Clustering Project

Resubmission by: Koto Andrew Omiloli – 15/4/2024
Feedback on my previous submission is appreciated.

1. Objective

This project focuses on the clustering of flowers corresponding to each different species they belong to for an Agro company that sells flowers to customers. From history, the company has recorded high sales from the hibiscus species of flowers hence they require an algorithm to detect and group each plant they purchase from farmers, according to the species they belong to, to decide on the mix of species of flowers they will require for the target market. Given this clustering, the benefit to the company will be that they will know the type and quantity of species they need to boost market sales.

2. Description of the data set

The data set consists of an image of an area in Wakulla State Forest comprising of different flower species see:
https://www.realsimple.com/thmb/6mCmm8k9kdZqK18VoPoIYdBh7I=/750x0/filters:no_upscale():max_bytes(150000):strip_icc():format(webp)/impossible-to-kill-outdoor-plants-1-2000
f513b0574cb04674a1bce40b832b28dd.jpg.

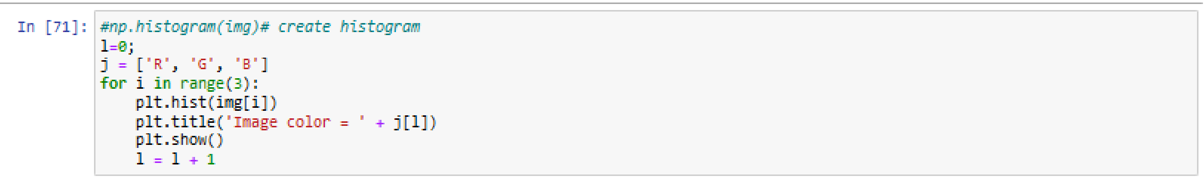
The image consists of 563 x 750 number of pixels with each pixel color corresponding to a combination of RGB (3 channels).

3. Data feature engineering (Data Exploration)

The following steps were carried out for the data exploration, cleaning and feature engineering

Step 1: Data Exploration

To understand the underlying features in the image data, visualization of the image was carried out using histogram plots. As can be seen in Figure 1, with the associated Jupiter notebook code. The data is skewed so it requires data transformation to make it balanced.



```
In [71]: #np.histogram(img)# create histogram
         l=0;
         j = ['R', 'G', 'B']
         for i in range(3):
             plt.hist(img[i])
             plt.title('Image color = ' + j[l])
             plt.show()
             l = l + 1
```

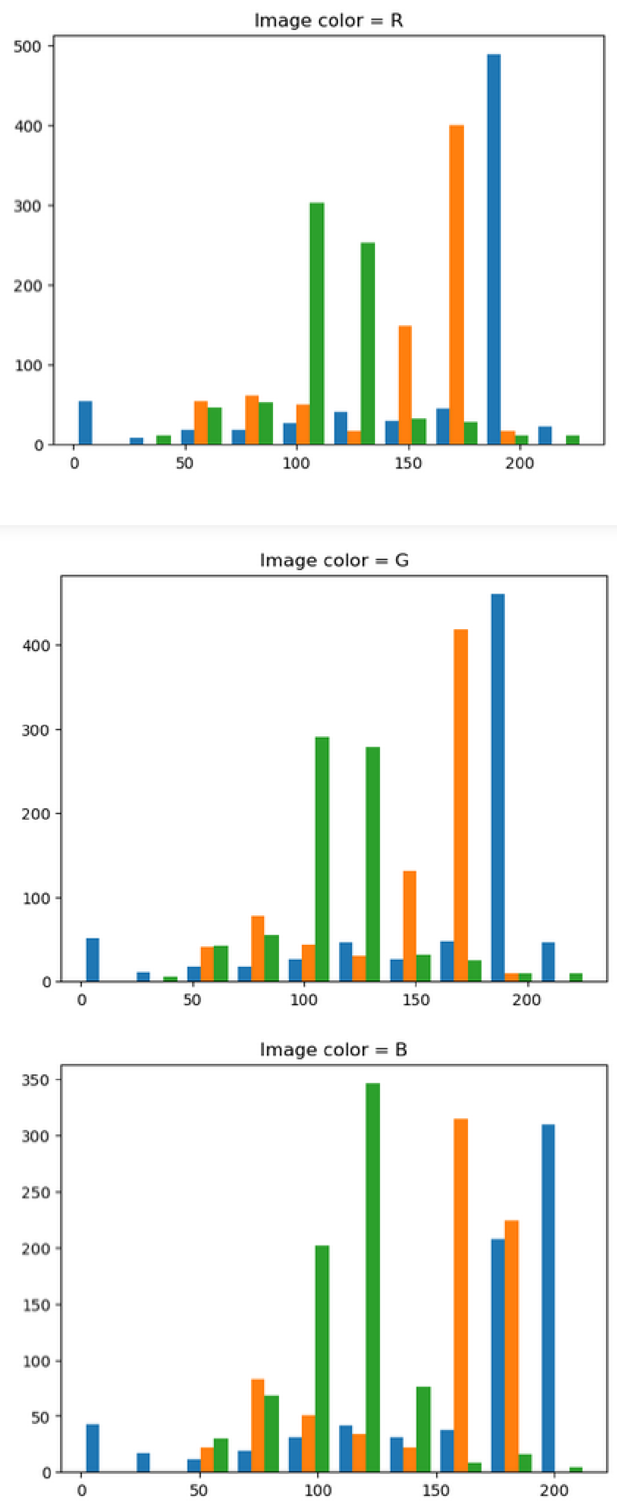
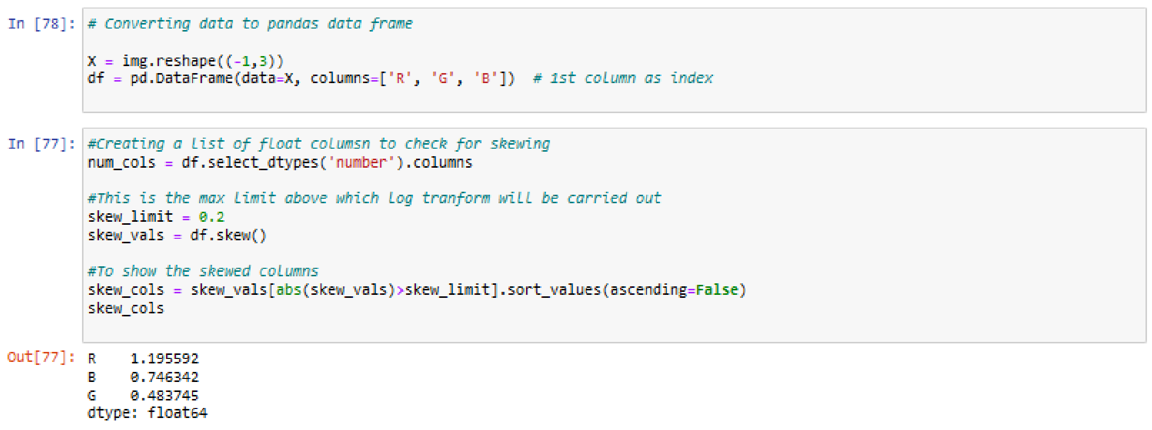


Figure 1: Image Visualization

Step 2: Checking for skewed data and applying a log transformation



```
In [78]: # Converting data to pandas data frame

         X = img.reshape((-1,3))
         df = pd.DataFrame(data=X, columns=['R', 'G', 'B'])  # 1st column as index
```

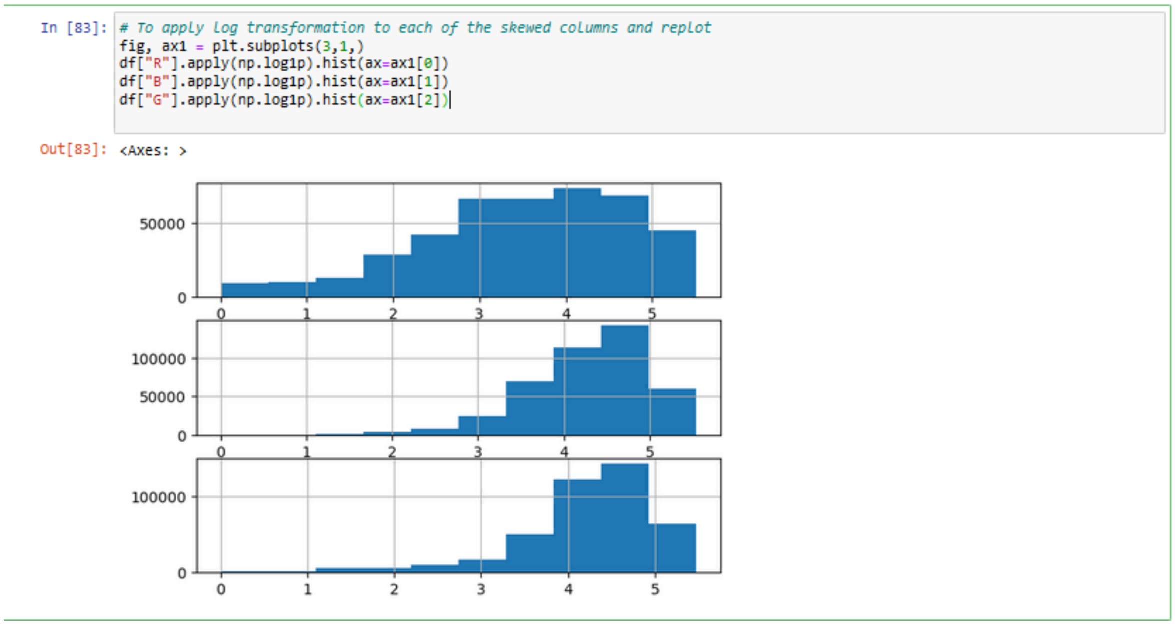
```
In [77]: #Creating a list of float columsn to check for skewing
         num_cols = df.select_dtypes('number').columns

         #This is the max limit above which log tranform will be carried out
         skew_limit = 0.2
         skew_vals = df.skew()

         #To show the skewed columns
         skew_cols = skew_vals[abs(skew_vals)>skew_limit].sort_values(ascending=False)
         skew_cols

Out[77]: R    1.195592
         B    0.746342
         G    0.483745
         dtype: float64
```

As seen above the skew limit of 0.2 is exceeded hence, it is important log transformation is carried out to get the right balance of the data set around the centre. The corresponding result after undergoing a transformation is shown in Figure 2.



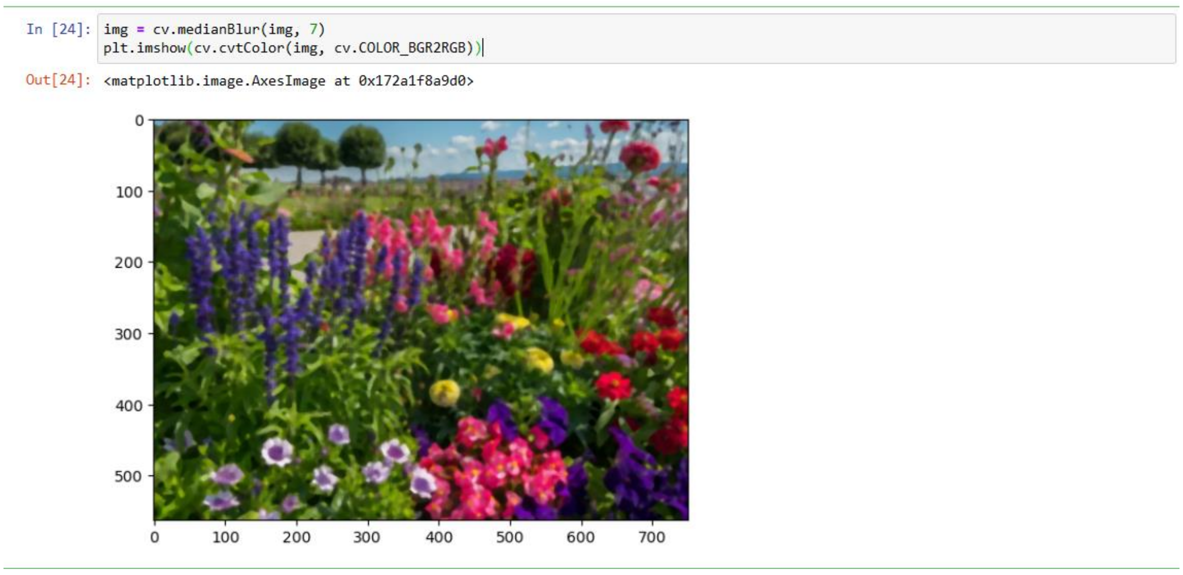
```
In [83]: # To apply log transformation to each of the skewed columns and replot
         fig, ax1 = plt.subplots(3,1,)
         df["R"].apply(np.log1p).hist(ax=ax1[0])
         df["B"].apply(np.log1p).hist(ax=ax1[1])
         df["G"].apply(np.log1p).hist(ax=ax1[2])

Out[83]: <Axes: >
```

Figure 2: Log transformed Image Data

Step 3: Image Cleaning

The feature engineering process requires smoothing the image as shown in the figure below (Figure 3) and then reshaping the number of pixels into 42225 rows corresponding to observations and the 3 channels corresponding to the number of columns. The final preprocessing step was converting the data to float type to increase the accuracy of the mean shift estimate bandwidth. The image plot is shown in 3D in Figure 4.



```
In [24]: img = cv.medianBlur(img, 7)
         plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))

Out[24]: <matplotlib.image.AxesImage at 0x172a1f8a9d0>
```

Figure 3: Smoothing of the image

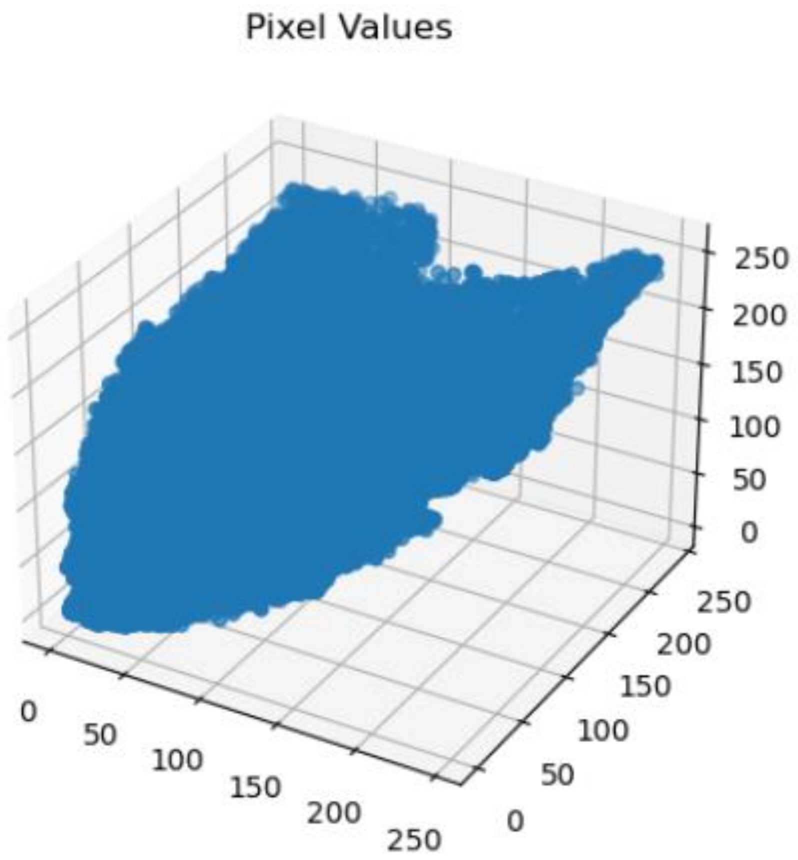


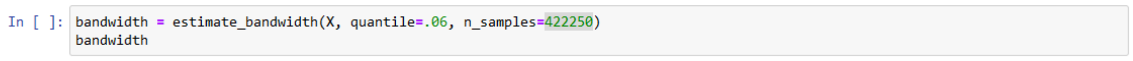
Figure 4: Image data plot in 3D

4. Summary of model training

Two clustering algorithms have been carried out in this project, the mean shift algorithm and kmeans clustering algorithm outlined below

A. Mean Shift Clustering Algorithm

The number of samples used for the mean shift clustering training is 42225 as shown in the Estimate_bandwith method declaration below.



```
In [ ]: bandwidth = estimate_bandwidth(X, quantile=.06, n_samples=422250)
        bandwidth
```

To test the performance of the mean shift clustering algorithm, the training was carried out on the hyperparameter, quantile. That is quantile parameter was set to 0.06 and 0.1 and the image clustering results is shown in Figure 5a and 5b respectively. The model with hyperparameter quantile set to 0.06 produced four (4) clustering results corresponding to the type of flower species in the image in contrast to the second model with a hyperparameter quantile set to 0.1 produced 3 clustering results.

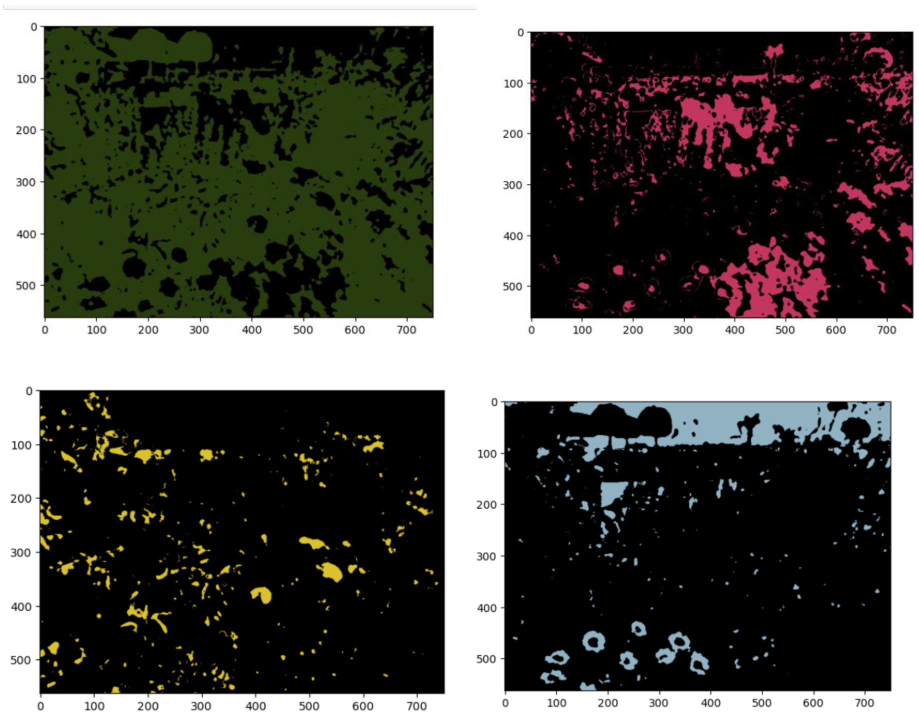


Figure 5a: Four clusters predicted with quantile = 0.06

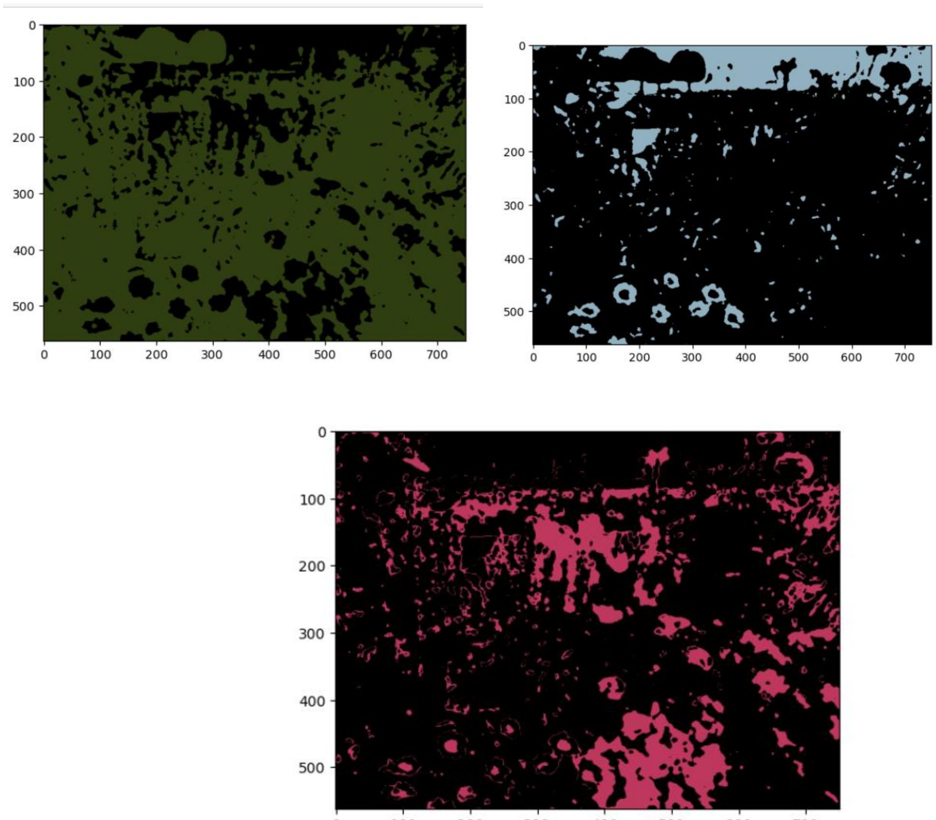
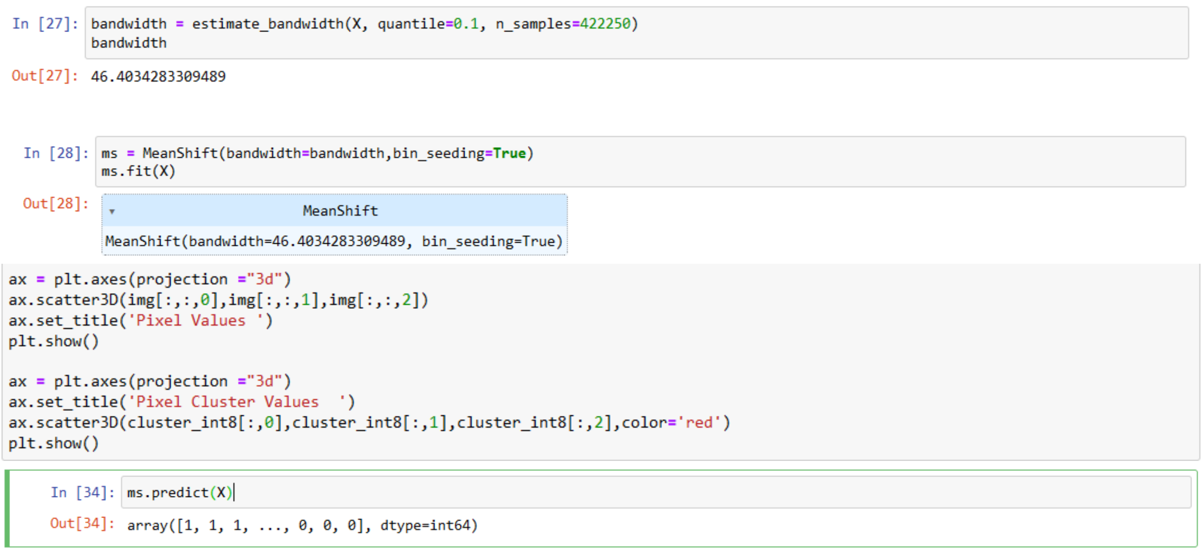


Figure 5b: Three clusters predicted with quantile = 0.1

The code snippet of the project with training, fitting, and predicting methods as well as assigning cluster labels to the examples is given below.



```
In [27]: bandwidth = estimate_bandwidth(X, quantile=0.1, n_samples=422250)
         bandwidth

Out[27]: 46.4034283309489
```

```
In [28]: ms = MeanShift(bandwidth=bandwidth,bin_seeding=True)
         ms.fit(X)

Out[28]:                       MeanShift
         MeanShift(bandwidth=46.4034283309489, bin_seeding=True)
```

```
ax = plt.axes(projection ="3d")
ax.scatter3D(img[:,:,0],img[:,:,1],img[:,:,2])
ax.set_title('Pixel Values ')
plt.show()

ax = plt.axes(projection ="3d")
ax.set_title('Pixel Cluster Values  ')
ax.scatter3D(cluster_int8[:,0],cluster_int8[:,1],cluster_int8[:,2],color='red')
plt.show()
```

```
In [34]: ms.predict(X)

Out[34]: array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

B. KMeans Clustering Algorithm

To carry out the K-means clustering algorithm a function is defined that returns the image (img) replaced by its cluster centers and the k-means inertia as shown in the code snippets below. The input arguments are the image (img) and number of clusters (k) required. Then the function can be called on any image.

```
In [47]: # Defining Clustering function
         def image_cluster(img,k):
             img_flat=img.reshape(img.shape[0]*img.shape[1],3)
             kmeans = KMeans(n_clusters=k, random_state=0).fit(img_flat)
             img_flat2 = img_flat.copy()

             # loops for each cluster center
             for i in np.unique(kmeans.labels_):
                 img_flat2[kmeans.labels_==i,:] = kmeans.cluster_centers_[i]

             img2 = img_flat2.reshape(img.shape)
             return img2, kmeans.inertia_
```

```
In [48]: # Calling the function for k between 2 and 20, and drawing inertia curve to determine optimum number of clusters
         k_vals = list(range(2,21,2))
         img_list = []
         inertia = []
         for k in k_vals:
             # print(k)
             img2, ine = image_cluster(img,k)
             img_list.append(img2)
             inertia.append(ine)
```

```
In [50]: # To find optimal number of clusters
         plt.plot(k_vals,inertia)
         plt.scatter(k_vals,inertia)
         plt.xlabel('k')
         plt.ylabel('Inertia');
```

The inertia is plotted against the k-number of clusters as shown in Figure 6. The figure shows an elbow occurs roughly at k=6 indicating the best number of clusters.

```
In [50]: # to find optimal number of clusters
         plt.plot(k_vals,inertia)
         plt.scatter(k_vals,inertia)
         plt.xlabel('k')
         plt.ylabel('Inertia');
```

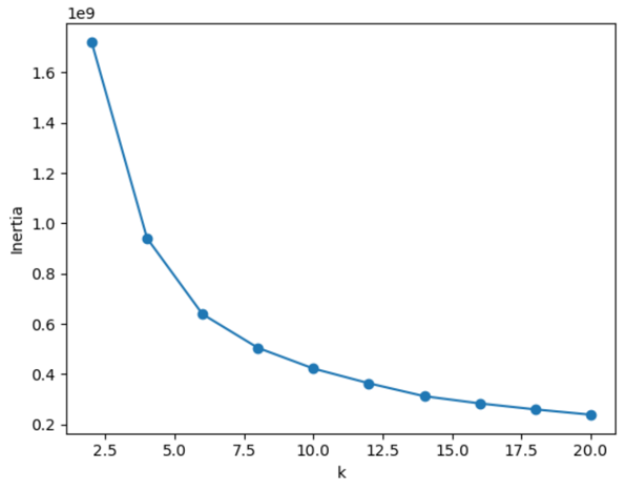
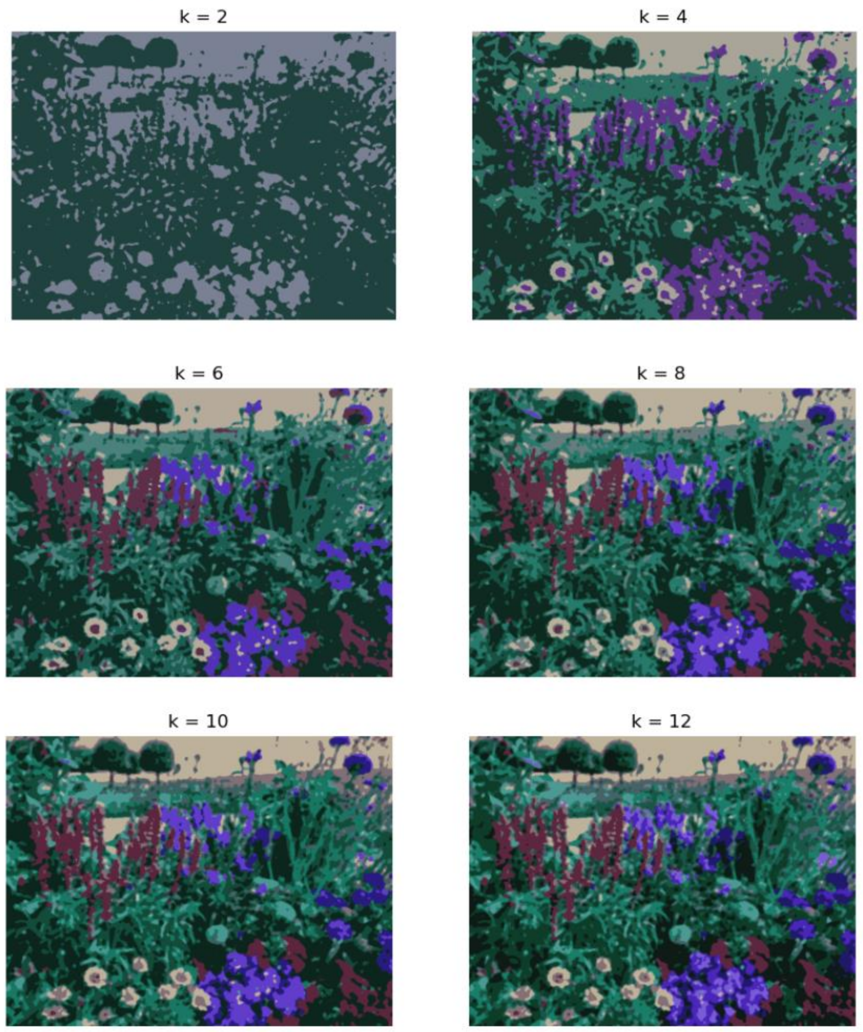


Figure 6: Inertia vs k-number of clusters

In [51]:

```python
# plot in a grid all the images for the different k values
plt.figure(figsize=[10,20])
for i in range(len(k_vals)):
    plt.subplot(5,2,i+1)
    plt.imshow(img_list[i])
    plt.title('k = ' + str(k_vals[i]) )
    plt.axis('off');
```



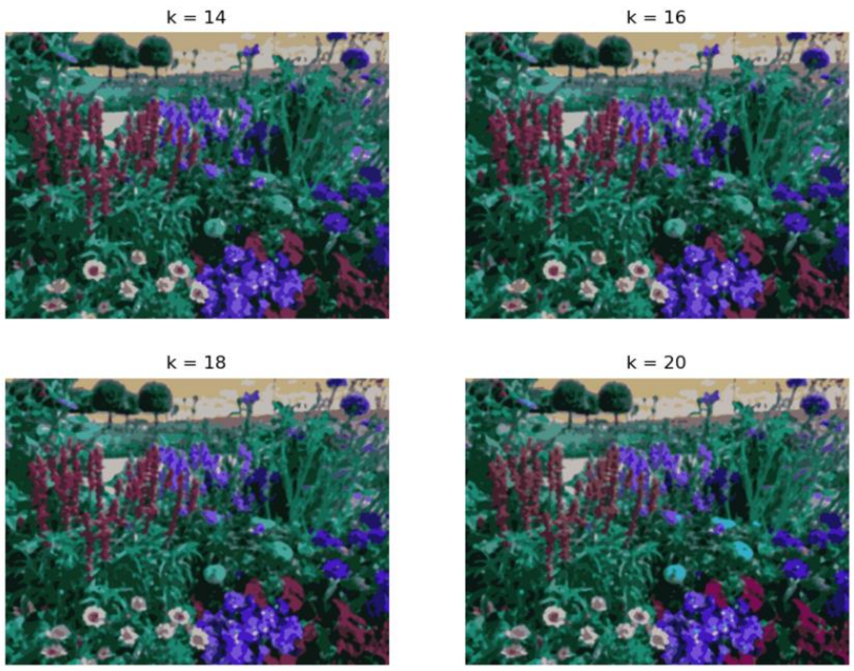


Figure 7: Results of the k-means clustering algorithm

5. Recommended model

From the clustering results, the model that best fits the image data with a balance of bias and variance is the k-means algorithm with 6 clusters as pointed out in the previous section. The k-means is preferred because it gives flexibility to the number of clusters that can be chosen. Comparing the image clustering results from the mean shift and k-means algorithm shows the number of clustering groups is less obvious than the k-means algorithm hence it is recommended for this project.

6. Key findings and insight

For the mean shift clustering algorithm, I observed reducing the quantile value in the estimate_bandwidth method produces a higher number of clusters but poor clustering results. For the k-means clustering algorithm, I observed setting high number of clusters caused more computational time in training.

7. Future work (next steps)

Future work will involve investigating a metric for evaluating the accuracy of the k-means and mean shift algorithm used in this project.