Machine Learning Lab 6

Альромхин Джорж, гр.858301

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io
```

Загружаем данные ex6data1.mat из файла.

In [3]:
```
data = scipy.io.loadmat('../Data/lab6/ex6data1.mat')
X = data['X']
```

Реализация функции случайной инициализации K центров кластеров.

In [4]:
```
def rand_centroids(K, X):
    rand_indices = np.arange(len(X))
    np.random.shuffle(rand_indices)
    centroids = X[rand_indices][:K]
    return centroids
```

Реализация функции определения принадлежности к кластерам.

In [7]:
```
def find_closest_centroids(X, centroids):
    distances = np.array([np.sqrt((X.T[0] - centroid[0])**2 + (X.T[1] -
    return distances.argmin(axis=0)
```

Реализация функции пересчета центров кластеров.

In [22]:
```
def compute_means(X, centroid_inds, K):
    centroids = []
    for k in range(K):
        t = X[centroid_inds == k]
        if t.size > 0:
            centroids.append(np.mean(t, axis=0))
        else:
            centroids.append(np.zeros((X.shape[1],)))

    return np.array(centroids)
```

Реализализация алгоритма K-средних.

In [9]:
```
def run_k_means(X, K, num_iterations=10):
    centroids = rand_centroids(K, X)
    centroids_history = [centroids]
    for i in range(num_iterations):
        centroid_indices = find_closest_centroids(X, centroids)
        centroids = compute_means(X, centroid_indices, K)
        centroids_history.append(centroids)

    return centroids, centroid_indices, centroids_history
```

In [10]:
```
def k_means_distortion(X, centroids, idx):
    K = centroids.shape[0]
    distortion = 0
    for i in range(K):
        distortion += np.sum((X[idx == i] - centroids[i])**2)
    distortion /= X.shape[0]
    return distortion
```

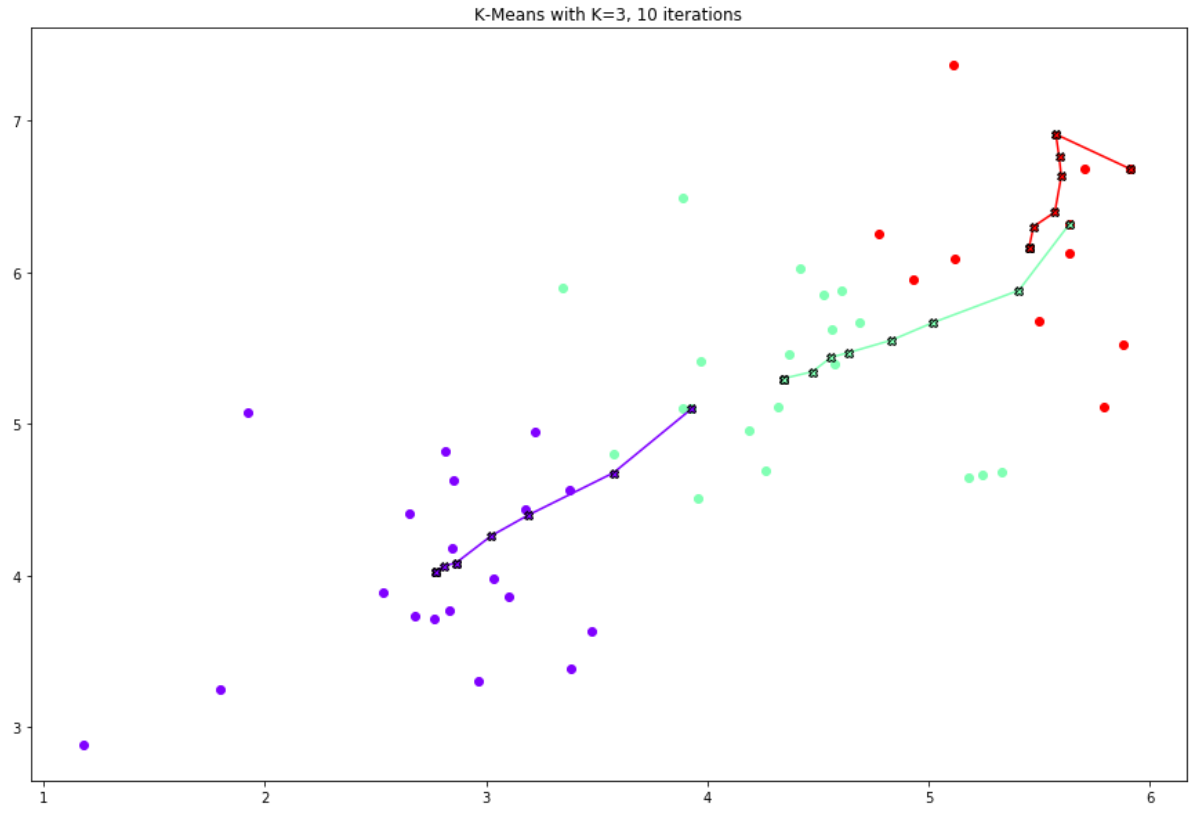
In [11]:
```
def find_best_k_means(X, K, num_iterations=100):
    result = np.inf
    r_centroids = None
    r_idx = None
    r_history = None
    for i in range(num_iterations):
        centroids, idx, history = run_k_means(X, K)
        d = k_means_distortion(X, centroids, idx)
        if d < result:
            print(f'K-Means found solution with distortion: {d}')
            r_centroids = centroids
            r_idx = idx
            r_history = history
            result = d

    return r_centroids, r_idx, r_history
```

График, на котором данные разделены на K=3 кластеров (при помощи различных маркеров или цветов), а также траекторию движения центров кластеров в процессе работы алгоритма.

In [20]:
```
import matplotlib.cm as cm

def plot_k_means(X, K, centroid_inds, centroids_history):
    plt.figure(figsize=(15,10))
    colors = cm.rainbow(np.linspace(0, 1, K))
    for k in range(K):
        plt.scatter(X[centroid_inds == k][:, 0], X[centroid_inds == k][:

    for i in range(K):
        vals = [centroids_points[i] for centroids_points in centroids_hi
        vals = np.array(vals)
        plt.plot(vals[:, 0], vals[:, 1], '-Xk', c=colors[i], markeredgec

    plt.title(f'K-Means with K={K}, {len(centroids_history)-1} iteratior
    plt.show()
```

In [21]:
```
K = 3
centroids, idx, history = find_best_k_means(X, K)
plot_k_means(X, K, idx, history)
```

```
K-Means found solution with distortion: 0.6126621462763979
K-Means found solution with distortion: 0.5927612624586158
K-Means found solution with distortion: 0.5927612624586157
```



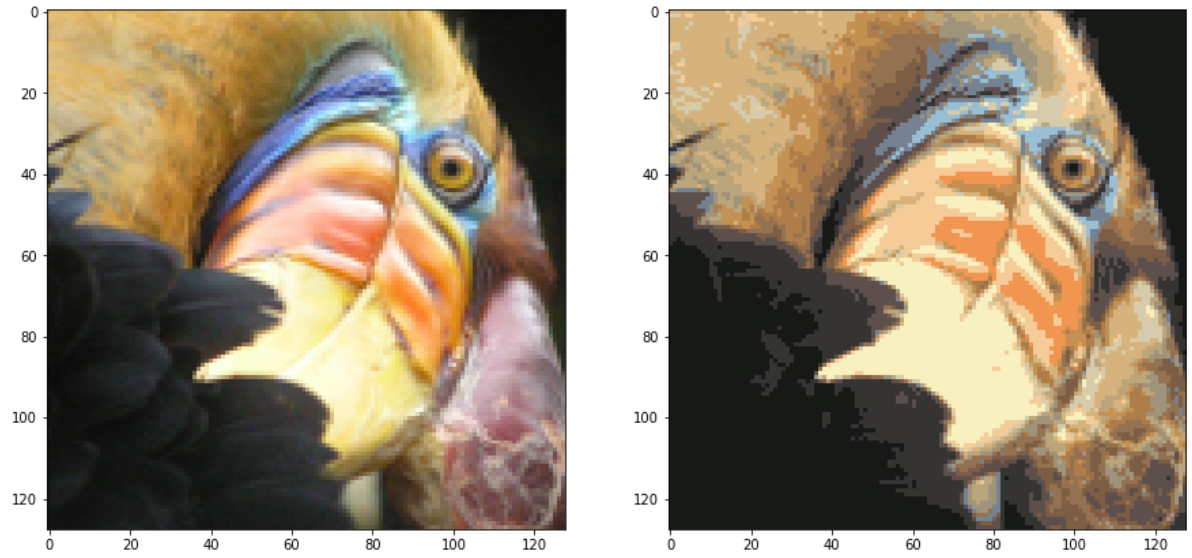
Загружаем данные bird_small.mat из файла.

In [23]:
```
img = scipy.io.loadmat('../Data/lab6/bird_small.mat')
A = np.reshape(img['A'], newshape=(-1, 3))
A = A.astype('float') / 255.0
```

С помощью алгоритма K-средних сжимаем картинку используя 16 цветов для кодирования пикселей.

In [27]:
```
K = 16
centroids, idx, _ = find_best_k_means(A, K)
A_recon = centroids[idx]
A_recon = A_recon.reshape(-1, 128, 3)
fig, axs = plt.subplots(1, 2, figsize=(15, 10))
axs.flat[0].imshow(img['A'])
axs.flat[1].imshow(A_recon)
plt.show()
```

```
K-Means found solution with distortion: 0.016891302207850418
K-Means found solution with distortion: 0.014941277276664229
K-Means found solution with distortion: 0.014298920142259376
K-Means found solution with distortion: 0.014085096261041737
K-Means found solution with distortion: 0.013890190065658212
```

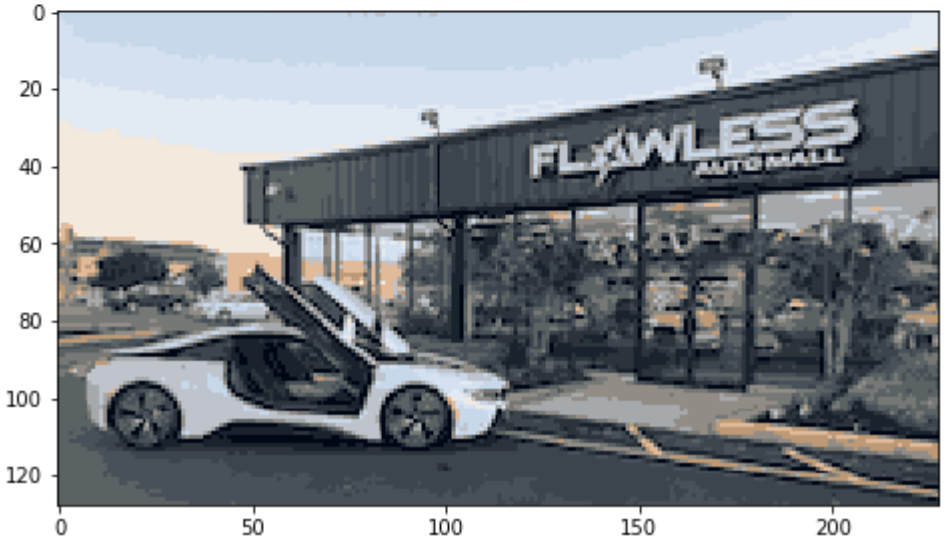


Оригинальная картинка использует 24 бита для каждого из 128 x 128 пикселей, полный размер данных 128 x 128 x 24 = 393,216 бит. Восстановленное изображение использует 16 цветов, каждый использует 24 бита, но сама картинка использует 4 бит на пиксель. Общий размер данных получается 16 x 24 + 128 x 128 x 4 = 65,920 бит, получается оригинальная картинка сжимается где-то в 6 раз.

Реализуем алгоритм K-средних на другом изображении.

In [36]:
```
import matplotlib.image as mpimg

data = mpimg.imread('test.png')
data = data[:, :, :3]
A = np.reshape(data, newshape=(-1, 3))
K = 16
centroids, idx, _ = find_best_k_means(A, K)
A_recon = centroids[idx]
A_recon = A_recon.reshape(-1, data.shape[1], 3)
fig, axs = plt.subplots(2, 1, figsize=(15, 10))
axs.flat[0].imshow(data)
axs.flat[1].imshow(A_recon)
plt.show()
```

```
K-Means found solution with distortion: 0.007162145008087943
K-Means found solution with distortion: 0.006366840743443422
K-Means found solution with distortion: 0.006155746711189286
K-Means found solution with distortion: 0.006093013406682171
K-Means found solution with distortion: 0.006038224308245015
```



Реализуем алгоритм иерархической кластеризации на том же изображении. Сравните полученные результаты.

In [37]:
```
from sklearn.cluster import AgglomerativeClustering

data = mpimg.imread('test.png')
data = data[:, :, :3]
A = np.reshape(data, newshape=(-1, 3))
K = 16
clustering = AgglomerativeClustering(n_clusters=K).fit(A)
idx = clustering.labels_
centroids = compute_means(A, idx, K)
A_recon = centroids[idx]
A_recon = A_recon.reshape(-1, data.shape[1], 3)
fig, axs = plt.subplots(2, 1, figsize=(15, 10))
axs.flat[0].imshow(data)
axs.flat[1].imshow(A_recon)
plt.show()
```