Scikit-Learn

•••

GT Big Data Club

Introduction to Scikit-Learn

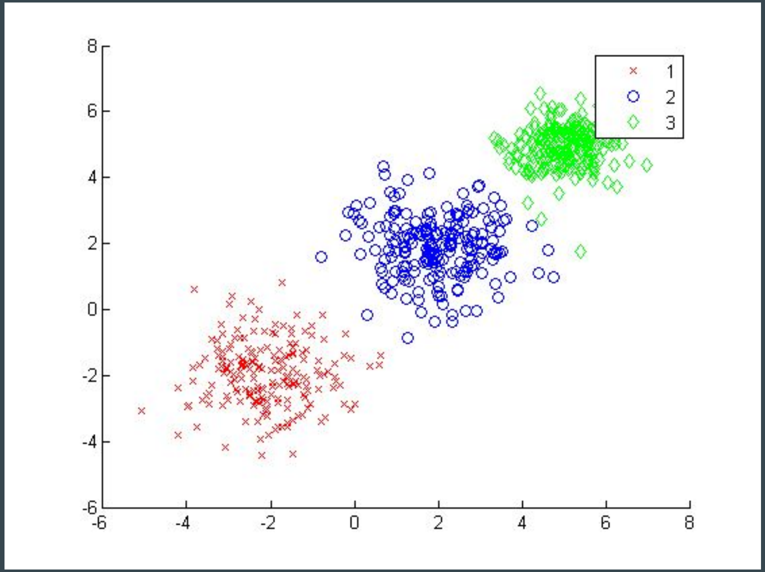
- A powerful, well-documented Machine Learning Library in Python
- Builds off of Numpy, Scipy, and Matplotlib
- Many built in datasets and good tutorials available
- `pip install scikit-learn`

Branches of Machine Learning

- Supervised Learning
  - Given vector of attributes $[x_1, x_2, \ldots x_n]$ can we determine the output value y?
  - Ex: We have pixel values for some handwritten text, can we figure out which digit/letter we have?
  - Ex: We have historical data on housing prices. Can we predict a home's value?
- Unsupervised Learning
  - Clustering: Can I group my data into similar sets?
  - Dimensionality Reduction: Can I get rid of redundant/useless data in my dataset?

Clustering

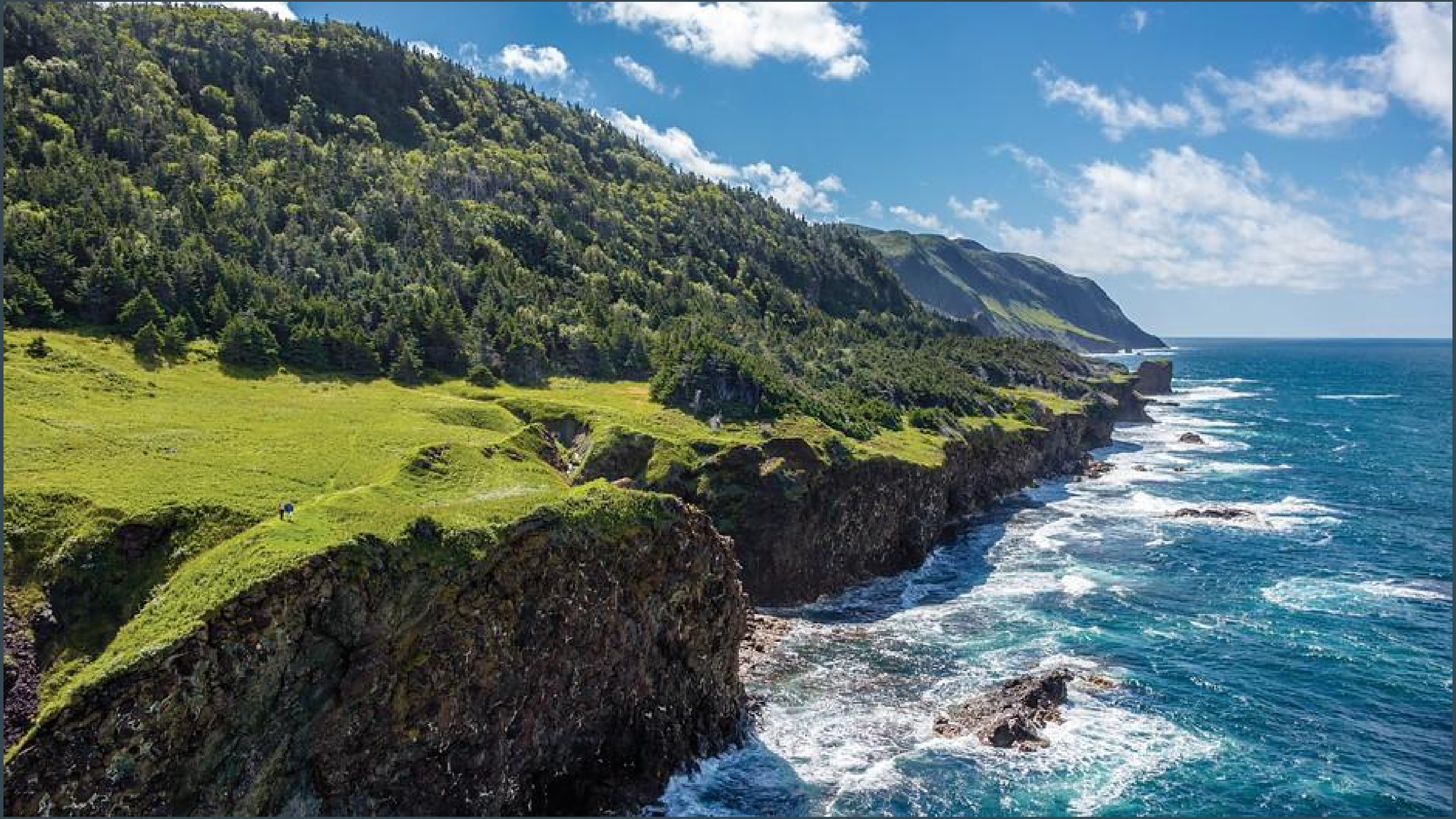
- Given our entire dataset, can we create “buckets” where each element in the bucket is “similar”
- A natural problem to “visualize” in 2D
- Hard to define precisely



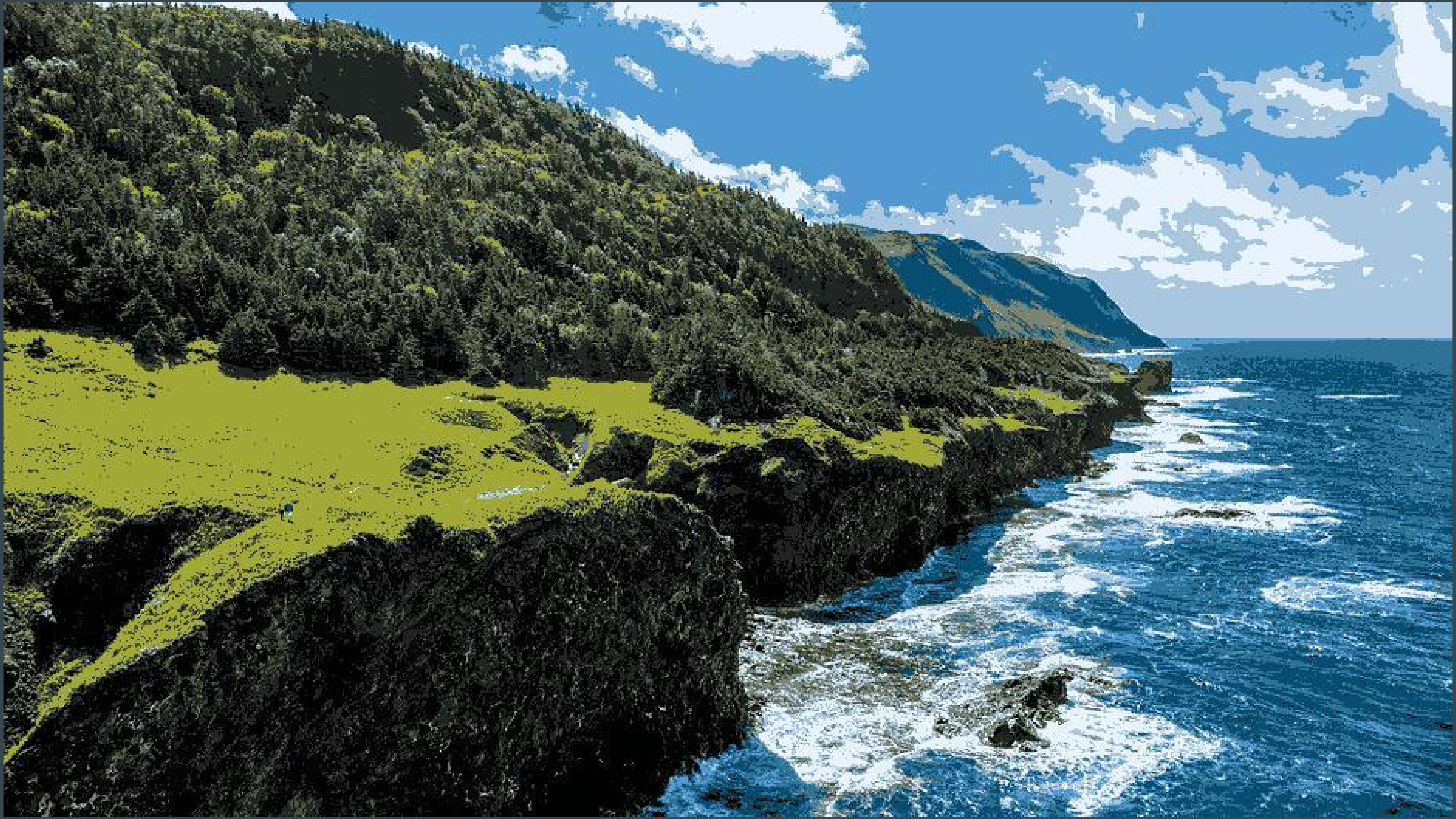
Today

- We’ll be using clustering to recolor a stock image using only 10 colors instead of the original `147,246` colors used
- Get the code on github now: http://github.com/gt-big-data/
- Related scikit-learn tutorial: http://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html#sphx-glr-auto-examples-cluster-plot-color-quantization-py

Before



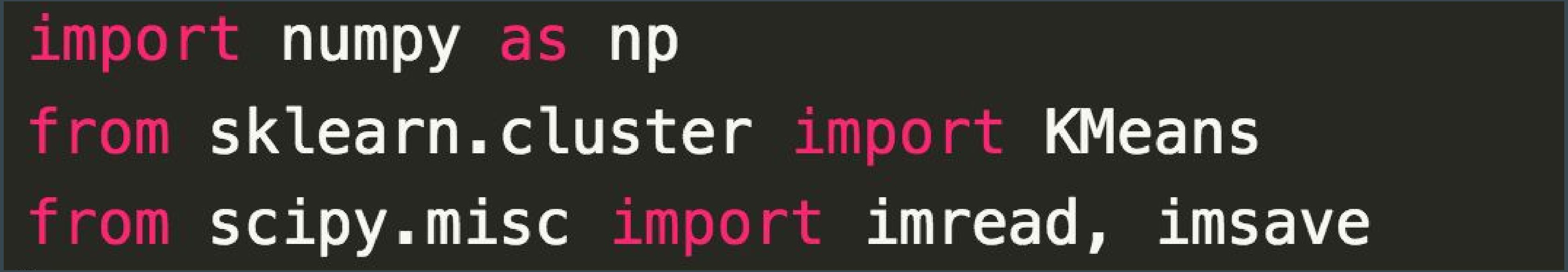
After



Dependencies

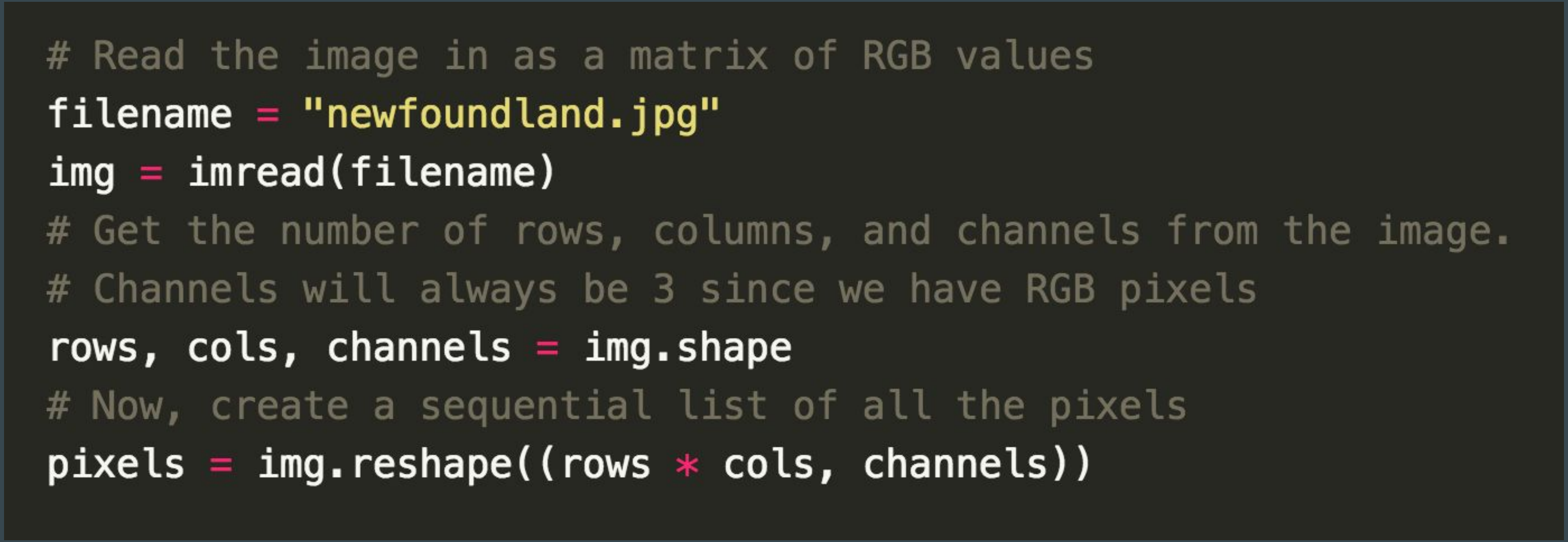
- You’ll need the following libraries: numpy, scipy, matplotlib, scikit-learn
- Easiest way is to install via `pip` through terminal:
    - `pip install numpy scipy matplotlib scikit-learn`

Import Dependencies



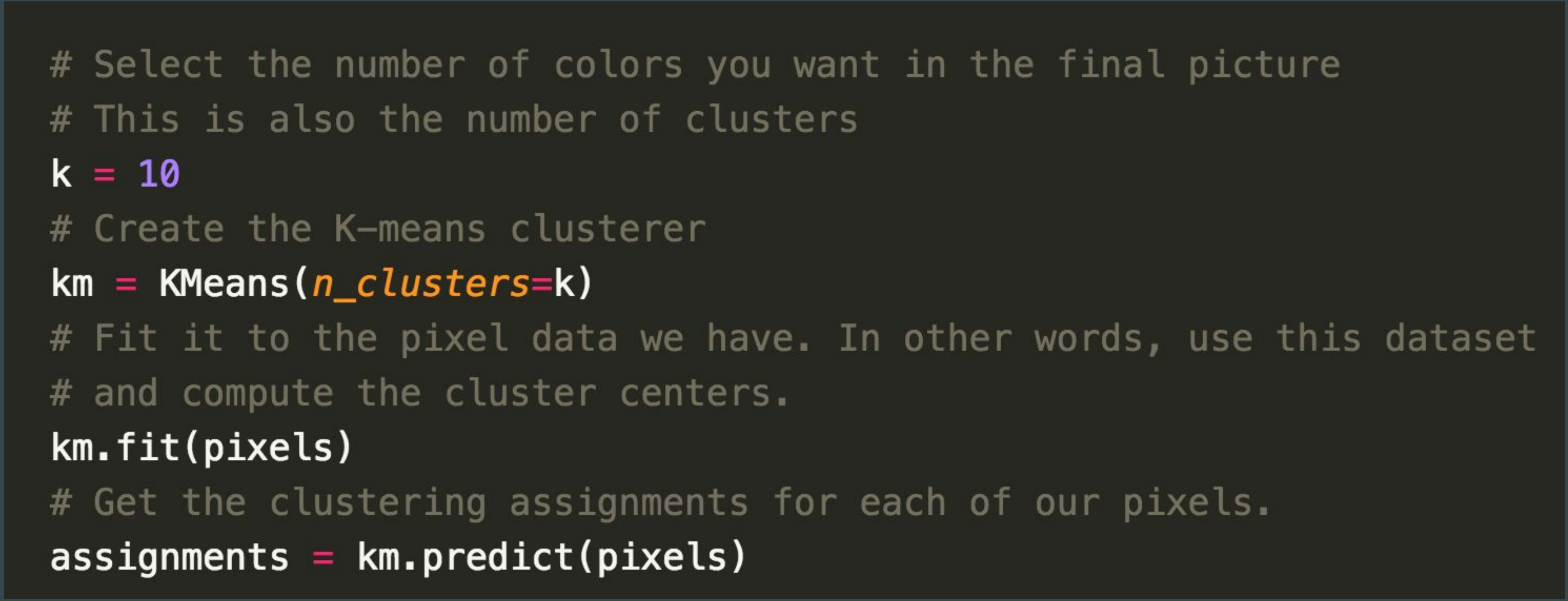
```python
import numpy as np
from sklearn.cluster import KMeans
from scipy.misc import imread, imsave
```

Loading An Image and Extract the Pixel Values



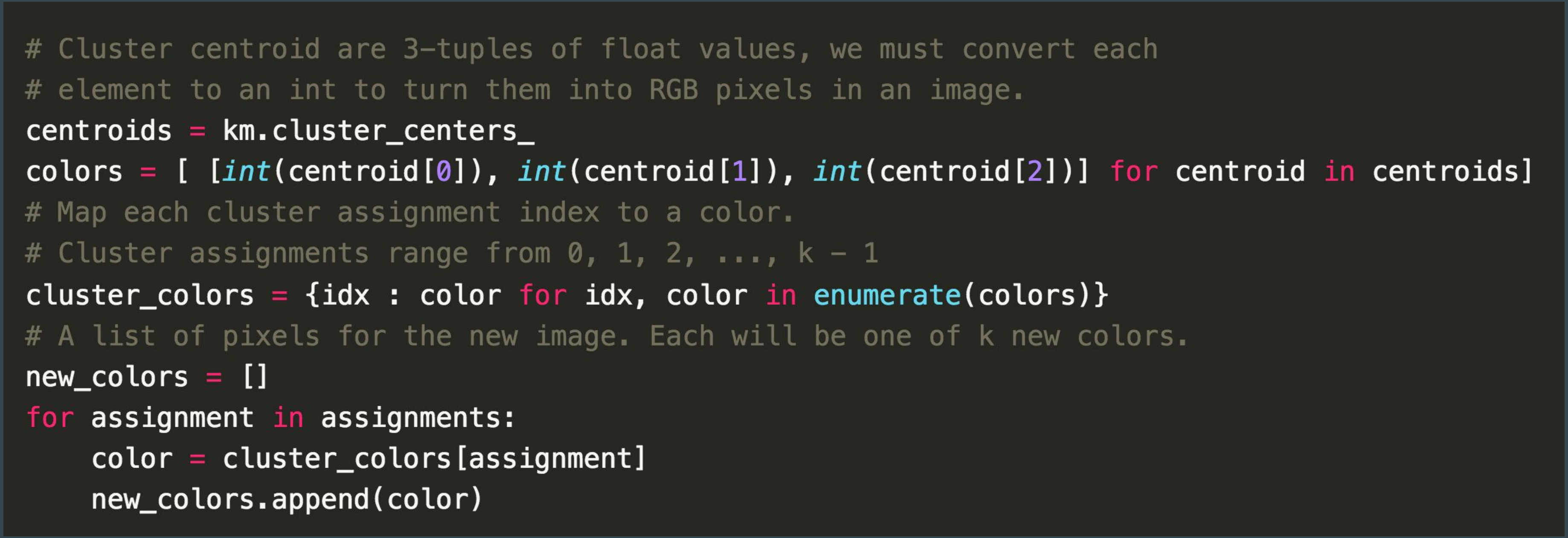
```
# Read the image in as a matrix of RGB values
filename = "newfoundland.jpg"
img = imread(filename)
# Get the number of rows, columns, and channels from the image.
# Channels will always be 3 since we have RGB pixels
rows, cols, channels = img.shape
# Now, create a sequential list of all the pixels
pixels = img.reshape((rows * cols, channels))
```

Begin Clustering



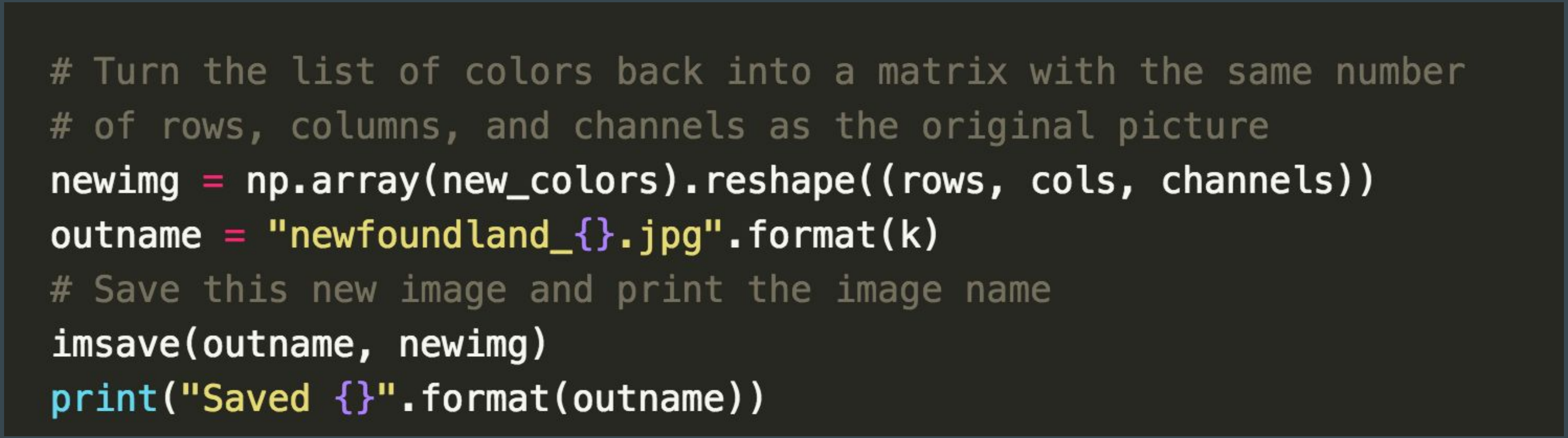
```
# Select the number of colors you want in the final picture
# This is also the number of clusters
k = 10
# Create the K-means clusterer
km = KMeans(n_clusters=k)
# Fit it to the pixel data we have. In other words, use this dataset
# and compute the cluster centers.
km.fit(pixels)
# Get the clustering assignments for each of our pixels.
assignments = km.predict(pixels)
```

Map Clustering Assignments to New Colors



```python
# Cluster centroid are 3-tuples of float values, we must convert each
# element to an int to turn them into RGB pixels in an image.
centroids = km.cluster_centers_
colors = [ [int(centroid[0]), int(centroid[1]), int(centroid[2])] for centroid in centroids]
# Map each cluster assignment index to a color.
# Cluster assignments range from 0, 1, 2, ..., k - 1
cluster_colors = {idx : color for idx, color in enumerate(colors)}
# A list of pixels for the new image. Each will be one of k new colors.
new_colors = []
for assignment in assignments:
    color = cluster_colors[assignment]
    new_colors.append(color)
```

Save the New Image



```
# Turn the list of colors back into a matrix with the same number
# of rows, columns, and channels as the original picture
newimg = np.array(new_colors).reshape((rows, cols, channels))
outname = "newfoundland_{}.jpg".format(k)
# Save this new image and print the image name
imsave(outname, newimg)
print("Saved {}".format(outname))
```

All Done

- To run the script, simply run the following:
    - `python script.py`