Лабораторная работа №7 “Метод главных компонент”

Лабораточная работа выполнена на языке **Python** с помощью интерактивной оболочки **Jupyter Notebook**. Исходный код работы - lab6.py. Файл jupyter notebook - lab6.ipynb.

Набор данных ex7data1.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит две переменные X1 и X2 - координаты точек, для которых необходимо выделить главные компоненты. Набор данных ex7faces.mat представляет собой файл формата* .mat (т.е. сохраненного из Matlab). Набор содержит 5000 изображений 32x32 в оттенках серого. Каждый пиксель представляет собой значение яркости (вещественное число). Каждое изображение сохранено в виде вектора из 1024 элементов. В результате загрузки набора данных должна быть получена матрица 5000x1024.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io
```

Загружаем данные ex7data1.mat из файла.

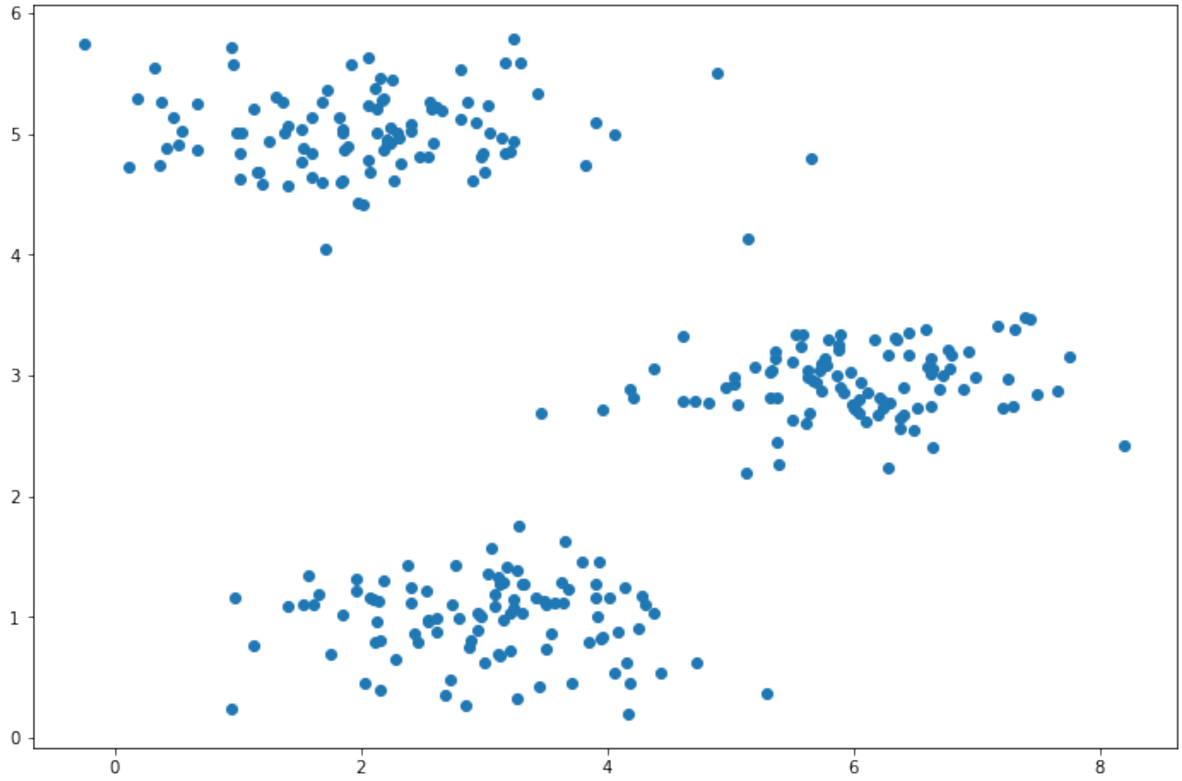
In [2]:

```
data = scipy.io.loadmat('ex7data1.mat')
X = data['X']
```

График загруженного набора данных.

In [22]:

```
plt.figure(figsize=(12, 8))
plt.scatter(X[:, 0], X[:, 1])
plt.show()
```



Реализация функции вычисления матрицы ковариации данных. Вычисляем координаты собственных векторов для набора данных с помощью сингулярного разложения матрицы ковариации используя библиотечную реализации матричных разложений numpy.linalg.svd.

In [4]:

```
def pca(X):
    m = X.shape[0]
    sigma = 1 / m * X.T.dot(X)
    U, S, V = np.linalg.svd(sigma)
    return U, S
```

In [5]:

```
def feature_normalization(X):
    norm = (X - X.mean(axis=0)) / X.std(axis=0)
    mu = X.mean(axis=0)
    sigma = X.std(axis=0)
    return norm, mu, sigma
```

In [9]:

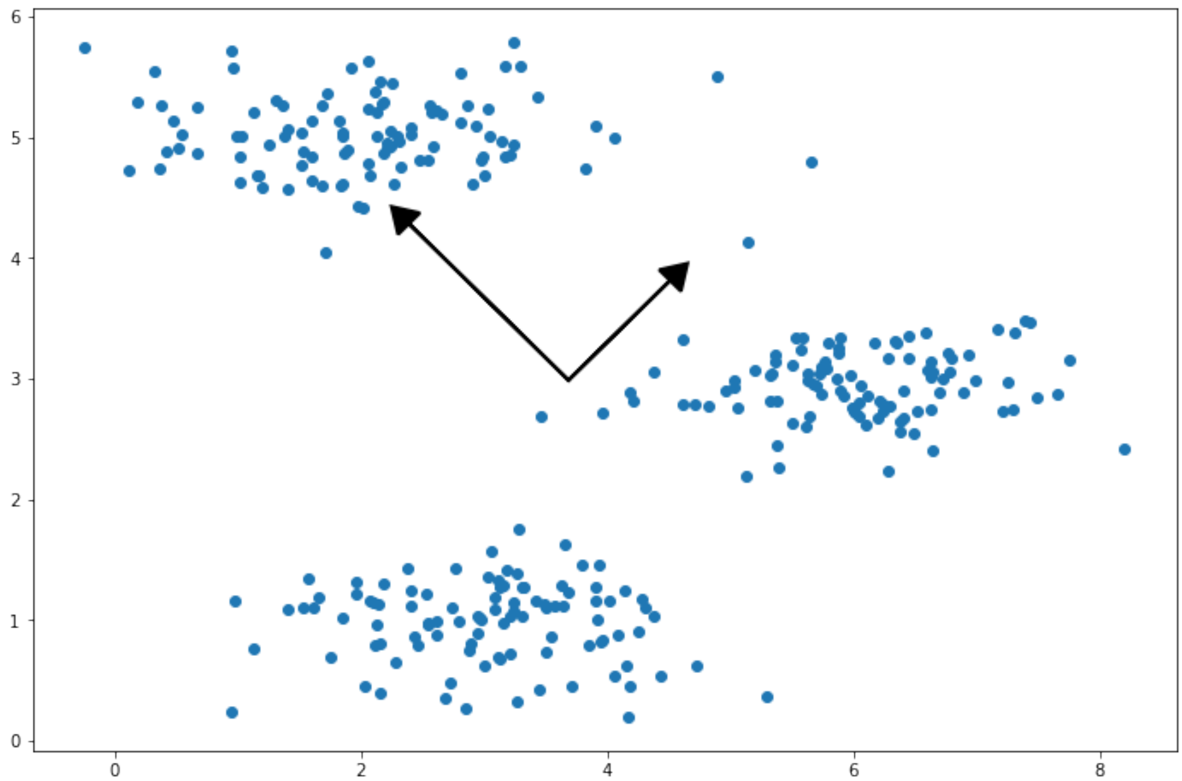
```
X_norm, mu, sig = feature_normalization(X)
U, S = pca(X_norm)
print(f'Top eigenvector: U[:, 0] = [{U[0, 0]:2.6}, {U[1, 0]:2.6}]')
```

```
Top eigenvector: U[:, 0] = [-0.707107, 0.707107]
```

Построенные на графике из пункта 2 собственные векторы матрицы ковариации.

In [21]:

```
plt.figure(figsize=(12, 8))
plt.scatter(X[:, 0], X[:, 1])
plt.arrow(mu[0], mu[1], 1.5 * S[0]*U[0, 0], 1.5 * S[0]*U[1, 0], head_width=0.2
5, head_length=0.2, fc='k', ec='k', lw=2, zorder=1000)
plt.arrow(mu[0], mu[1], 1.5 * S[1]*U[0, 1], 1.5 * S[1]*U[1, 1], head_width=0.2
5, head_length=0.2, fc='k', ec='k', lw=2, zorder=1000)
plt.show()
```



Реализация функци проекции из пространства большей размерности в пространство меньшей размерности с помощью метода главных компонент.

In [11]:

```
def project_data(X, U, K):
    return X.dot(U[:, :K])
```

Реализация функции вычисления обратного преобразования.

In [12]:

```
def recover_data(Z, U, K):
    return Z.dot(U[:, :K].T)
```

График исходных точек и их проекций на пространство меньшей размерности (с линиями проекций).

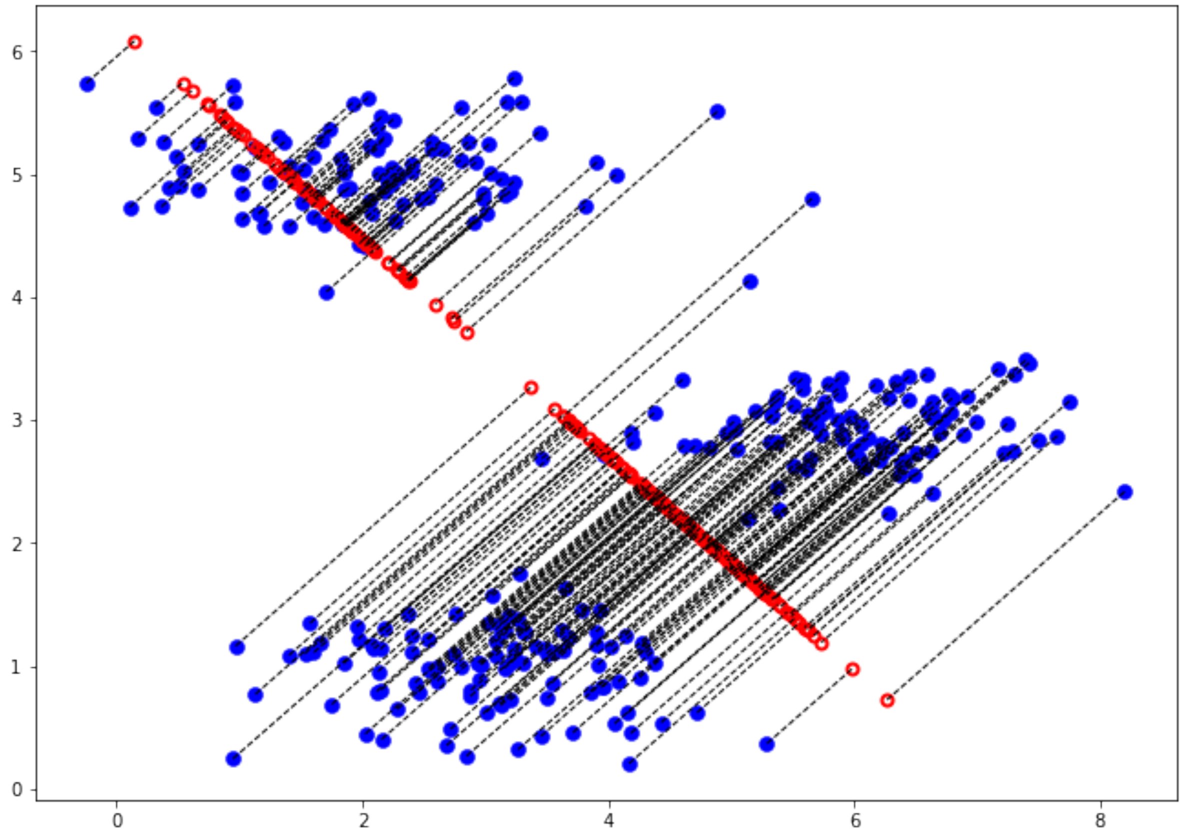
In [14]:

```
Z = project_data(X_norm, U, 1)
X_rec = recover_data(Z, U, 1)
X_rec = X_rec * sig + mu
print(f'Approximation of the first example: [{X_rec[0, 0]:2.6}, {X_rec[0, 1]:2
.6}]')
```

```
Approximation of the first example: [1.83735, 4.60343]
```

In [18]:

```
fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(X[:, 0], X[:, 1], 'bo', ms=8, mec='b', mew=0.5)
ax.set_aspect('equal')
ax.grid(False)
ax.plot(X_rec[:, 0], X_rec[:, 1], 'ro', mec='r', mew=2, mfc='none')
for xnorm, xrec in zip(X, X_rec):
    ax.plot([xnorm[0], xrec[0]], [xnorm[1], xrec[1]], '--k', lw=1)

plt.show()
```



Загружаем данные ex7faces.mat из файла.

In [23]:

```
data = scipy.io.loadmat('ex7faces.mat')
X = data['X']
```

Визуализируем 100 случайных изображений из набора данных.

In [39]:

```
def plot_faces(X):
    m, n = X.shape
    dim = int(np.sqrt(m))
    fig, axs = plt.subplots(dim, dim, figsize=(8, 8))
    fig.subplots_adjust(wspace=0.025, hspace=0.025)
    example_width = int(np.round(np.sqrt(n)))
    example_height = int(n / example_width)
    for i, ax in enumerate(axs.flat):
        ax.axis('off')
        w = int(np.sqrt(n))
        ax.imshow(X[i].reshape(example_height, example_width, order='F'), cmap
='gray')

    plt.show()
```

In [27]:

```
plot_faces(X[:100, :])
```

С помощью метода главных компонент вычисляем собственные векторы.

In [28]:

```
X_norm, mu, sigma = feature_normalization(X)
U, S = pca(X_norm)
```

Визуализируем 36 главных компонент с наибольшей дисперсией.

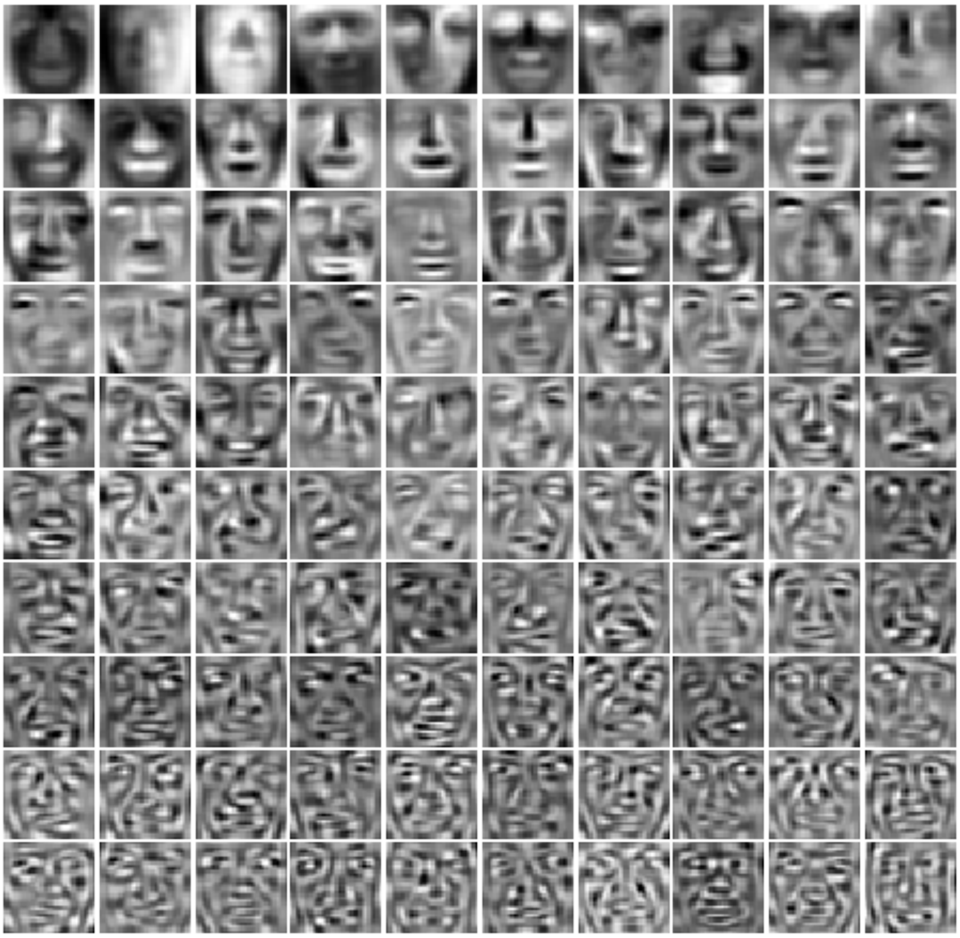
In [30]:

```
plot_faces(U[:, :36].T)
```

Визуализируем 100 главных компонент с наибольшей дисперсией.

In [31]:

```
plot_faces(U[:, :100].T)
```



Заметно как изменились изображения, они все менее похожи на лица.

In [33]:

```
K = 100
Z = project_data(X_norm, U, K)
X_rec = recover_data(Z, U, K)
```

In [40]:

```
plot_faces(X_norm[:100, :])
print('Original data')
```

Original data

In [41]:

```
plot_faces(X_rec[:100, :])
print('Reconstructed data')
```

```
Reconstructed data
```

Используя изображение, сжатое в лабораторной работе №5, с помощью метода главных компонент визуализируем данное изображение в 3D и 2D.

In [44]:

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

In [45]:

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

In [46]:

```
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)
```

In [50]:

```
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 [47]:

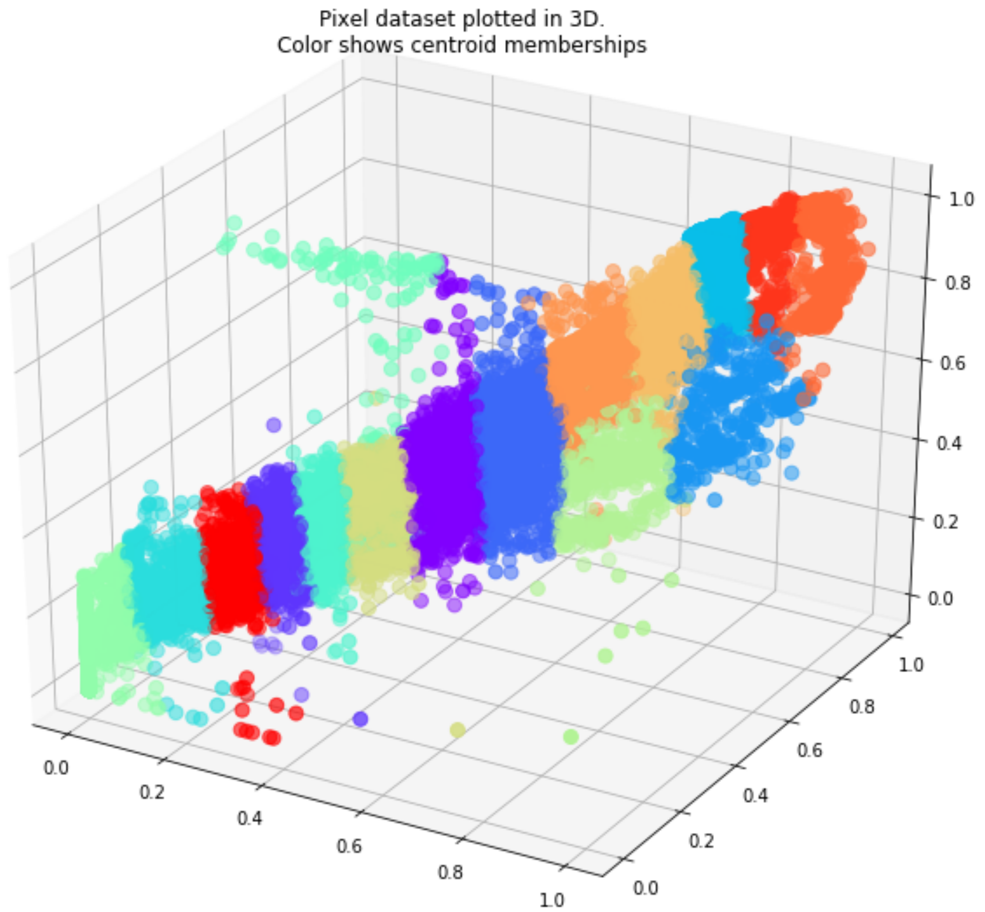
```
def run_k_means(X, K, num_iterations=10):
    centroids = rand_centroids(K, X)
    centroids_history = [centroids]
    for i in range(num_iterations):
        centroid_indices = find_closest_centroids(X, centroids)
        centroids = compute_means(X, centroid_indices, K)
        centroids_history.append(centroids)

    return centroids, centroid_indices, centroids_history
```

In [48]:

```
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:
            r_centroids = centroids
            r_idx = idx
            r_history = history
            result = d

    return r_centroids, r_idx, r_history
```

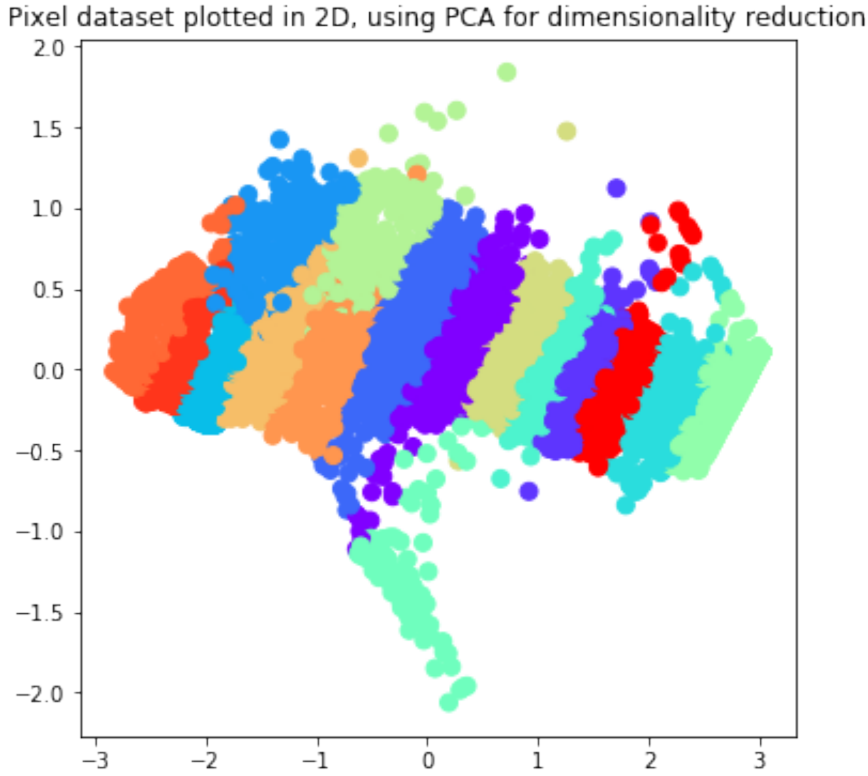
In [52]:

```
import matplotlib.image as mpimg
from mpl_toolkits.mplot3d import Axes3D

data = mpimg.imread('test.png')
data = data[:, :, :3]
X = np.reshape(data, newshape=(-1, 3))
K = 16
centroids, idx, _ = find_best_k_means(X, K)
X_rec = centroids[idx]
X_rec = X_rec.reshape(-1, data.shape[1], 3)

fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:, 0], X[:, 1], X[:, 2], cmap='rainbow', c=idx, s=8**2)
ax.set_title('Pixel dataset plotted in 3D.\nColor shows centroid memberships')
plt.show()
```



In [53]:

```
X_norm, mu, sig = feature_normalization(X)
U, S = pca(X_norm)
Z = project_data(X_norm, U, K)

fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111)
ax.scatter(Z[:, 0], Z[:, 1], cmap='rainbow', c=idx, s=64)
ax.set_title('Pixel dataset plotted in 2D, using PCA for dimensionality reduct
ion')
ax.grid(False)
plt.show()
```



По визуализации видно что изображение в 2D соответсвует проекции в 3D.

Вывод

В данной работе была показа работа метода главных компонент для уменьшения размерности данных, сжатия картинок, показан результат работы метода главных компонент в сравнении с методом K средних.