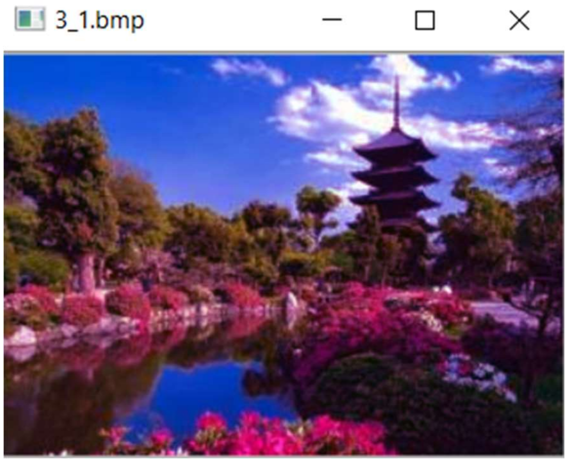
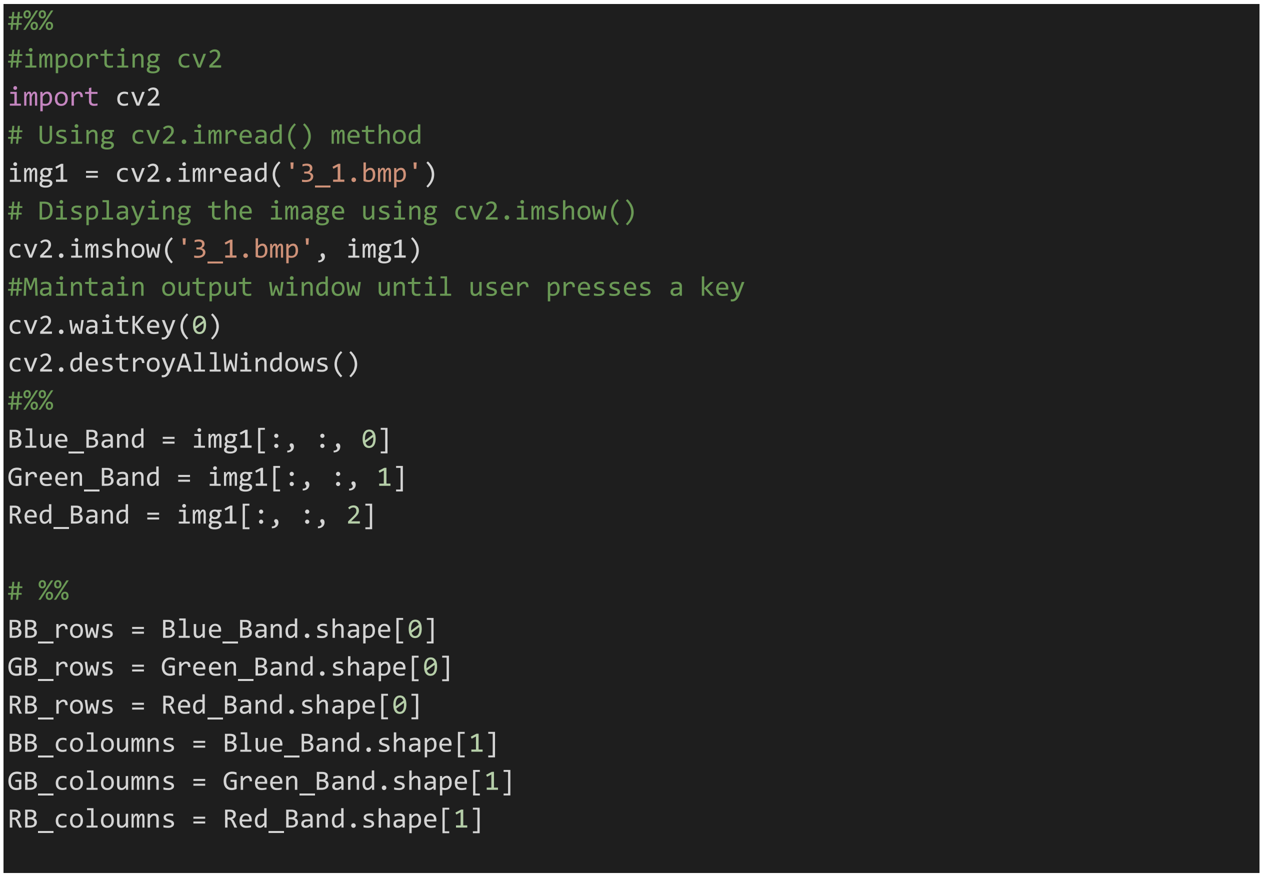
Homework Three:

Problem 01:

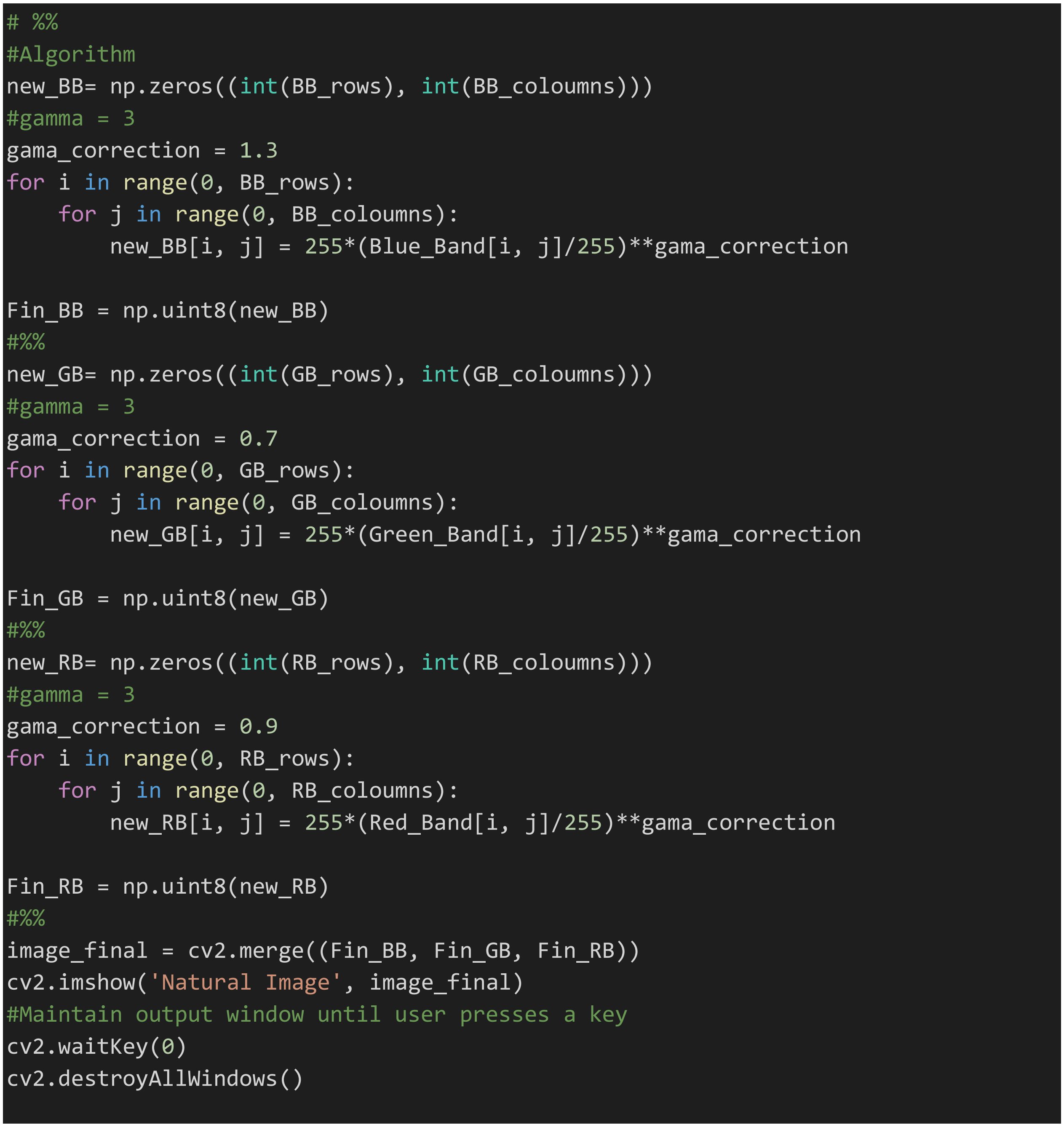
Original Image:



Code:

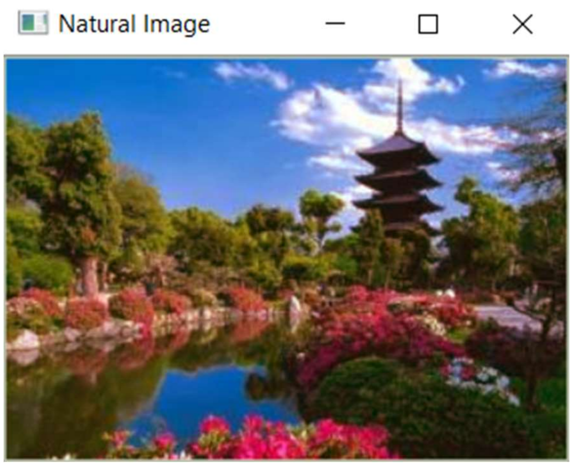


```
#%%
#importing cv2
import cv2
# Using cv2.imread() method
img1 = cv2.imread('3_1.bmp')
# Displaying the image using cv2.imshow()
cv2.imshow('3_1.bmp', img1)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
#%%
Blue_Band = img1[:, :, 0]
Green_Band = img1[:, :, 1]
Red_Band = img1[:, :, 2]

# %%
BB_rows = Blue_Band.shape[0]
GB_rows = Green_Band.shape[0]
RB_rows = Red_Band.shape[0]
BB_coloumns = Blue_Band.shape[1]
GB_coloumns = Green_Band.shape[1]
RB_coloumns = Red_Band.shape[1]
```



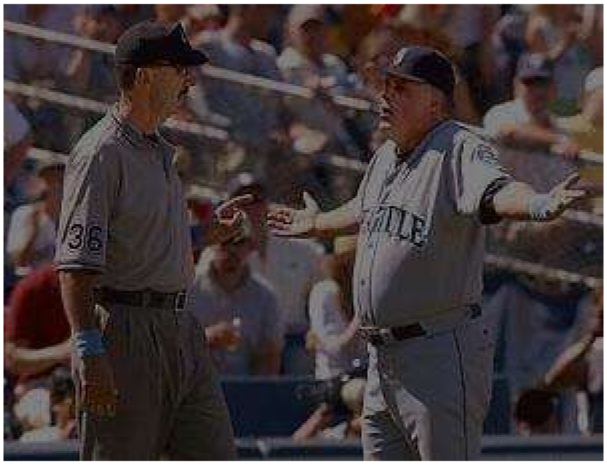
```
# %%
#Algorithm
new_BB= np.zeros((int(BB_rows), int(BB_coloumns)))
#gamma = 3
gama_correction = 1.3
for i in range(0, BB_rows):
    for j in range(0, BB_coloumns):
        new_BB[i, j] = 255*(Blue_Band[i, j]/255)**gama_correction

Fin_BB = np.uint8(new_BB)
#%%
new_GB= np.zeros((int(GB_rows), int(GB_coloumns)))
#gamma = 3
gama_correction = 0.7
for i in range(0, GB_rows):
    for j in range(0, GB_coloumns):
        new_GB[i, j] = 255*(Green_Band[i, j]/255)**gama_correction

Fin_GB = np.uint8(new_GB)
#%%
new_RB= np.zeros((int(RB_rows), int(RB_coloumns)))
#gamma = 3
gama_correction = 0.9
for i in range(0, RB_rows):
    for j in range(0, RB_coloumns):
        new_RB[i, j] = 255*(Red_Band[i, j]/255)**gama_correction

Fin_RB = np.uint8(new_RB)
#%%
image_final = cv2.merge((Fin_BB, Fin_GB, Fin_RB))
cv2.imshow('Natural Image', image_final)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natural Image:

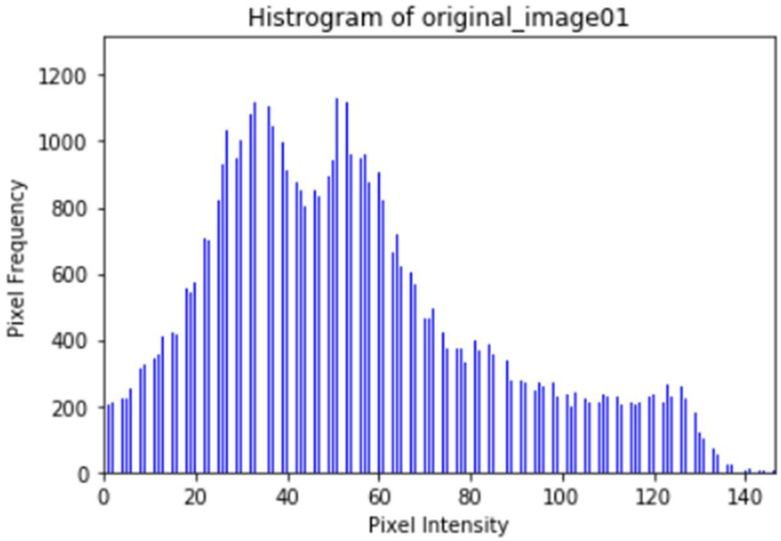


Problem 2:

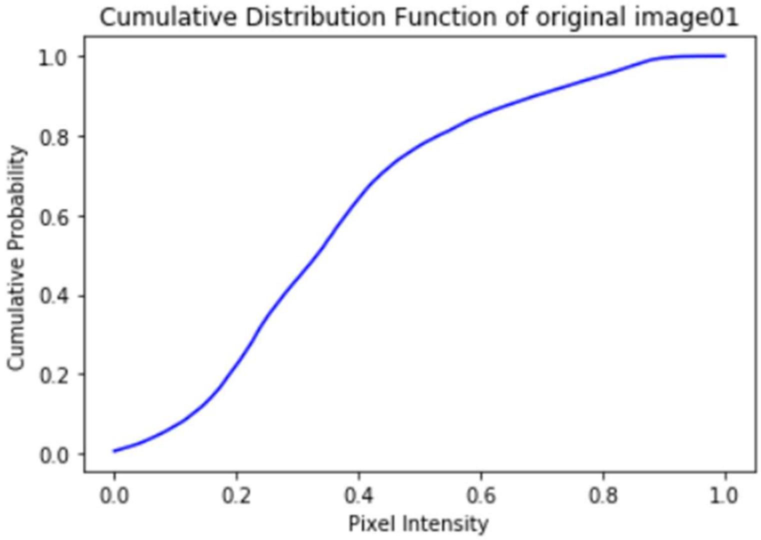
Original Image 01:



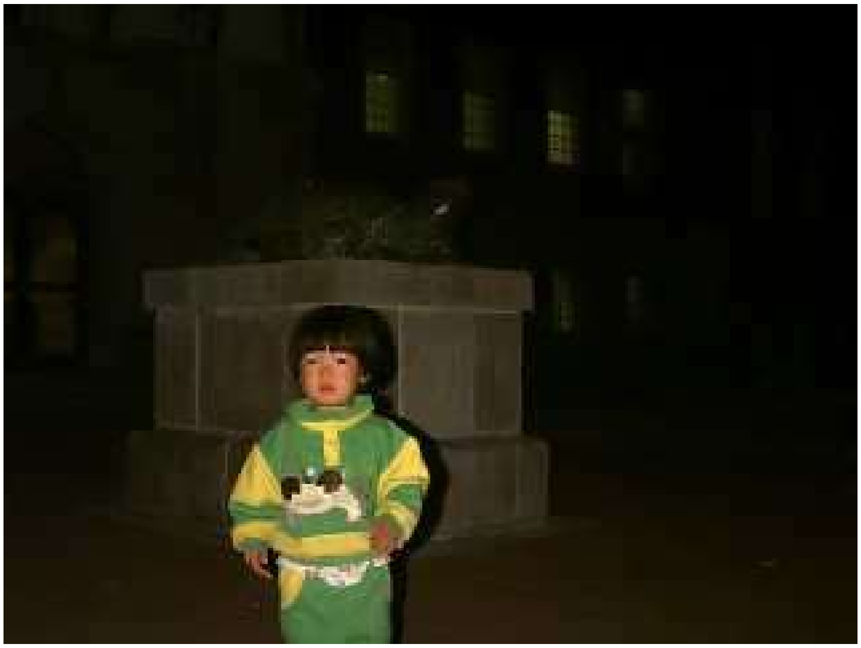
Original Image 01 Histogram:



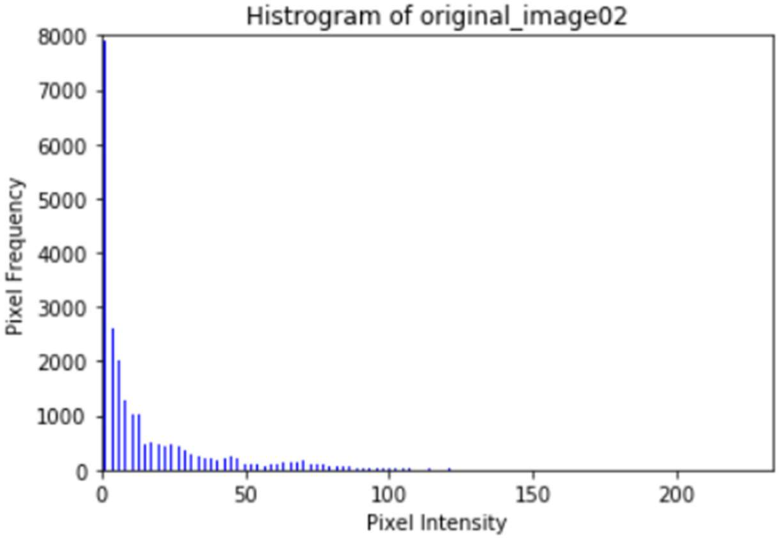
Original Image 01 CDF:



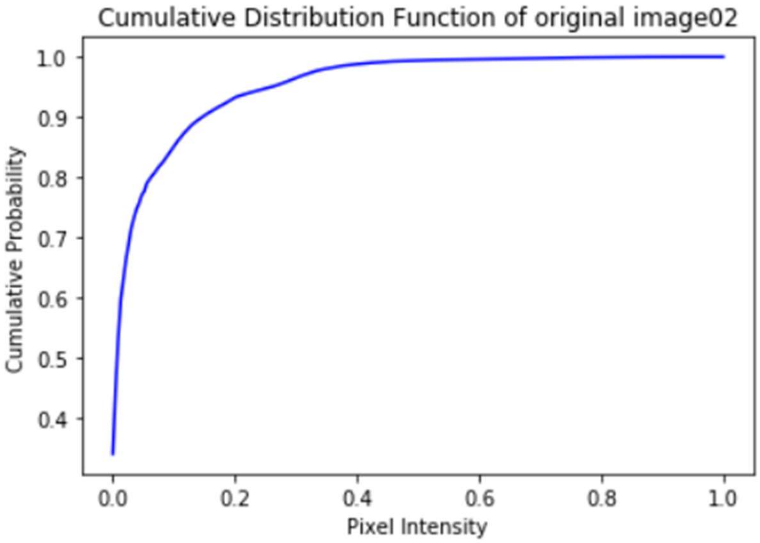
Original Image 02:



Histogram original image 02:

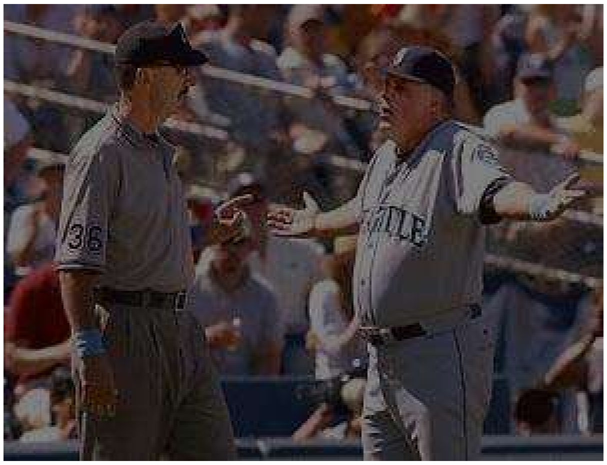


CDF original image 02:



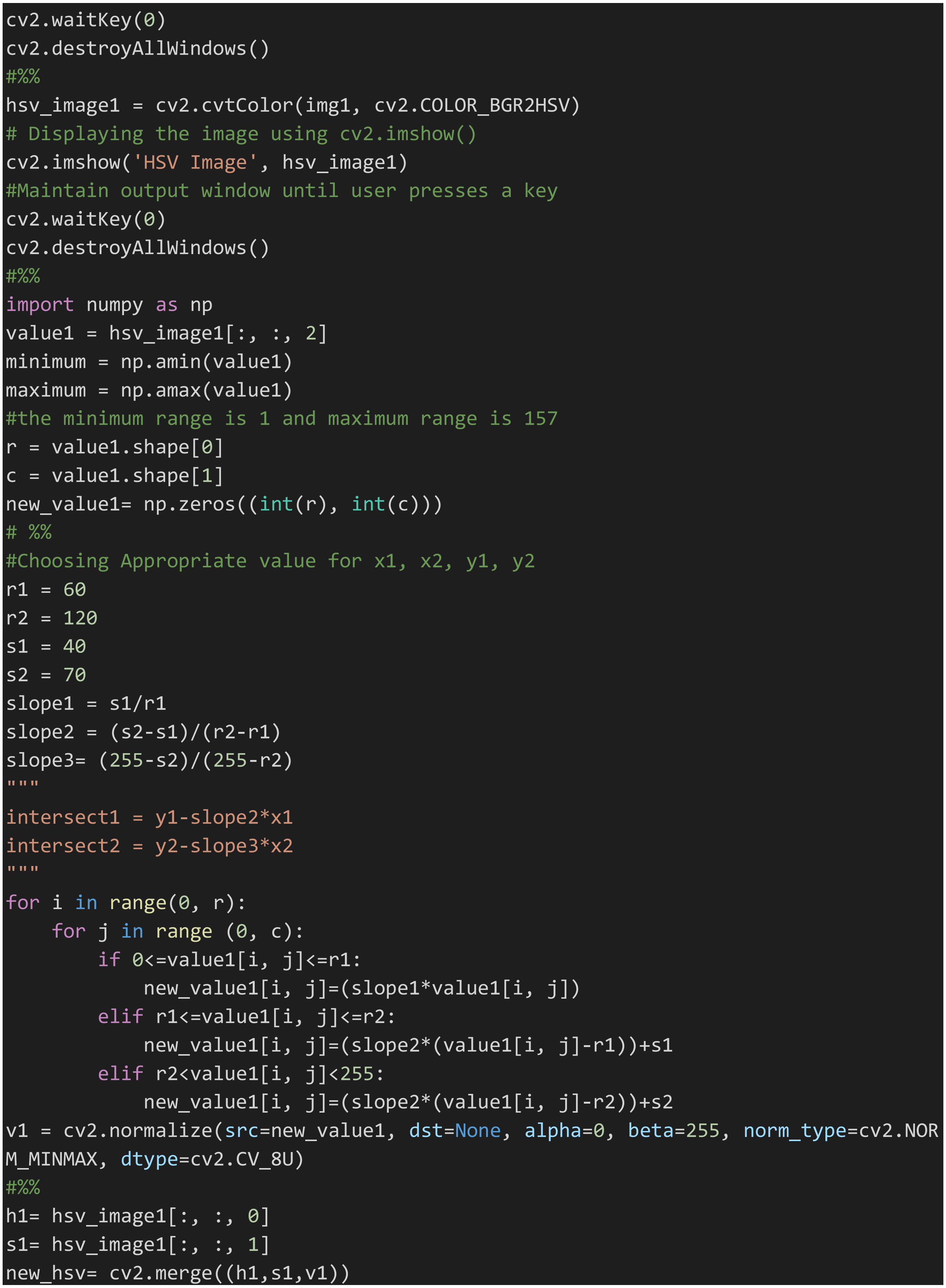
Linear Stretching:

Original Image 01



Code:

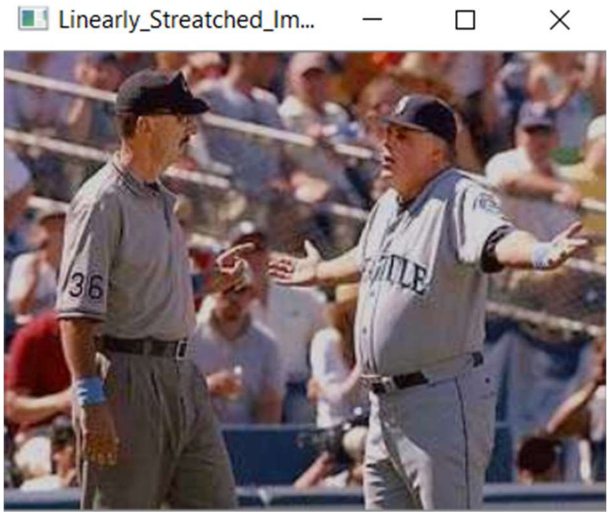
```
#%%
#Problem 2
#Linear Streatching
#importing cv2
import cv2
# Using cv2.imread() method
img1 = cv2.imread('3_2.jpg')
# Displaying the image using cv2.imshow()
cv2.imshow('3_2.jpg', img1)
#Maintain output window until user presses a key
```



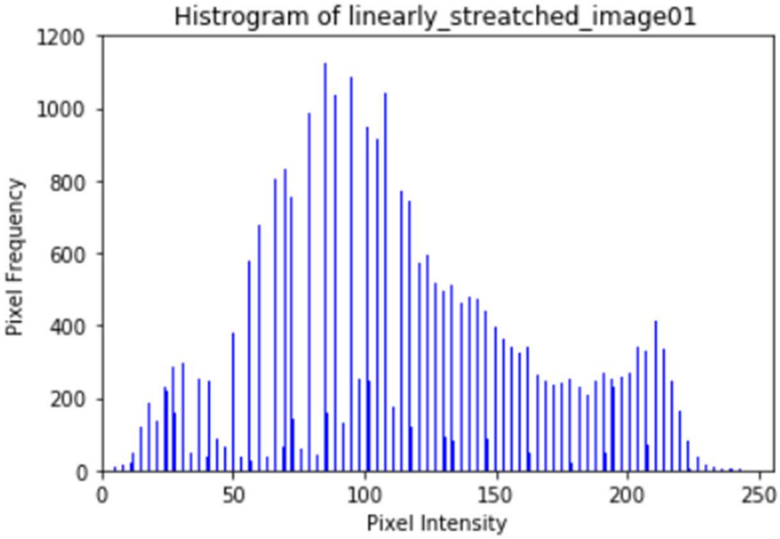
```
cv2.waitKey(0)
cv2.destroyAllWindows()
#%%
hsv_image1 = cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)
# Displaying the image using cv2.imshow()
cv2.imshow('HSV Image', hsv_image1)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
#%%
import numpy as np
value1 = hsv_image1[:, :, 2]
minimum = np.amin(value1)
maximum = np.amax(value1)
#the minimum range is 1 and maximum range is 157
r = value1.shape[0]
c = value1.shape[1]
new_value1= np.zeros((int(r), int(c)))
# %%
#Choosing Appropriate value for x1, x2, y1, y2
r1 = 60
r2 = 120
s1 = 40
s2 = 70
slope1 = s1/r1
slope2 = (s2-s1)/(r2-r1)
slope3= (255-s2)/(255-r2)
"""
intersect1 = y1-slope2*x1
intersect2 = y2-slope3*x2
"""
for i in range(0, r):
    for j in range (0, c):
        if 0<=value1[i, j]<=r1:
            new_value1[i, j]=(slope1*value1[i, j])
        elif r1<=value1[i, j]<=r2:
            new_value1[i, j]=(slope2*(value1[i, j]-r1))+s1
        elif r2<value1[i, j]<255:
            new_value1[i, j]=(slope2*(value1[i, j]-r2))+s2
v1 = cv2.normalize(src=new_value1, dst=None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
#%%
h1= hsv_image1[:, :, 0]
s1= hsv_image1[:, :, 1]
new_hsv= cv2.merge((h1,s1,v1))
```

```
final_rgb = cv2.cvtColor(new_hsv, cv2.COLOR_HSV2BGR)
# Displaying the image using cv2.imshow()
cv2.imshow('Linearly_Streatched_Image1', final_rgb)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
#%%
#Collecting values from 2D array
image_first_band = final_rgb[:,:,2]
rows = image_first_band.shape[0]
coloumns= image_first_band.shape[1]
values = []
for i in range(0, rows):
    for j in range(0, coloumns):
        values.append(image_first_band[i, j])
frequencies = {x:values.count(x) for x in values}
import collections
od = collections.OrderedDict(sorted(frequencies.items()))
#%%
#Histogram
import matplotlib.pyplot as plt
ax = plt.subplot(111)
w = 0.3
ax.bar(list(od.keys()), list(od.values()) , width=w, color='b', align='center')
ax.autoscale(tight=True)
plt.title("Histrogram of linearly_streatched_image01")
plt.ylim([0, 1200])
plt.xlabel("Pixel Intensity")
plt.ylabel("Pixel Frequency")
plt.show()
#%%
#CDF
import numpy as np
probability = []
for item in list(od.values()):
    probability.append(item/sum(list(od.values())))
cp = np.cumsum(probability).tolist()
od_list = list(od.keys())
amin, amax = min(od_list), max(od_list)
for i, val in enumerate(od_list):
    od_list[i] = (val-amin) / (amax-amin)
plt.xlabel("Pixel Intensity")
plt.ylabel("Cumulative Probability")
plt.title("Cumulative Distribution Function of linearly_streatched_image01")
plt.plot(od_list, cp, c='blue')
plt.show()
```

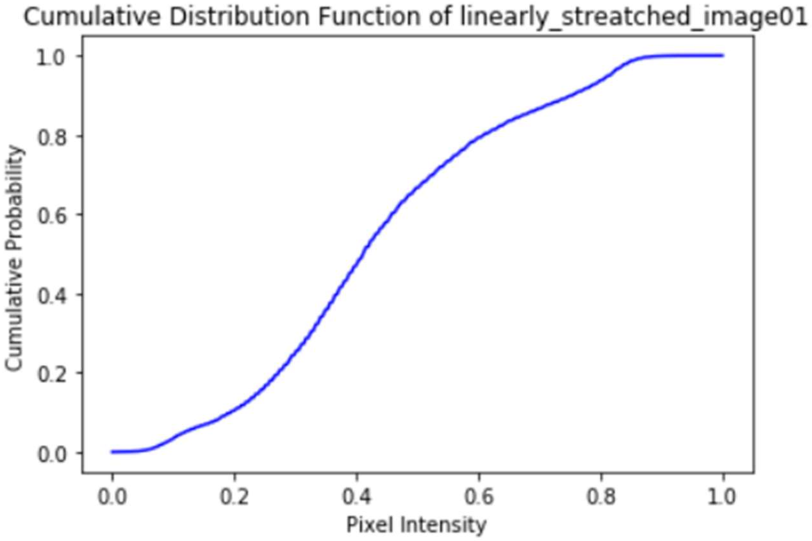
Resulting Image:



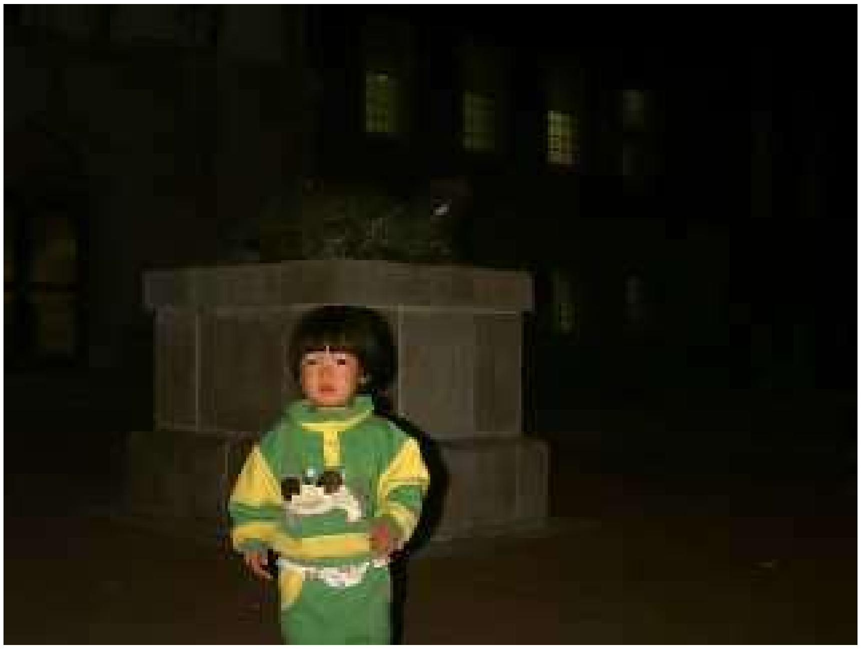
Linearly Stretched Image 01 histogram:



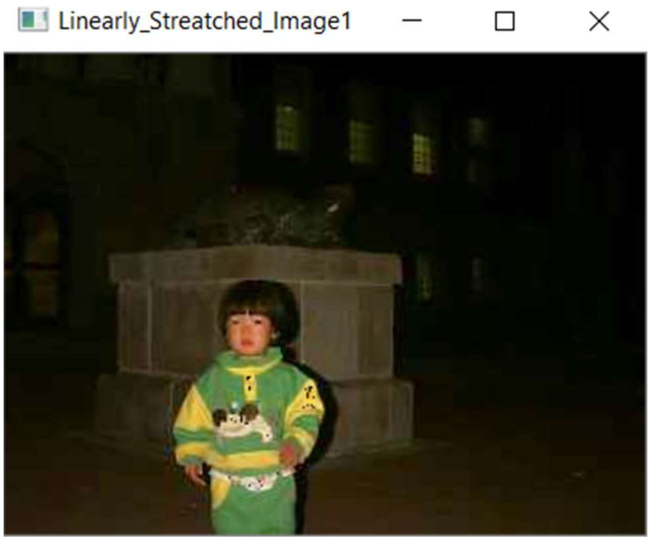
Linearly Stretched Image 01 CDF:



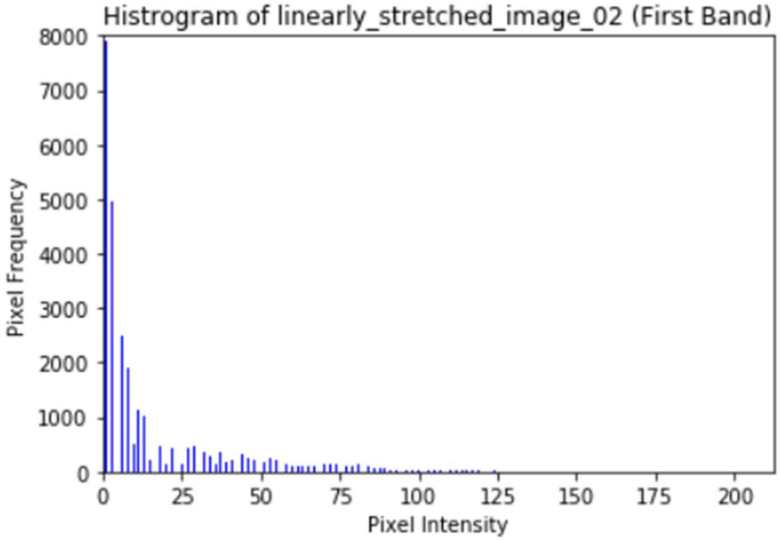
Original Image 02:



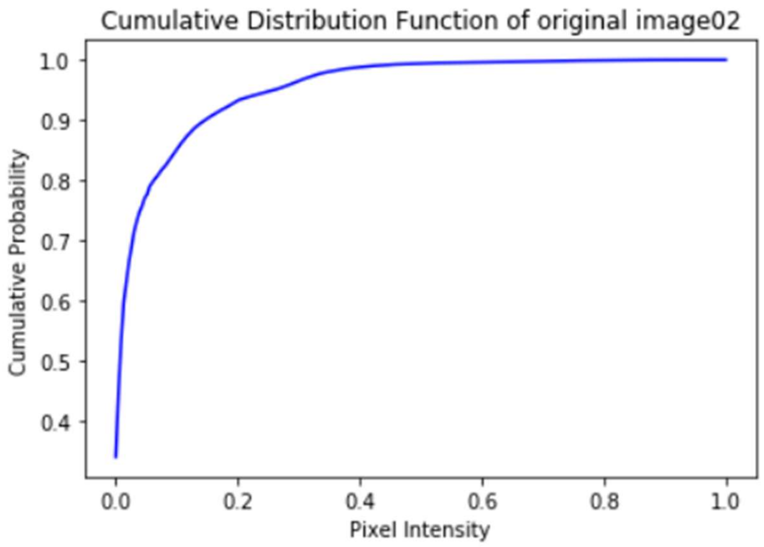
Linearly stretched image 02:



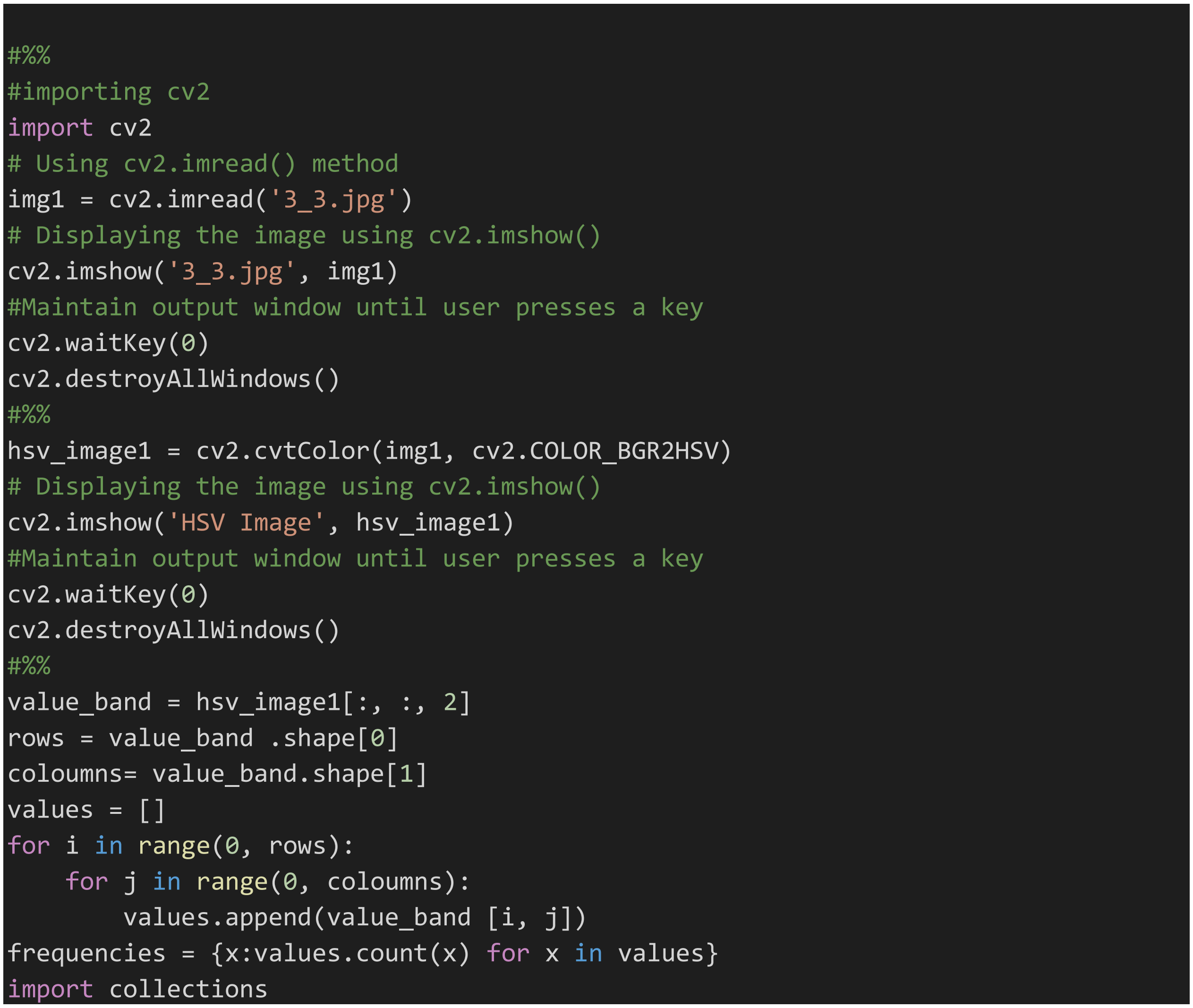
Linearly Stretched Image 02 histogram:



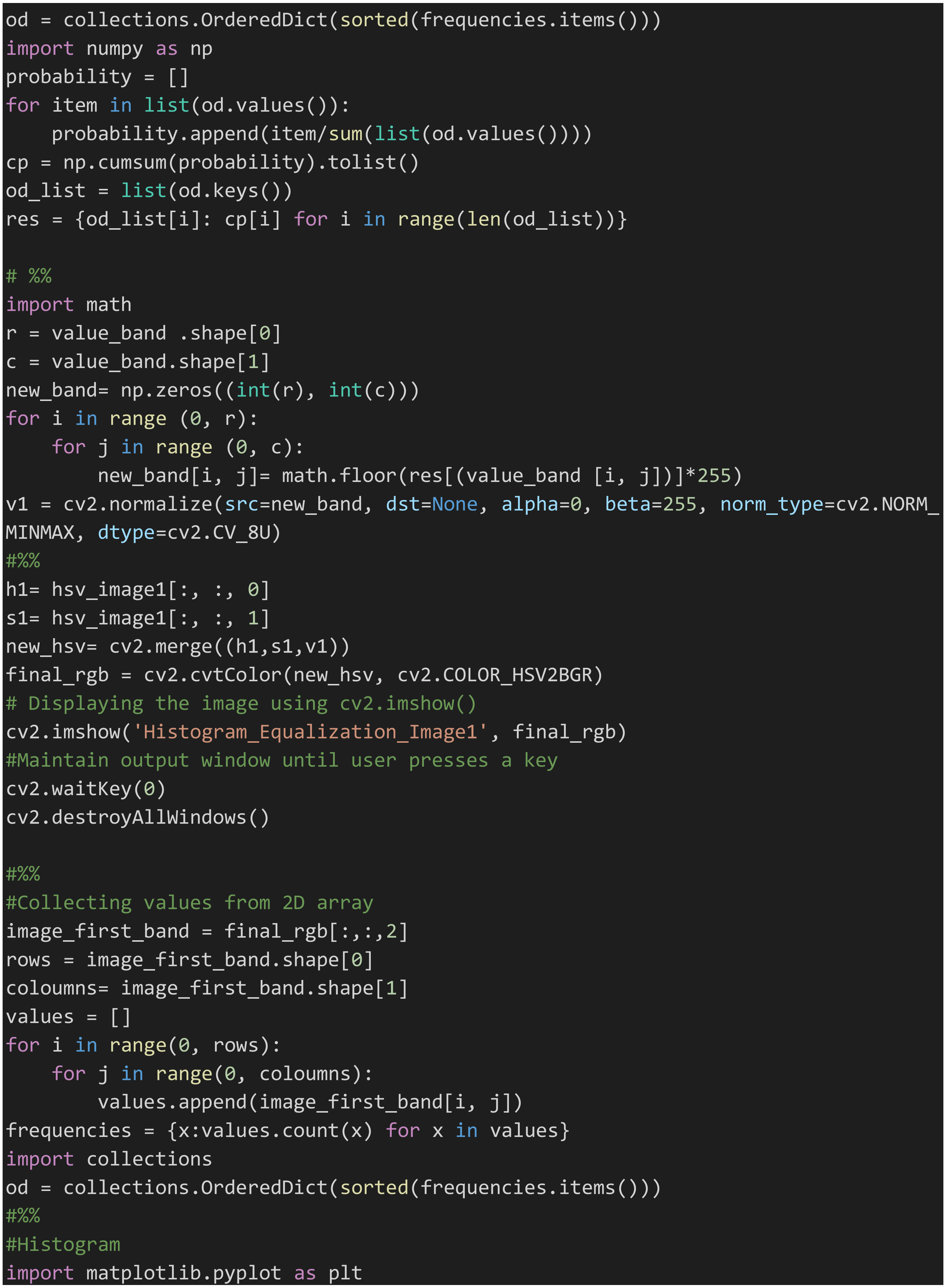
Linearly Stretched Image 01 CDF:



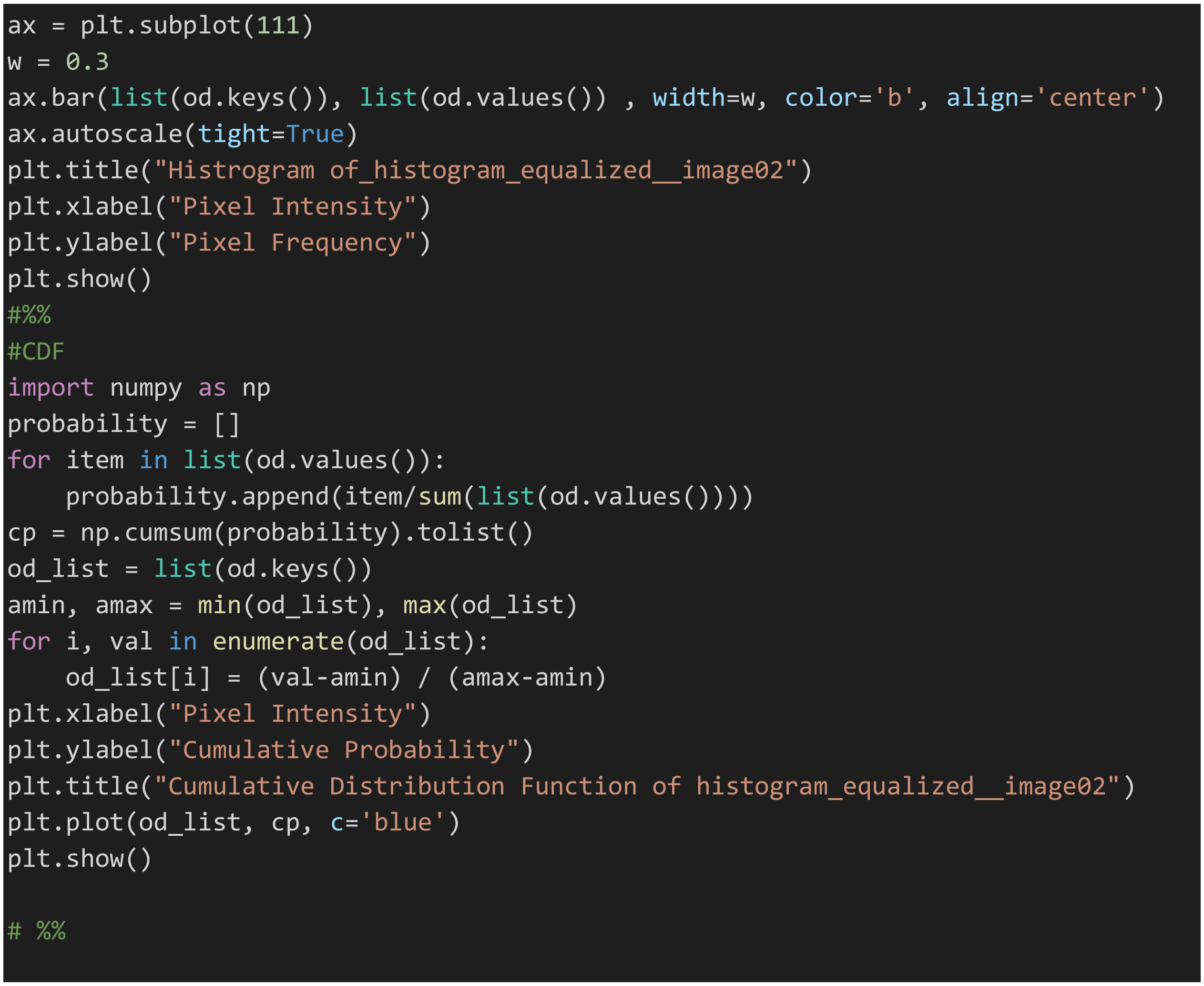
Histogram Equalization:



```
#%%
#importing cv2
import cv2
# Using cv2.imread() method
img1 = cv2.imread('3_3.jpg')
# Displaying the image using cv2.imshow()
cv2.imshow('3_3.jpg', img1)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
#%%
hsv_image1 = cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)
# Displaying the image using cv2.imshow()
cv2.imshow('HSV Image', hsv_image1)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
#%%
value_band = hsv_image1[:, :, 2]
rows = value_band .shape[0]
coloumns= value_band.shape[1]
values = []
for i in range(0, rows):
    for j in range(0, coloumns):
        values.append(value_band [i, j])
frequencies = {x:values.count(x) for x in values}
import collections
```

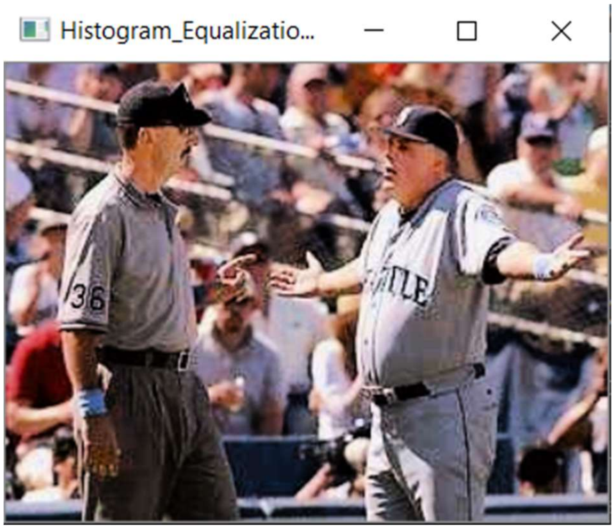


```
od = collections.OrderedDict(sorted(frequencies.items()))
import numpy as np
probability = []
for item in list(od.values()):
    probability.append(item/sum(list(od.values())))
cp = np.cumsum(probability).tolist()
od_list = list(od.keys())
res = {od_list[i]: cp[i] for i in range(len(od_list))}

# %%
import math
r = value_band .shape[0]
c = value_band.shape[1]
new_band= np.zeros((int(r), int(c)))
for i in range (0, r):
    for j in range (0, c):
        new_band[i, j]= math.floor(res[(value_band [i, j])]*255)
v1 = cv2.normalize(src=new_band, dst=None, alpha=0, beta=255, norm_type=cv2.NORM_
MINMAX, dtype=cv2.CV_8U)
#%%
h1= hsv_image1[:, :, 0]
s1= hsv_image1[:, :, 1]
new_hsv= cv2.merge((h1,s1,v1))
final_rgb = cv2.cvtColor(new_hsv, cv2.COLOR_HSV2BGR)
# Displaying the image using cv2.imshow()
cv2.imshow('Histogram_Equalization_Image1', final_rgb)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()

#%%
#Collecting values from 2D array
image_first_band = final_rgb[:,:,2]
rows = image_first_band.shape[0]
coloumns= image_first_band.shape[1]
values = []
for i in range(0, rows):
    for j in range(0, coloumns):
        values.append(image_first_band[i, j])
frequencies = {x:values.count(x) for x in values}
import collections
od = collections.OrderedDict(sorted(frequencies.items()))
#%%
#Histogram
import matplotlib.pyplot as plt
```

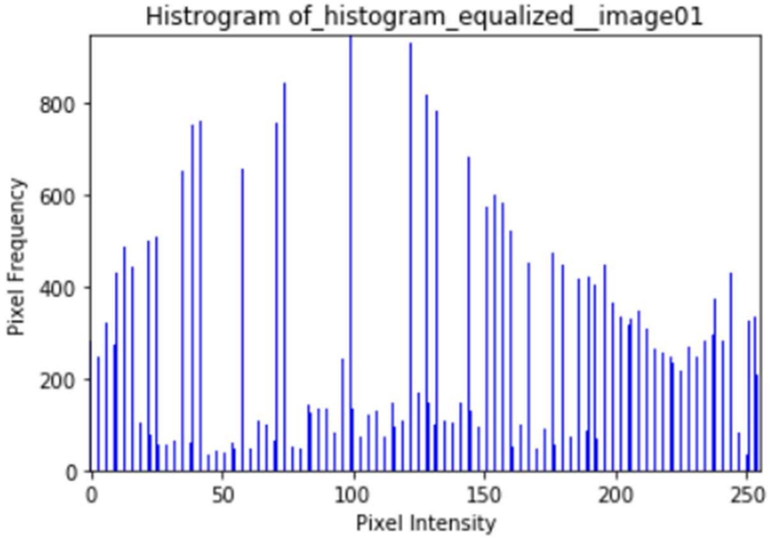


```
ax = plt.subplot(111)
w = 0.3
ax.bar(list(od.keys()), list(od.values()) , width=w, color='b', align='center')
ax.autoscale(tight=True)
plt.title("Histrogram of_histogram_equalized__image02")
plt.xlabel("Pixel Intensity")
plt.ylabel("Pixel Frequency")
plt.show()
#%%
#CDF
import numpy as np
probability = []
for item in list(od.values()):
    probability.append(item/sum(list(od.values())))
cp = np.cumsum(probability).tolist()
od_list = list(od.keys())
amin, amax = min(od_list), max(od_list)
for i, val in enumerate(od_list):
    od_list[i] = (val-amin) / (amax-amin)
plt.xlabel("Pixel Intensity")
plt.ylabel("Cumulative Probability")
plt.title("Cumulative Distribution Function of histogram_equalized__image02")
plt.plot(od_list, cp, c='blue')
plt.show()

# %%
```

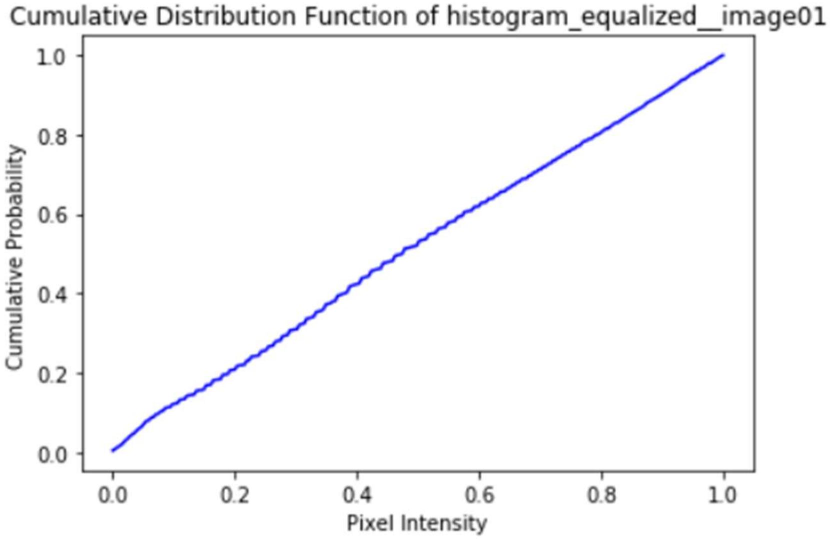
Resulting Image 01:



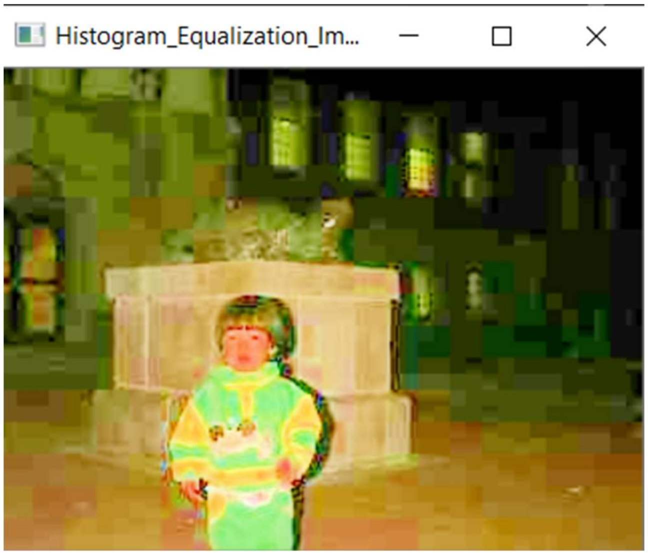
Histogram of histogram equalized image01:



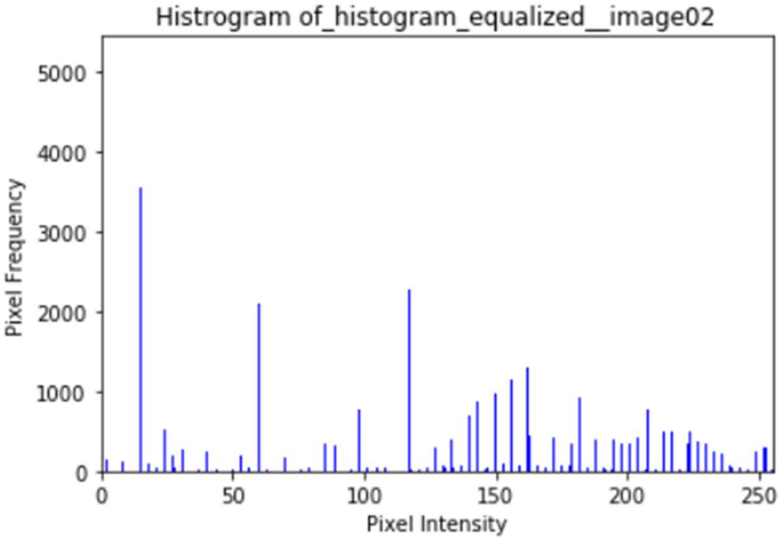
CDF of histogram equalized image01:



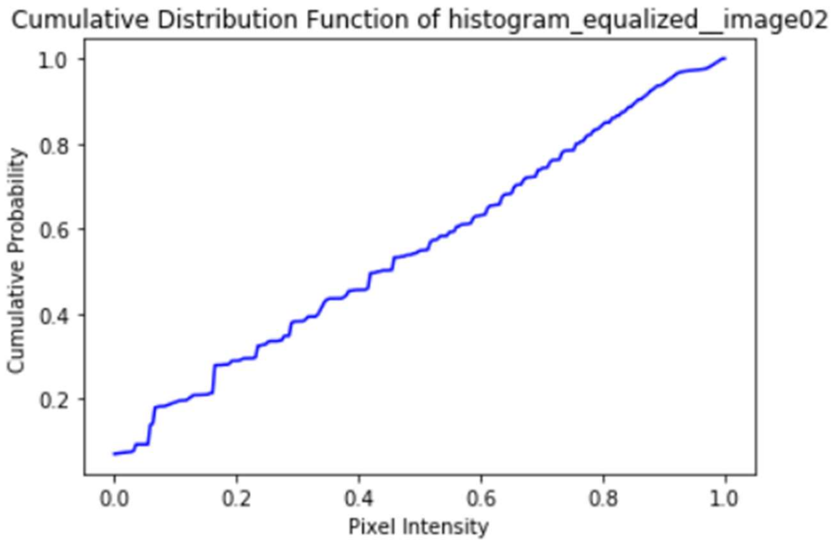
Resulting Image 02:



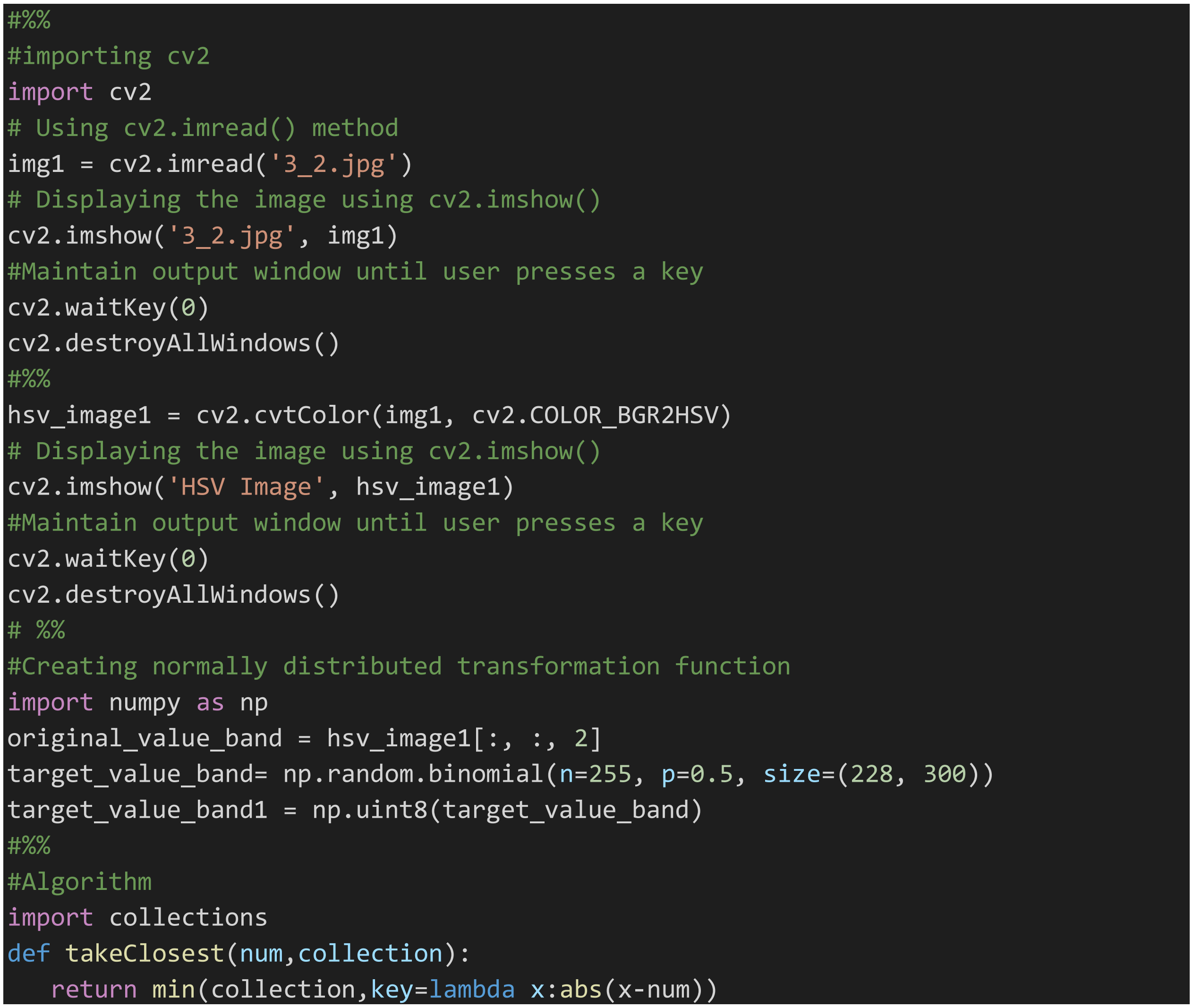
Histogram of histogram equalized image02:



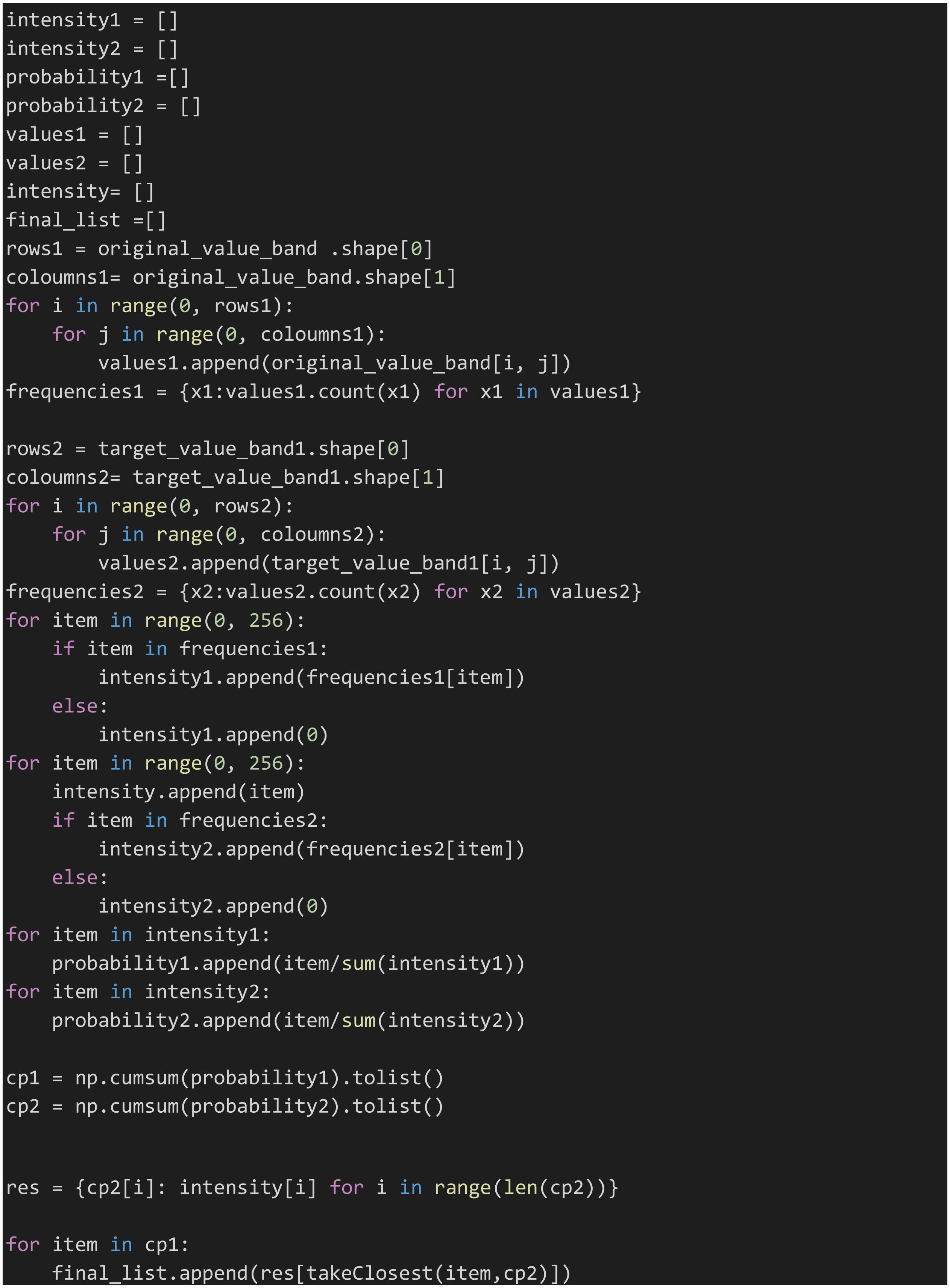
CDFof histogram equalized image02:



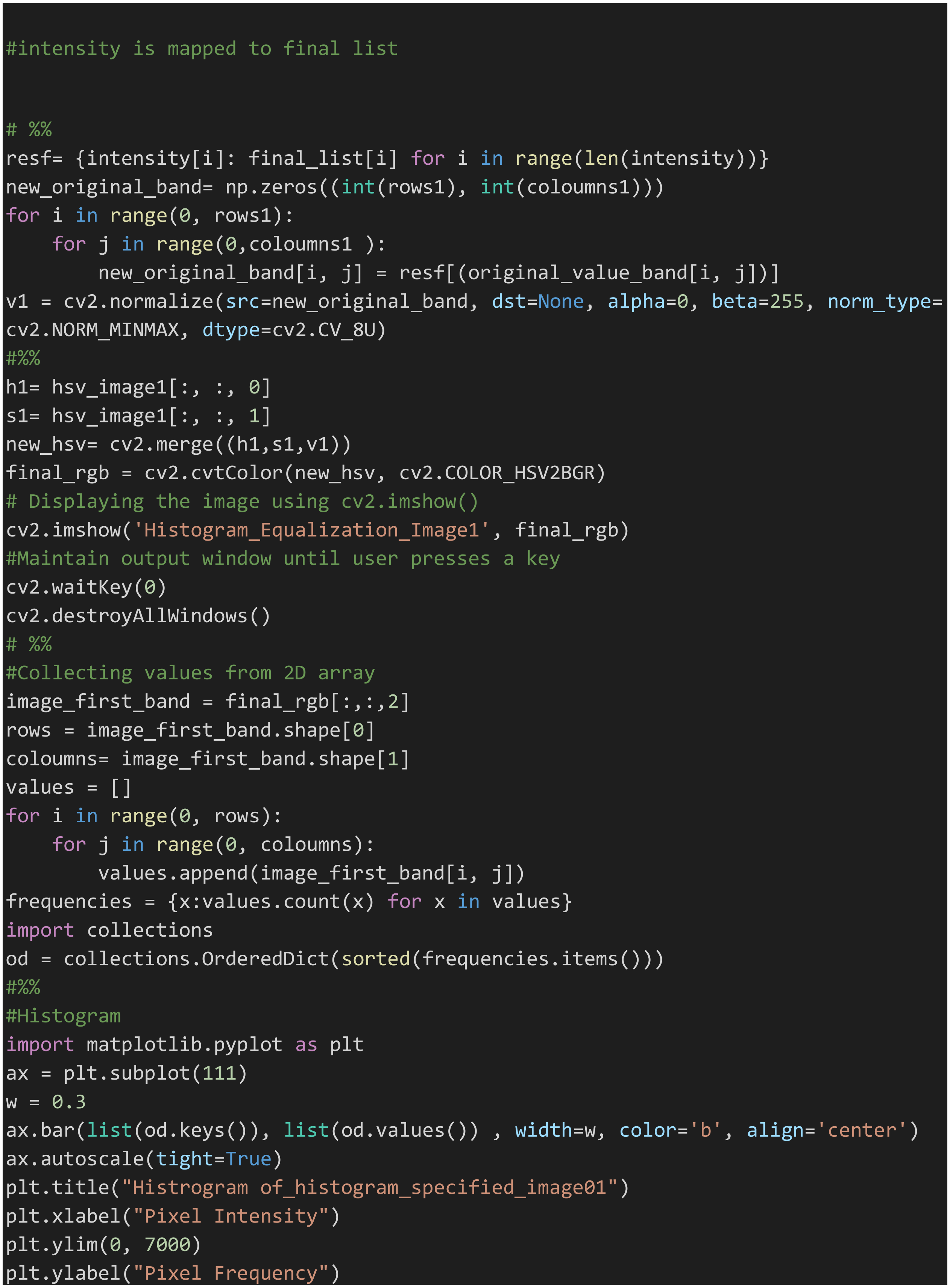
Histogram Specification:



```
#%%
#importing cv2
import cv2
# Using cv2.imread() method
img1 = cv2.imread('3_2.jpg')
# Displaying the image using cv2.imshow()
cv2.imshow('3_2.jpg', img1)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
#%%
hsv_image1 = cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)
# Displaying the image using cv2.imshow()
cv2.imshow('HSV Image', hsv_image1)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
# %%
#Creating normally distributed transformation function
import numpy as np
original_value_band = hsv_image1[:, :, 2]
target_value_band= np.random.binomial(n=255, p=0.5, size=(228, 300))
target_value_band1 = np.uint8(target_value_band)
#%%
#Algorithm
import collections
def takeClosest(num,collection):
   return min(collection,key=lambda x:abs(x-num))
```

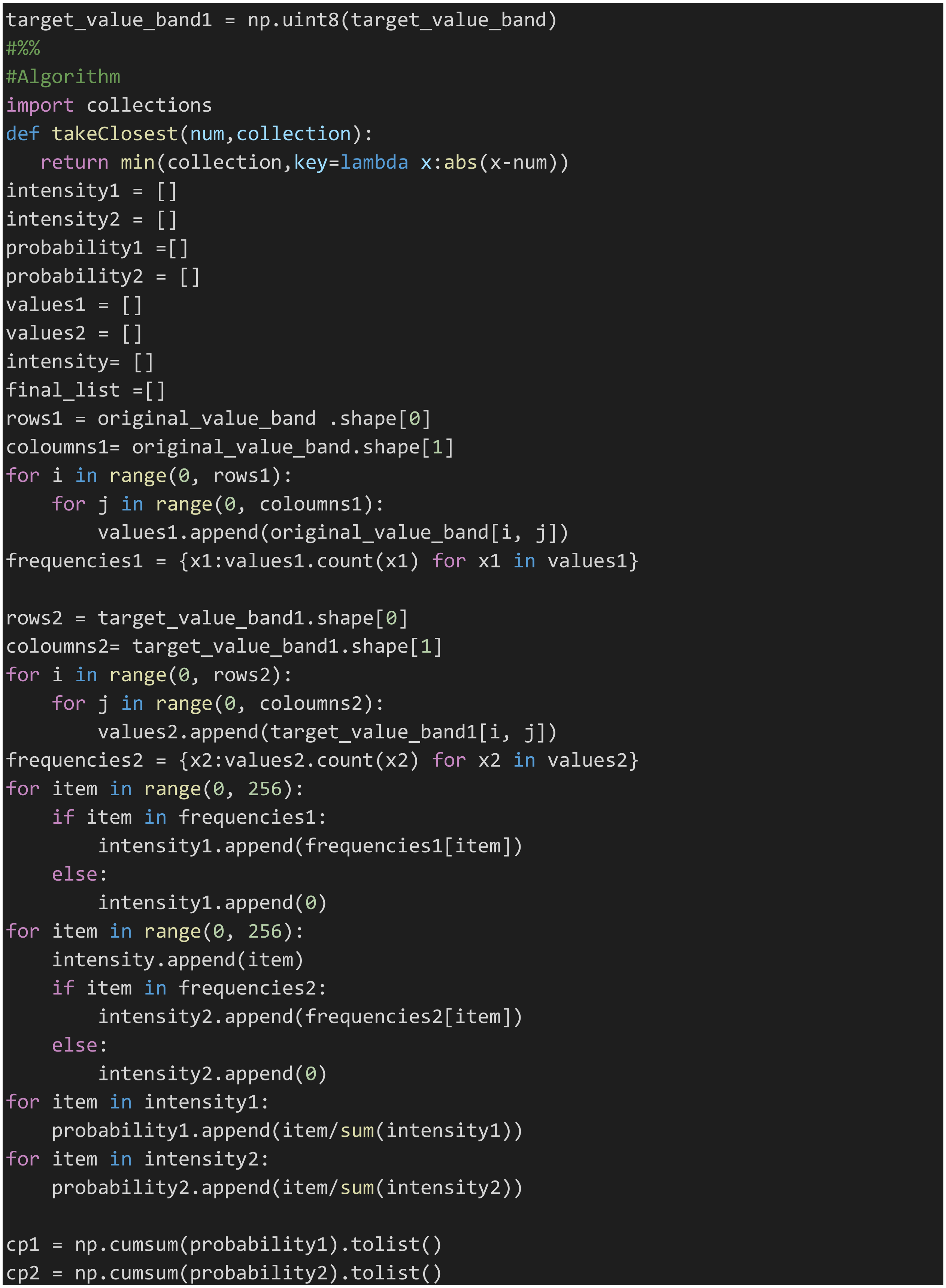


```python
intensity1 = []
intensity2 = []
probability1 =[]
probability2 = []
values1 = []
values2 = []
intensity= []
final_list =[]
rows1 = original_value_band .shape[0]
coloumns1= original_value_band.shape[1]
for i in range(0, rows1):
    for j in range(0, coloumns1):
        values1.append(original_value_band[i, j])
frequencies1 = {x1:values1.count(x1) for x1 in values1}

rows2 = target_value_band1.shape[0]
coloumns2= target_value_band1.shape[1]
for i in range(0, rows2):
    for j in range(0, coloumns2):
        values2.append(target_value_band1[i, j])
frequencies2 = {x2:values2.count(x2) for x2 in values2}
for item in range(0, 256):
    if item in frequencies1:
        intensity1.append(frequencies1[item])
    else:
        intensity1.append(0)
for item in range(0, 256):
    intensity.append(item)
    if item in frequencies2:
        intensity2.append(frequencies2[item])
    else:
        intensity2.append(0)
for item in intensity1:
    probability1.append(item/sum(intensity1))
for item in intensity2:
    probability2.append(item/sum(intensity2))

cp1 = np.cumsum(probability1).tolist()
cp2 = np.cumsum(probability2).tolist()


res = {cp2[i]: intensity[i] for i in range(len(cp2))}

for item in cp1:
    final_list.append(res[takeClosest(item,cp2)])
```

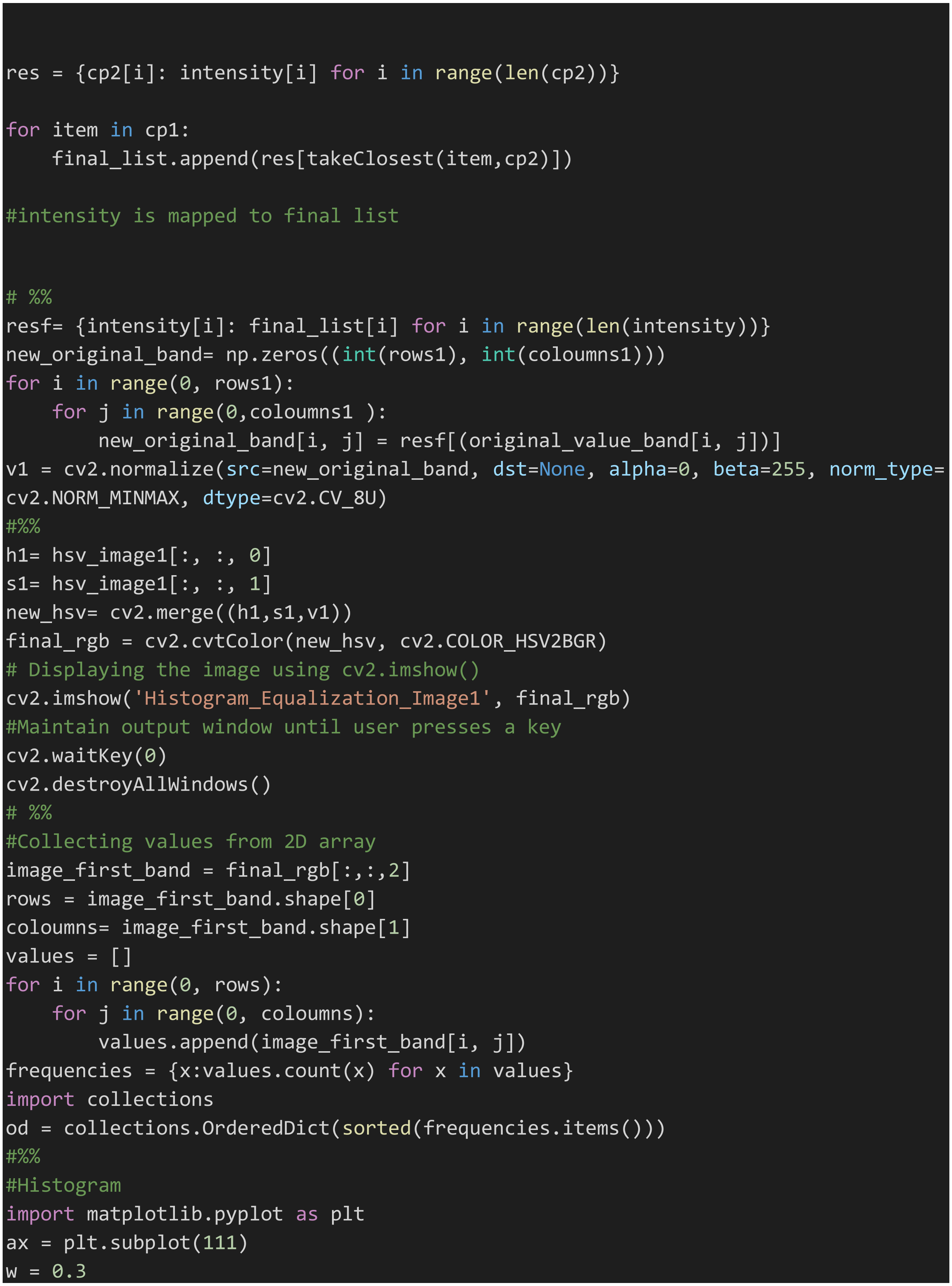


```
#intensity is mapped to final list


# %%
resf= {intensity[i]: final_list[i] for i in range(len(intensity))}
new_original_band= np.zeros((int(rows1), int(coloumns1)))
for i in range(0, rows1):
    for j in range(0,coloumns1 ):
        new_original_band[i, j] = resf[(original_value_band[i, j])]
v1 = cv2.normalize(src=new_original_band, dst=None, alpha=0, beta=255, norm_type=
cv2.NORM_MINMAX, dtype=cv2.CV_8U)
#%%
h1= hsv_image1[:, :, 0]
s1= hsv_image1[:, :, 1]
new_hsv= cv2.merge((h1,s1,v1))
final_rgb = cv2.cvtColor(new_hsv, cv2.COLOR_HSV2BGR)
# Displaying the image using cv2.imshow()
cv2.imshow('Histogram_Equalization_Image1', final_rgb)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
# %%
#Collecting values from 2D array
image_first_band = final_rgb[:,:,2]
rows = image_first_band.shape[0]
coloumns= image_first_band.shape[1]
values = []
for i in range(0, rows):
    for j in range(0, coloumns):
        values.append(image_first_band[i, j])
frequencies = {x:values.count(x) for x in values}
import collections
od = collections.OrderedDict(sorted(frequencies.items()))
#%%
#Histogram
import matplotlib.pyplot as plt
ax = plt.subplot(111)
w = 0.3
ax.bar(list(od.keys()), list(od.values()) , width=w, color='b', align='center')
ax.autoscale(tight=True)
plt.title("Histrogram of_histogram_specified_image01")
plt.xlabel("Pixel Intensity")
plt.ylim(0, 7000)
plt.ylabel("Pixel Frequency")
plt.show()
#%%
#CDF
import numpy as np
probability = []
for item in list(od.values()):
    probability.append(item/sum(list(od.values())))
cp = np.cumsum(probability).tolist()
od_list = list(od.keys())
amin, amax = min(od_list), max(od_list)
for i, val in enumerate(od_list):
    od_list[i] = (val-amin) / (amax-amin)
plt.xlabel("Pixel Intensity")
plt.ylabel("Cumulative Probability")
plt.title("Cumulative Distribution Function of histogram_specified_image01")
plt.plot(od_list, cp, c='blue')
plt.show()


# %%
```

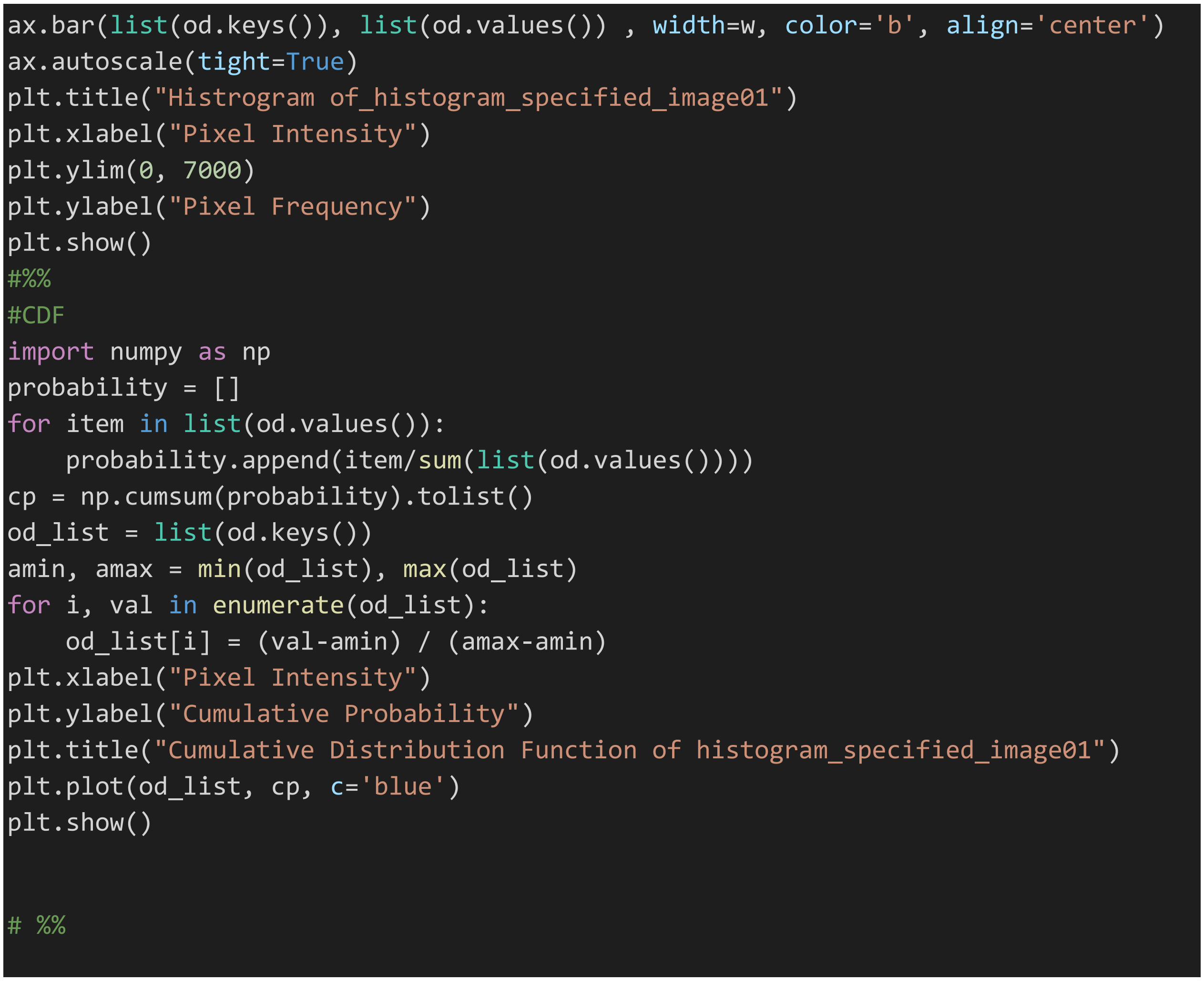
```
Resulting Image 01:  #%%

#importing cv2
import cv2
# Using cv2.imread() method
img1 = cv2.imread('3_2.jpg')
# Displaying the image using cv2.imshow()
cv2.imshow('3_2.jpg', img1)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
#%%
hsv_image1 = cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)
# Displaying the image using cv2.imshow()
cv2.imshow('HSV Image', hsv_image1)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
# %%
#Creating normally distributed transformation function
import numpy as np
original_value_band = hsv_image1[:, :, 2]
target_value_band= np.random.binomial(n=255, p=0.5, size=(228, 300))
```



```
target_value_band1 = np.uint8(target_value_band)
#%%
#Algorithm
import collections
def takeClosest(num,collection):
   return min(collection,key=lambda x:abs(x-num))
intensity1 = []
intensity2 = []
probability1 =[]
probability2 = []
values1 = []
values2 = []
intensity= []
final_list =[]
rows1 = original_value_band .shape[0]
coloumns1= original_value_band.shape[1]
for i in range(0, rows1):
    for j in range(0, coloumns1):
        values1.append(original_value_band[i, j])
frequencies1 = {x1:values1.count(x1) for x1 in values1}

rows2 = target_value_band1.shape[0]
coloumns2= target_value_band1.shape[1]
for i in range(0, rows2):
    for j in range(0, coloumns2):
        values2.append(target_value_band1[i, j])
frequencies2 = {x2:values2.count(x2) for x2 in values2}
for item in range(0, 256):
    if item in frequencies1:
        intensity1.append(frequencies1[item])
    else:
        intensity1.append(0)
for item in range(0, 256):
    intensity.append(item)
    if item in frequencies2:
        intensity2.append(frequencies2[item])
    else:
        intensity2.append(0)
for item in intensity1:
    probability1.append(item/sum(intensity1))
for item in intensity2:
    probability2.append(item/sum(intensity2))

cp1 = np.cumsum(probability1).tolist()
cp2 = np.cumsum(probability2).tolist()
```

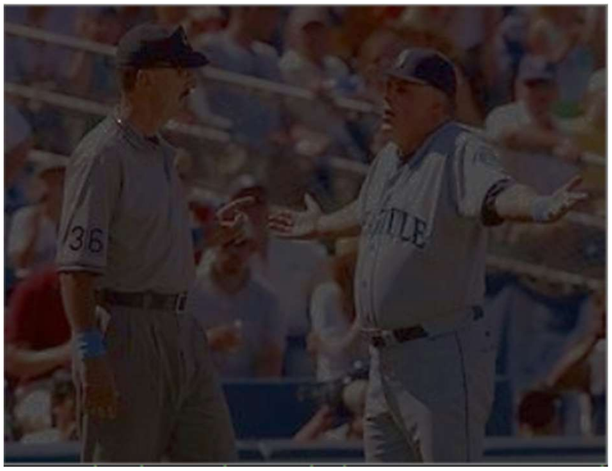


```
res = {cp2[i]: intensity[i] for i in range(len(cp2))}

for item in cp1:
    final_list.append(res[takeClosest(item,cp2)])

#intensity is mapped to final list


# %%
resf= {intensity[i]: final_list[i] for i in range(len(intensity))}
new_original_band= np.zeros((int(rows1), int(coloumns1)))
for i in range(0, rows1):
    for j in range(0,coloumns1 ):
        new_original_band[i, j] = resf[(original_value_band[i, j])]
v1 = cv2.normalize(src=new_original_band, dst=None, alpha=0, beta=255, norm_type=
cv2.NORM_MINMAX, dtype=cv2.CV_8U)
#%%
h1= hsv_image1[:, :, 0]
s1= hsv_image1[:, :, 1]
new_hsv= cv2.merge((h1,s1,v1))
final_rgb = cv2.cvtColor(new_hsv, cv2.COLOR_HSV2BGR)
# Displaying the image using cv2.imshow()
cv2.imshow('Histogram_Equalization_Image1', final_rgb)
#Maintain output window until user presses a key
cv2.waitKey(0)
cv2.destroyAllWindows()
# %%
#Collecting values from 2D array
image_first_band = final_rgb[:,:,2]
rows = image_first_band.shape[0]
coloumns= image_first_band.shape[1]
values = []
for i in range(0, rows):
    for j in range(0, coloumns):
        values.append(image_first_band[i, j])
frequencies = {x:values.count(x) for x in values}
import collections
od = collections.OrderedDict(sorted(frequencies.items()))
#%%
#Histogram
import matplotlib.pyplot as plt
ax = plt.subplot(111)
w = 0.3
```

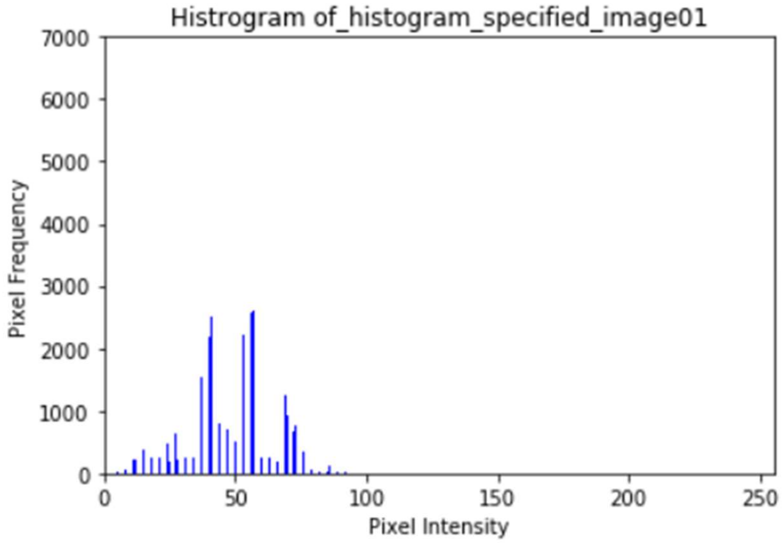


```
ax.bar(list(od.keys()), list(od.values()) , width=w, color='b', align='center')
ax.autoscale(tight=True)
plt.title("Histrogram of_histogram_specified_image01")
plt.xlabel("Pixel Intensity")
plt.ylim(0, 7000)
plt.ylabel("Pixel Frequency")
plt.show()
#%%
#CDF
import numpy as np
probability = []
for item in list(od.values()):
    probability.append(item/sum(list(od.values())))
cp = np.cumsum(probability).tolist()
od_list = list(od.keys())
amin, amax = min(od_list), max(od_list)
for i, val in enumerate(od_list):
    od_list[i] = (val-amin) / (amax-amin)
plt.xlabel("Pixel Intensity")
plt.ylabel("Cumulative Probability")
plt.title("Cumulative Distribution Function of histogram_specified_image01")
plt.plot(od_list, cp, c='blue')
plt.show()


# %%
```

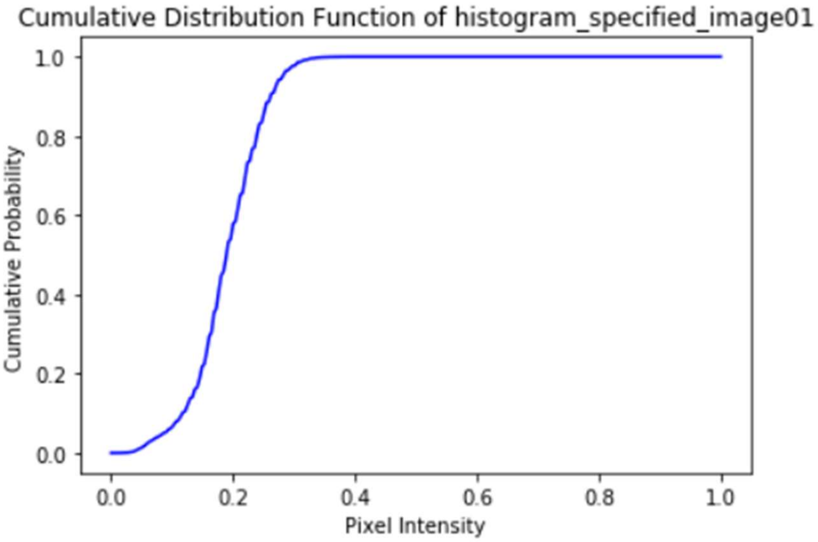
Resulting Image 01:



Histogram

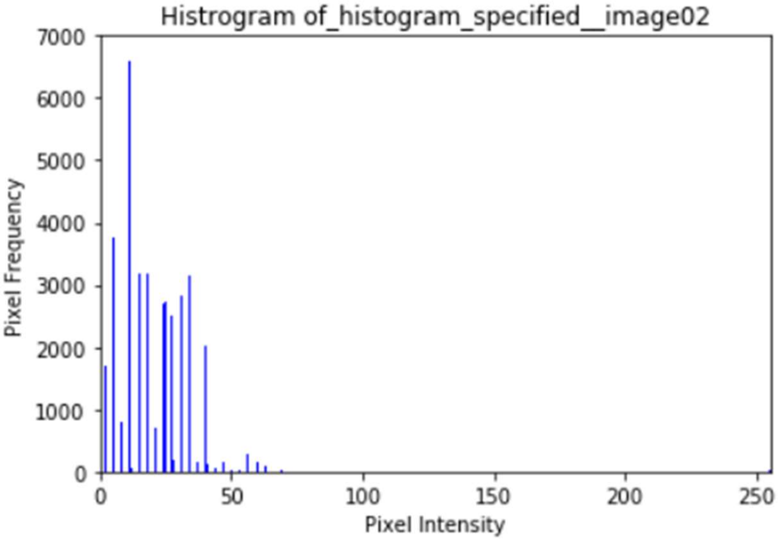


CDF:

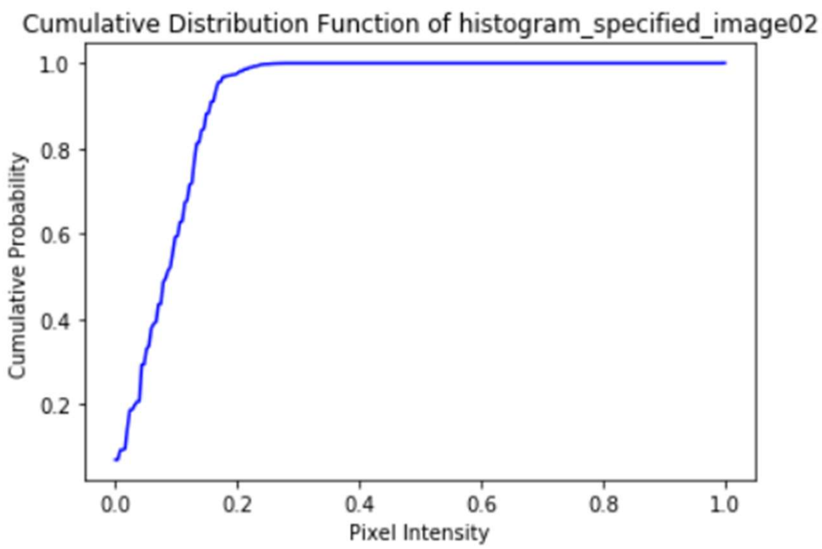


Resulting Image 02:

Histogram histogram specified_image 02:

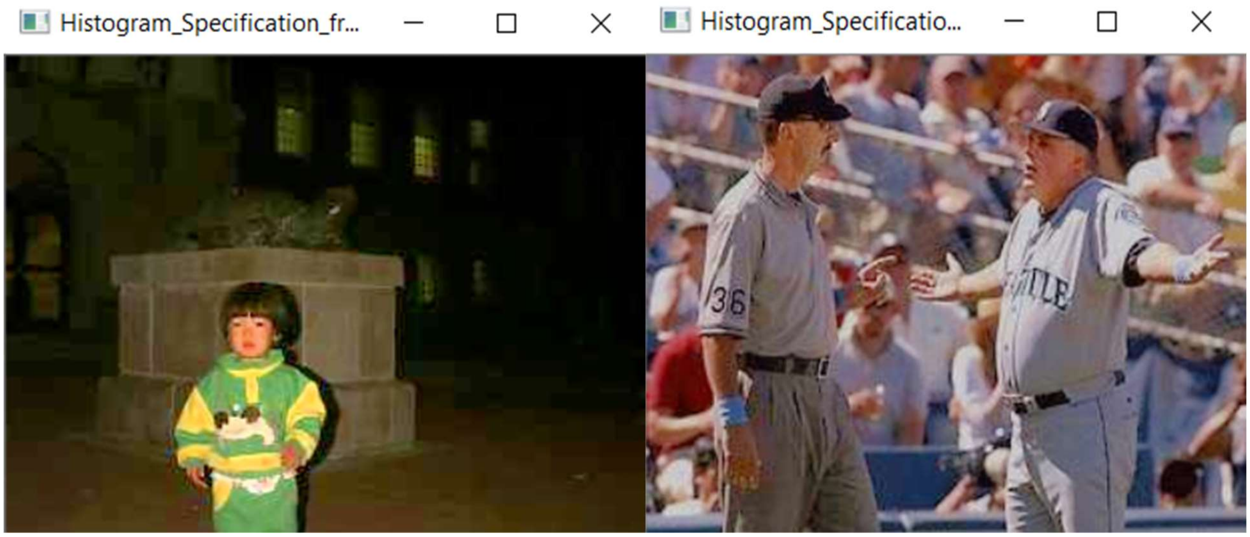


CDF of histogram specified image:



Between these three techniques of linear stretching, histogram equalization and histogram specification, it depends on image type and our desired output to find out which technique is good. In this case for first image, linear stretching technique worked well. For second image, again linear stretching technique worked better. After histogram equalization both images were too bright. So, I used a 2d array which is normally distributed as a target transformation function in order to implement histogram specification. But this time both images were too dark. I got better results when I didn't use the normally distributed array but an array from a target image. Then histogram specification worked.

Histogram Specification while used target image's array as transformation function:


Histogram_Specification_fr...
Histogram_Specificatio...
36