Import Modules

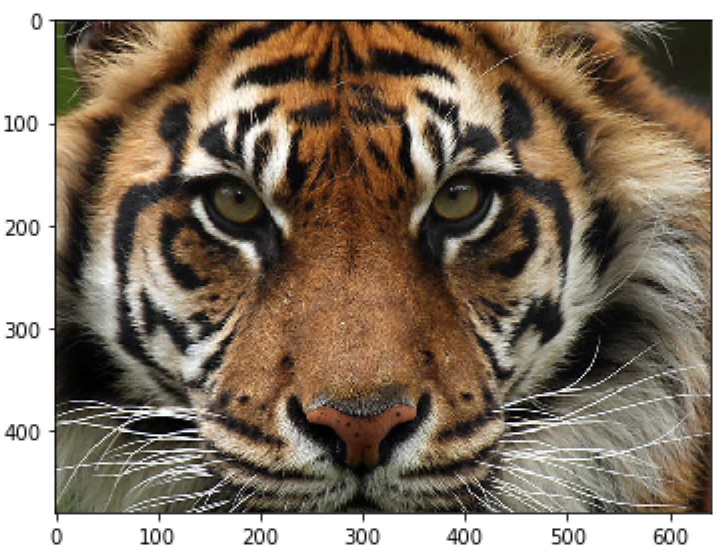
In [1]:

```
import os
import random
import numpy as np
import numpy.matlib

from skimage import io
```

Show original image

In [2]:

```
original_image = io.imread('../images/before-compression/Sumatran-Tiger-Hero.jpg')

io.imshow(original_image)
io.show()
```



Get width and length from image

In [3]:

```
length         = original_image.shape[0]
width          = original_image.shape[1]
original_image = original_image / 255

X = original_image.reshape(original_image.shape[0]*original_image.shape[1], 3)
```

Determine Cluster and Max Iteration

In [4]:

```
number_of_cluster = 16
max_iters         = 50
```

Function Initialize centroid

In [5]:

```
def init_centroids(X, number_of_clusters):
    c = random.sample(list(X), number_of_clusters)

    return c
```

Determine closest centroid for each datapoint (pixel)

In [6]:

```
def closest_centroids(X, c):
    number_of_cluster = np.size(c,0)
    idx               = np.zeros((np.size(X,0),1))
    arr               = np.empty((np.size(X,0),1))

    for i in range(0, number_of_cluster):
        y    = c[i]
        temp = np.ones((np.size(X,0),1))*y
        b    = np.power(np.subtract(X,temp),2)
        a    = np.sum(b,axis = 1)
        a    = np.asarray(a)

        a.resize((np.size(X,0),1))
        arr  = np.append(arr, a, axis=1)

    arr = np.delete(arr,0,axis=1)
    idx = np.argmin(arr, axis=1)

    return idx
```

Determine Centroid

In [7]:

```
def compute_centroids(X, idx, number_of_cluster):
    n = np.size(X, 1)
    centroids = np.zeros((number_of_cluster, n))

    for i in range(0, number_of_cluster):
        ci           = idx==i
        ci           = ci.astype(int)
        total_number = sum(ci)

        ci.resize((np.size(X,0),1))

        total_matrix = np.matlib.repmat(ci,1,n)
        ci           = np.transpose(ci)
        total        = np.multiply(X,total_matrix)
        centroids[i] = (1 / total_number) * np.sum(total, axis=0)

    return centroids
```

KMeans Clustering

In [8]:

```
def run_kMean(X, initial_centroids, max_iters):
    m                  = np.size(X, 0)
    n                  = np.size(X, 1)
    print(m)
    print(n)
    number_of_cluster  = np.size(initial_centroids, 0)
    centroids          = initial_centroids
    previous_centroids = centroids
    idx                = np.zeros((m,1))


    for i in range(1,max_iters):
        idx = closest_centroids(X, centroids)
        centroids = compute_centroids(X, idx, number_of_cluster)

    return centroids,idx
```

Generate Centroid

In [9]:

```
initial_centroids = init_centroids(X, number_of_cluster)
centroids, idx    = run_kMean(X, initial_centroids, max_iters)

print(np.shape(centroids))
print(np.shape(idx))
```

```
307200
3
(8, 3)
(307200,)
```

Generate Compression Image

In [10]:

```
idx         = closest_centroids(X, centroids)
X_recovered = centroids[idx]
X_recovered = np.reshape(X_recovered, (length, width, 3))

print(np.shape(X_recovered))
print(np.shape(X_recovered))
```

```
(480, 640, 3)
(480, 640, 3)
```

Show image compressed

In [11]:

```
image_compressed = io.imsave('../images/after-compression/Sumatran-Tiger-Hero-Result.
```

```
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to
saving to suppress this warning.
```

In [12]:

```
image_compressed = io.imread('../images/after-compression/Sumatran-Tiger-Hero-Result.

io.imshow(image_compressed)
io.show()
```

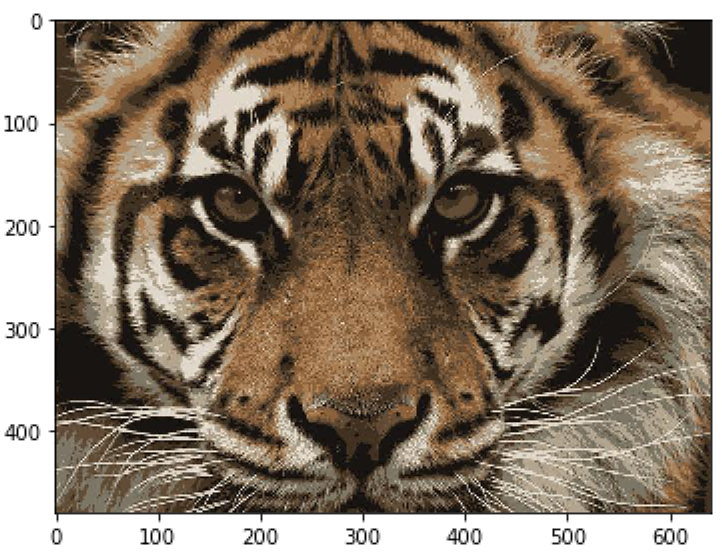


Image size before and after compression with KMeans

In [13]:

```
size_before_compression = os.stat('../images/before-compression/Sumatran-Tiger-Hero.j
size_after_compression = os.stat('../images/after-compression/Sumatran-Tiger-Hero-Resu

print("Size of image before running K-mean algorithm: ", size_before_compression.st_s
print("Size of image after running K-mean algorithm: ", size_after_compression.st_size
```

```
Size of image before running K-mean algorithm:  90.515625 KB
Size of image after running K-mean algorithm:  83.76953125 KB
```