0.34308333 0.92160553
  0.74855869 0.69424804 0.16956913 0.24891166 0.48605276 0.36848106
  0.60431324 0.71463652 0.24507106 0.62427142 0.18879625 0.44737595]
 [0.98963507 0.61959867 0.92961668 0.49464221 0.36272798 0.22983842
  0.17847595 0.69356753 0.6213863
  0.05143963 0.88064578 0.7724707
  0.51597322 0.63211486 0.4678138
  0.85790426 0.81631493 0.86433901 0.43542295 0.51044643 0.32814576
```

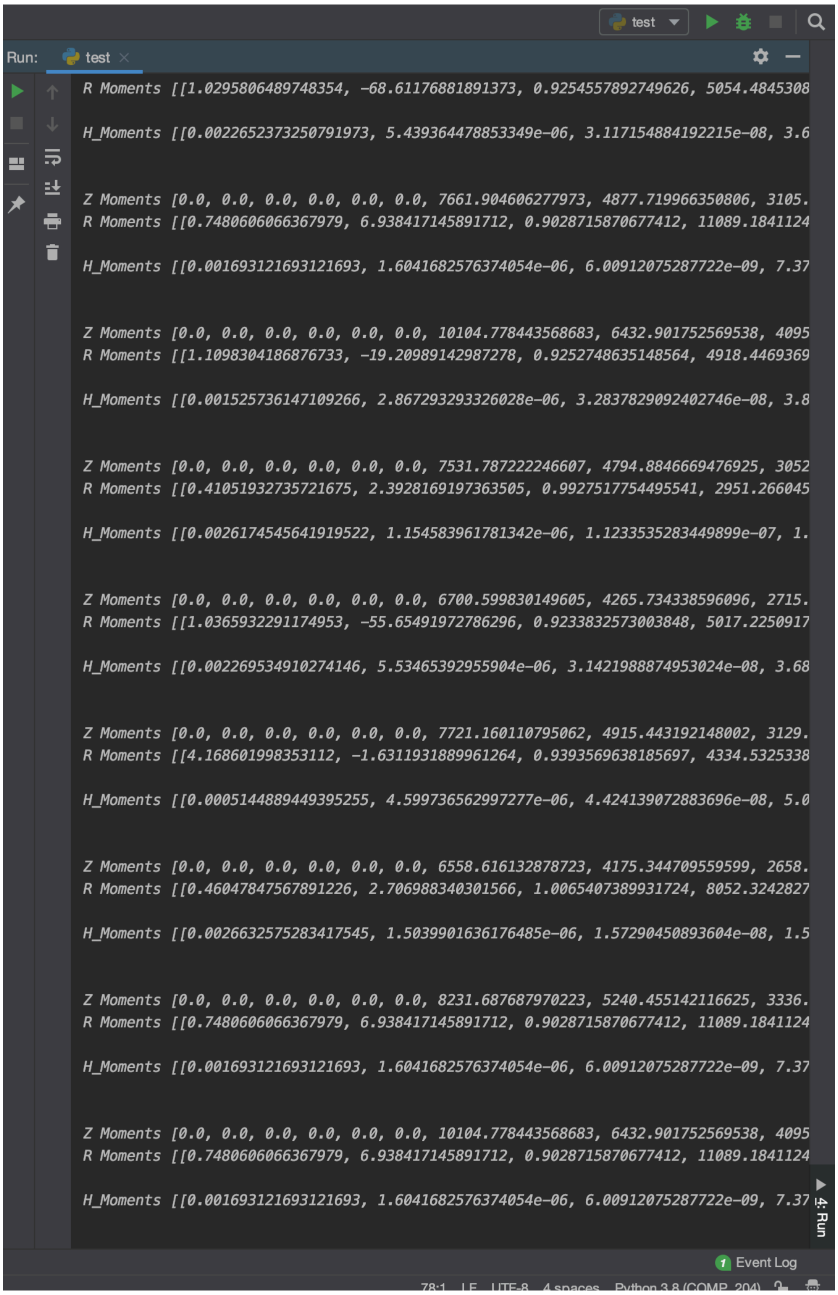
```
Test Set  [[0.48259866 0.26834068 0.37057389 0.91298863 0.23384694 0.87064158
  0.5561205  0.85339014 0.86302858 0.108736   0.75134657 0.78703688
  0.97133944 0.08671638 0.82265458 0.74219344 0.30064014 0.39616317
  0.30522846 0.62815474 0.44745547 0.76242023 0.97492968 0.06761713
  0.42846681 0.3561137  0.32500941 0.75999103 0.15776453 0.84719605
  0.10098593 0.10669307 0.18984651 0.78607232 0.60633253 0.95156979]
 [0.67503565 0.09769119 0.08457547 0.36970604 0.65363577 0.29365734
  0.82507064 0.11136555 0.6544571  0.99565763 0.10679163 0.60883147
  0.93573182 0.67817042 0.66386287 0.88337693 0.96662837 0.85751496
  0.78982034 0.54104809 0.32819801 0.84515914 0.66221008 0.64394265
  0.67573412 0.48893679 0.02784852 0.44402724 0.93187809 0.40445131
  0.86627133 0.89976973 0.10798592 0.73510146 0.00578168 0.24092021]
 [0.15023899 0.69608675 0.02807609 0.64310179 0.0577184  0.10286479
  0.44689667 0.49941734 0.67939827 0.67701856 0.73113451 0.20492994
  0.83705262 0.66722028 0.08635217 0.58311406 0.29613265 0.4618449
  0.32957546 0.0208304  0.30711822 0.28736112 0.02423855 0.91412864
  0.6112209  0.63639116 0.59286739 0.61102791 0.151353   0.00904043
  0.58614031 0.76711047 0.11867153 0.55220093 0.18446108 0.50915627]
 [0.909635   0.1105254  0.79458865 0.33728867 0.1425055  0.24753248
  0.05459723 0.86247549 0.28223112 0.93566087 0.90357083 0.47108441
  0.24374769 0.3031753  0.88595535 0.48600797 0.69689031 0.5893723
  0.39123552 0.0888484  0.08339999 0.29856547 0.33235369 0.9972294
  0.94488095 0.19436681 0.06351519 0.35686799 0.13583683 0.109268
  0.13246927 0.29235054 0.4357862  0.95817644 0.00907958 0.51702197]
 [0.26018369 0.08543971 0.6950642  0.19768029 0.7763645  0.52203717
  0.803789   0.668922   0.26123409 0.78890047 0.89246822 0.06216956
  0.30811666 0.51150094 0.32009755 0.92255745 0.77131209 0.89323486
  0.14372774 0.04622406 0.04656431 0.66660654 0.56762638 0.52779035
  0.33597534 0.64997766 0.28361672 0.78075389 0.53513686 0.15072414
  0.63445814 0.09903529 0.6416078

Process finished with exit code 0
```

IDE and Plugin Updates
PyCharm is ready to update.

Event Log

5:22 LF UTF-8 4 spaces Python 3.8 (COMP_204)


test
Run:
test
R Moments [[1.0295806489748354, -68.61176881891373, 0.9254557892749626, 5054.4845308
H_Moments [[0.0022652373250791973, 5.439364478853349e-06, 3.117154884192215e-08, 3.6
Z Moments [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7661.904606277973, 4877.719966350806, 3105.
R Moments [[0.7480606066367979, 6.938417145891712, 0.9028715870677412, 11089.1841124
H_Moments [[0.001693121693121693, 1.6041682576374054e-06, 6.00912075287722e-09, 7.37
Z Moments [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 10104.778443568683, 6432.901752569538, 4095
R Moments [[1.1098304186876733, -19.20989142987278, 0.9252748635148564, 4918.4469369
H_Moments [[0.001525736147109266, 2.867293293326028e-06, 3.2837829092402746e-08, 3.8
Z Moments [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7531.787222246607, 4794.8846669476925, 3052
R Moments [[0.41051932735721675, 2.3928169197363505, 0.9927517754495541, 2951.266045
H_Moments [[0.0026174545641919522, 1.154583961781342e-06, 1.1233535283449899e-07, 1.
Z Moments [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6700.599830149605, 4265.734338596096, 2715.
R Moments [[1.0365932291174953, -55.65491972786296, 0.9233832573003848, 5017.2250917
H_Moments [[0.002269534910274146, 5.53465392955904e-06, 3.1421988874953024e-08, 3.68
Z Moments [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7721.160110795062, 4915.443192148002, 3129.
R Moments [[4.168601998353112, -1.6311931889961264, 0.9393569638185697, 4334.5325338
H_Moments [[0.0005144889449395255, 4.599736562997277e-06, 4.424139072883696e-08, 5.0
Z Moments [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6558.616132878723, 4175.344709559599, 2658.
R Moments [[0.46047847567891226, 2.706988340301566, 1.0065407389931724, 8052.3242827
H_Moments [[0.0026632575283417545, 1.5039901636176485e-06, 1.57290450893604e-08, 1.5
Z Moments [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8231.687687970223, 5240.455142116625, 3336.
R Moments [[0.7480606066367979, 6.938417145891712, 0.9028715870677412, 11089.1841124
H_Moments [[0.001693121693121693, 1.6041682576374054e-06, 6.00912075287722e-09, 7.37
Z Moments [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 10104.778443568683, 6432.901752569538, 4095
R Moments [[0.7480606066367979, 6.938417145891712, 0.9028715870677412, 11089.1841124
H_Moments [[0.001693121693121693, 1.6041682576374054e-06, 6.00912075287722e-09, 7.37
4: Run
Event Log