Hands-On with Unsupervised Learning

A quick tutorial on k-means clustering and principal component analysis (PCA).

Marco Peixeiro Follow
Jan 31 · 4 min read

Photo by Ryoji Iwata on Unsplash

In a previous post, unsupervised learning was introduced as a set of statistical tools for scenarios in which there is a set of features, but no targets. Therefore, this tutorial will be different than other ones, since we will not be able to make predictions.

Instead, we will work with k-means clustering to perform **color quantization** on an image.

Then, we will use PCA for dimensionality reduction and visualization of a dataset.

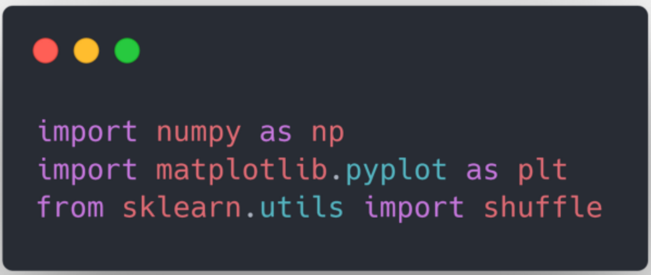
The full notebook is available here.

Spin up your Jupyter notebook, and let's go!

Unlike a chainsaw, you can use this tutorial unsupervised

Setup

Before starting on any implementation, we will import a few libraries that will become handy later on:



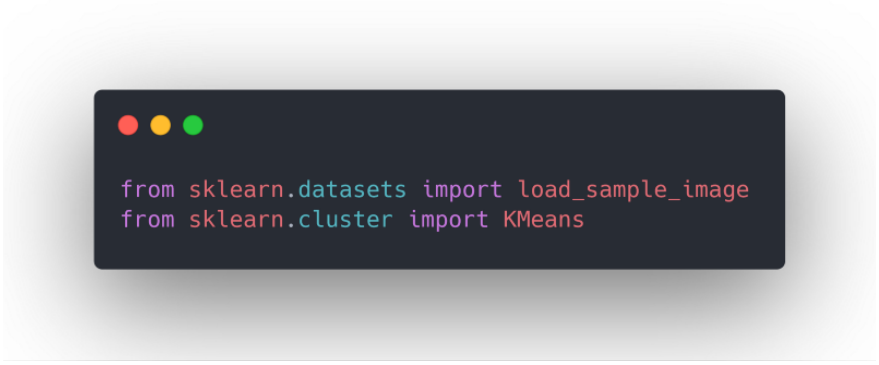
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
```

Unlike previous tutorials, we will not import datasets. Instead, we will use data provided by the *scikit-learn* library.

Color quantization—k-means clustering

Quickly, color quantization is technique to reduce the number of distinct colors used in an image. This is especially useful to compress images while keeping the integrity of the image.

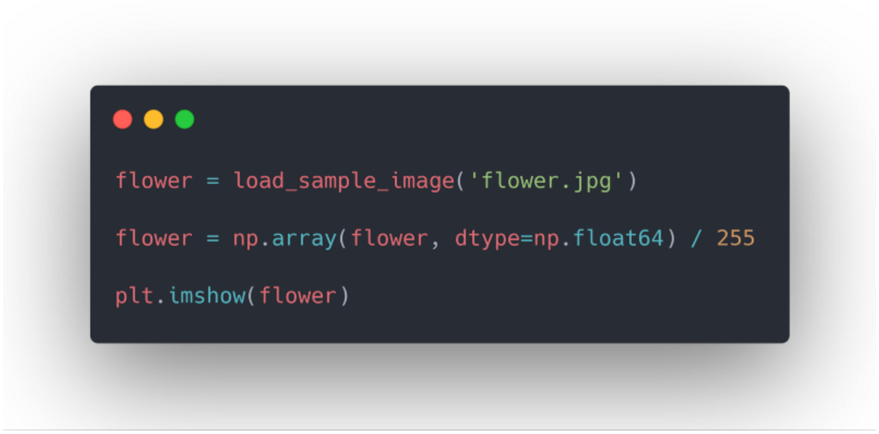
To get started, we import the following libraries:



```python
from sklearn.datasets import load_sample_image
from sklearn.cluster import KMeans
```

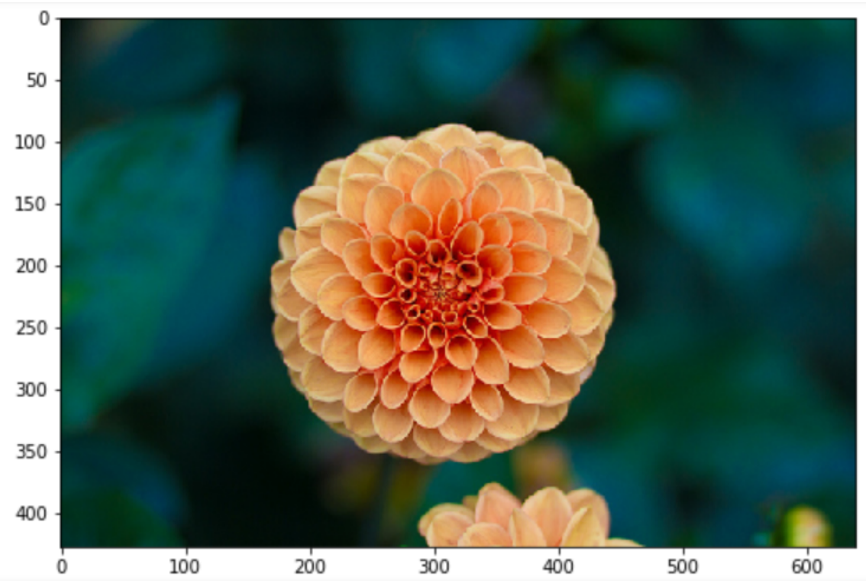
Notice that we import a sample dataset called *load_sample_image*. This simply contains two images. We will use one of them to perform color quantization.

So, let's show the image we will use for this exercise:



```python
flower = load_sample_image('flower.jpg')

flower = np.array(flower, dtype=np.float64) / 255

plt.imshow(flower)
```

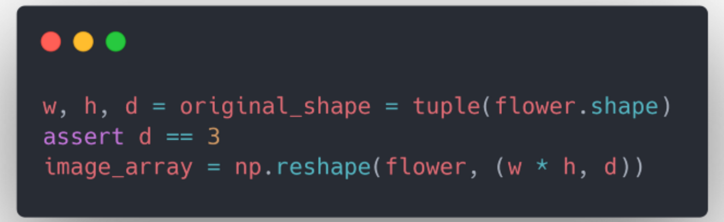
And you should see:



Original image

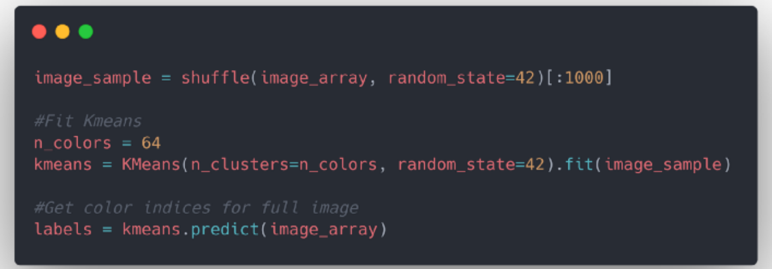
Now, for color quantization, different steps must be followed.

First, we need to change the image into a 2D matrix for manipulation:



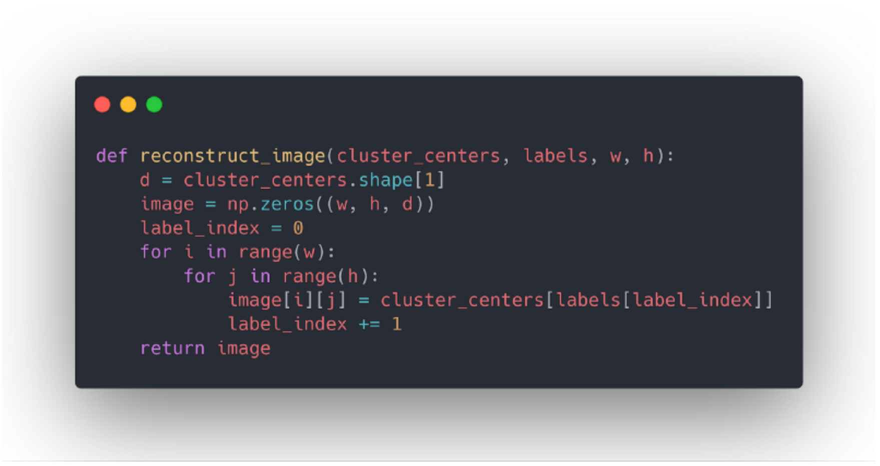
```
w, h, d = original_shape = tuple(flower.shape)
assert d == 3
image_array = np.reshape(flower, (w * h, d))
```

Then, we train our model to aggregate colors in order to have 64 distinct colors in the image:



```
image_sample = shuffle(image_array, random_state=42)[:1000]

#Fit Kmeans
n_colors = 64
kmeans = KMeans(n_clusters=n_colors, random_state=42).fit(image_sample)

#Get color indices for full image
labels = kmeans.predict(image_array)
```

Then, we build a helper function to help us reconstruct the image with the number of specified colors:



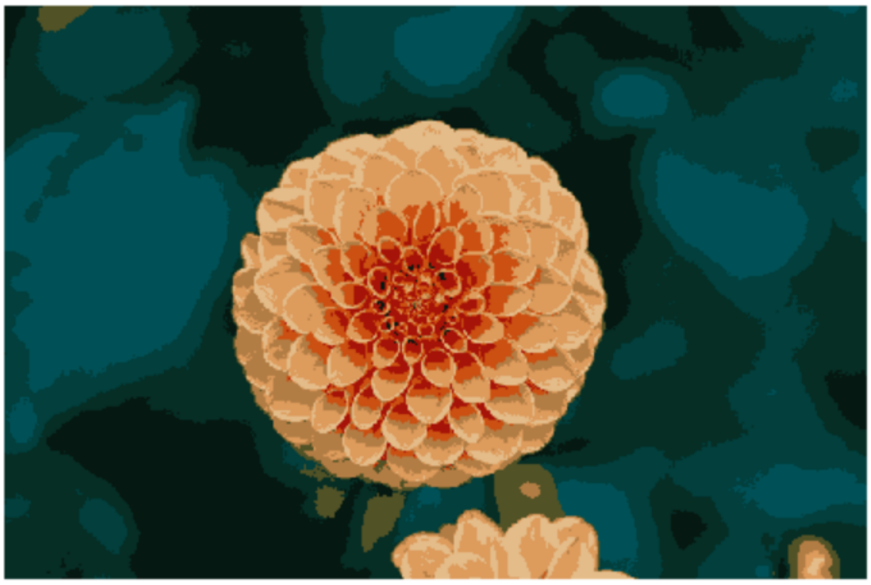
```python
def reconstruct_image(cluster_centers, labels, w, h):
    d = cluster_centers.shape[1]
    image = np.zeros((w, h, d))
    label_index = 0
    for i in range(w):
        for j in range(h):
            image[i][j] = cluster_centers[labels[label_index]]
            label_index += 1
    return image
```

Finally, we can now visualize how the image looks with only 64 colors, and how it compares to the original one:

Original image with 96 615 colors

Reconstructed image with 64 colors

Of course, we can see some differences, but overall, the integrity of the image is conserved! Do explore different number of clusters! For example, here is what you get if you specify 10 colors:

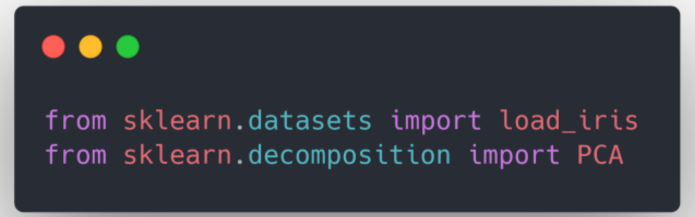


Reconstructed image with 10 colors

Dimensionality reduction — PCA

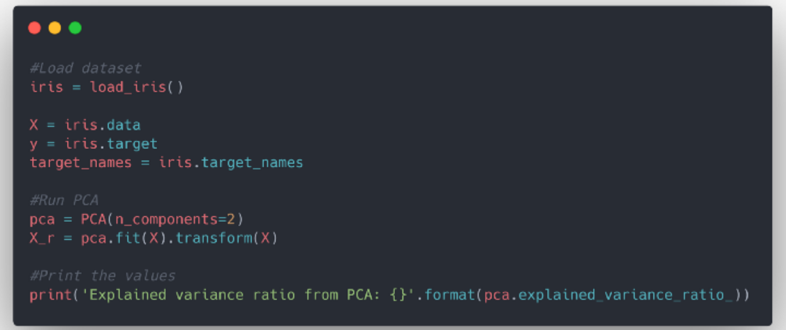
For this exercise, we will use PCA to reduce the dimensions of a dataset so we can easily visualize it.

Therefore, let's import the iris dataset from *scikit-learn*:



```python
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
```

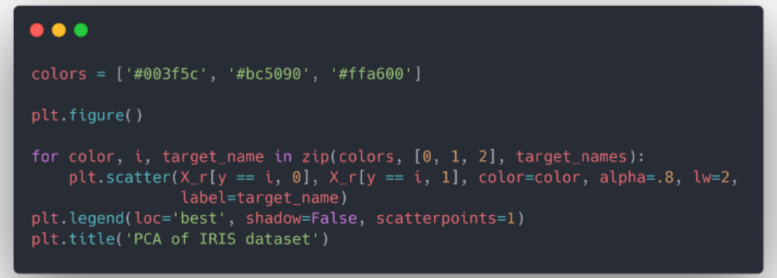
Now, we will compute the first two principal components and see what proportion of the variance can be explained by each:



```python
#Load dataset
iris = load_iris()

X = iris.data
y = iris.target
target_names = iris.target_names

#Run PCA
pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)

#Print the values
print('Explained variance ratio from PCA: {}'.format(pca.explained_variance_ratio_))
```

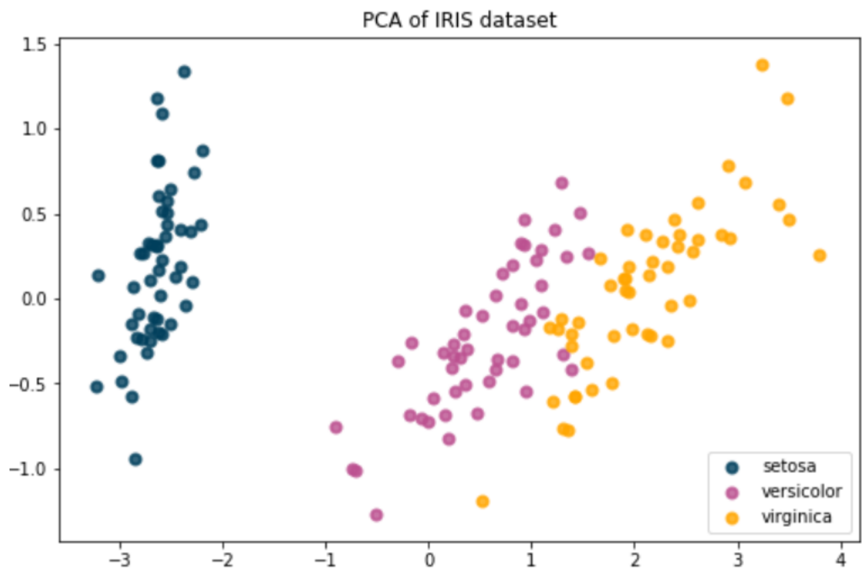
From the above code block, you should see that the first principal component contains 92% of the variance, while the second accounts for 5% of the variance. Therefore, this means that only two features are sufficient to explain 97% of the variance in the dataset!

Now, we can use this to easily plot the data in two dimensions:



```python
colors = ['#003f5c', '#bc5090', '#ffa600']

plt.figure()

for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_r[y == i, 0], X_r[y == i, 1], color=color, alpha=.8, lw=2,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA of IRIS dataset')
```

And you get:



. . .

That's it! You now know how to implement k-means and PCA! Again, keep in mind that unsupervised learning is hard, because there is no error metric to evaluate how well the algorithm performed. Also, these techniques are usually used in exploratory data analysis prior to making supervised learning.

Leave me a comment to ask a question or to tell me how to improve!

Keep working hard!

> *These exercises were examples available on the* scikit-learn *website. I simply tried to explain them with more details.*