Jin Chen
50200769

Task 2

For Question 1-3:

Algorithm:

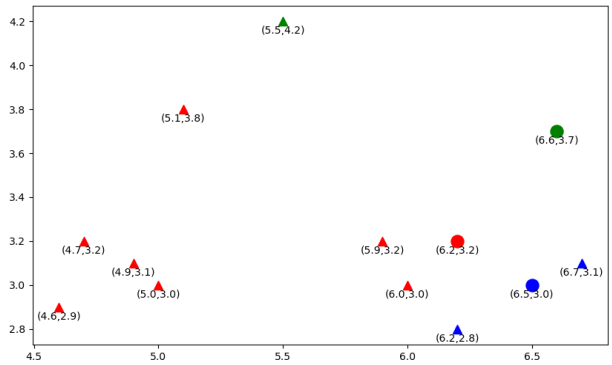
k-means core algorithm function at code line: 52-83

This function logic:

1. Take k, X, number of iterations and initial centroid as its inputs
2. If there is no initial centroid, it will generate one
3. Set a while loop to limit iterations
4. Create empty np.array, **dis**, with shape of (k, N), **N = len(X)
5. Loop through k centroids, and compute the Euclidean distance between every points in X and each centroid. After that, store the distance in the array **dis**, and its shape is ((k, N)).
6. Transpose **dis (shape now is (N,k))** and find the minimum value/distance for each row and store label to the row. This will give me the classification vector with shape (N,1), and each element is the label/cluster. If k=3, then the label will be 0,1,2.

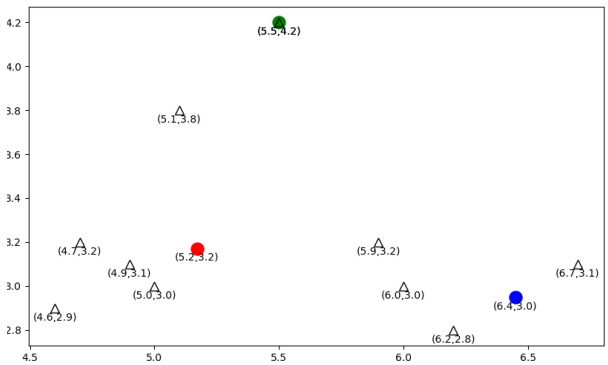
Result Plots:

task2_iter1_a.png



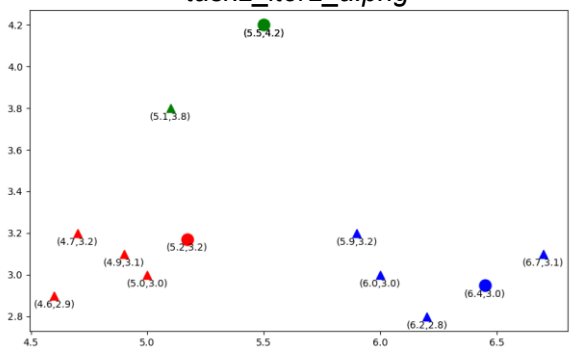
Fill empty triangles with initial centroid

task2_iter1_b.png



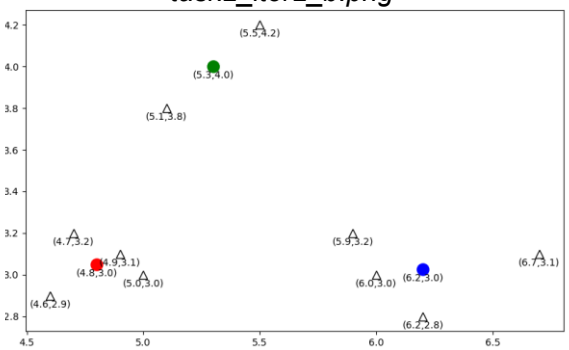
Plot new centroid computed from new cluster

task2_iter2_a.png



Fill empty triangles with 1st iteration centroid

task2_iter2_b.png



Plot new centroid computed from new cluster

For Question 4:

** Note:

- The image directory is '*Project3_clustering/baboon.png*' by default, you can change it at *code line: 209*
- Detail of processes for question 4 will print out while running task2.py

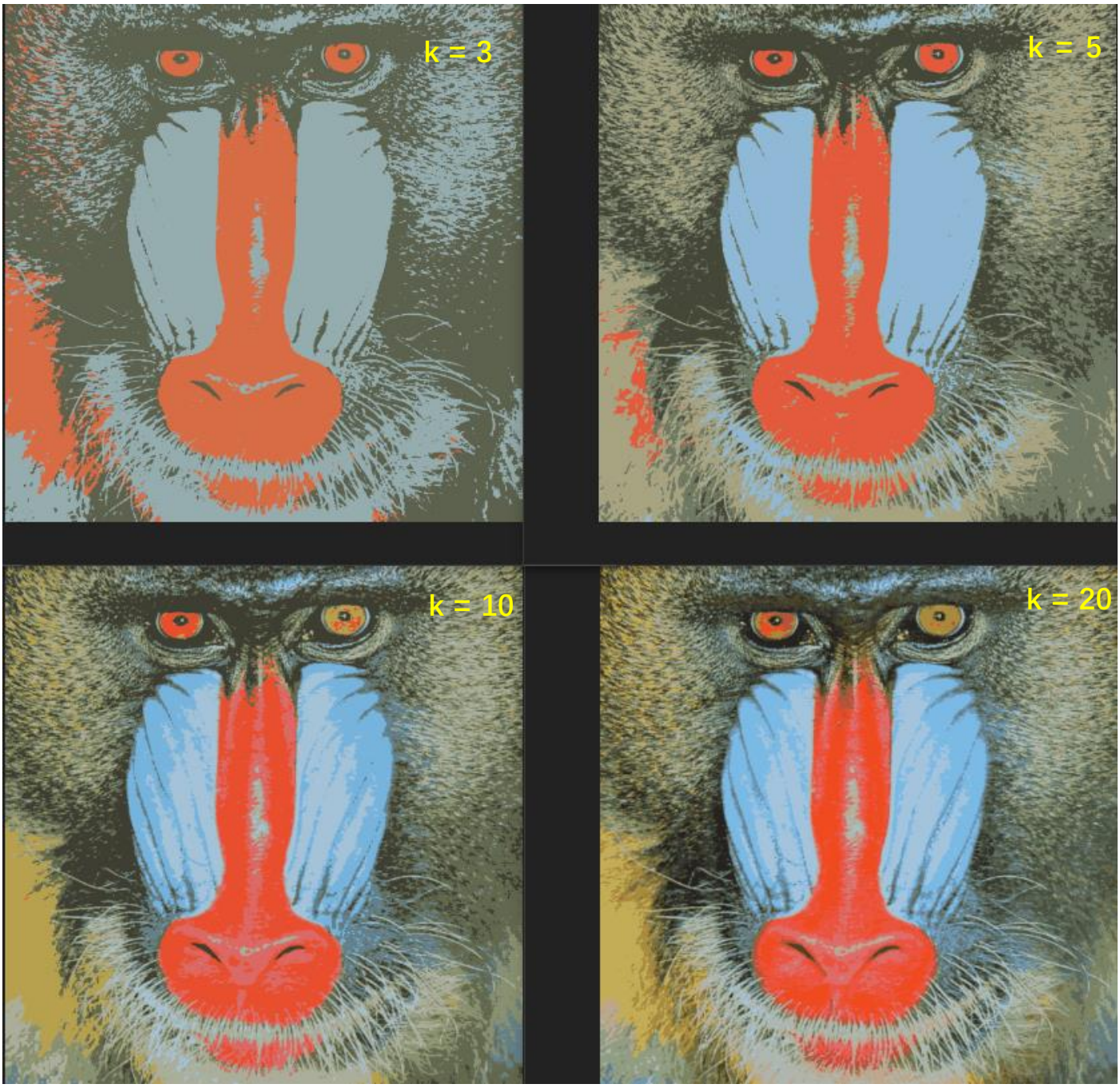
Algorithm:

Color Quantization k-means algorithm function at code line: 147-202

This function logic is the same as question 1-3, the only differences are:

1. Instead of computing the distance between [x, y] in X and [x, y] in centroid, this function will compute the distance between [b, g, r] in each pixel and [b, g, r] in centroid.
2. At the end, this function will reform the pixels/image by replacing pixels with the centroid it belongs to, and then return the reformed image.

Result Pictures:



Sample Result

---------- Color Quantization Start ----------

---------- When k=3 ----------
updated μ: [[69. 109. 216.]
[174. 173. 149.]
[78. 98. 92.]]
---------- Total Running for k=3: 38.912s ----------
/task2_baboon_3.jpg saved at current directory

---------- When k=5 ----------
updated μ: [[60. 90. 227.]

```
 [129. 164. 168.]
 [100. 121. 111.]
 [214. 185. 143.]
 [ 56.  71.  69.]]
---------- Total Running for k=5: 53.094s ----------
/task2_baboon_5.jpg saved at current directory

---------- When k=10 ----------
updated μ:  [[222. 180. 118.]
 [155. 174. 171.]
 [113. 145. 137.]
 [ 50.  62.  62.]
 [102. 105. 221.]
 [ 78. 161. 184.]
 [ 46.  79. 233.]
 [ 79. 105.  99.]
 [138. 128. 100.]
 [217. 195. 164.]]
---------- Total Running for k=10: 98.593s ----------
/task2_baboon_10.jpg saved at current directory

---------- When k=20 ----------
updated μ:  [[ 66.  82.  74.]
 [131. 158. 146.]
 [ 88. 103.  90.]
 [129. 125. 101.]
 [ 87. 174. 195.]
 [228. 186. 122.]
 [143. 167. 189.]
 [107. 108. 222.]
 [ 60.  82. 183.]
 [ 39.  67.  76.]
 [ 64.  84. 236.]
 [180. 151. 109.]
 [174. 182. 175.]
 [221. 196. 164.]
 [ 48.  55.  54.]
 [ 60. 106. 117.]
 [ 31.  34.  38.]
 [ 97. 132. 126.]
 [ 32.  74. 239.]
 [ 62. 139. 168.]]
---------- Total Running for k=20: 186.199s ----------
/task2_baboon_20.jpg saved at current directory
```

----------All Process Finished!! -- Run time: 377.191s ----------