2.1) We can display images in python using `plt.imshow(x)`, where x is the image. Show the first image from the dataset to understand what the data looks like.

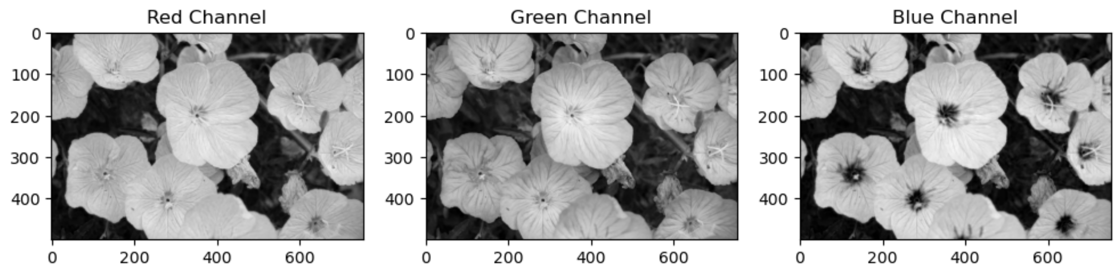
```
In [8]: index = 0 #first image
        x, y = dataset_train[index] #x: image, y: the label, classes[y]: the actual name of the flower

        ######## Fill in the code below ########
        plt.imshow(x)
        plt.title(classes[y])
        ########################################
```

```
Out[8]: Text(0.5, 1.0, 'pink primrose')
```

2.2) Print the shape and type of the image. Generate three plots, each showing the image only in one color channel in gray scale (`cmap='gray'`)!

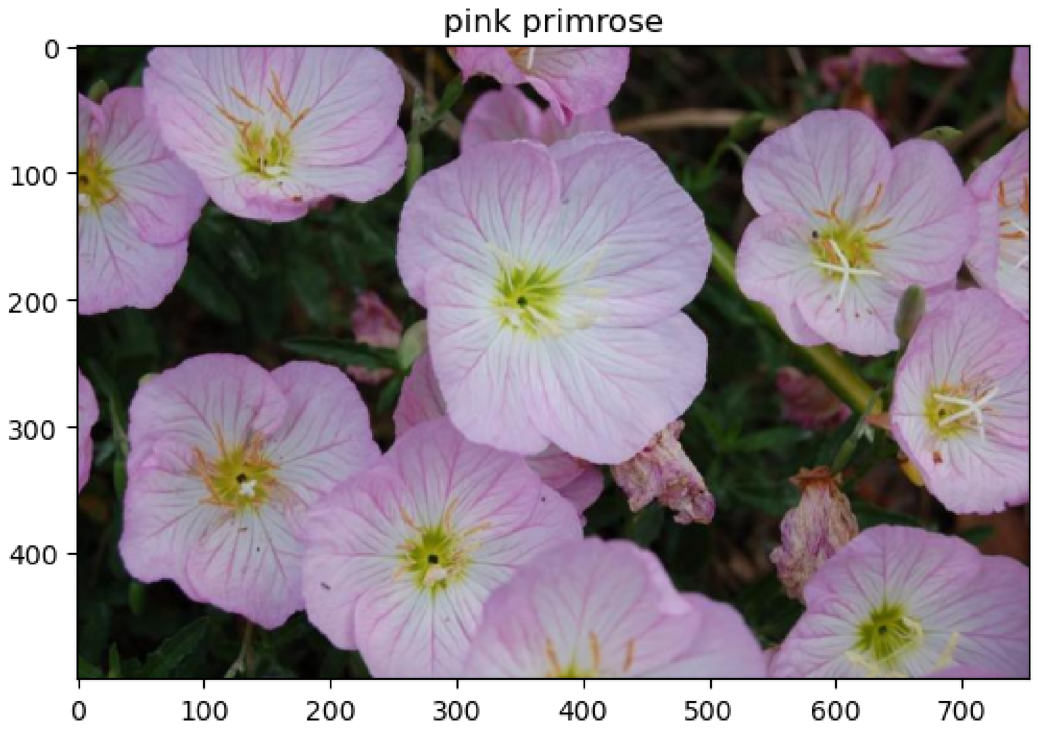
```
In [9]: ######## Fill in the code below ########
        print("Shape:", x.shape)
        print("Type:", type(x))

        fig, axes = plt.subplots(1, 3, figsize=(12, 4))
        # R channel
        axes[0].imshow(x[:, :, 0], cmap='gray')
        axes[0].set_title('Red Channel')

        # G channel
        axes[1].imshow(x[:, :, 1], cmap='gray')
        axes[1].set_title('Green Channel')

        # B channel
        axes[2].imshow(x[:, :, 2], cmap='gray')
        axes[2].set_title('Blue Channel')

        plt.show()
        ########################################
```

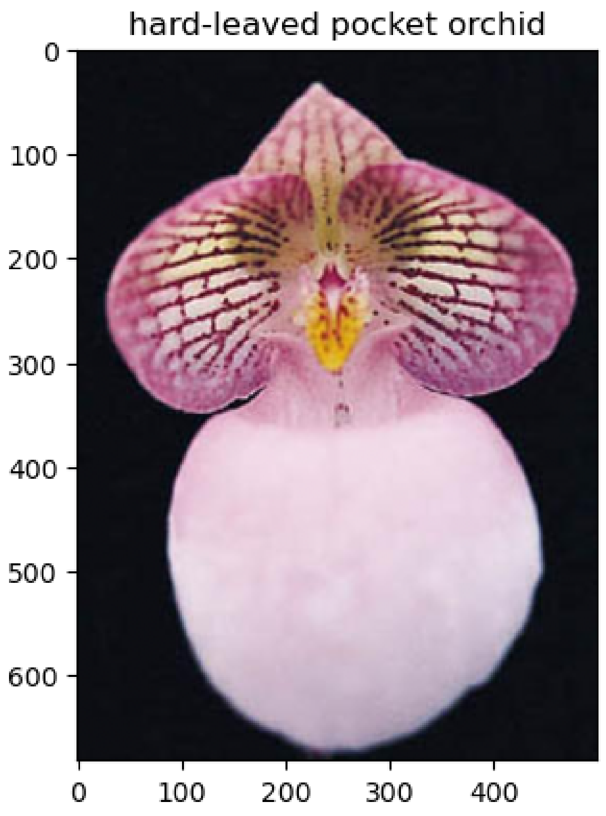
```
Shape: (500, 754, 3)
Type: <class 'numpy.ndarray'>
```

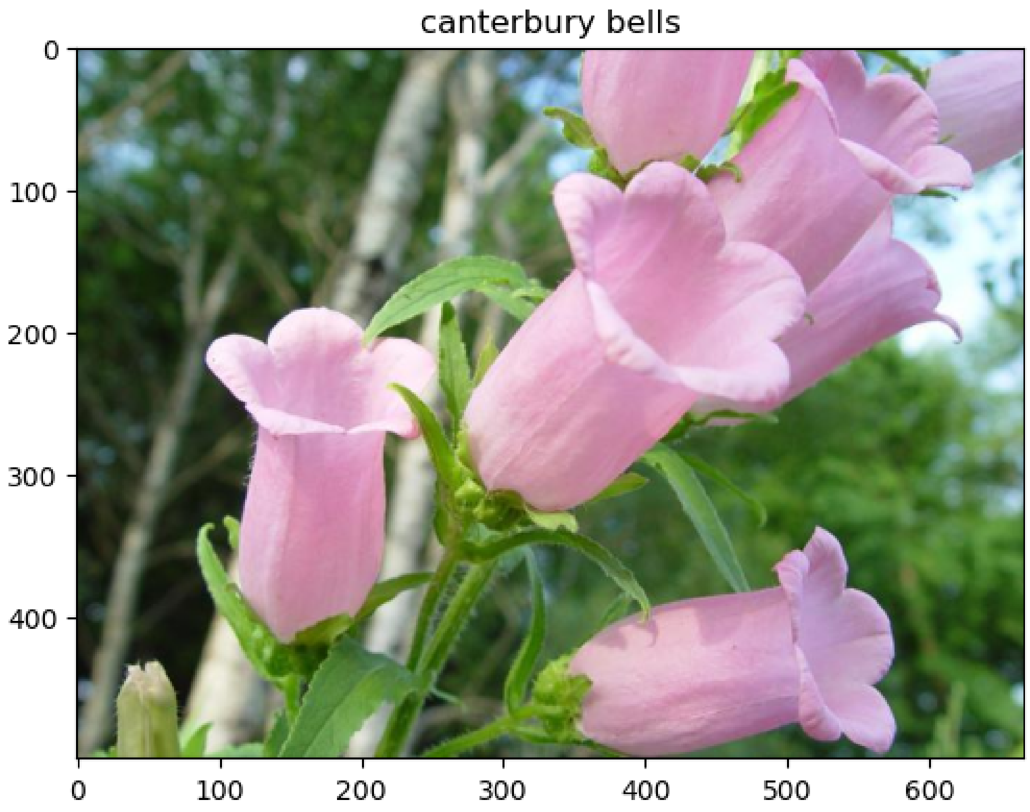


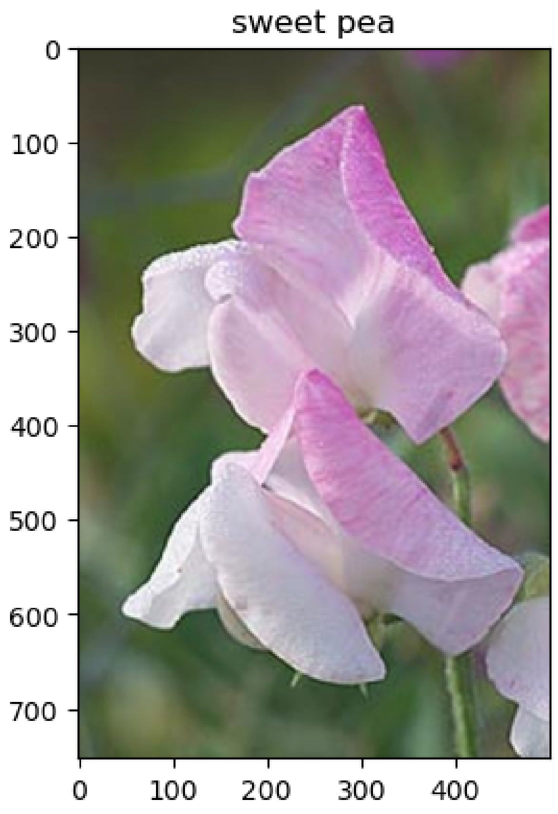
2.3) Show an image in the first 8 classes from the dataset to understand what the data looks like.

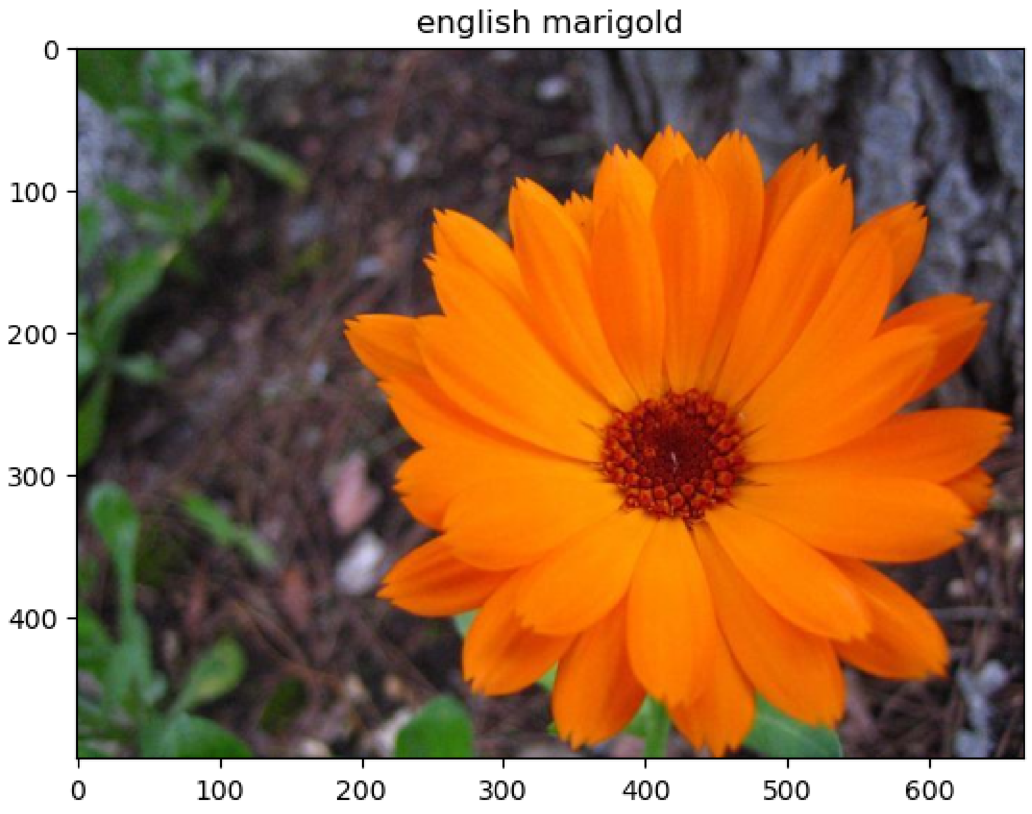
```
In [10]: indices = [0, 10, 20, 30, 40, 50, 60, 70]

         ######## Fill in the code below ########
         for idx in indices:
             x, y = dataset_train[idx]
             plt.figure()
             plt.imshow(x)
             plt.title(classes[y])
             plt.show()
         ########################################
```




hard-leaved pocket orchid
0
100
200
300
400
500
600
0
100
200
300
400


canterbury bells
0
100
200
300
400
0
100
200
300
400
500
600

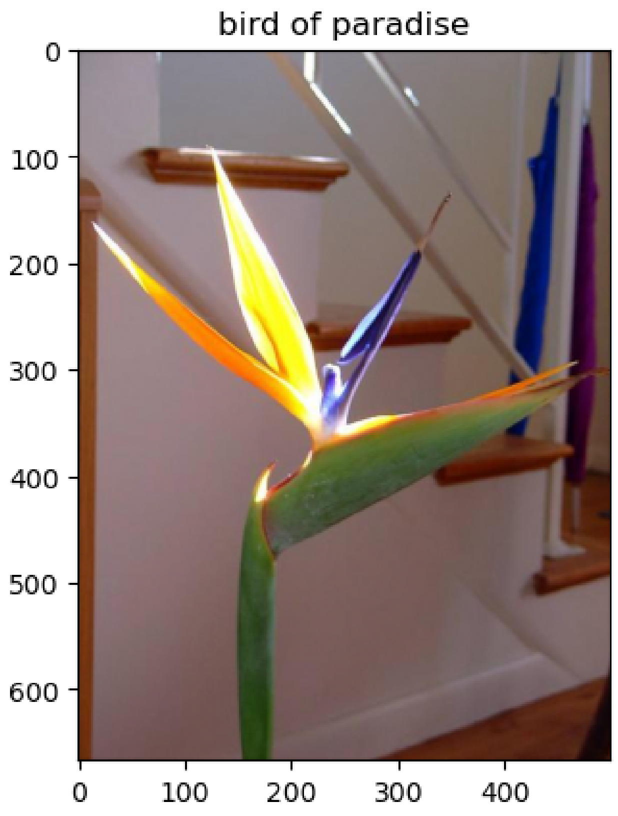

sweet pea
0
100
200
300
400
500
600
700
0
100
200
300
400


english marigold
0
100
200
300
400
0
100
200
300
400
500
600

tiger lily

Copyright (c) kdjoergensen
http://davesgarden.com/members/kdjoergensen/
Unauthorized Use Prohibited
0
100
200
300
400
0
100
200
300
400
500
600

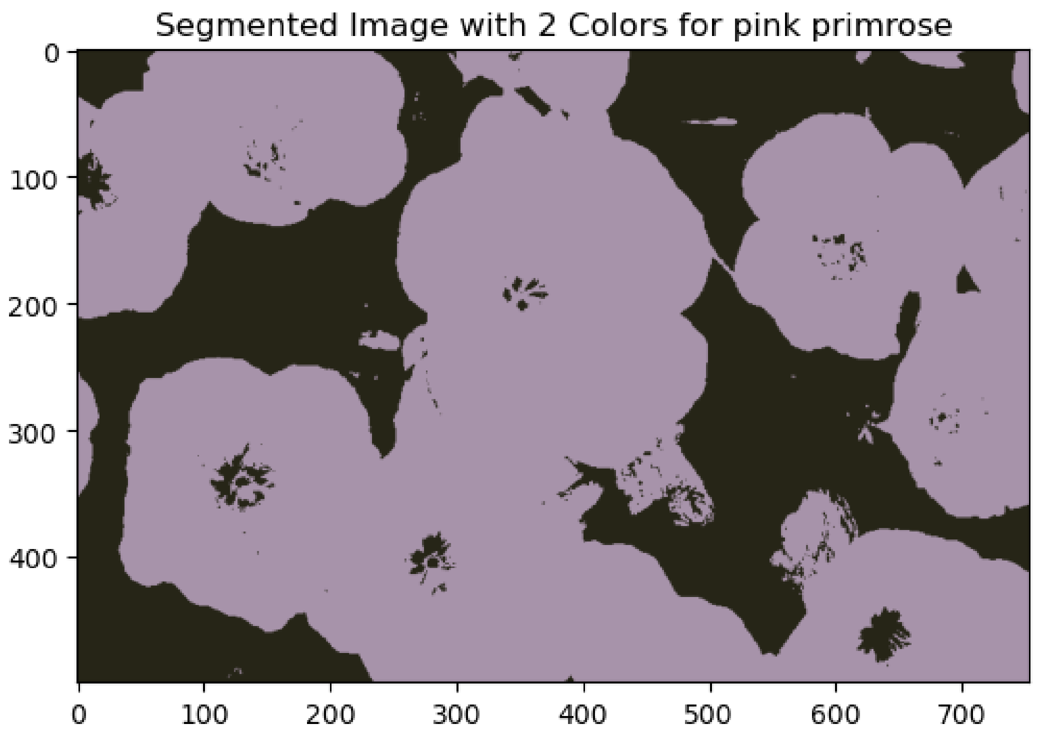
moon orchid
0
100
200
300
400
0
100
200
300
400
500
600
700


bird of paradise
0
100
200
300
400
500
600
0
100
200
300
400

0.0.1 3) Image Segmentation

3.1) Segment the first image in the training set using K-means clustering. Reshape the image into a 2D array where each row represents a pixel and each column represents a color channel (RGB), and then apply K-means clustering to cluster the pixels. Visualize the segmented image and discuss what you observe.

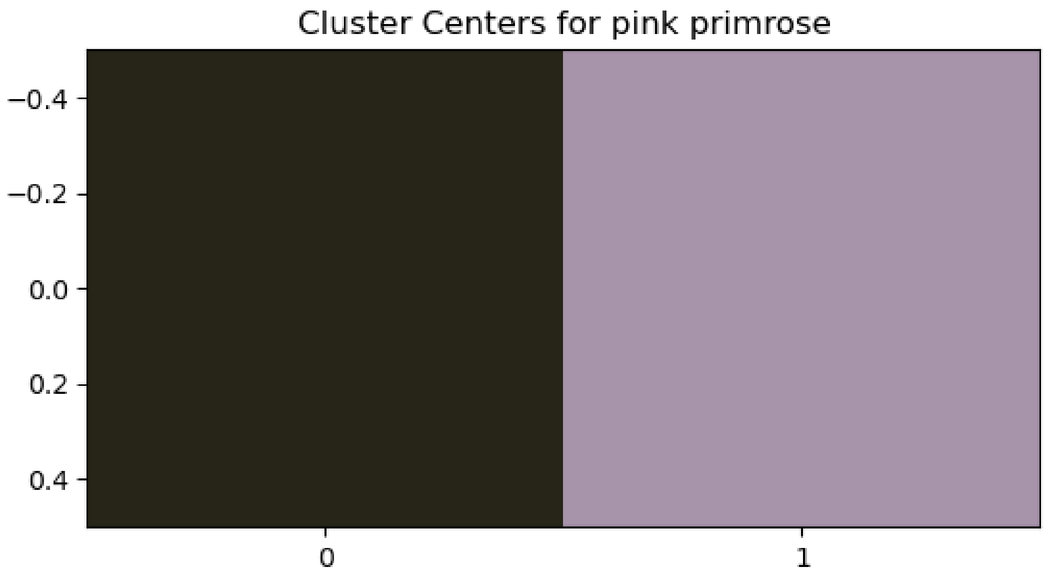
```
In [11]: index = 0

         ######## Fill in the code below ########
         K = 2
         x, y = dataset_train[index]
         x = x.reshape(-1, 3)
         kmeans = KMeans(n_clusters=K, random_state=rng_seed)
         kmeans.fit(x)
         labels = kmeans.labels_
         colors = kmeans.cluster_centers_
         segmented_img = colors[labels].reshape(dataset_train[0][0].shape)
         plt.imshow(segmented_img)
         plt.title(f'Segmented Image with {K} Colors for {classes[y]}')
         plt.show()
         ########################################
         #When using K-means clustering to segment this image, K=2 can roughly separate the flowers and
         #When K=3 or higher, it can roughly separate different parts of the flowers in the image and r
         #(such as the problem of classifying the flower's stamen as background in the image).
```



The segmented image shows distinct color regions corresponding to different parts of the flower and background. Visually, the first cluster represents the background, the second cluster represents the flower, and the third cluster represents the stems. This segmentation helps in identifying different components of the image based on color similarity.

3.2) Visualize the center of each cluster as a color using `plt.imshow(colors.reshape(.., .., ..))`. What does this tell you about the image?

```
In [12]: ######## Fill in the code below ########
         # Visualize cluster centers as colors
         colors_reshaped = colors.reshape(1, K, 3)
         plt.imshow(colors_reshaped)
         plt.title(f'Cluster Centers for {classes[y]}')
         plt.xticks(range(K))
         plt.show()
         ########################################
         #When clustering with K=2, the image is divided into two categories: background and flowers.
         #The former has lower brightness, while the latter has higher brightness.
         #The clustering mainly occurs in the brightness direction.
```

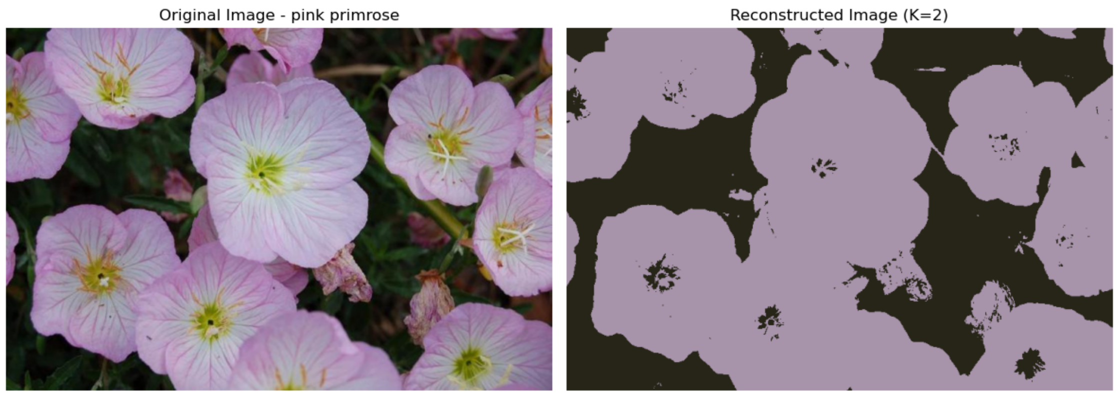


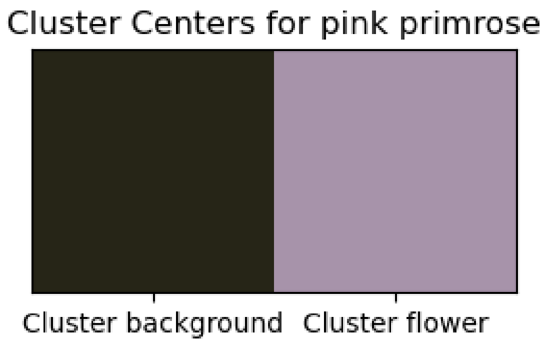
The cluster centers represent the dominant colors in the image. The first cluster center is a shade of dark green corresponding to the ground, the second cluster center is a shade of pinkish purple corresponding to the flower, and the third cluster center is a shade of brown corresponding to the stems. This indicates that the image has distinct color regions that can be effectively captured using K-means clustering.

3.3) Reconstruct the image using the cluster centers and the labels assigned to each pixel. Does the reconstructed image capture the main features of the original image? Discuss your observations.

Type your answer here, replacing this text.

```
In [13]: ######## Fill in the code below ########
         segmentation = ['background', 'flower']
         # Reconstruct the image using cluster centers
         original_shape = dataset_train[index][0].shape
         reconstructed_img = colors[labels].reshape(original_shape)

         # Display original and reconstructed images side by side
         fig, axes = plt.subplots(1, 2, figsize=(12, 5))

         axes[0].imshow(x.reshape(original_shape))
         axes[0].set_title(f'Original Image - {classes[y]}')
         axes[0].axis('off')

         axes[1].imshow(reconstructed_img)
         axes[1].set_title(f'Reconstructed Image (K={K})')
         axes[1].axis('off')

         plt.tight_layout()
         plt.show()

         # Display cluster centers with labels
         fig, ax = plt.subplots(figsize=(8, 2))
         colors_reshaped = colors.reshape(1, K, 3)
         ax.imshow(colors_reshaped)
         ax.set_title(f'Cluster Centers for {classes[y]}')
         ax.set_xticks(range(K))
         ax.set_xticklabels([f'Cluster {segmentation[i]}' for i in range(K)])
         ax.set_yticks([])
         plt.tight_layout()
         plt.show()
         ########################################

         #The clustering results here are relatively clear, with background and flowers, but because th
         #the centers of flowers with colors close to the background are also classified as background,
         #while petals in the background with colors close to flowers can also be classified into the f
```


Original Image - pink primrose
Reconstructed Image (K=2)


Cluster Centers for pink primrose
Cluster background
Cluster flower

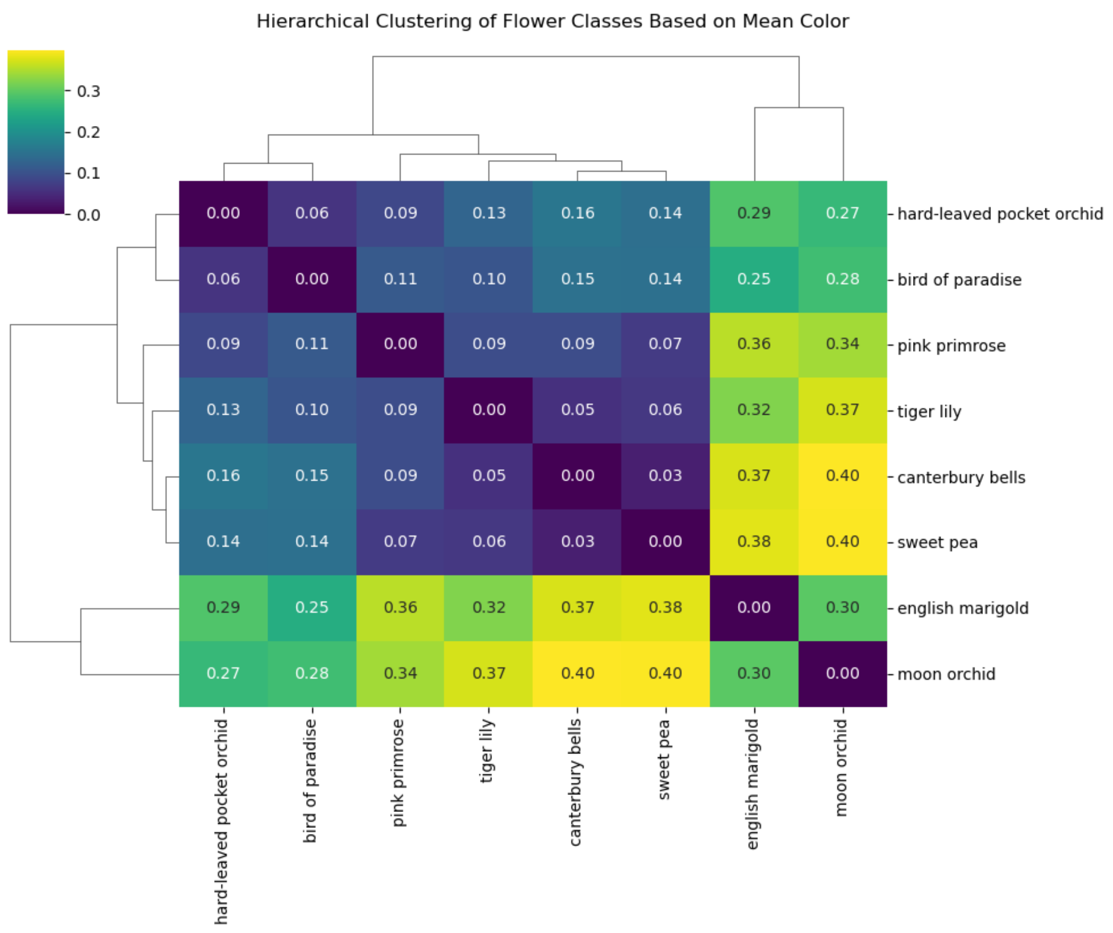
0.0.2 4) Image Clustering

4.1) Use hierarchical clustering (`sns.clustermap`) to cluster the first 8 classes of images in the training set. Compute the mean color of the first image in the class and use the mean colors as features. Next, based on the mean colors, generate a distance square matrix (you can use `pdist(your_means, metric='euclidean')`) for clustering. Visualize the resulting dendrogram and discuss what you observe.

Using average color classification, group most flowers with similar colors into similar parts, and use hierarchical clustering to classify them from a color perspective, and be able to distinguish how large the color difference is between the two types of flowers. For example, although English marigold and moon orchid are grouped together, the distance between them is very large, and they are actually grouped together due to brightness.

```
In [14]: indices = [0, 10, 20, 30, 40, 50, 60, 70]
         ticks = classes[labels_train[indices]]

         ######## Fill in the code below ########
         #mean color
         mean_colors = []
         for idx in indices:
             x, y = dataset_train[idx]
             mean_color = x.reshape(-1, 3).mean(axis=0)
             mean_colors.append(mean_color)

         mean_colors = np.array(mean_colors)

         #pairwise distance matrix
         distances = pdist(mean_colors, metric='euclidean')
         distance_matrix = squareform(distances)

         sns.clustermap(distance_matrix,
                        xticklabels=ticks,
                        yticklabels=ticks,
                        cmap='viridis',
                        figsize=(10, 8),
                        annot=True,
                        fmt='.2f')
         plt.suptitle('Hierarchical Clustering of Flower Classes Based on Mean Color', y=1.02)
         plt.show()
         ########################################
```

```
/Users/zhengxiaoyang/miniconda/lib/python3.13/site-packages/seaborn/matrix.py:530: ClusterWarning: The 
  linkage = hierarchy.linkage(self.array, method=self.method,
/Users/zhengxiaoyang/miniconda/lib/python3.13/site-packages/seaborn/matrix.py:530: ClusterWarning: The 
  linkage = hierarchy.linkage(self.array, method=self.method,
```


Hierarchical Clustering of Flower Classes Based on Mean Color
0.3
0.2
0.1
0.0
hard-leaved pocket orchid 0.00 0.06 0.09 0.13 0.16 0.14 0.29 0.27
bird of paradise 0.06 0.00 0.11 0.10 0.15 0.14 0.25 0.28
pink primrose 0.09 0.11 0.00 0.09 0.09 0.07 0.36 0.34
tiger lily 0.13 0.10 0.09 0.00 0.05 0.06 0.32 0.37
canterbury bells 0.16 0.15 0.09 0.05 0.00 0.03 0.37 0.40
sweet pea 0.14 0.14 0.07 0.06 0.03 0.00 0.38 0.40
english marigold 0.29 0.25 0.36 0.32 0.37 0.38 0.00 0.30
moon orchid 0.27 0.28 0.34 0.37 0.40 0.40 0.30 0.00
hard-leaved pocket orchid
bird of paradise
pink primrose
tiger lily
canterbury bells
sweet pea
english marigold
moon orchid

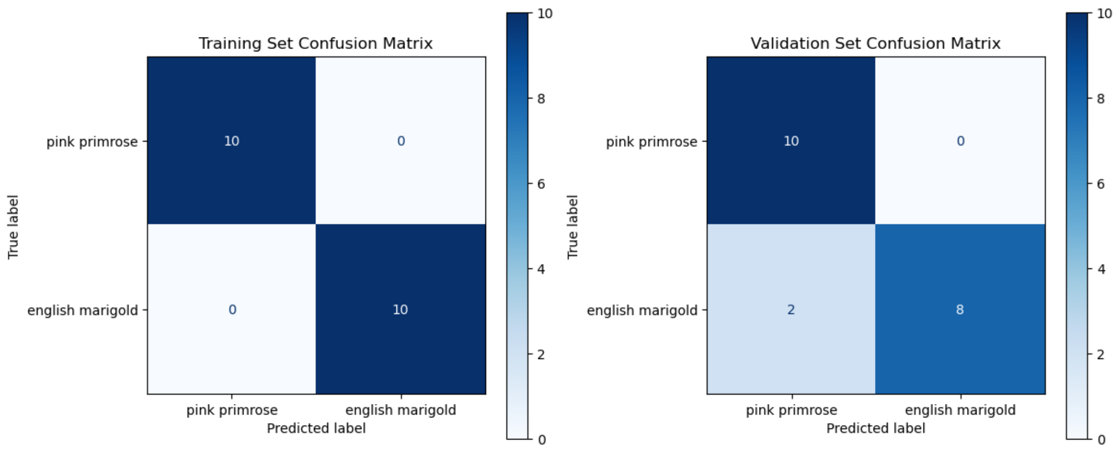
0.0.3 5) Classification

5.1) Perform decision tree classification using `DecisionTreeClassifier` to predict the flower species based on the features from the K-means clustering. Train the decision tree classifier for class 0 (pink primrose) and class 4 (english marigold) on the training set and evaluate its performance on the validation set by creating a **confusion matrix plot** for both, training and evaluation set. Discuss why or why not the features from K-means clustering might be effective for this classification task.

Features from K-means clustering can be effective for classification tasks as they capture the dominant color characteristics of the images, which can be indicative of different flower species. But relying solely on color features may not be sufficient for accurate classification, as different species may share similar colors or have variations in lighting and background that can affect the clustering results.

```
In [15]: indices_train = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49
         indices_val = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])

         ######## Fill in the code below ########
         def extract_features(image, n_clusters=2):
             x = image.reshape(-1, 3)
             kmeans = KMeans(n_clusters=n_clusters, random_state=rng_seed)
             kmeans.fit(x)
             labels = kmeans.labels_
             colors = kmeans.cluster_centers_
             hist, _ = np.histogram(labels, bins=np.arange(n_clusters + 1))
             hist = hist.astype("float")
             hist /= hist.sum()
             features = np.concatenate([hist, colors.flatten()])
             return features

         x_train = np.array([extract_features(dataset_train[i][0]) for i in indices_train])
         y_train = labels_train[indices_train]

         x_val = np.array([extract_features(dataset_val[i][0]) for i in indices_val])
         y_val = labels_val[indices_val]

         tree = DecisionTreeClassifier(random_state=rng_seed)
         tree.fit(x_train, y_train)

         # Predictions
         y_train_pred = tree.predict(x_train)
         y_val_pred = tree.predict(x_val)

         # Training set confusion matrix
         cm_train = confusion_matrix(y_train, y_train_pred)
         disp_train = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=['pink primrose'
         fig, axes = plt.subplots(1, 2, figsize=(12, 5))
         disp_train.plot(ax=axes[0], cmap='Blues')
         axes[0].set_title('Training Set Confusion Matrix')

         # Validation set confusion matrix
         cm_val = confusion_matrix(y_val, y_val_pred)
```

```
disp_val = ConfusionMatrixDisplay(confusion_matrix=cm_val, display_labels=['pink primrose', 'en
disp_val.plot(ax=axes[1], cmap='Blues')
axes[1].set_title('Validation Set Confusion Matrix')

plt.tight_layout()
plt.show()
########################################
```

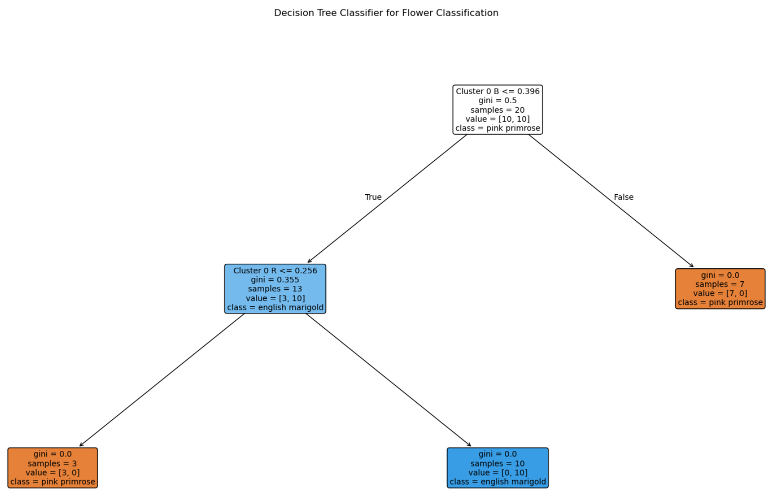


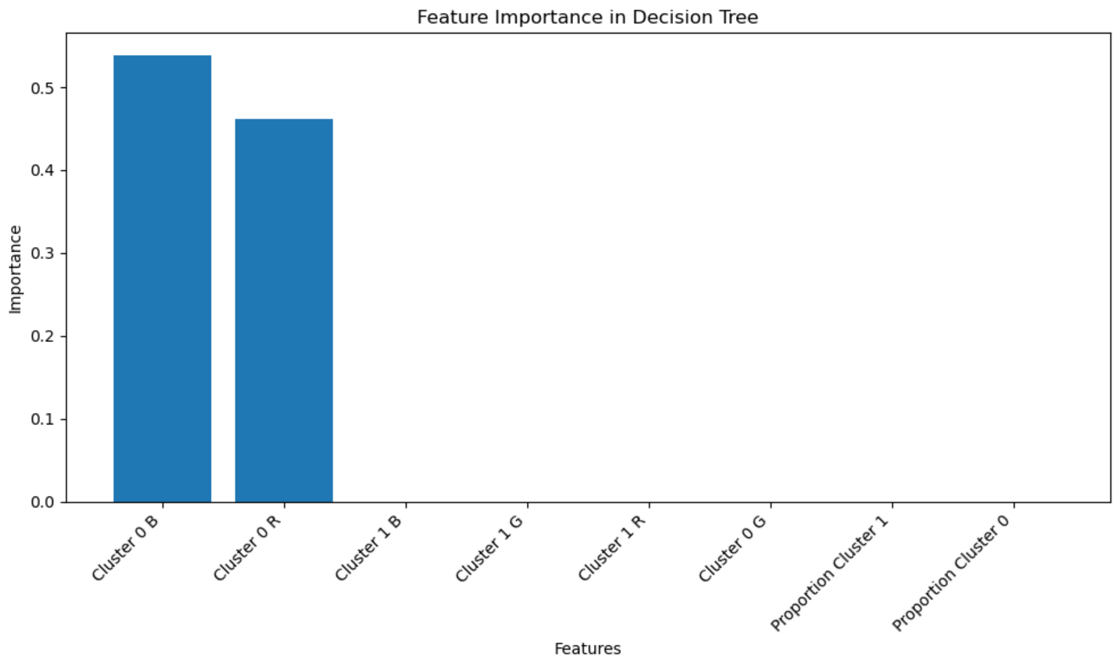
5.2) Visualize the decision tree and interpret the results. Discuss which features are most important for classification and how well the model performs. You can visualize the features as colors to help interpret the decision tree.

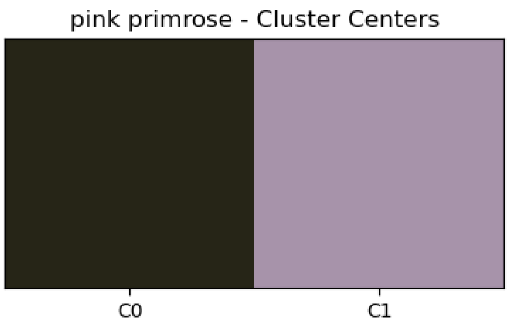
The decision tree visualization shows how the classifier makes decisions based on the K-means clustering features. The features include cluster proportions (how much of the image belongs to each cluster) and the RGB values of each cluster center. Features related to cluster center colors (particularly the dominant colors) appear to be most important for separating pink primrose from english marigold. The model performs well on the training set but may show some overfitting, as indicated by the deeper branches in the tree.

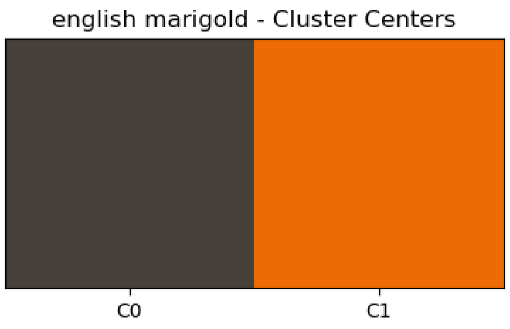
```
In [16]: ######## Fill in the code below ########
         # Visualize the decision tree
         plt.figure(figsize=(20, 10))
         feature_names = ['Proportion Cluster 0', 'Proportion Cluster 1'] + \
                         [f'Cluster {i//3} {["R", "G", "B"][i%3]}' for i in range(6)]
         plot_tree(tree,
                   filled=True,
                   feature_names=feature_names,
                   class_names=['pink primrose', 'english marigold'],
                   rounded=True,
                   fontsize=10)
         plt.title('Decision Tree Classifier for Flower Classification')
         plt.tight_layout()
         plt.show()

         # Feature importance
         importances = tree.feature_importances_
         indices = np.argsort(importances)[::-1]

         plt.figure(figsize=(10, 6))
         plt.bar(range(len(importances)), importances[indices])
         plt.xticks(range(len(importances)), [feature_names[i] for i in indices], rotation=45, ha='righ
         plt.xlabel('Features')
         plt.ylabel('Importance')
         plt.title('Feature Importance in Decision Tree')
         plt.tight_layout()
         plt.show()

         # Visualize cluster centers as colors for interpretation
         n_clusters = 2
         fig, axes = plt.subplots(1, 2, figsize=(10, 3))
         # Use actual class labels: 0 for pink primrose, 4 for english marigold
         class_labels = [0, 4]
         for idx, (class_label, class_name) in enumerate(zip(class_labels, ['pink primrose', 'english ma
             sample_idx = indices_train[y_train == class_label][0]
             x_sample, _ = dataset_train[sample_idx]
             x_reshaped = x_sample.reshape(-1, 3)
             kmeans_temp = KMeans(n_clusters=n_clusters, random_state=rng_seed)
             kmeans_temp.fit(x_reshaped)
             colors_temp = kmeans_temp.cluster_centers_

             axes[idx].imshow(colors_temp.reshape(1, n_clusters, 3))
```

```
    axes[idx].set_title(f'{class_name} - Cluster Centers')
    axes[idx].set_xticks(range(n_clusters))
    axes[idx].set_xticklabels([f'C{i}' for i in range(n_clusters)])
    axes[idx].set_yticks([])

plt.show()
########################################
```






pink primrose - Cluster Centers
C0
C1

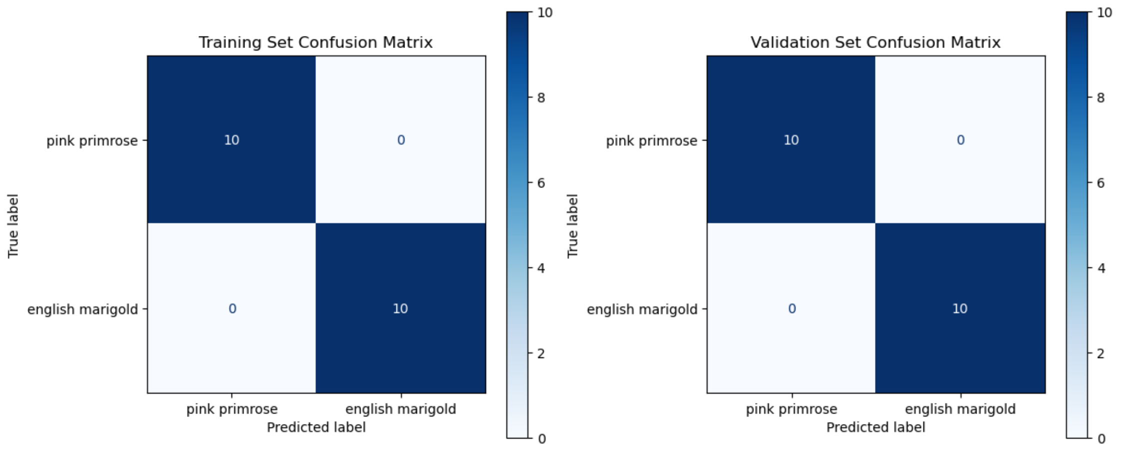

english marigold - Cluster Centers
C0
C1

5.3) Try a little more feature engineering by, for example, converting the RGB color space to HSV color space (`image = rgb_to_hsv(image)`) before applying K-means clustering, as well as sorting the cluster centers before flattening them into a feature vector. See if these changes improve the classification performance.

According to the results, converting the RGB color space to HSV color space before applying K-means clustering and sorting the cluster centers before flattening them into a feature vector improved the classification performance. But the overall accuracy remains similar, indicating that the original RGB features were already quite effective for this classification task.

```
In [ ]: indices_train = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]
        indices_val = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])

        ######## Fill in the code below ########
        def extract_features(image, n_clusters=2):
            image = rgb_to_hsv(image)
            x = image.reshape(-1, 3)
            kmeans = KMeans(n_clusters=n_clusters, random_state=rng_seed)
            kmeans.fit(x)
            labels = kmeans.labels_
            colors = kmeans.cluster_centers_

            # Sort by brightness
            sorted_indices = np.argsort(colors[:, 2])
            colors = colors[sorted_indices]

            label_mapping = {old_idx: new_idx for new_idx, old_idx in enumerate(sorted_indices)}
            labels = np.array([label_mapping[label] for label in labels])

            hist, _ = np.histogram(labels, bins=np.arange(n_clusters + 1))
            hist = hist.astype("float")
            hist /= hist.sum()
            features = np.concatenate([hist, colors.flatten()])
            return features

        x_train = np.array([extract_features(dataset_train[i][0]) for i in indices_train])
        y_train = labels_train[indices_train]

        x_val = np.array([extract_features(dataset_val[i][0]) for i in indices_val])
        y_val = labels_val[indices_val]

        tree = DecisionTreeClassifier(random_state=rng_seed)
        tree.fit(x_train, y_train)

        # Predictions
        y_train_pred = tree.predict(x_train)
        y_val_pred = tree.predict(x_val)

        # Training set confusion matrix
        cm_train = confusion_matrix(y_train, y_train_pred)
        disp_train = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=['pink primrose',
        fig, axes = plt.subplots(1, 2, figsize=(12, 5))
        disp_train.plot(ax=axes[0], cmap='Blues')
```

```
axes[0].set_title('Training Set Confusion Matrix')

# Validation set confusion matrix
cm_val = confusion_matrix(y_val, y_val_pred)
disp_val = ConfusionMatrixDisplay(confusion_matrix=cm_val, display_labels=['pink primrose', 'eng
disp_val.plot(ax=axes[1], cmap='Blues')
axes[1].set_title('Validation Set Confusion Matrix')

plt.tight_layout()
plt.show()
########################################
```

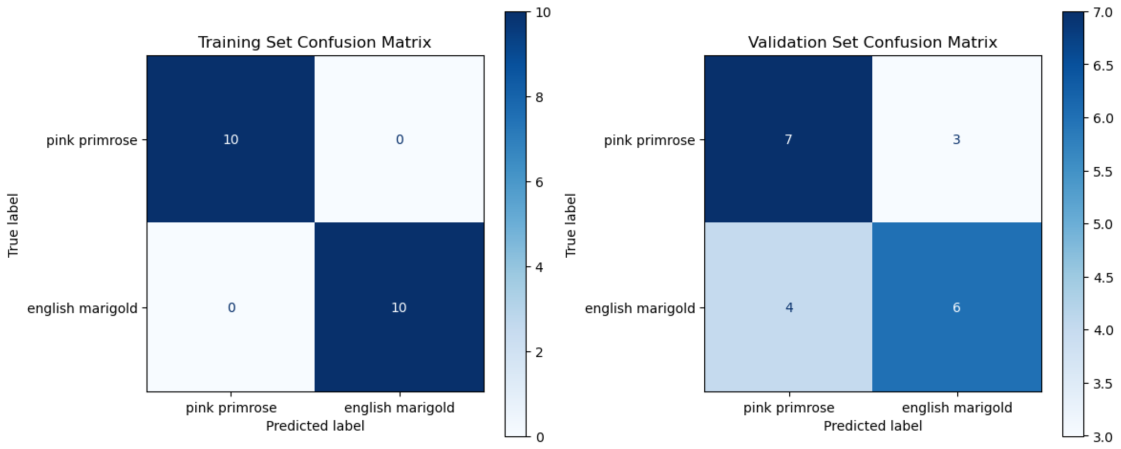


5.4) Try your final classification method on class 0 (pink primrose) and class 1 (hard-leaved pocket orchid). How well does it perform? Discuss what kind of features we have not considered that might help improve the classification.

It's not performing as well as expected. Possible reasons include the similarity in color features between the two flower species, which makes it difficult for the decision tree to distinguish between them based solely on K-means clustering features. Additional features such as texture, shape, or more advanced image descriptors could help improve classification performance by providing more discriminative information about the flowers.

```
In [22]: indices_train = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
         indices_val = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

         ######## Fill in the code below ########
         def extract_features(image, n_clusters=2):
             image = rgb_to_hsv(image)
             x = image.reshape(-1, 3)
             kmeans = KMeans(n_clusters=n_clusters, random_state=rng_seed)
             kmeans.fit(x)
             labels = kmeans.labels_
             colors = kmeans.cluster_centers_

             # Sort by brightness
             sorted_indices = np.argsort(colors[:, 2])
             colors = colors[sorted_indices]
             label_mapping = {old_idx: new_idx for new_idx, old_idx in enumerate(sorted_indices)}
             labels = np.array([label_mapping[label] for label in labels])

             hist, _ = np.histogram(labels, bins=np.arange(n_clusters + 1))
             hist = hist.astype("float")
             hist /= hist.sum()
             features = np.concatenate([hist, colors.flatten()])
             return features

         x_train = np.array([extract_features(dataset_train[i][0]) for i in indices_train])
         y_train = labels_train[indices_train]

         x_val = np.array([extract_features(dataset_val[i][0]) for i in indices_val])
         y_val = labels_val[indices_val]

         tree = DecisionTreeClassifier(random_state=rng_seed)
         tree.fit(x_train, y_train)

         # Predictions
         y_train_pred = tree.predict(x_train)
         y_val_pred = tree.predict(x_val)

         # Training set confusion matrix
         cm_train = confusion_matrix(y_train, y_train_pred)
         disp_train = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=['pink primrose'
         fig, axes = plt.subplots(1, 2, figsize=(12, 5))
         disp_train.plot(ax=axes[0], cmap='Blues')
```

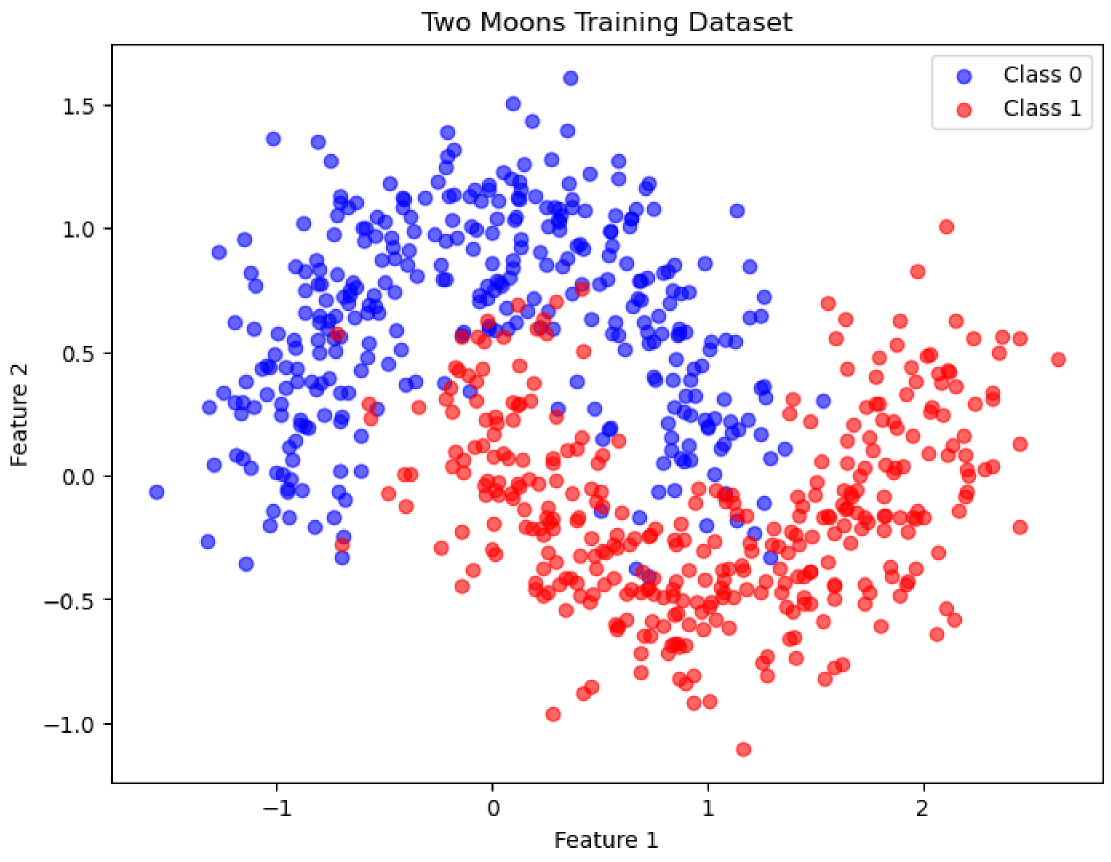
```
axes[0].set_title('Training Set Confusion Matrix')

# Validation set confusion matrix
cm_val = confusion_matrix(y_val, y_val_pred)
disp_val = ConfusionMatrixDisplay(confusion_matrix=cm_val, display_labels=['pink primrose', 'en
disp_val.plot(ax=axes[1], cmap='Blues')
axes[1].set_title('Validation Set Confusion Matrix')

plt.tight_layout()
plt.show()
########################################
```



0.0.4 1) Generate and visualize the two-moons dataset

The dataset is generated through make_moons. You will need to split the dataset using train_test_split, with a 0.3 test size and setting the random_state to rng_seed. Plot a figure of your training data, with a different colour for each class.

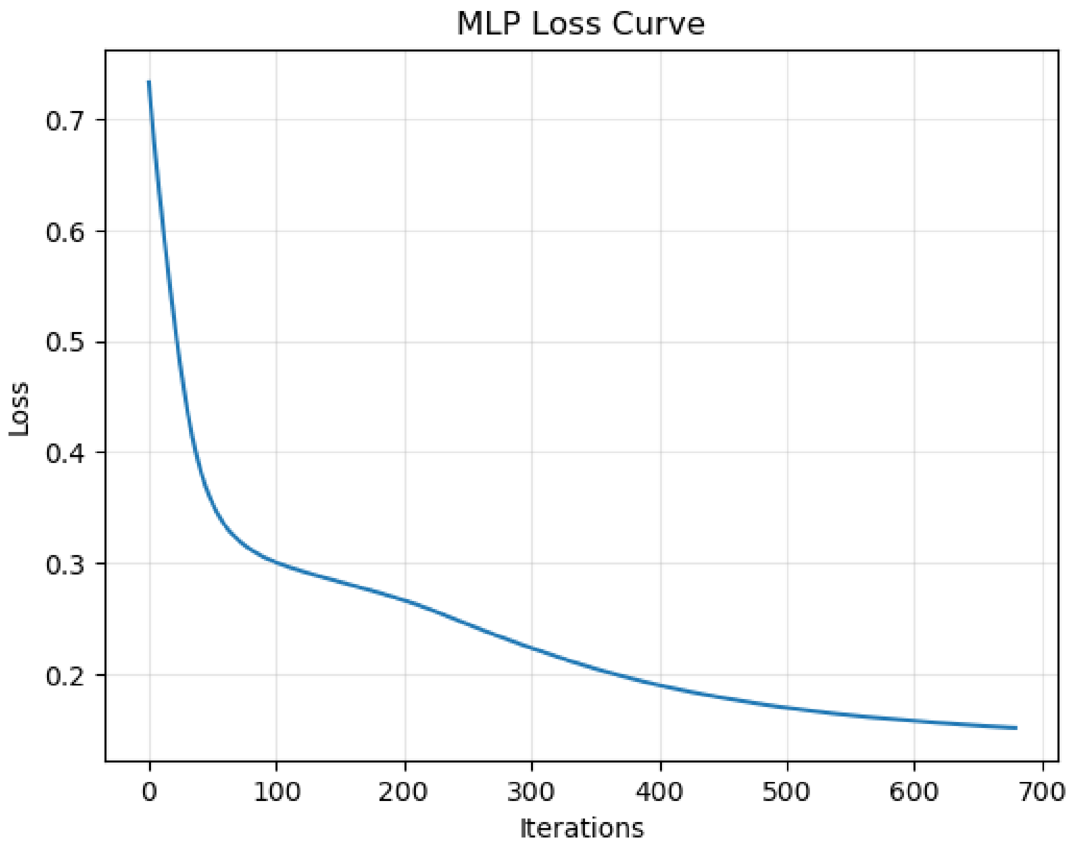
```
In [26]: X, y = make_moons(n_samples=1000, noise=0.25, random_state=rng_seed)
         X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=rng_seed)

         # Plot training data with different colors for each class
         plt.figure(figsize=(8, 6))
         plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], c='blue', label='Class 0', alph
         plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], c='red', label='Class 1', alpha
         plt.xlabel('Feature 1')
         plt.ylabel('Feature 2')
         plt.title('Two Moons Training Dataset')
         plt.legend()
         plt.show()
```



0.0.5 3) Inspect the training loss curve

The attribute `mlp.loss_curve_` stores the loss per iteration. Plot the training loss curve

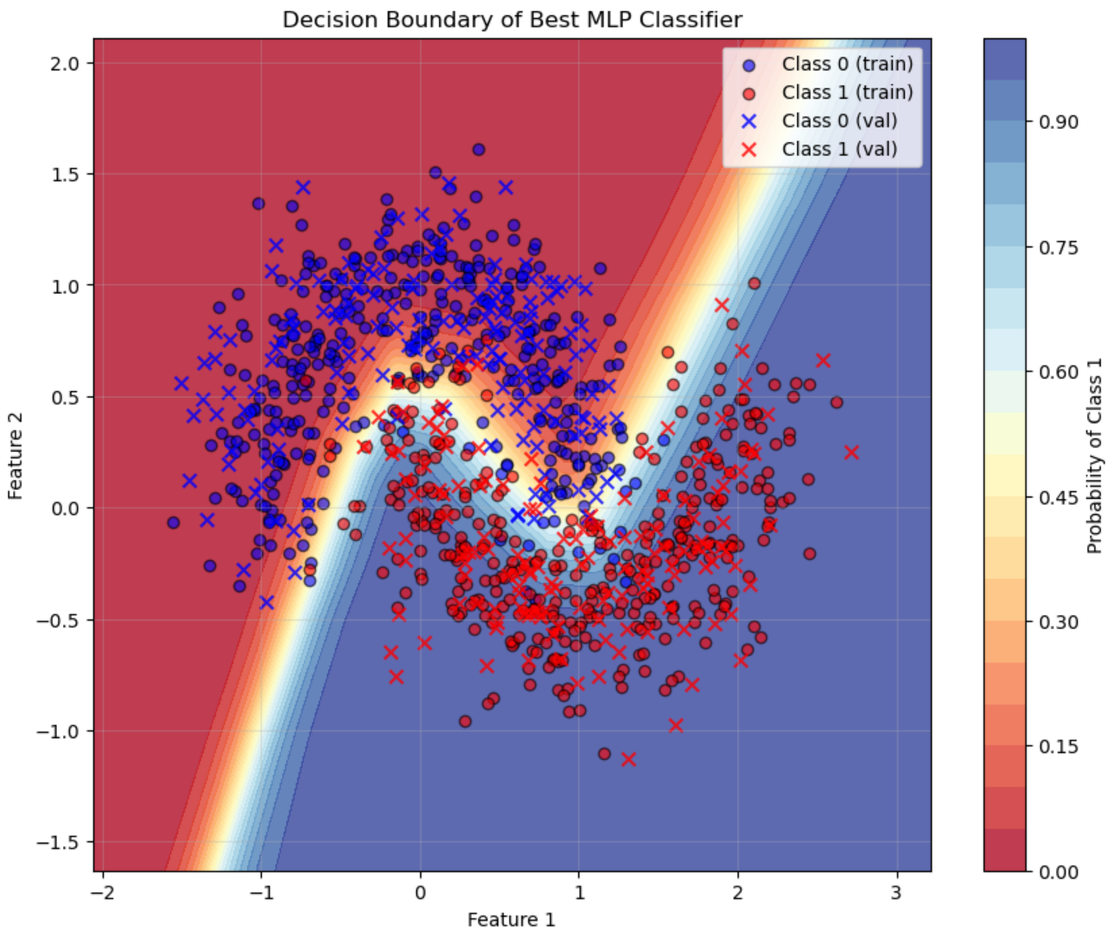
```
In [38]: plt.plot(mlp.loss_curve_)
         plt.title('MLP Loss Curve')
         plt.xlabel('Iterations')
         plt.ylabel('Loss')
         plt.grid(alpha=0.3)
         plt.show()
```



0.0.6 5) Visualize the decision boundary

Write some code to visiualize the decision boundary using your best MLPClassifier. Hint: you can do this by first making a np.meshgrid of values for x1, x2 and classifying every point on the grid with a probability. You can then use a contourf plot to plot the result.

```
In [41]: # Create a meshgrid for decision boundary visualization
         x1_min, x1_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
         x2_min, x2_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
         xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max, 200),
                                np.linspace(x2_min, x2_max, 200))

         # Predict probability for each point in the meshgrid
         Z = best.predict_proba(np.c_[xx1.ravel(), xx2.ravel()])[:, 1]
         Z = Z.reshape(xx1.shape)

         # Plot the decision boundary
         plt.figure(figsize=(10, 8))
         plt.contourf(xx1, xx2, Z, levels=20, cmap='RdYlBu', alpha=0.8)
         plt.colorbar(label='Probability of Class 1')

         # Overlay the training data
         plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1],
                    c='blue', edgecolors='k', label='Class 0 (train)', alpha=0.6)
         plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1],
                    c='red', edgecolors='k', label='Class 1 (train)', alpha=0.6)

         # Overlay the validation data with different marker
         plt.scatter(X_val[y_val == 0, 0], X_val[y_val == 0, 1],
                    c='blue', marker='x', label='Class 0 (val)', alpha=0.8, s=50)
         plt.scatter(X_val[y_val == 1, 0], X_val[y_val == 1, 1],
                    c='red', marker='x', label='Class 1 (val)', alpha=0.8, s=50)

         plt.xlabel('Feature 1')
         plt.ylabel('Feature 2')
         plt.title('Decision Boundary of Best MLP Classifier')
         plt.legend()
         plt.grid(alpha=0.3)
         plt.show()
```

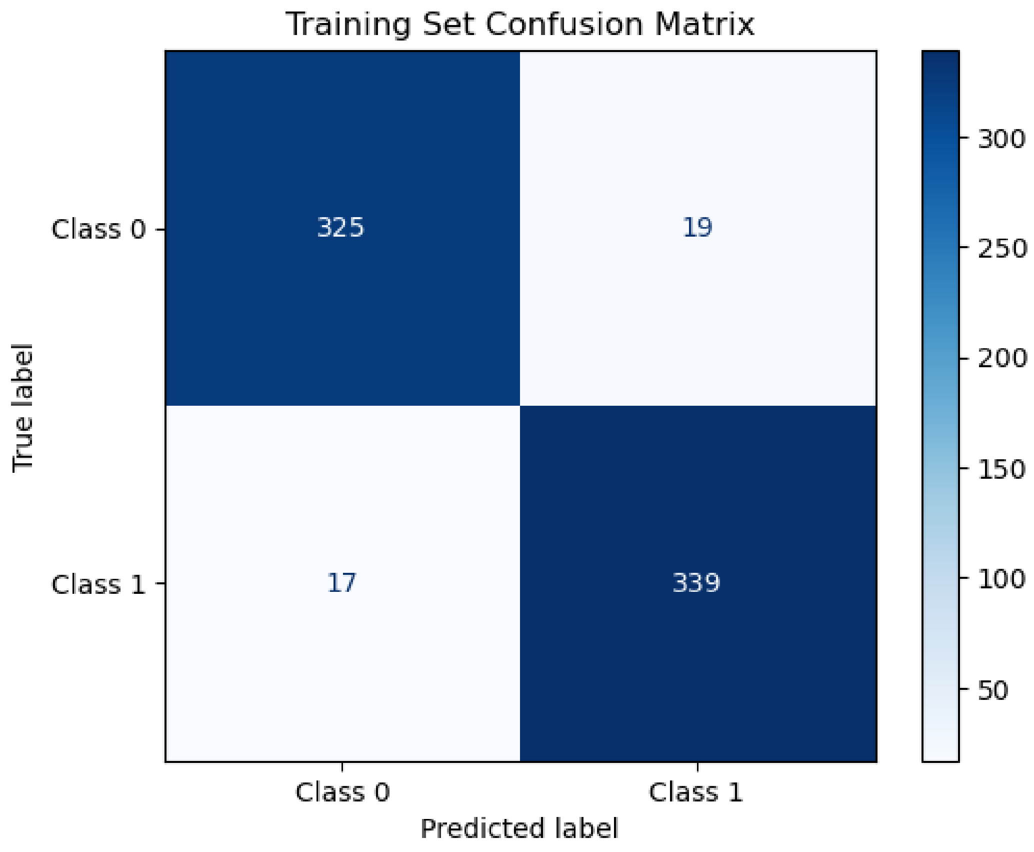

Decision Boundary of Best MLP Classifier
Class 0 (train)
Class 1 (train)
Class 0 (val)
Class 1 (val)
Feature 1
Feature 2
Probability of Class 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2
3
0.90
0.75
0.60
0.45
0.30
0.15
0.00

0.0.7 6) Confusion Matrix

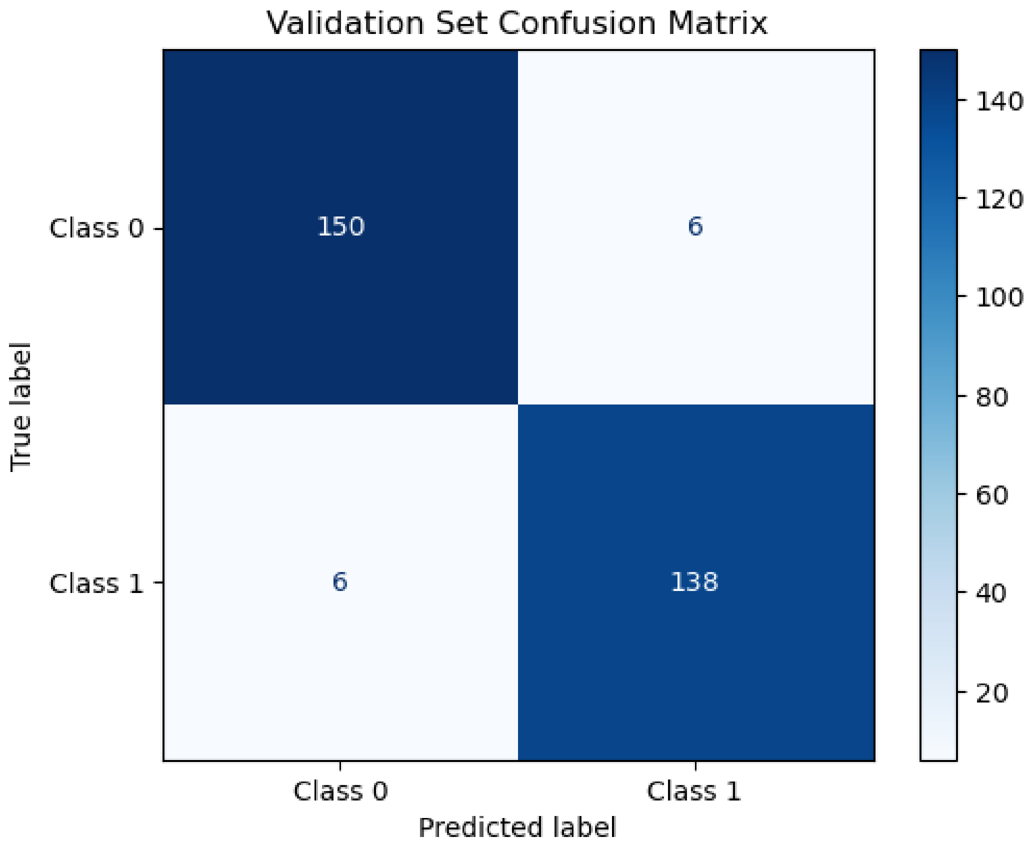
Plot the confusion matrix on the train and prediction data, using the best model from before.

```
In [42]: print('Training accuracy:', best.score(X_train, y_train))
         cm = confusion_matrix(y_train, best.predict(X_train))
         disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Class 0', 'Class 1'])
         disp.plot(cmap='Blues')
         plt.title('Training Set Confusion Matrix')
         plt.show()

         print('Validation accuracy:', best.score(X_val, y_val))
         cm = confusion_matrix(y_val, best.predict(X_val))
         disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Class 0', 'Class 1'])
         disp.plot(cmap='Blues')
         plt.title('Validation Set Confusion Matrix')
         plt.show()
```

```
Training accuracy: 0.9485714285714286
```



```
Validation accuracy: 0.96
```


Validation Set Confusion Matrix
True label
Predicted label
Class 0
Class 1
Class 0
Class 1