**Title:** Assignment # 3

**Name:** Umm-e-Hani

**Roll No.:** 21I-1715

**Section:** M

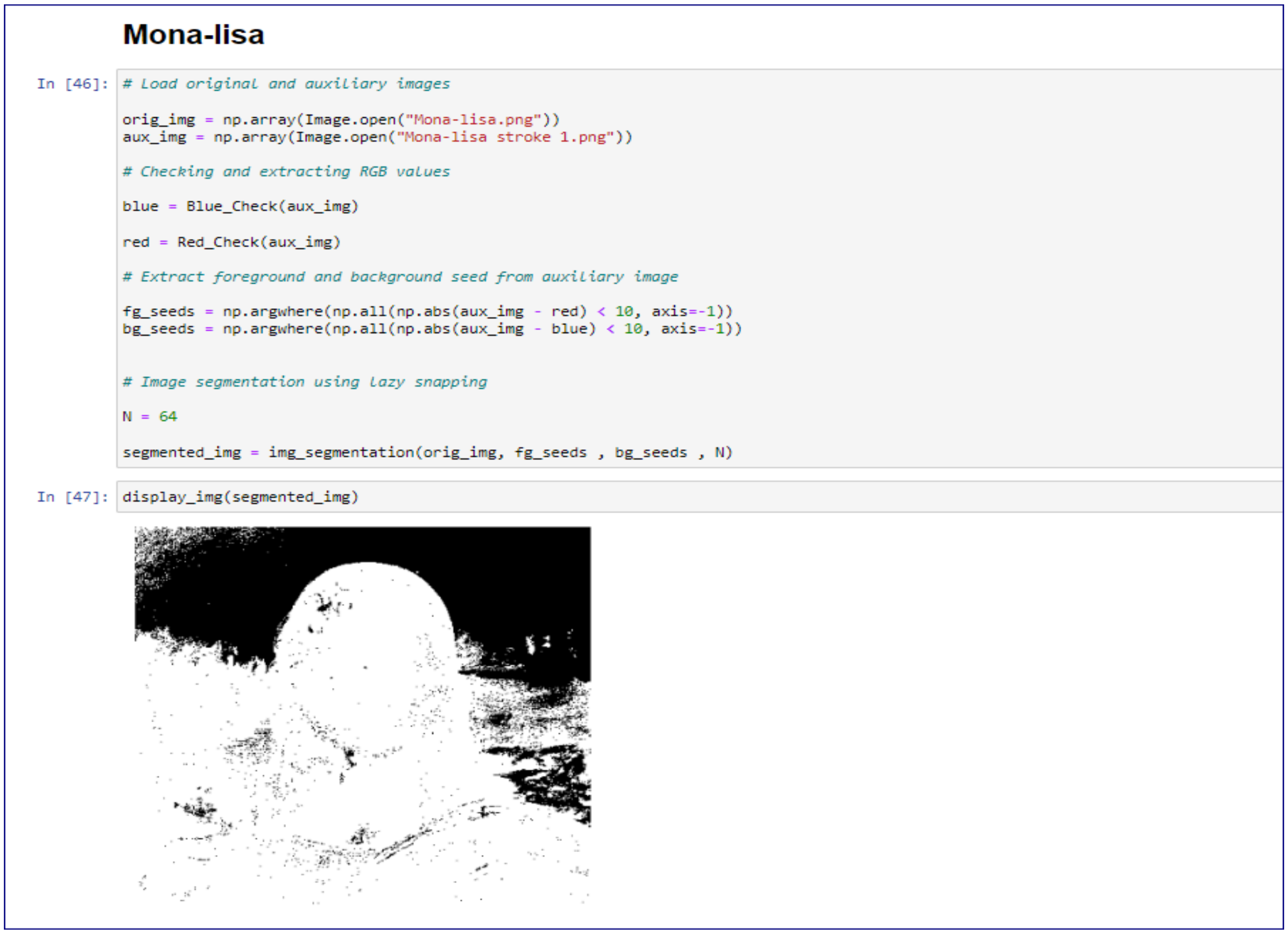
**Submitted to:**
Maam Ayesha Kamran

# REPORT

## Q1. Interactive Foreground Segmentation Using K-Means Clustering

In the given question our task is to implement image segmentation using k-means.The given dataset contains original images and auxiliary images. The auxiliary images have been derived from a technique called lazy snapping. The auxiliary images contain red and blue strokes i.e (foreground and background). The requirement is to classify the given image into two clusters i.e foreground and background to get image segmentation.

The following is the output when N = 64

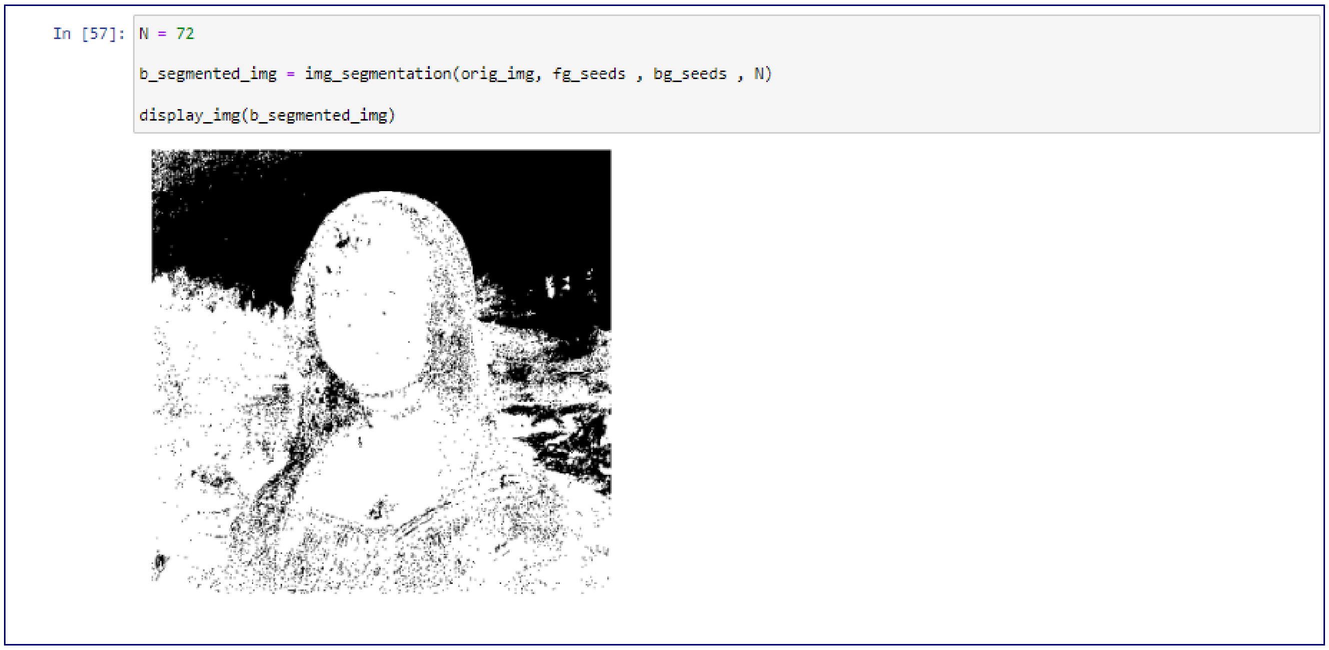


**Mona-lisa**

```
In [46]: # Load original and auxiliary images

         orig_img = np.array(Image.open("Mona-lisa.png"))
         aux_img = np.array(Image.open("Mona-lisa stroke 1.png"))

         # Checking and extracting RGB values

         blue = Blue_Check(aux_img)

         red = Red_Check(aux_img)

         # Extract foreground and background seed from auxiliary image

         fg_seeds = np.argwhere(np.all(np.abs(aux_img - red) < 10, axis=-1))
         bg_seeds = np.argwhere(np.all(np.abs(aux_img - blue) < 10, axis=-1))


         # Image segmentation using lazy snapping

         N = 64

         segmented_img = img_segmentation(orig_img, fg_seeds , bg_seeds , N)
```

```
In [47]: display_img(segmented_img)
```

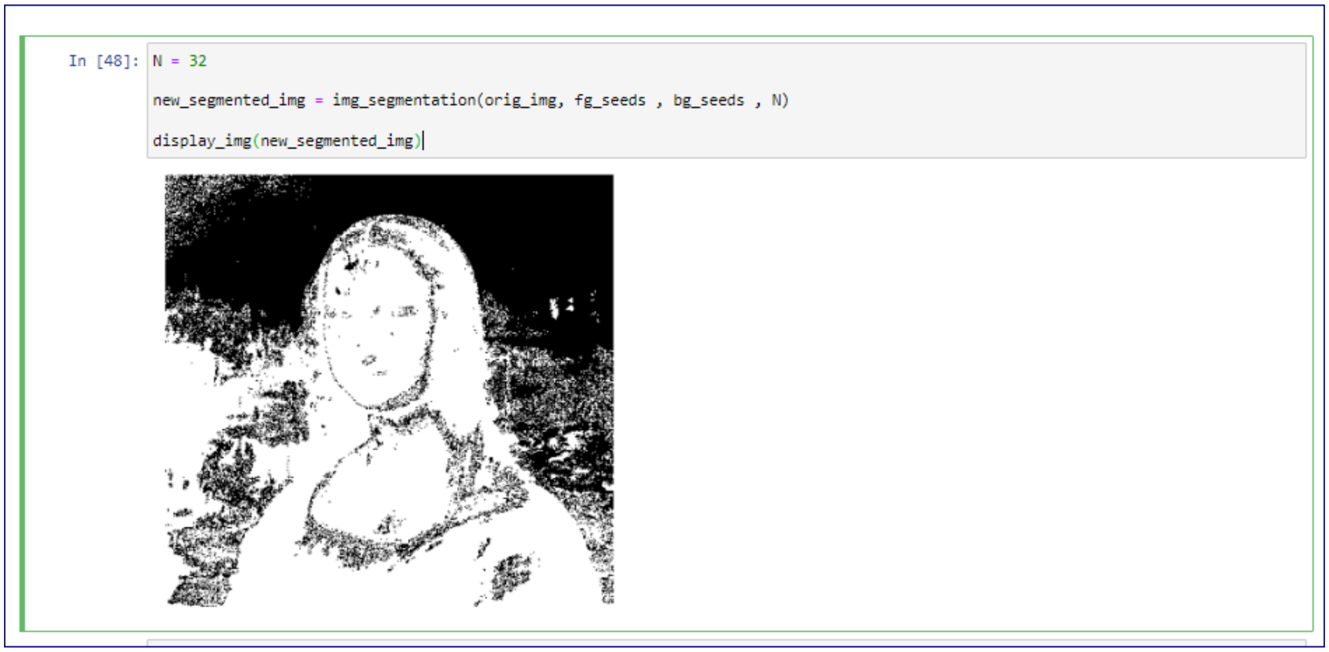
The resultant image isn't very clear due to a large number of clusters. Similarly let's observe another output when we further increase the cluster size N = 72.



```
In [57]: N = 72
         b_segmented_img = img_segmentation(orig_img, fg_seeds , bg_seeds , N)
         display_img(b_segmented_img)
```

The image gets more and more faded as we increase the cluster size. Hence increasing isn't optimal in this case.

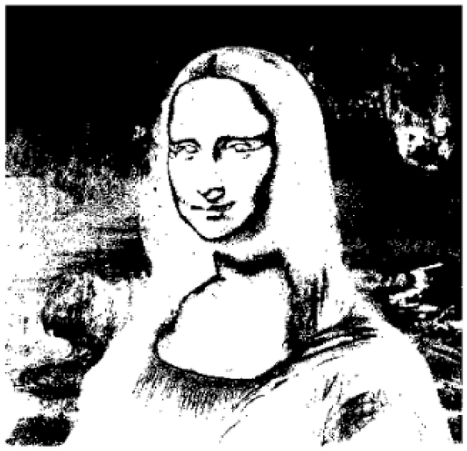
Now let's check for when we decrease the cluster size for N = 32. We observe that the image gets clearer.



```
In [48]: N = 32
         new_segmented_img = img_segmentation(orig_img, fg_seeds , bg_seeds , N)
         display_img(new_segmented_img)
```

We get the best result at N = 16.

```
In [50]: N = 16

         new_segmented_img = img_segmentation(orig_img, fg_seeds , bg_seeds , N)

         display_img(new_segmented_img)
```

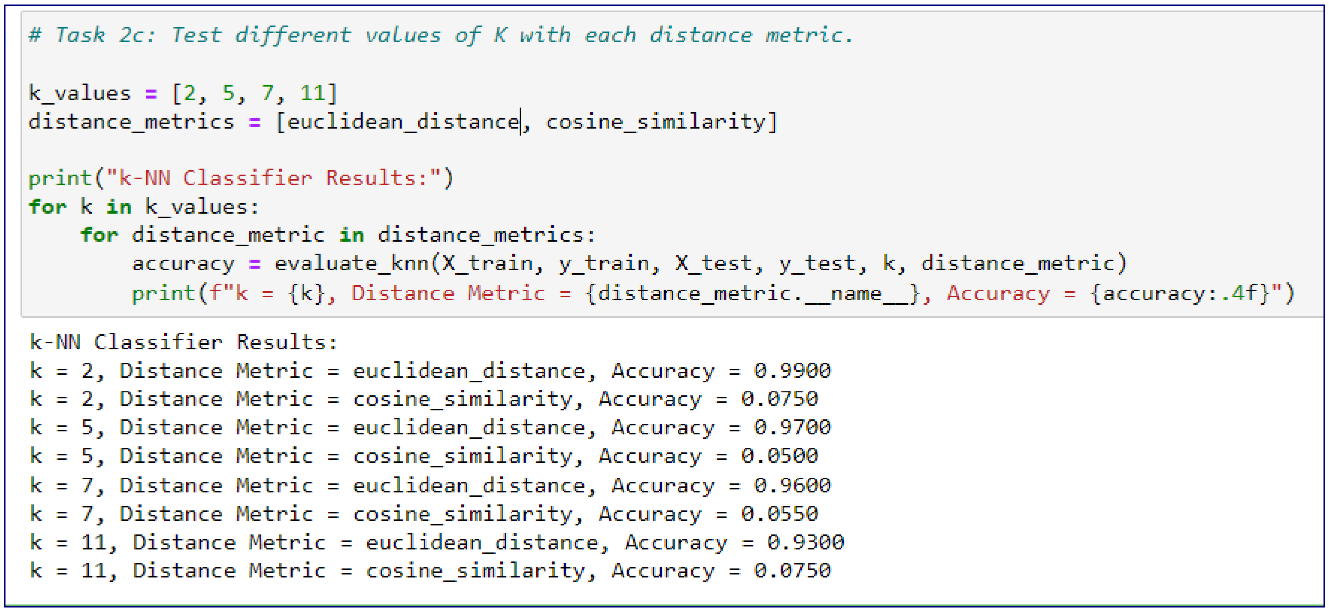


## Q2. Face Recognition Using K-NN

The CMU Pose, Illumination, and Expression (PIE) Dataset is a valuable resource for research in facial recognition and related fields.

**c. Write results in the report for k-values 2, 5, 7, and 11 with each distance metric.**

**Output:**



```
# Task 2c: Test different values of K with each distance metric.

k_values = [2, 5, 7, 11]
distance_metrics = [euclidean_distance, cosine_similarity]

print("k-NN Classifier Results:")
for k in k_values:
    for distance_metric in distance_metrics:
        accuracy = evaluate_knn(X_train, y_train, X_test, y_test, k, distance_metric)
        print(f"k = {k}, Distance Metric = {distance_metric.__name__}, Accuracy = {accuracy:.4f}")
```

```
k-NN Classifier Results:
k = 2, Distance Metric = euclidean_distance, Accuracy = 0.9900
k = 2, Distance Metric = cosine_similarity, Accuracy = 0.0750
k = 5, Distance Metric = euclidean_distance, Accuracy = 0.9700
k = 5, Distance Metric = cosine_similarity, Accuracy = 0.0500
k = 7, Distance Metric = euclidean_distance, Accuracy = 0.9600
k = 7, Distance Metric = cosine_similarity, Accuracy = 0.0550
k = 11, Distance Metric = euclidean_distance, Accuracy = 0.9300
k = 11, Distance Metric = cosine_similarity, Accuracy = 0.0750
```

This output presents the results of running a k-Nearest Neighbors (k-NN) classifier on a dataset, evaluating different values of k and using two different distance metrics (Euclidean distance and cosine similarity).

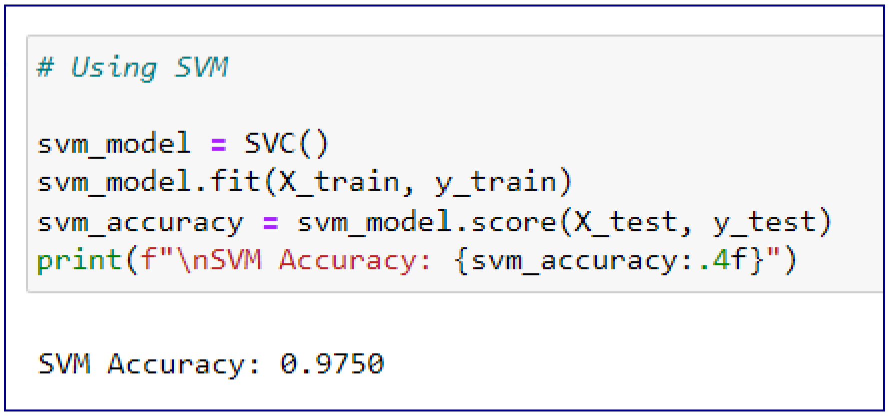
Here's what each line of the output means:

- When using k = 2 and Euclidean distance as the distance metric, the classifier achieved an accuracy of 99.00%. This means that when considering the two nearest neighbors of each test instance, the majority vote of their labels was correct 99.00% of the time.
- When using k = 2 and cosine similarity as the distance metric, the classifier achieved an accuracy of 7.50%. This indicates that cosine similarity performed poorly compared to Euclidean distance for this dataset.
- When using k = 5 and Euclidean distance, the classifier achieved an accuracy of 97.00%.
- When using k = 5 and cosine similarity, the accuracy dropped to 5.00%.
- Accuracy for k = 7 and Euclidean distance is 96.00%.
- Accuracy for k = 7 and cosine similarity is 5.50%.
- Accuracy for k = 11 and Euclidean distance is 93.00%.
- Accuracy for k = 11 and cosine similarity is 7.50%.

From these results, we can observe how different values of k and distance metrics impact the performance of the K-NN classifier. In this case, Euclidean distance generally performs better than cosine similarity, and lower values of k tend to produce higher accuracies.

**Q : Use Sklearn to apply SVM and GaussianNB on this dataset and compare the accuracy with K-NN in the report.**

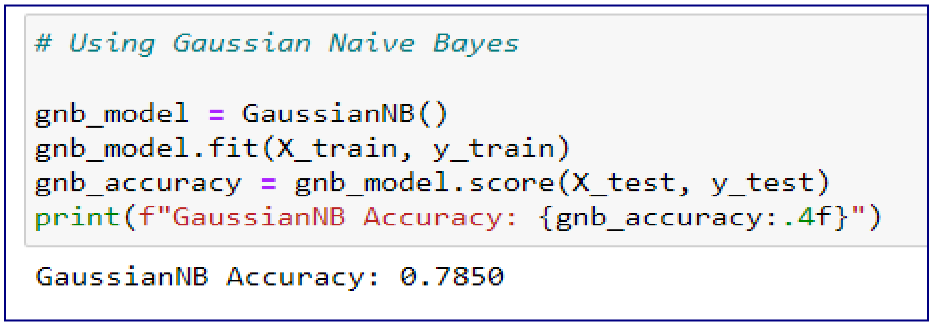
**SVM Output:**



```
# Using SVM

svm_model = SVC()
svm_model.fit(X_train, y_train)
svm_accuracy = svm_model.score(X_test, y_test)
print(f"\nSVM Accuracy: {svm_accuracy:.4f}")
```

```
SVM Accuracy: 0.9750
```

**Gaussian Naive Bayes Output:**



```
# Using Gaussian Naive Bayes

gnb_model = GaussianNB()
gnb_model.fit(X_train, y_train)
gnb_accuracy = gnb_model.score(X_test, y_test)
print(f"GaussianNB Accuracy: {gnb_accuracy:.4f}")
```

```
GaussianNB Accuracy: 0.7850
```

To compare the accuracy of k-NN with SVM and Gaussian Naive Bayes (GaussianNB) on this dataset, we can summarize the results as follows:

**K-NN Accuracy:**

- k-NN achieved accuracies ranging from 93.00% to 99.00%, depending on the choice of k and distance metric.
- The highest accuracy obtained by k-NN was 99.00% when using k = 2 and Euclidean distance.
- Overall, k-NN performed well on the dataset, with accuracies consistently above 90%.

**SVM Accuracy:**

- SVM achieved an accuracy of 97.50% on the dataset.
- SVM outperformed GaussianNB but had slightly lower accuracy compared to the best-performing K-NN configuration (99.00% accuracy).

**GaussianNB Accuracy:**

- GaussianNB achieved an accuracy of 78.50% on the dataset.
- GaussianNB had lower accuracy compared to both K-NN and SVM.

**Comparison:**

- K-NN had the highest accuracy among the three classifiers, achieving 99.00% accuracy with the appropriate distance metric.
- SVM performed slightly worse than the best-performing K-NN configuration but still had high accuracy at 97.50%.
- GaussianNB had the lowest accuracy among the three classifiers, achieving 78.50%.

Overall, K-NN showed superior performance on this dataset compared to SVM and GaussianNB.