YASSIN SHEHAB - 231003610

INTRO TO ARTIFICIAL INTELLIGENCE

12TH WEEK PROJECT

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score

raw_data = pd.read_csv('Q1_data.csv')
feature_matrix = raw_data.drop('class', axis=1).values
label_vector = np.where(raw_data['class'] == 'normal', 0, 1)

print(raw_data.head());
```

```
   duration  src_bclasstes  dst_bclasstes  count  srv_count
serror_rate  \
0         0            520              0    428        428
0.0
1         0              0              0    131         18
0.0
2         0              0              0     20          8
1.0
3         0           1235            404      1          4
0.0
4         0            224           1415      1          1
0.0

   srv_serror_rate  rerror_rate  srv_rerror_rate
same_srv_rate  ...  \
0              0.0          0.0              0.0           1.00  ...

1              0.0          1.0              1.0           0.14  ...

2              1.0          0.0              0.0           0.40  ...

3              0.0          0.0              0.0           1.00  ...

4              0.0          0.0              0.0           1.00  ...


   dst_host_srv_count  dst_host_same_srv_rate  dst_host_diff_srv_rate
\
0                 255                    1.00                    0.00

1                  18                    0.07                    0.06

2                  68                    0.27                    0.02

3                 179                    0.72                    0.12

4                  48                    1.00                    0.00
```

```
   dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
0                         1.00                         0.00
1                         0.00                         0.00
2                         0.01                         0.00
3                         0.04                         0.02
4                         0.02                         0.00

   dst_host_serror_rate  dst_host_srv_serror_rate
dst_host_rerror_rate  \
0                   0.0                      0.00
0.00
1                   0.0                      0.00
1.00
2                   1.0                      1.00
0.00
3                   0.0                      0.01
0.04
4                   0.0                      0.00
0.00

   dst_host_srv_rerror_rate        class
0                       0.0  anomalclass
1                       1.0  anomalclass
2                       0.0  anomalclass
3                       0.0       normal
4                       0.0       normal

[5 rows x 23 columns]
```

```python
print(raw_data.info());
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 23 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   duration                     200 non-null    int64
 1   src_bclasstes                200 non-null    int64
 2   dst_bclasstes                200 non-null    int64
 3   count                        200 non-null    int64
 4   srv_count                    200 non-null    int64
 5   serror_rate                  200 non-null    float64
 6   srv_serror_rate              200 non-null    float64
 7   rerror_rate                  200 non-null    float64
 8   srv_rerror_rate              200 non-null    float64
 9   same_srv_rate                200 non-null    float64
 10  diff_srv_rate                200 non-null    float64
 11  srv_diff_host_rate           200 non-null    float64
```

```
 12  dst_host_count               200 non-null    int64
 13  dst_host_srv_count           200 non-null    int64
 14  dst_host_same_srv_rate       200 non-null    float64
 15  dst_host_diff_srv_rate       200 non-null    float64
 16  dst_host_same_src_port_rate  200 non-null    float64
 17  dst_host_srv_diff_host_rate  200 non-null    float64
 18  dst_host_serror_rate         200 non-null    float64
 19  dst_host_srv_serror_rate     200 non-null    float64
 20  dst_host_rerror_rate         200 non-null    float64
 21  dst_host_srv_rerror_rate     200 non-null    float64
 22  class                        200 non-null    object
dtypes: float64(15), int64(7), object(1)
memory usage: 36.1+ KB
None
```

```
scaler = StandardScaler() # to normalize all the data
feature_matrix = scaler.fit_transform(feature_matrix)
print(feature_matrix); # feature_matrix has the training data without class field from csv
```

```
[[-0.1180733  -0.07777228 -0.2117447  ... -0.67984097 -0.4467788
  -0.43567987]
 [-0.1180733  -0.07810672 -0.2117447  ... -0.67984097  2.41169081
   2.35177931]
 [-0.1180733  -0.07810672 -0.2117447  ...  1.47462258 -0.4467788
  -0.43567987]
 ...
 [-0.1180733  -0.07810672 -0.2117447  ...  1.47462258 -0.4467788
  -0.43567987]
 [-0.1180733  -0.07800703 -0.15243546 ... -0.65829634 -0.4181941
  -0.40780528]
 [-0.1180733  -0.07808743 -0.19969542 ... -0.67984097 -0.4467788
  -0.43567987]]
```

```
# split data
training_features, testing_features, training_labels, testing_labels = train_test_split(
    feature_matrix,
    label_vector,
    test_size=0.3,
    stratify=label_vector,
    random_state=665
    );

print(f"Training Set: Normal Count = {np.sum(training_labels == 0)}")
print(f"Training Set: Anomaly Count = {np.sum(training_labels == 1)}")
print(f"Testing Set:  Normal Count = {np.sum(testing_labels == 0)}")
print(f"Testing Set:  Anomaly Count = {np.sum(testing_labels == 1)}")
```

```
Training Set: Normal Count = 70
Training Set: Anomaly Count = 70
Testing Set:  Normal Count = 30
Testing Set:  Anomaly Count = 30

def activation(val):
    # step function
    if val >= 0: return 1;
    return 0;

def train_perceptron(features, labels, learning_rate=0.1,
max_iterations=1000):
    num_samples, num_features = features.shape

    # init weights and threshold
    weights = np.zeros(num_features)
    bias = 0.0

    for it in range(max_iterations):
        for idx in range(num_samples):
            curr_sample = features[idx];
            true_label = labels[idx];

            # fn = x0 * w0 + x1 * w1 + .... + bias
            fn = np.dot(curr_sample, weights) + bias;
            predicted_val = activation(fn);

            # update weigths
            if true_label != predicted_val:
                err = true_label - predicted_val;

                weights = weights + (learning_rate * err *
curr_sample);
                bias = bias + (learning_rate * err);

    return weights, bias

def predict(features, weights, bias):
    pred_list = [];
    num_samples = len(features);

    for idx in range(num_samples):
        curr_sample = features[idx];

        fn = np.dot(curr_sample, weights) + bias;
        pred_val = activation(fn);

        pred_list.append(pred_val);

    return np.array(pred_list);
```

```
# training
trained_weights, trained_bias = train_perceptron(training_features,
training_labels)

predicted_test_labels = predict(testing_features, trained_weights,
trained_bias)
perceptron_accuracy = accuracy_score(testing_labels,
predicted_test_labels)

print(f"Perceptron Classification Accuracy: {perceptron_accuracy *
100:.2f}%")
```

```
Perceptron Classification Accuracy: 93.33%
```

```
best_k_value = 0
highest_training_accuracy = 0.0

# try different num of clusters
for num_clusters in [2, 3, 4]:

    # Initialize and fit K-Means
    kmeans_algorithm = KMeans(n_clusters=num_clusters,
random_state=665, n_init=10)
    kmeans_algorithm.fit(training_features)

    cluster_assignments = kmeans_algorithm.labels_

    print(f"\nanalysis for K = {num_clusters}:")

    # This map will store which class (0 or 1) each cluster represents
    id_class_map = {}

    for cluster_id in range(num_clusters):
        # get all points that belong to this cluster
        indices_in_cluster = np.where(cluster_assignments ==
cluster_id)
        true_labels_in_cluster = training_labels[indices_in_cluster]

        # calculate the num of elements in cluster
        normal_count = np.sum(true_labels_in_cluster == 0)
        anomaly_count = np.sum(true_labels_in_cluster == 1)
        total_count_in_cluster = normal_count + anomaly_count

        # make sure the cluster isn't empty
        if total_count_in_cluster == 0:
            continue;

        percent_normal = (normal_count / total_count_in_cluster) *
100;
        percent_anomaly = (anomaly_count / total_count_in_cluster) *
```

```
100;

        print(f"  cluster id {cluster_id}: normal={percent_normal:.1f}
%, anomaly={percent_anomaly:.1f}%")

        # give the clutter the label of the majority
        if normal_count > anomaly_count:
            id_class_map[cluster_id] = 0 # normal
        else:
            id_class_map[cluster_id] = 1 # anomaly

    # calc accuracy
    predicted_labels_from_clustering = []
    for assigned_cluster in cluster_assignments:
        predicted_label = id_class_map[assigned_cluster]
        predicted_labels_from_clustering.append(predicted_label)

    current_accuracy = accuracy_score(training_labels,
predicted_labels_from_clustering)
    print(f"  classification accuracy (on train set):
{current_accuracy * 100:.2f}%");

    # Keep track of the best K
    if current_accuracy > highest_training_accuracy:
        highest_training_accuracy = current_accuracy;
        best_k_value = num_clusters;

print(f"\noptimal number of clusters (K): {best_k_value}");
```

```
analysis for K = 2:
  cluster id 0: normal=0.0%, anomaly=100.0%
  cluster id 1: normal=71.4%, anomaly=28.6%
  classification accuracy (on train set): 80.00%

analysis for K = 3:
  cluster id 0: normal=25.0%, anomaly=75.0%
  cluster id 1: normal=0.0%, anomaly=100.0%
  cluster id 2: normal=86.5%, anomaly=13.5%
  classification accuracy (on train set): 88.57%

analysis for K = 4:
  cluster id 0: normal=25.0%, anomaly=75.0%
  cluster id 1: normal=0.0%, anomaly=100.0%
  cluster id 2: normal=86.3%, anomaly=13.7%
  cluster id 3: normal=100.0%, anomaly=0.0%
  classification accuracy (on train set): 88.57%

optimal number of clusters (K): 3
```

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

customer_data = pd.read_csv('mall_customer.csv')
print(customer_data.info());
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   CustomerID           500 non-null    object
 1   Name                 500 non-null    object
 2   Age                  500 non-null    int64
 3   Gender               500 non-null    object
 4   MembershipLevel      500 non-null    object
 5   IncomeLevel          500 non-null    float64
 6   ElectronicsSpending  500 non-null    float64
 7   ClothingSpending     500 non-null    float64
 8   GrocerySpending      500 non-null    float64
 9   HomeSpending         500 non-null    float64
 10  Visits               500 non-null    int64
 11  PurchaseFrequency    500 non-null    int64
 12  OnlineActivity       500 non-null    float64
 13  EmailOpens           500 non-null    float64
 14  AppUsage             500 non-null    float64
 15  LoyaltyPoints        500 non-null    float64
dtypes: float64(9), int64(3), object(4)
memory usage: 62.6+ KB
None
```

```python
numeric_feature_names = [
    'Age', 'IncomeLevel', 'ElectronicsSpending', 'ClothingSpending',
    'GrocerySpending', 'HomeSpending', 'Visits', 'PurchaseFrequency',
    'OnlineActivity', 'EmailOpens', 'AppUsage', 'LoyaltyPoints'
]
categorical_feature_names = ['Gender', 'MembershipLevel']

# normalize data
scaler = StandardScaler()
numeric_data_scaled = pd.DataFrame(
    scaler.fit_transform(customer_data[numeric_feature_names]),
    columns=numeric_feature_names
)
print(numeric_data_scaled);
```

```
          Age  IncomeLevel  ElectronicsSpending  ClothingSpending  \
0   -2.022622    -1.076784            -1.561765         -1.303146
```

```
1   -2.217499    -0.549924            -1.457846         -1.254796
2   -2.022622    -1.178372            -1.237088         -0.945250
3   -2.412375    -1.252029            -1.413130         -1.262717
4   -1.730308    -0.846171            -1.668460         -1.609342
..        ...          ...                  ...               ...
495  1.290277     1.326355             1.042406          1.756637
496  1.095400     1.262265             1.077671          1.980948
497  1.290277     0.930803             1.400057          1.462284
498  0.900524     1.545417             1.630423          1.561354
499  0.900524     1.239021             1.374612          1.505411

     GrocerySpending  HomeSpending    Visits  PurchaseFrequency  \
0          -0.924782     -1.478844  0.901474           0.302569
1          -0.802603     -1.258900 -0.182897          -1.416574
2          -1.324214     -1.399553 -0.544354          -1.416574
3          -0.584143     -1.884581  0.178560          -0.843527
4          -2.215617     -1.884581  0.178560          -1.416574
..               ...           ...       ...                ...
495        -1.761730      0.188825  0.178560           0.875617
496        -1.194257      1.297081  0.178560          -0.843527
497        -0.198839     -0.014037  0.178560           0.302569
498         0.402067      0.473768 -0.544354           0.875617
499         0.111258      0.274206  0.178560          -0.270479

     OnlineActivity  EmailOpens  AppUsage  LoyaltyPoints
0         -1.622613    0.713398 -0.528785      -1.207362
1         -0.531652    0.760257 -0.479183      -1.180360
2         -0.764920    1.418271 -0.271860      -1.461425
3         -0.368898    0.756074  0.631222      -0.952202
4         -0.367671    1.249417 -0.321725      -1.050133
..              ...         ...       ...            ...
495        1.013074    1.636709  1.074382       1.758903
496        1.910496    1.113341  1.217708       1.834472
497        2.033284    1.493938  0.944123       1.682072
498        1.508789    0.715367  1.077459       1.644895
499        0.841951    0.533986  0.415132       1.468352

[500 rows x 12 columns]
```

```python
# encode categorical data (0s or 1s for categories, etc...)
categorical_data_encoded = pd.get_dummies(
    customer_data[categorical_feature_names],
    drop_first=False
);
print(categorical_data_encoded);
```

```
     Gender_Female  Gender_Male  MembershipLevel_Bronze
MembershipLevel_Gold  \
0            False         True                    True
False
```

```
1               True        False                        True
False
2               True        False                       False
False
3               True        False                       False
False
4              False         True                       False
False
..               ...          ...                         ...
...
495            False         True                        True
False
496             True        False                       False
True
497            False         True                        True
False
498             True        False                        True
False
499             True        False                        True
False

     MembershipLevel_Silver
0                     False
1                     False
2                      True
3                      True
4                      True
..                      ...
495                   False
496                   False
497                   False
498                   False
499                   False

[500 rows x 5 columns]

# combine numerical and categorical data
clustering_features = pd.concat([numeric_data_scaled,
categorical_data_encoded], axis=1)
print(clustering_features);

          Age  IncomeLevel  ElectronicsSpending  ClothingSpending  \
0   -2.022622    -1.076784            -1.561765         -1.303146
1   -2.217499    -0.549924            -1.457846         -1.254796
2   -2.022622    -1.178372            -1.237088         -0.945250
3   -2.412375    -1.252029            -1.413130         -1.262717
4   -1.730308    -0.846171            -1.668460         -1.609342
..        ...          ...                  ...               ...
495  1.290277     1.326355             1.042406          1.756637
496  1.095400     1.262265             1.077671          1.980948
```

```
497  1.290277     0.930803             1.400057         1.462284
498  0.900524     1.545417             1.630423         1.561354
499  0.900524     1.239021             1.374612         1.505411

     GrocerySpending  HomeSpending    Visits  PurchaseFrequency  \
0          -0.924782     -1.478844  0.901474           0.302569
1          -0.802603     -1.258900 -0.182897          -1.416574
2          -1.324214     -1.399553 -0.544354          -1.416574
3          -0.584143     -1.884581  0.178560          -0.843527
4          -2.215617     -1.884581  0.178560          -1.416574
..               ...           ...       ...                ...
495        -1.761730      0.188825  0.178560           0.875617
496        -1.194257      1.297081  0.178560          -0.843527
497        -0.198839     -0.014037  0.178560           0.302569
498         0.402067      0.473768 -0.544354           0.875617
499         0.111258      0.274206  0.178560          -0.270479

     OnlineActivity  EmailOpens  AppUsage  LoyaltyPoints
Gender_Female  \
0         -1.622613    0.713398 -0.528785      -1.207362
False
1         -0.531652    0.760257 -0.479183      -1.180360
True
2         -0.764920    1.418271 -0.271860      -1.461425
True
3         -0.368898    0.756074  0.631222      -0.952202
True
4         -0.367671    1.249417 -0.321725      -1.050133
False
..              ...         ...       ...            ...             ..
.
495        1.013074    1.636709  1.074382       1.758903
False
496        1.910496    1.113341  1.217708       1.834472
True
497        2.033284    1.493938  0.944123       1.682072
False
498        1.508789    0.715367  1.077459       1.644895
True
499        0.841951    0.533986  0.415132       1.468352
True

     Gender_Male  MembershipLevel_Bronze  MembershipLevel_Gold  \
0           True                    True                 False
1          False                    True                 False
2          False                   False                 False
3          False                   False                 False
4           True                   False                 False
..           ...                     ...                   ...
495         True                    True                 False
```

```
496        False                       False                    True
497         True                        True                   False
498        False                        True                   False
499        False                        True                   False

     MembershipLevel_Silver
0                     False
1                     False
2                      True
3                      True
4                      True
..                      ...
495                   False
496                   False
497                   False
498                   False
499                   False

[500 rows x 17 columns]

k_vals = [2, 3, 4, 5];

# kmeans
for k in k_vals:

    print("-----------------------------");
    print(f"analysis for K = {k} clusters");

    kmeans_model = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans_model.fit(clustering_features)

    cluster_labels = kmeans_model.labels_

    analysis_df = customer_data.copy()
    analysis_df['Cluster_ID'] = cluster_labels

    for cluster_id in range(k):
        # filter data for the current cluster
        cluster_segment = analysis_df[analysis_df['Cluster_ID'] ==
cluster_id]

        customer_count = len(cluster_segment)

        # calc key metrics
        avg_income = cluster_segment['IncomeLevel'].mean()
        avg_loyalty = cluster_segment['LoyaltyPoints'].mean()

        # calc spending habits
        avg_electronics =
cluster_segment['ElectronicsSpending'].mean()
```

```
        avg_clothing = cluster_segment['ClothingSpending'].mean()
        avg_grocery = cluster_segment['GrocerySpending'].mean()
        avg_home = cluster_segment['HomeSpending'].mean()

        print(f"\n[ Cluster {cluster_id} ] - {customer_count}
Customers")
        print(f"  -> avg income:         ${avg_income:,.2f}")
        print(f"  -> avg loyalty points: {avg_loyalty:.2f}")
        print(f"  -> avg spending:")
        print(f"      - electronics: ${avg_electronics:,.2f}")
        print(f"      - clothing:    ${avg_clothing:,.2f}")
        print(f"      - grocery:     ${avg_grocery:,.2f}")
        print(f"      - home:        ${avg_home:,.2f}")

----------------------------
analysis for K = 2 clusters

[ Cluster 0 ] - 200 Customers
  -> avg income:         $44,598.70
  -> avg loyalty points: 161.79
  -> avg spending:
      - electronics: $728.51
      - clothing:    $546.14
      - grocery:     $236.86
      - home:        $291.79

[ Cluster 1 ] - 300 Customers
  -> avg income:         $75,503.31
  -> avg loyalty points: 404.43
  -> avg spending:
      - electronics: $1,364.83
      - clothing:    $866.32
      - grocery:     $397.96
      - home:        $825.79
----------------------------
analysis for K = 3 clusters

[ Cluster 0 ] - 100 Customers
  -> avg income:         $82,392.57
  -> avg loyalty points: 539.76
  -> avg spending:
      - electronics: $1,617.17
      - clothing:    $1,320.66
      - grocery:     $251.12
      - home:        $824.88

[ Cluster 1 ] - 200 Customers
  -> avg income:         $44,598.70
  -> avg loyalty points: 161.79
  -> avg spending:
```

```
      - electronics: $728.51
      - clothing:    $546.14
      - grocery:     $236.86
      - home:        $291.79

[ Cluster 2 ] - 200 Customers
  -> avg income:         $72,058.68
  -> avg loyalty points: 336.77
  -> avg spending:
      - electronics: $1,238.66
      - clothing:    $639.15
      - grocery:     $471.37
      - home:        $826.24
----------------------------
analysis for K = 4 clusters

[ Cluster 0 ] - 100 Customers
  -> avg income:         $43,704.85
  -> avg loyalty points: 181.65
  -> avg spending:
      - electronics: $969.13
      - clothing:    $911.21
      - grocery:     $275.09
      - home:        $448.76

[ Cluster 1 ] - 200 Customers
  -> avg income:         $72,058.68
  -> avg loyalty points: 336.77
  -> avg spending:
      - electronics: $1,238.66
      - clothing:    $639.15
      - grocery:     $471.37
      - home:        $826.24

[ Cluster 2 ] - 100 Customers
  -> avg income:         $45,492.55
  -> avg loyalty points: 141.93
  -> avg spending:
      - electronics: $487.90
      - clothing:    $181.07
      - grocery:     $198.62
      - home:        $134.81

[ Cluster 3 ] - 100 Customers
  -> avg income:         $82,392.57
  -> avg loyalty points: 539.76
  -> avg spending:
      - electronics: $1,617.17
      - clothing:    $1,320.66
      - grocery:     $251.12
```

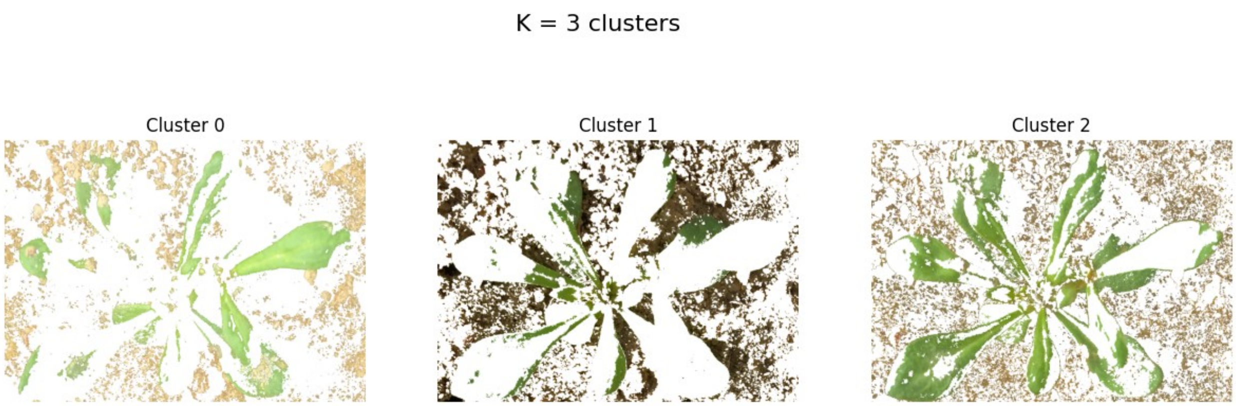
```
      - home:        $824.88
----------------------------
analysis for K = 5 clusters

[ Cluster 0 ] - 100 Customers
  -> avg income:         $43,704.85
  -> avg loyalty points: 181.65
  -> avg spending:
      - electronics: $969.13
      - clothing:    $911.21
      - grocery:     $275.09
      - home:        $448.76

[ Cluster 1 ] - 100 Customers
  -> avg income:         $60,324.04
  -> avg loyalty points: 368.41
  -> avg spending:
      - electronics: $898.78
      - clothing:    $768.02
      - grocery:     $491.03
      - home:        $619.98

[ Cluster 2 ] - 100 Customers
  -> avg income:         $45,492.55
  -> avg loyalty points: 141.93
  -> avg spending:
      - electronics: $487.90
      - clothing:    $181.07
      - grocery:     $198.62
      - home:        $134.81

[ Cluster 3 ] - 100 Customers
  -> avg income:         $82,392.57
  -> avg loyalty points: 539.76
  -> avg spending:
      - electronics: $1,617.17
      - clothing:    $1,320.66
      - grocery:     $251.12
      - home:        $824.88

[ Cluster 4 ] - 100 Customers
  -> avg income:         $83,793.32
  -> avg loyalty points: 305.13
  -> avg spending:
      - electronics: $1,578.53
      - clothing:    $510.28
      - grocery:     $451.72
      - home:        $1,032.49
```

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans

imgfiles = [
    "Q3_data/image_001.png",
    "Q3_data/image_002.png",
    "Q3_data/image_003.png",
    "Q3_data/image_004.png",
    "Q3_data/image_005.png",
    "Q3_data/image_006.png"
    ]

def process_and_segment_image(file_path):
    print(f"\nProcessing Image: {file_path}")

    # load the Image
    try:
        original_image = plt.imread(file_path)
    except FileNotFoundError:
        print(f"Error: File {file_path} not found. Please upload it or 
check the name.")
        return

    # check image color data type
    if original_image.dtype == np.uint8:
        max_color_value = 255
    else:
        max_color_value = 1.0

    image_height, image_width, color_channels = original_image.shape

    # for k means we need it to be a 2d array (matrix)
    pixel_data_matrix = original_image.reshape(-1, 3)

    # normal ahh k means again
    k_values_to_test = [2, 3, 4, 5]

    for k in k_values_to_test:
        print(f"\n--- segmentation with K = {k} ---")

        kmeans_algorithm = KMeans(n_clusters=k, random_state=42, 
n_init=10)
        kmeans_algorithm.fit(pixel_data_matrix)

        # get the cluster ID (0 to k-1) for every single pixel
        pixel_cluster_ids = kmeans_algorithm.labels_

        # show the img
        plt.figure(figsize=(15, 5))
```

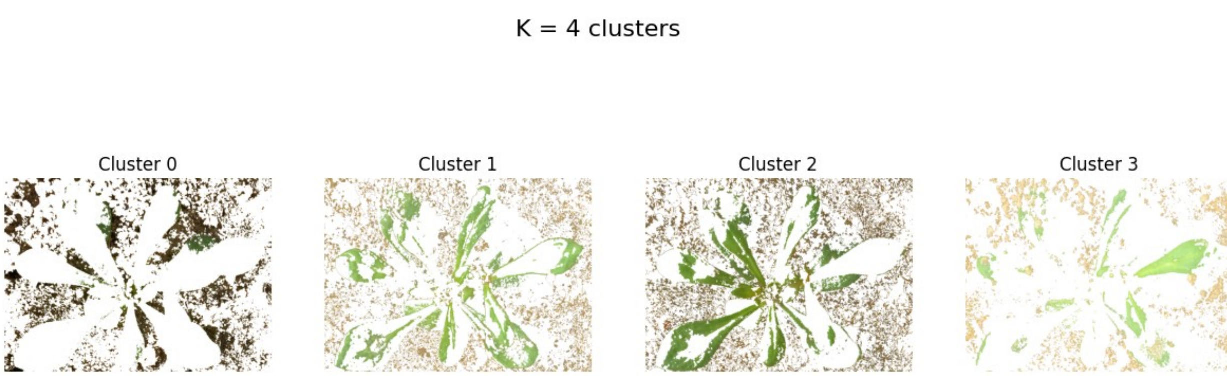
```
        plt.suptitle(f"K = {k} clusters", fontsize=16)

        for cluster_id in range(k):
            # blank white image
            segmented_image_flat = np.full_like(pixel_data_matrix,
max_color_value)

            # masking pixels which don't belong to this clutter
            indices_in_current_cluster = (pixel_cluster_ids ==
cluster_id)

            # get colors from image, but only in the mask (photoshop
type stuff)
            segmented_image_flat[indices_in_current_cluster] =
pixel_data_matrix[indices_in_current_cluster]

            # get back normal image format (Height x Width x 3)
            final_segmented_image =
segmented_image_flat.reshape(image_height, image_width,
color_channels)

            # show the img
            plt.subplot(1, k, cluster_id + 1)
            plt.imshow(final_segmented_image)
            plt.title(f"Cluster {cluster_id}")
            plt.axis('off') # Hide axis numbers

        plt.show()

for imgfile in imgfiles:
    process_and_segment_image(imgfile)
```

```
Processing Image: plant_dataset/image_001.png

--- segmentation with K = 2 ---
```


```
--- segmentation with K = 3 ---
```



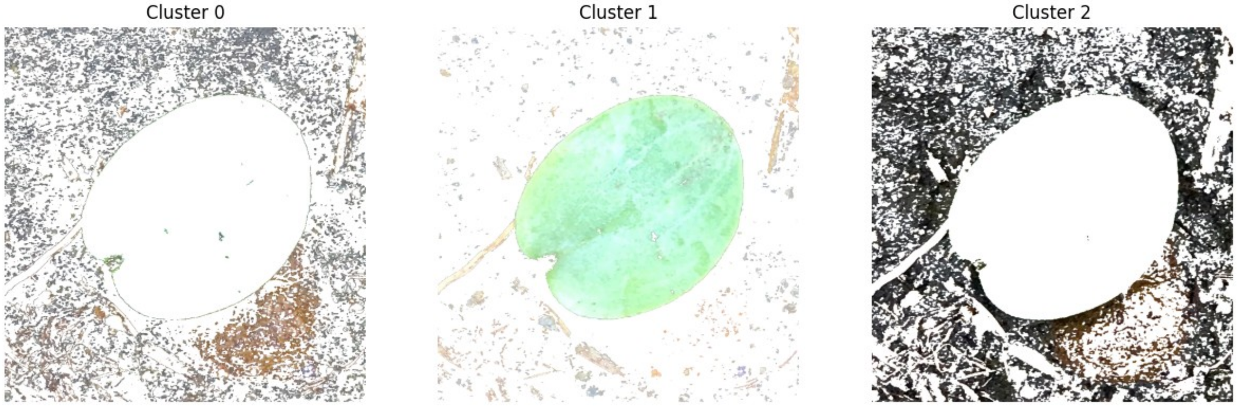

```
--- segmentation with K = 4 ---
```




```
--- segmentation with K = 5 ---
```

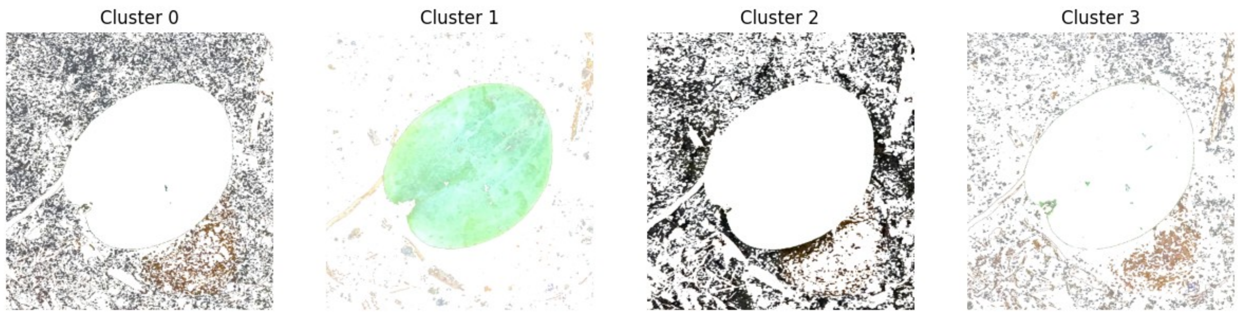

```
Processing Image: plant_dataset/image_002.png

--- segmentation with K = 2 ---
```


```
--- segmentation with K = 3 ---
```



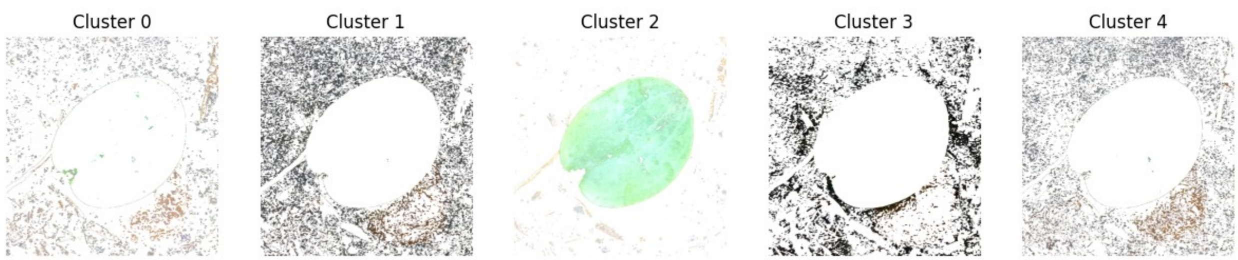
```
--- segmentation with K = 4 ---
```

K = 4 clusters

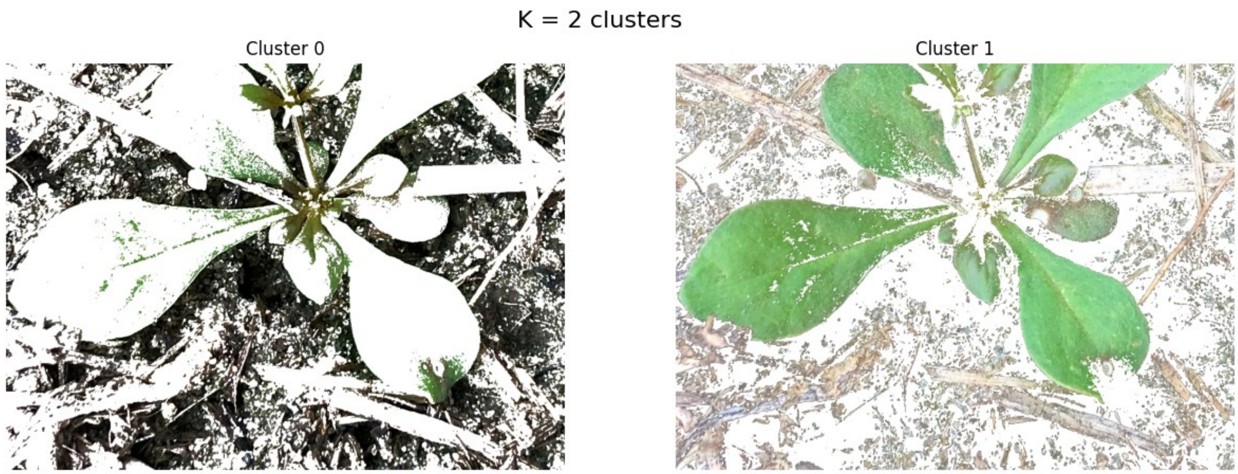



```
--- segmentation with K = 5 ---
```

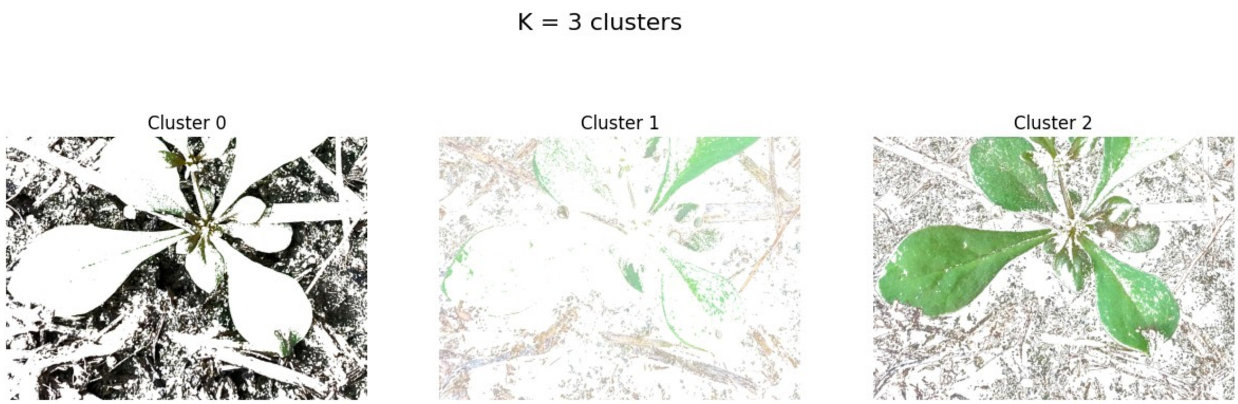
K = 5 clusters



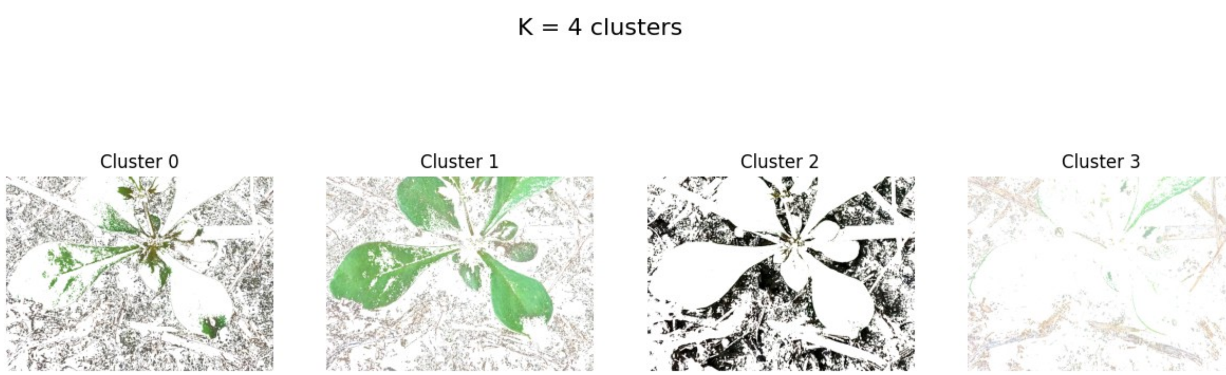

```
Processing Image: plant_dataset/image_003.png

--- segmentation with K = 2 ---
```




```
--- segmentation with K = 3 ---
```

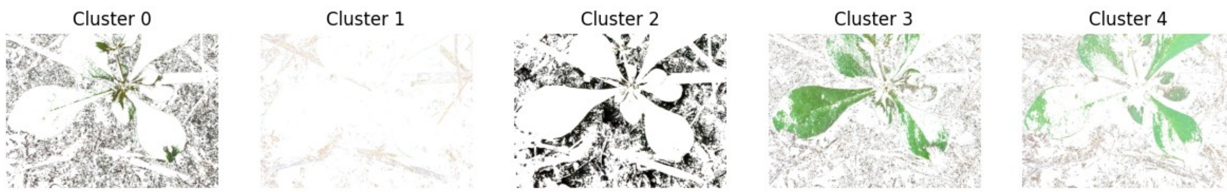



```
--- segmentation with K = 4 ---
```




```
--- segmentation with K = 5 ---
```

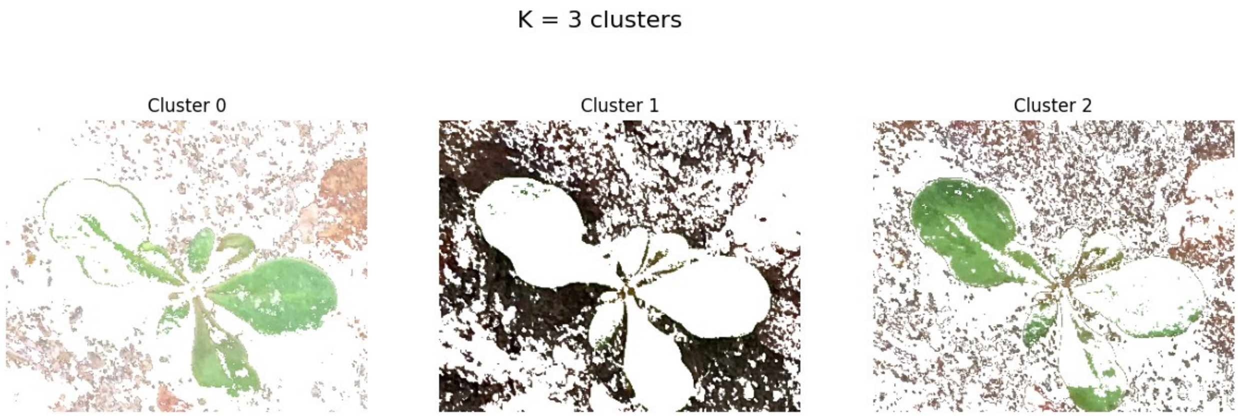
K = 5 clusters



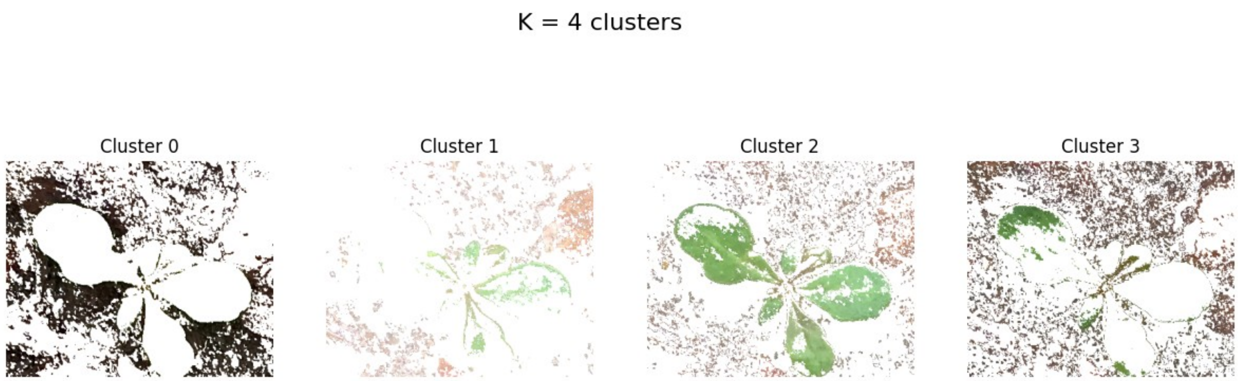

```
Processing Image: plant_dataset/image_004.png
Error: File plant_dataset/image_004.png not found. Please upload it or
check the name.

Processing Image: plant_dataset/image_005.png

--- segmentation with K = 2 ---
```

K = 2 clusters

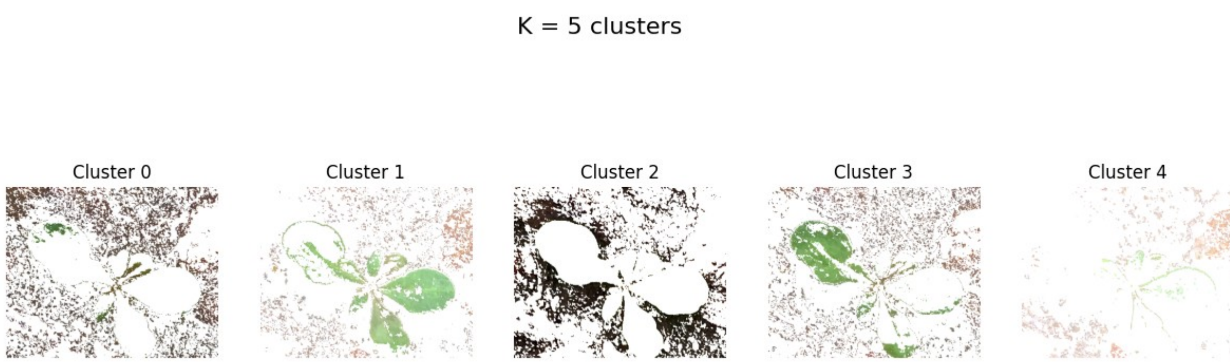

```
--- segmentation with K = 3 ---
```



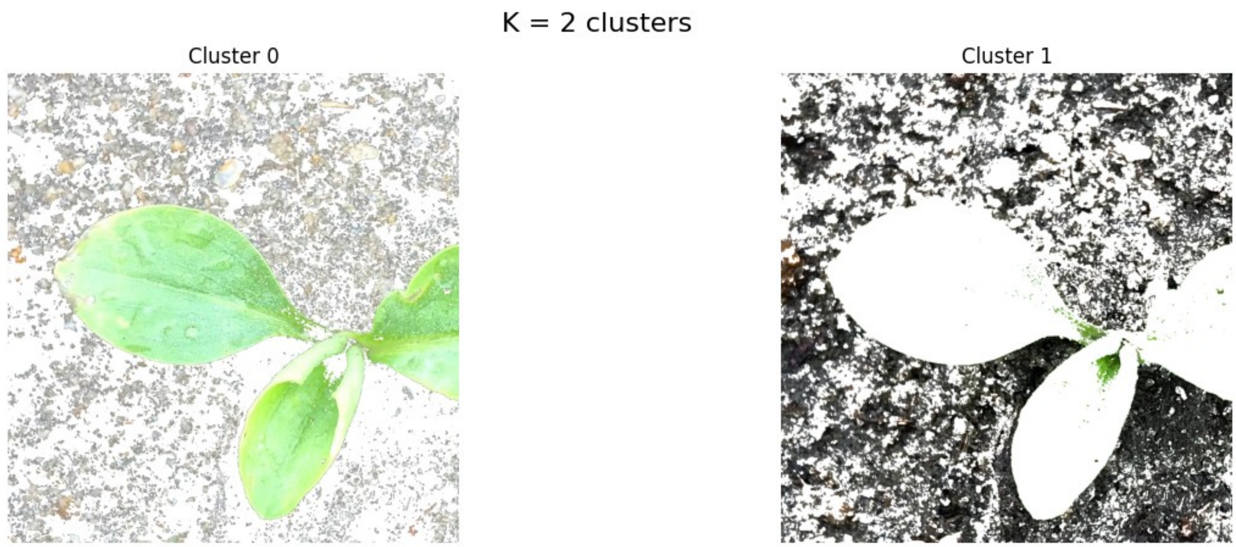

```
--- segmentation with K = 4 ---
```



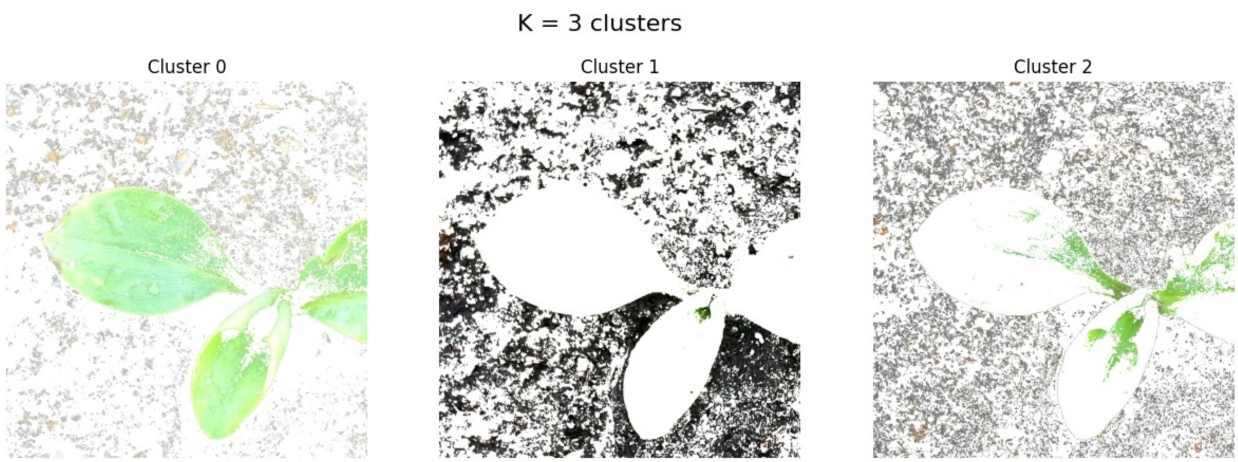

```
--- segmentation with K = 5 ---
```



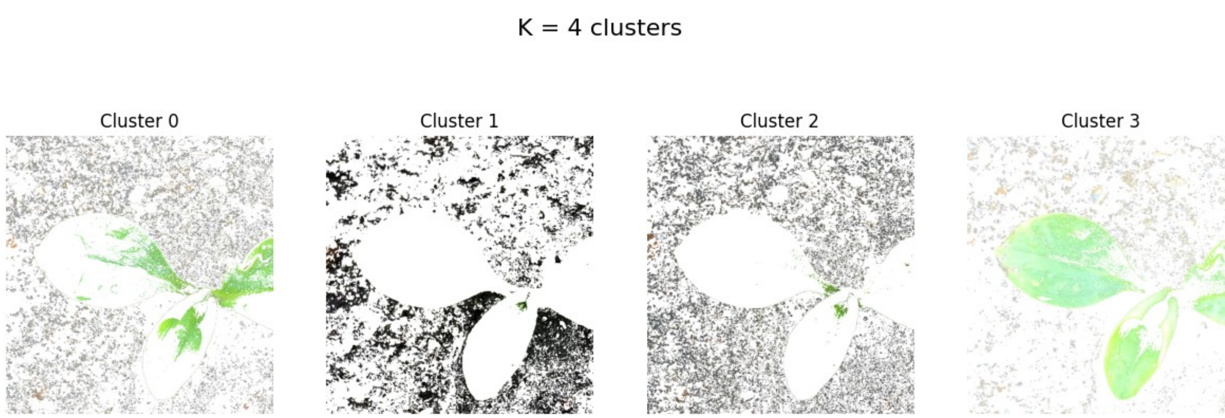

```
Processing Image: plant_dataset/image_006.png

--- segmentation with K = 2 ---
```




```
--- segmentation with K = 3 ---
```




```
--- segmentation with K = 4 ---
```



```
--- segmentation with K = 5 ---
```