In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.sparse.linalg
import random
```

In [ ]:
```
chosen_numbers = [0, 5, 6, 9]
```

In [ ]:
```
data = pd.read_csv('./data.csv')
```

In [ ]:
```
print(data.shape)
data.head()
```

```
(42000, 785)
```

Out[ ]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel77
0	1	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

In [ ]:
```
A = np.array(data)
X_true = A[:, 1:].T
Y_true = A[:, 0]
```

In [ ]:
```
def get_chosen(X, Y, chosen_numbers = chosen_numbers):
    idx = [index for index, elem in enumerate(Y) if elem in chosen_numbers]
    X = X[:, idx]
    Y = Y[idx]
    return X, Y
```

In [ ]:
```
def x_split(X, Y, N_train):
    d, N = X.shape

    idx = np.arange(N)
    np.random.shuffle(idx)

    train_idx = idx[:N_train]
    test_idx = idx[N_train:]

    X_train = X[:, train_idx]
    Y_train = Y[train_idx]

    X_test = X[:, test_idx]
    Y_test = Y[test_idx]
```

```
    return X_train, X_test, Y_train, Y_test
```

In [ ]:

```
def PCA(X, k):
    c_X = np.mean(X, axis = 1)
    c_X = np.reshape(c_X, (len(c_X), 1))

    X_c = X - c_X
    U, _, _ = np.linalg.svd(X_c, full_matrices = False)

    U_k = U[:, :k]

    Z_k = U_k.T @ X

    return Z_k, U_k.T
```

In [ ]:

```
def c_k(X, Y, k):
    I = (Y == k)
    tmp_X = X[:, I]
    return np.mean(tmp_X, axis = 1)

def is_pos(X):
    return np.all(np.linalg.eigvals(X) > 0)

def LDA(X, Y, k, chosen_numbers = chosen_numbers):
    cs = []
    for num in chosen_numbers:
        c = c_k(X, Y, num)
        cs.append(np.reshape(c, (len(c), 1)))

    glob_c = np.mean(X, axis = 1)
    glob_c = np.reshape(glob_c, (len(glob_c), 1))

    Xcs = []
    for i, _ in enumerate(cs):
        num = chosen_numbers[i]
        c = cs[i]

        I = (Y == num)
        tmp_X = X[:, I]
        Xcs.append(tmp_X - c)

    X_w = np.concatenate([Xc for Xc in Xcs], axis=1)

    S_w = np.dot(X_w, X_w.T)

    gXs = []
    for c in cs:
        gXs.append(np.full(X.shape, c))

    gX = np.concatenate([gX for gX in gXs], axis=1)
    gX_c = gX - glob_c

    S_b = np.dot(gX_c, gX_c.T)

    L = []

    if is_pos(S_w):
        L = np.linalg.cholesky(S_w)
    else:
        tmp = S_w.copy()
```

```
        tmp += np.eye(S_w.shape[0])
        L = np.linalg.cholesky(tmp)

    L_inv = np.linalg.inv(L)
    simil_H = L_inv @ S_b @ L

    W = scipy.sparse.linalg.eigs(simil_H, k=k)[1]
    W = np.real(W)

    Q = L_inv.T @ W

    Z = Q.T @ X

    return Z, Q.T
```

In [ ]:

```
def avg_centroid_dist(Z, c, three_dim=False):
    tmp = []
    c = np.array(c)
    c = np.reshape(c, (len(c), 1))
    if three_dim:
        for Z_coord in zip(Z[0, :], Z[1, :], Z[2, :]):
            Z_coord = np.array(Z_coord)
            Z_coord = np.reshape(Z_coord, (len(Z_coord), 1))
            tmp.append(np.linalg.norm((Z_coord - c))**2)
    else:
        for Z_coord in zip(Z[0, :], Z[1, :]):
            Z_coord = np.array(Z_coord)
            Z_coord = np.reshape(Z_coord, (len(Z_coord), 1))
            tmp.append(np.linalg.norm((Z_coord - c))**2)
    return np.mean(tmp)
```

In [ ]:

```
def my_plot(X, Y, three_dim = False, c_dist = False, chosen_numbers = chosen_numbers
    fig = plt.figure(figsize = (12, 12))
    ax = fig.add_subplot()
    if three_dim:
        ax = fig.add_subplot(projection="3d")
        ax.scatter(X[0, :], X[1, :], X[2, :], c=Y)
    else:
        ax.scatter(X[0, :], X[1, :], c=Y)
    for c in chosen_numbers:
        ax.scatter(*(c_k(X, Y, c)), marker = "x", color="red")
        if c_dist:
            print(f"Average distance from centroid for digit {str(c)} = {avg_centroi
    plt.title(title)
    plt.show()
```

In [ ]:

```
X, Y = get_chosen(X_true, Y_true)
```

In [ ]:

```
X_train, X_test, Y_train, Y_test = x_split(X, Y, int(X.shape[1]*2/3))
```

In [ ]:

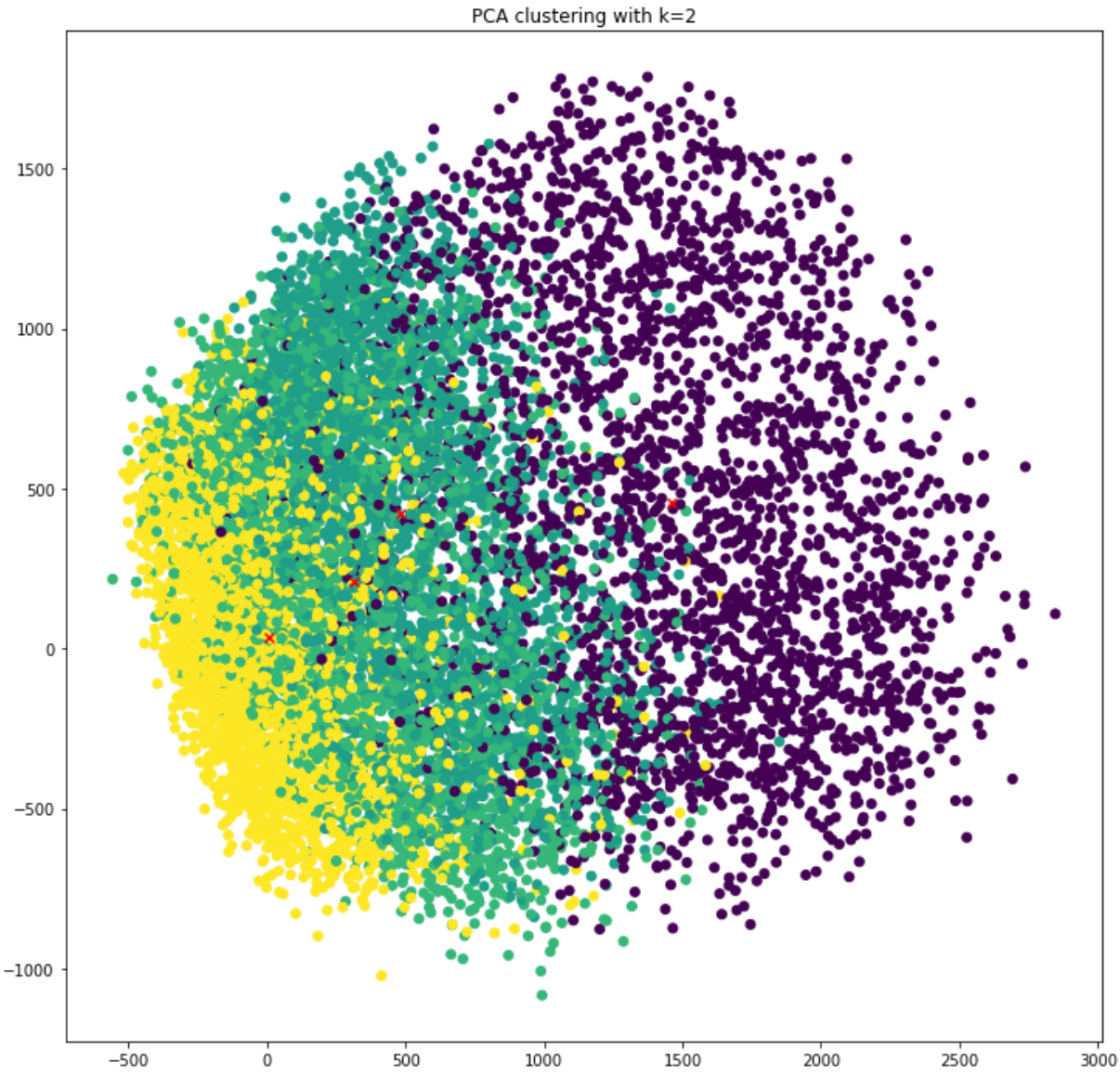
```
Z_pca, P_pca = PCA(X_train, 2)
Z_lda, P_lda = LDA(X_train, Y_train, 2)
```

In [ ]:

```
my_plot(Z_pca, Y_train, c_dist=True, title = "PCA clustering with k=2")
```

```
Average distance from centroid for digit 0 = 1598042.0775582837
```

```
Average distance from centroid for digit 5 = 792782.9317928992
Average distance from centroid for digit 6 = 831546.5778938432
Average distance from centroid for digit 9 = 1132791.3772359805
```



In [ ]:

```
my_plot(Z_lda, Y_train, c_dist=True, title = "LDA clustering with k=2")
```

```
Average distance from centroid for digit 0 = 0.00262146749255668
Average distance from centroid for digit 5 = 0.0013456414745566926
Average distance from centroid for digit 6 = 0.002616778404977008
Average distance from centroid for digit 9 = 0.0028289779661626047
```

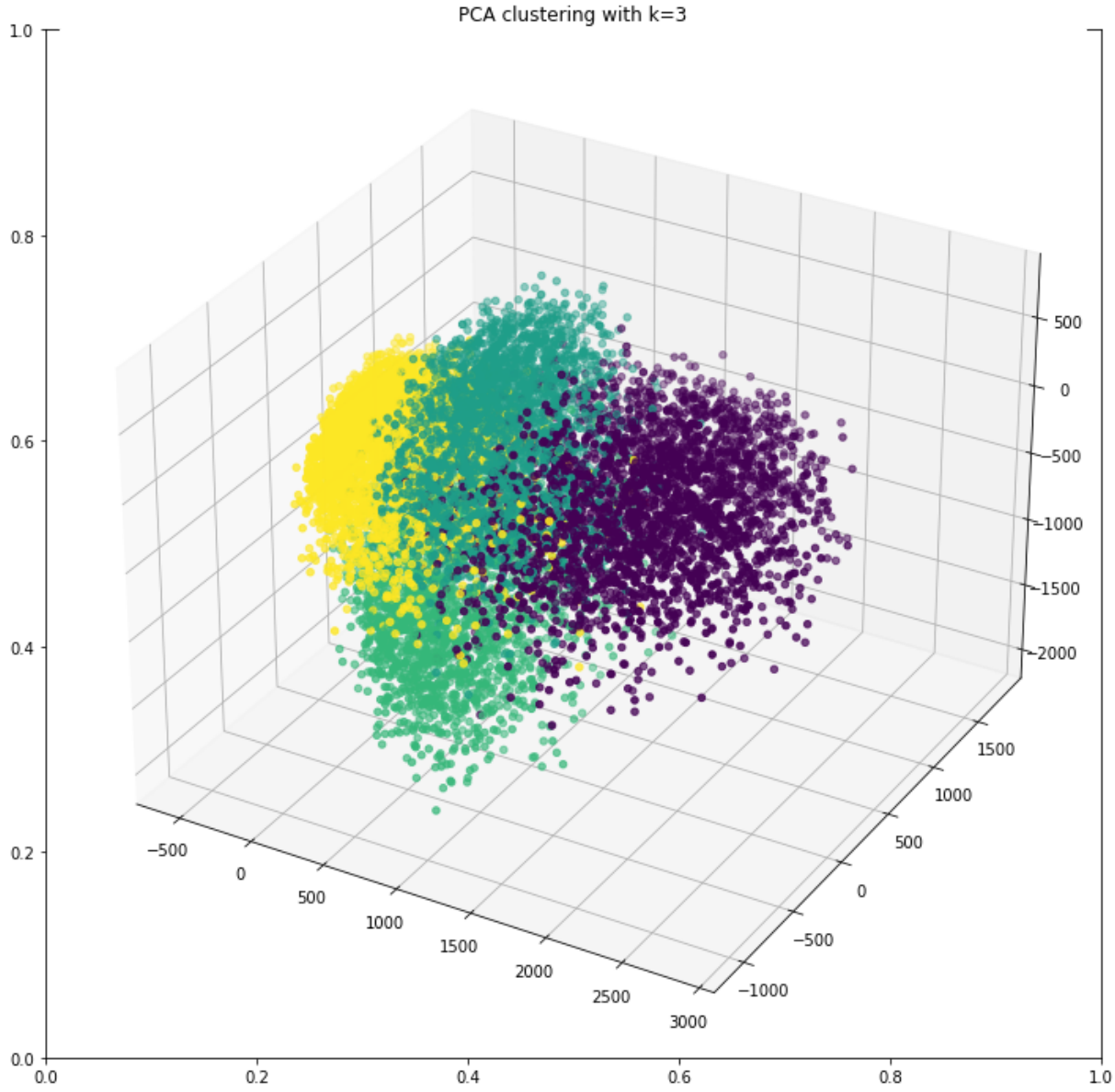
In [ ]:

```
Z_pca_3, P_pca_3 = PCA(X_train, k=3)
Z_lda_3, P_lda_3 = LDA(X_train, Y_train, k=3)
```

In [ ]:

```
my_plot(Z_pca_3, Y_train, c_dist=True, three_dim = True, title = "PCA clustering wit
```

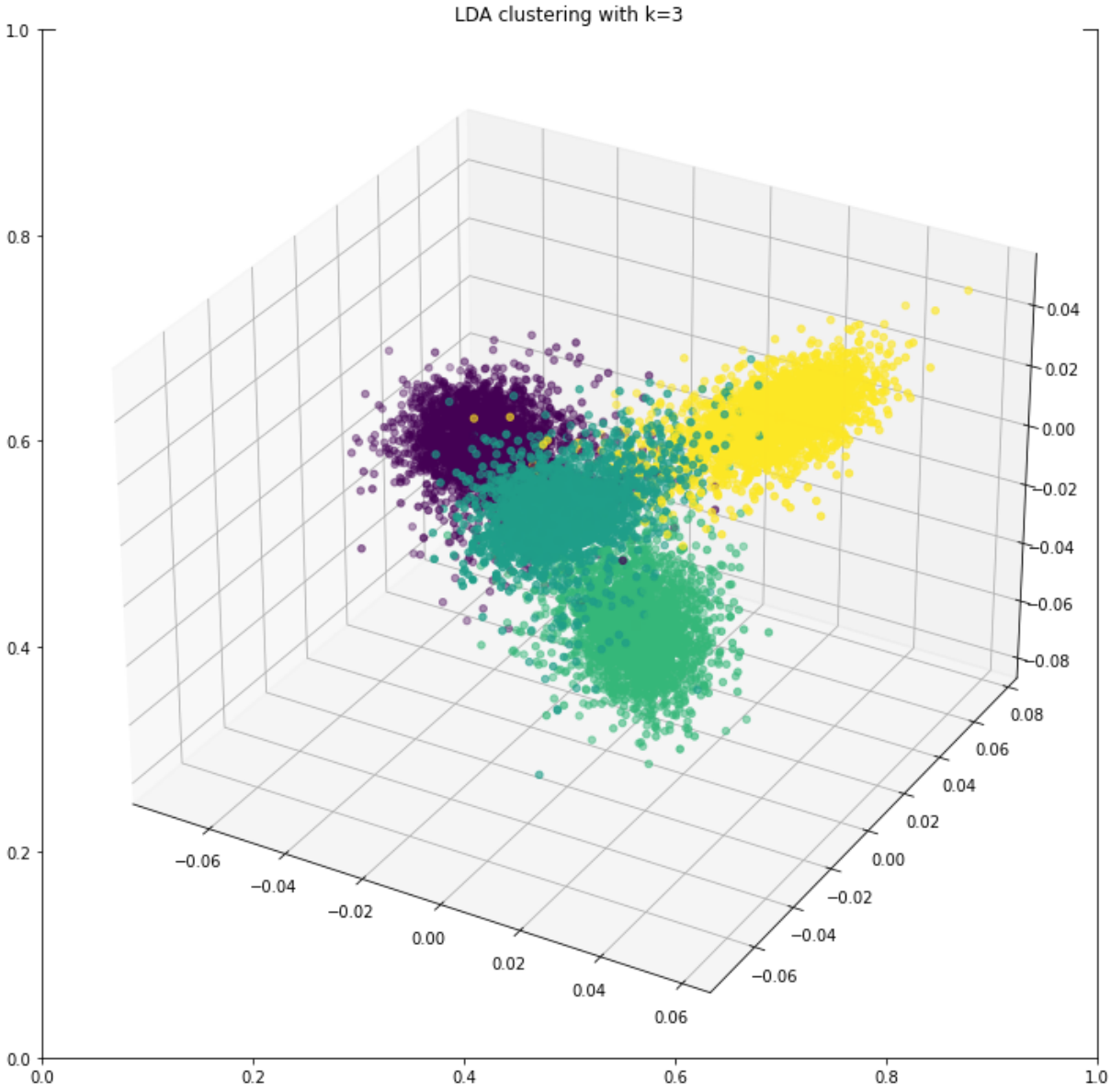
```
Average distance from centroid for digit 0 = 1880928.730481753
Average distance from centroid for digit 5 = 1187029.4741519378
Average distance from centroid for digit 6 = 1598409.0445846848
Average distance from centroid for digit 9 = 1510235.4151528922
```



In [ ]:

```
my_plot(Z_lda_3, Y_train, c_dist=True, three_dim = True, title = "LDA clustering wit
```

```
Average distance from centroid for digit 0 = 0.0030059520250964363
Average distance from centroid for digit 5 = 0.002215743584037764
Average distance from centroid for digit 6 = 0.0030172896312881605
Average distance from centroid for digit 9 = 0.0036900555888931675
```



In [ ]:

```python
def classify(Z, Y, P, x, chosen_numbers = chosen_numbers):
    z = P @ x
    cs = [c_k(Z, Y, c) for c in chosen_numbers]
    ds = [np.linalg.norm((z - c)) for c in cs]

    idx = np.argmin(ds)
    return chosen_numbers[idx]
```

In [ ]:

```python
hitting = {"pca": 0, "pca_3": 0, "lda": 0, "lda_3": 0}
for index, elem in enumerate(X_test.T):
    true_digit = Y_test[index]
    if classify(Z_pca, Y_train, P_pca, elem) == true_digit:
        hitting["pca"]+=1
    if classify(Z_pca_3, Y_train, P_pca_3, elem) == true_digit:
        hitting["pca_3"]+=1
    if classify(Z_lda, Y_train, P_lda, elem) == true_digit:
        hitting["lda"]+=1
    if classify(Z_lda_3, Y_train, P_lda_3, elem) == true_digit:
        hitting["lda_3"]+=1
```

In [ ]:

```python
accuracy = {"pca": 0, "pca_3": 0, "lda": 0, "lda_3": 0}
for idx in {"pca", "pca_3", "lda", "lda_3"}:
    accuracy[idx] = hitting[idx]/len(X_test.T)*100
```

In [ ]:
```
print(f"Hitting values are = {hitting}")
print(f"Accuracy values are = {accuracy}")
```

```
Hitting values are = {'pca': 3354, 'pca_3': 4444, 'lda': 5004, 'lda_3': 5162}
Accuracy values are = {'pca': 61.904761904761905, 'pca_3': 82.02288667404946, 'lda':
92.35880398671097, 'lda_3': 95.2750092284976}
```

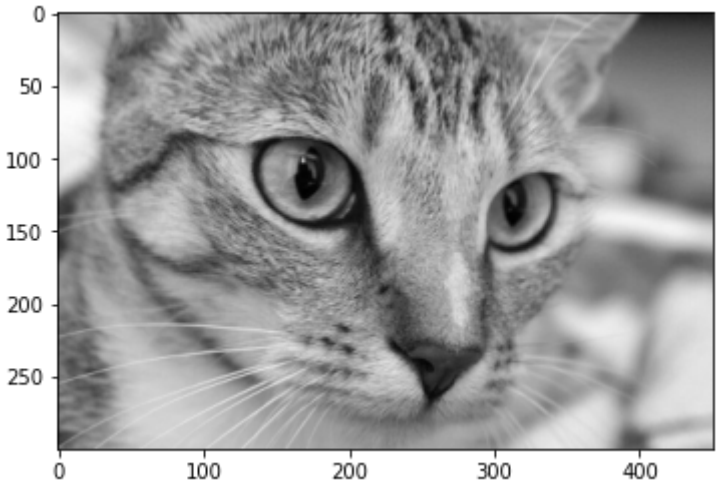
Visualizing Dyad

In [ ]:
```
import skimage
import numpy as np

import matplotlib.pyplot as plt
import skimage.io
from skimage import data
```

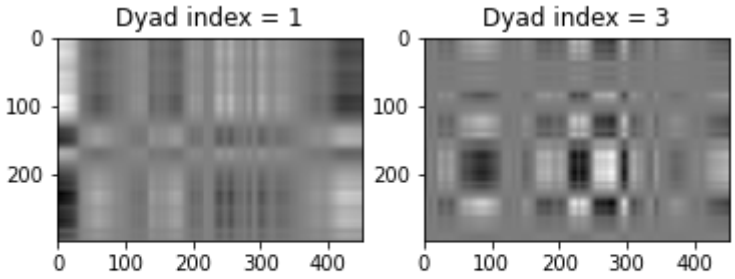
In [ ]:
```
img = data.chelsea()[:, :, 1]
```

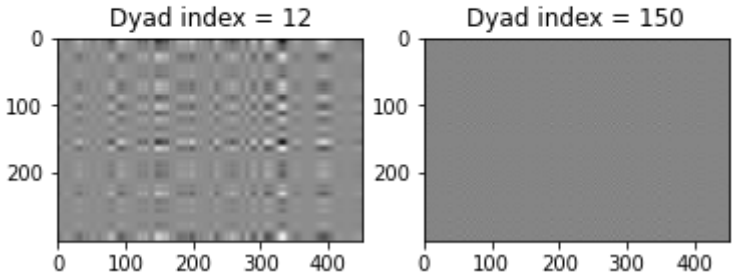
In [ ]:
```
plt.imshow(img, cmap="gray")
print()
```



In [ ]:
```
U, s, VT = np.linalg.svd(img, full_matrices = False)
```

In [ ]:
```
indices = [1, 3, 12, 150]
fig = plt.figure(figsize = (4*img.shape[0]/200, 4*img.shape[1]/200))
rows = 2
columns = 2
j = 1
for i in indices:
    M_tmp = s[i] * np.outer(U.T[i], VT[i])
    fig.add_subplot(rows, columns, j).set_title("Dyad index = " + str(i))
    j+=1
    plt.imshow(M_tmp, cmap="gray")
plt.show()
```





In [ ]:

```
print(f"Singular values of X = {[sv for sv in s]}")
```

```
Singular values of X = [41308.42858057524, 5576.383506047912, 4354.477252545097, 319
9.7095132191203, 2955.0902593304418, 2619.6891998433025, 2301.5715287749263, 1878.35
72250006255, 1760.6777916492456, 1714.0428596724296, 1444.9362842408327, 1348.635358
1060423, 1280.1637298828357, 1256.4396519172278, 1123.3875314519128, 1080.9241585010
504, 991.3170433741433, 964.8891009452345, 892.2689694081091, 854.5629953875249, 82
0.9061626276886, 771.3347793160058, 749.0899473868143, 704.2772068951738, 633.718723
0919791, 619.8969622622639, 602.502307845833, 578.575961711794, 560.4374295218403, 5
50.3061870224385, 527.6476924602772, 505.3788521700552, 497.46720510467003, 481.4336
0353995485, 475.6007128534145, 457.47023268252656, 443.32877218894066, 436.379619835
21114, 425.4636041174901, 420.72154320310153, 416.4528219001583, 403.9376865271829,
395.15429178895073, 383.18458824113145, 379.9314916639518, 360.68764978638876, 353.3
7224919130915, 346.7299176757595, 341.20132765334506, 335.9359271760259, 327.0579034
223399, 320.65579298232393, 315.1342427319289, 310.4013884457481, 306.7994525018537
5, 301.57530528089666, 296.17583961490465, 293.03091190333254, 284.9176700783528, 27
8.07793948828044, 268.67222470182145, 263.78681238306757, 261.3828023914313, 258.746
57891635303, 254.51066694435553, 248.0565140089941, 246.7534400392861, 242.676563758
7703, 240.55582082336375, 236.0993884274856, 232.39038042117429, 229.1960307597998,
227.85030762517454, 224.59562526023447, 219.823226888667, 215.86079704465448, 207.89
955652287097, 207.29665278456952, 205.79719758730482, 203.48920468202996, 203.050240
32417649, 196.51927347734426, 191.20275418948177, 188.55573880799048, 185.9380294957
5397, 182.18717607871565, 179.39592659523933, 178.60191707685905, 176.4680461057298
8, 175.95833843631885, 173.30961461353704, 171.48749695811875, 169.794422478539, 16
9.16488900847315, 166.7101505476015, 163.55778512498776, 162.57652638023913, 157.878
20693424706, 155.83910106636262, 154.82279916938822, 153.66938828727382, 150.7514458
6017214, 149.4983171708511, 147.92370423692532, 146.2129106607471, 144.8979624450496
7, 144.1960366580345, 141.82516852875384, 138.9953604798291, 137.57610829996491, 13
4.59156684512675, 133.7335048114339, 132.90441534589598, 131.6448289930466, 129.0739
0660441996, 127.50671702034677, 125.12813512255768, 124.85776547659303, 123.59005665
5148, 122.10573118195133, 121.84317331927711, 119.49691873949057, 118.1326975024326
2, 115.48979014062338, 114.04218854255701, 113.39493058476356, 110.71953737698453, 1
09.96810456304432, 109.03790372423624, 108.13343251327245, 106.08021124014019, 104.6
738783566544, 104.06706622943447, 102.5432089727671, 101.05085993517788, 99.75946651
969865, 98.69850718076928, 97.77715833063942, 96.0108270636279, 95.12008918590895, 9
3.89585485400309, 93.22863350581548, 92.97076204288176, 91.74048074222847, 90.721146
```

```
43059038, 89.21708351688419, 87.7849514453068, 85.86998277132793, 85.57682737595536,
84.59986183915795, 83.74111719260624, 83.01226525627631, 82.26629422110605, 81.46611
814259143, 81.00274826280337, 78.57079407703634, 77.55390038464495, 76.1359764731318
6, 75.79905024080361, 73.72173194468579, 73.58282041909688, 73.00673177924344, 71.94
28600909553, 71.3474516181386, 70.47905206337396, 69.8910601799078, 69.171851983980
9, 67.97896520689343, 67.535713972242, 67.02807340229327, 66.14867906294309, 65.5050
6122634711, 64.0699156084704, 63.515227785292076, 62.86049870203895, 61.450006876351
91, 60.810631352968166, 60.30021047494613, 59.237975013534594, 58.38142766305791, 5
7.849002310094136, 57.29937792236274, 56.8013935607049, 56.35376162708958, 55.021781
95190759, 53.931603899539034, 53.38190417948369, 53.04810882990342, 52.895697498827
4, 51.947919472457905, 51.18615575336739, 50.731430445258816, 49.65071662506056, 49.
378331757953134, 48.47350833222354, 47.918667775785, 47.32112589154909, 46.571031796
468816, 46.0611121038635, 45.176831945406335, 44.72771015596874, 43.68181621810806,
43.633768603939664, 42.80750935441311, 42.704607707102326, 42.48644243789605, 42.120
80857284759, 41.375706626929684, 40.88289240929202, 40.56873978701075, 39.9998122846
66206, 39.10342909161797, 38.69365725105461, 38.60393072200382, 37.44808044649448, 3
7.32799988520649, 36.739161109719014, 36.28122089427487, 35.74845406870593, 35.24265
5807648845, 34.68288474155795, 34.379914924125465, 33.39354731299273, 33.12526964218
561, 32.713661153387356, 32.39373760395119, 31.895924253134183, 31.28543052916164, 3
0.649968229548815, 30.239944760770957, 29.911961703199623, 29.25431697274126, 28.888
059474840112, 28.395464542738637, 28.154562236302123, 27.782054351053215, 27.2430898
4250166, 27.150427072219276, 26.29238375046674, 25.91541249902864, 25.58210026180290
3, 25.096992436248673, 24.547820956478315, 24.15604078287414, 24.05355605523715, 23.
982015345833886, 23.285412520858966, 23.253566643317853, 22.223438383553756, 21.9660
65151647754, 21.83633588910206, 21.46827469148174, 21.284877940880992, 20.8412534742
11786, 20.34257312289985, 20.21176143652864, 19.8688298014808, 19.343713752474372, 1
9.033540330520843, 18.88721401725114, 18.270435149199482, 17.70964546842904, 17.4194
56301745964, 16.998533180028573, 16.805226253864912, 16.621894756382257, 15.96247127
077147, 15.939269836049661, 15.709809440430075, 15.396504202952123, 14.8309018511212
8, 14.752470753367206, 14.203469388716945, 13.897621645247119, 13.491334236084722, 1
3.343475647649706, 12.949047685585436, 12.702355333569004, 12.47500946830709, 12.083
66563162109, 12.053625771242565, 11.40165004239627, 11.16726216393847, 10.9749274600
31694, 10.779052460305621, 10.434816342018497, 10.40293767939899, 10.15825646474015
7, 9.677476651296551, 9.62686254892282, 9.224510007062138, 8.69819096195722, 8.48927
2320768317, 8.240076493284151, 8.078050496471185, 7.834352343047424, 7.1999681320726
37, 6.6874969499879935, 6.222831550021803, 6.033755018119643]
```

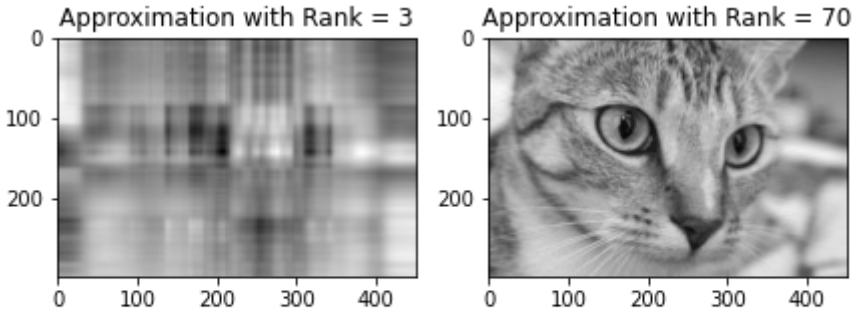
In [ ]:

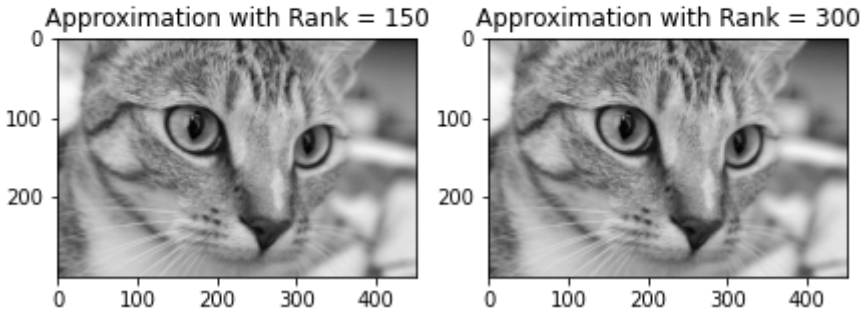
```
def rank_approximation(U, s, VT, A = None, ks = [3, 15, 30, 170], toshow = True):
    if len(ks) > 4 and toshow:
        raise Exception(f"A maximum number of 4 trials of approximation must be pass
    if toshow:
        fig = plt.figure(figsize = (4*img.shape[0]/170, 4*img.shape[1]/170))
        rows = 2
        columns = 2
        j = 1
    errs = []
    for k in ks:
        approx = s[0] * np.outer(U.T[0], VT[0])
        for i in range(1, k):
            approx += s[i] * np.outer(U.T[i], VT[i])

        if toshow:
            fig.add_subplot(rows, columns, j).set_title("Approximation with Rank = "
            j+=1
            plt.imshow(approx, cmap="gray")

        if A is not None:
            errs.append(np.linalg.norm(A - approx))
        else:
            errs = None
    if toshow:
        plt.show()
    return ks, errs
```

In [ ]:
```
ks, errs = rank_approximation(U, s, VT, img, ks=[3, 70, 150, 300])
ks_full, errs_full = rank_approximation(U, s, VT, img, ks=np.arange(start=1, stop=30
```

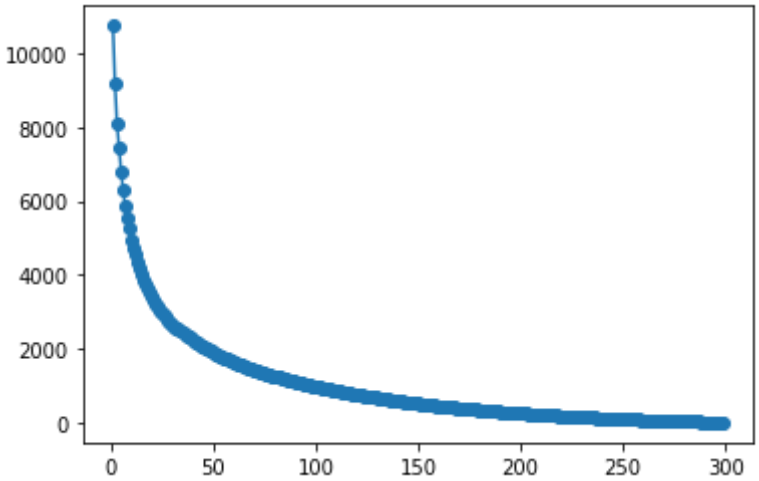




In [ ]:
```
for item in zip(ks, errs):
    print(f"Error of {item[1]} with {item[0]}-rank approximation.")
```

```
Error of 8081.498441134487 with 3-rank approximation.
Error of 1432.2311363729689 with 70-rank approximation.
Error of 520.8644647073336 with 150-rank approximation.
Error of 1.1608323369164794e-10 with 300-rank approximation.
```

In [ ]:
```
plt.plot(ks_full, errs_full, '-o')
plt.show()
```



In [ ]:
```
def compr_factor(img, k):
    m, n = img.shape
    return (m*n)/k

plt.plot(ks_full, compr_factor(img, ks_full), '-r')
plt.show()
```