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

Importing data

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

Constructing data matrices for three digits (0, 6, 9)

In [ ]:
```
A = np.array(data)

X_true = A[:,1:].T
Y_true = A[:,0]
```

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:]

    Xtrain = X[:, train_idx]
    Ytrain = Y[train_idx]

    Xtest = X[:, test_idx]
    Ytest = Y[test_idx]

    return Xtrain, Xtest, Ytrain, Ytest


Xtrain, Xtest, Ytrain, Ytest = x_split(X_true, Y_true, 30000)
```

Implementing PCA

In [ ]:
```
def PCA(X, Y, k, choosen_numbers = [0, 6, 9]):

    indeces = [index for index, elem in enumerate(Y) if elem in choosen_n
    X = X[:, indeces]
    Y = Y[indeces]


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


    X_c = X - c_X


    U, s, VT = np.linalg.svd(X, full_matrices = False)


    U_k = np.resize(U, (len(U), k))


    Z_k = U_k.T @ X

    return Z_k, Y, U_k.T
```

Implementing LDA

In [ ]:
```python
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, choosen_numbers = [0, 6, 9]):

    if len(choosen_numbers) == 3:
        indeces_1 = (Y == choosen_numbers[0])
        indeces_2 = (Y == choosen_numbers[1])
        indeces_3 = (Y == choosen_numbers[2])
    else:
        raise Exception(f"Choosen Numbers must be 3, but {len(choosen_num


    X1 = X[:, indeces_1]
    Y1 = Y[indeces_1]

    X2 = X[:, indeces_2]
    Y2 = Y[indeces_2]

    X3 = X[:, indeces_3]
    Y3 = Y[indeces_3]


    X_join = np.concatenate((X1, X2, X3), axis=1)
    Y_join = np.concatenate((Y1, Y2, Y3))


    c_1 = c_k(X, Y, choosen_numbers[0])
    c_1 = np.reshape(c_1, (len(c_1), 1))
    c_2 = c_k(X, Y, choosen_numbers[1])
    c_2 = np.reshape(c_2, (len(c_2), 1))
    c_3 = c_k(X, Y, choosen_numbers[2])
    c_3 = np.reshape(c_3, (len(c_3), 1))


    global_c = np.mean(X_join, axis=1)
    global_c = np.reshape(global_c, (len(global_c),1))


    X1_c = X1 - c_1
    X2_c = X2 - c_2
    X3_c = X3 - c_3


    X_w = np.concatenate((X1_c, X2_c, X3_c), axis=1)


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


    gX_1 = np.full( Xtrain.shape, c_1)
    gX_2 = np.full( Xtrain.shape, c_2)
    gX_3 = np.full( Xtrain.shape, c_3)
    gX = np.concatenate((gX_1, gX_2, gX_3), axis=1)


    gX_c = gX - global_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_join

    return Z, Y_join, 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)
```

Testing PCA and LDA and visualizing results

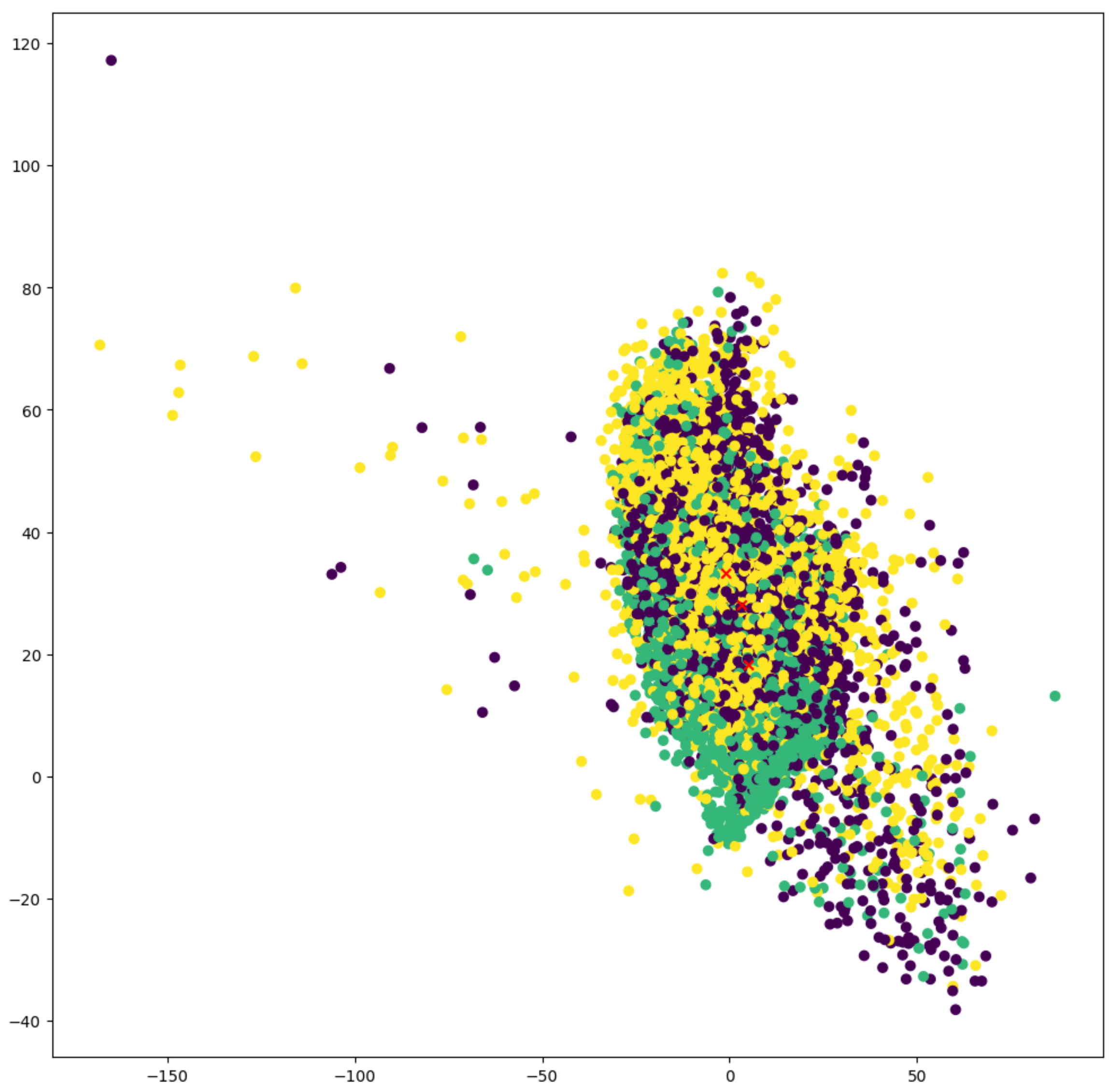
In [ ]:
```
def my_plot(X, Y, three_dim=False, c_dist=False):
    fig = plt.figure(figsize = (12, 12))
    ax = fig.add_subplot()
    if three_dim:
        ax = fig.add_subplot(projection="3d")
        ax.scatter(X[0, :], X[1, :], X[2, :], c=Y)
    else:
        ax.scatter(X[0,:], X[1,:], c=Y)
    ax.scatter(*(c_k(X, Y, 0)), marker="x", color="red")
    ax.scatter(*(c_k(X, Y, 6)), marker="x", color="red")
    ax.scatter(*(c_k(X, Y, 9)), marker="x", color="red")
    if c_dist:
        print(f"Average distance from centroid for digit 0 = {avg_centroi
        print(f"Average distance from centroid for digit 6 = {avg_centroi
        print(f"Average distance from centroid for digit 9 = {avg_centroi
    plt.show()
```

In [ ]:
```
Z_pca, Y_pca, _ = PCA(Xtrain, Ytrain, k=2)
```

In [ ]:
```
Z_pca_3, Y_pca_3, _ = PCA(Xtrain, Ytrain, k=3)
```

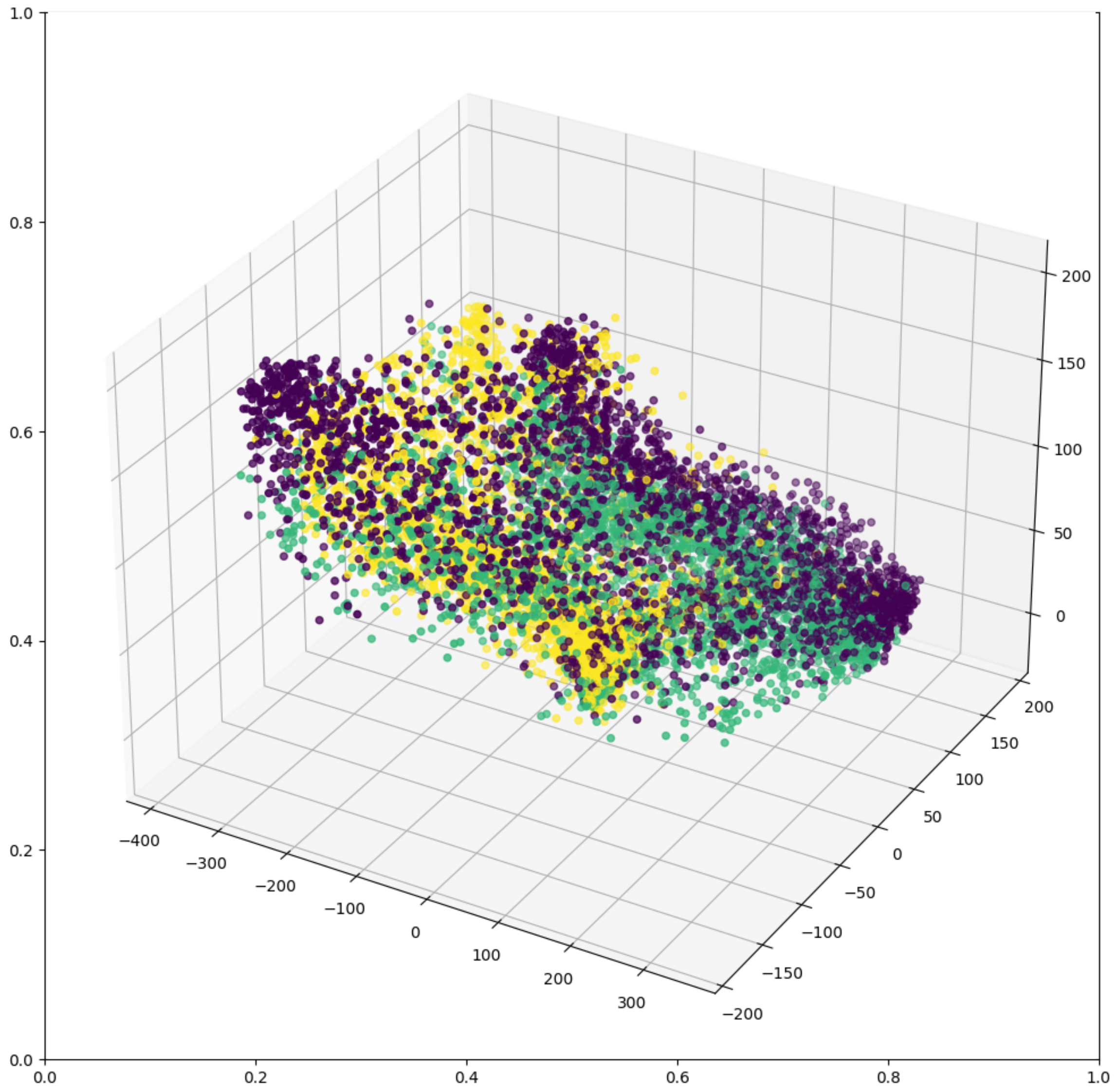
In [ ]:
```
my_plot(Z_pca, Y_pca, c_dist=True)
```

```
Average distance from centroid for digit 0 = 746.7235947438776
Average distance from centroid for digit 6 = 819.2256183994555
Average distance from centroid for digit 9 = 799.6707146301981
```



```
In [ ]: my_plot(Z_pca_3, Y_pca_3, three_dim=True, c_dist=True)
```

```
Average distance from centroid for digit 0 = 34363.71594678581
Average distance from centroid for digit 6 = 37317.7466143958
Average distance from centroid for digit 9 = 39630.51569530052
```

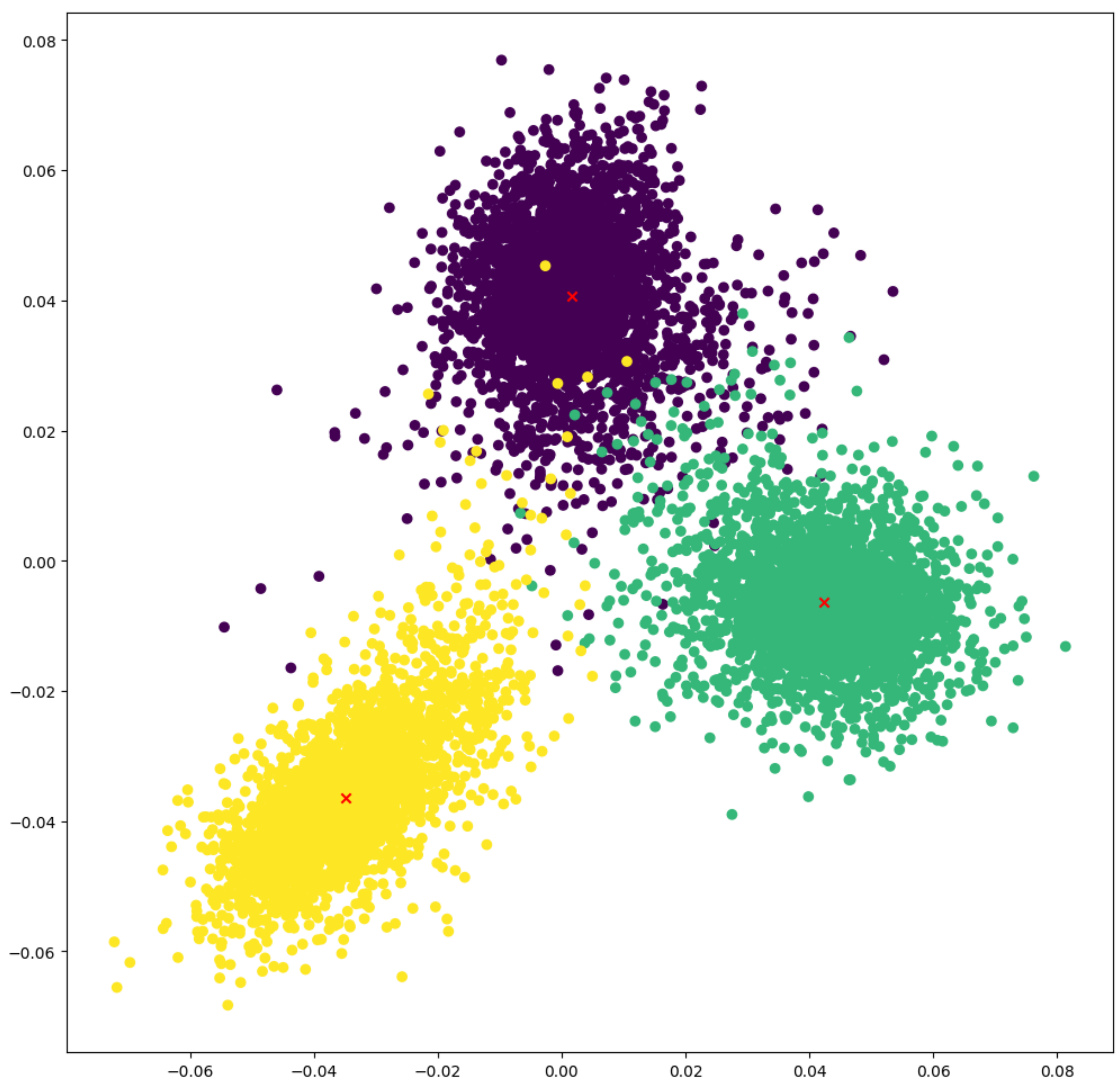


```
In [ ]: Z, Y_join, _ = LDA(Xtrain, Ytrain, k=2)
```

```
In [ ]: Z_3, Y_join_3, _ = LDA(Xtrain, Ytrain, k=3)
```

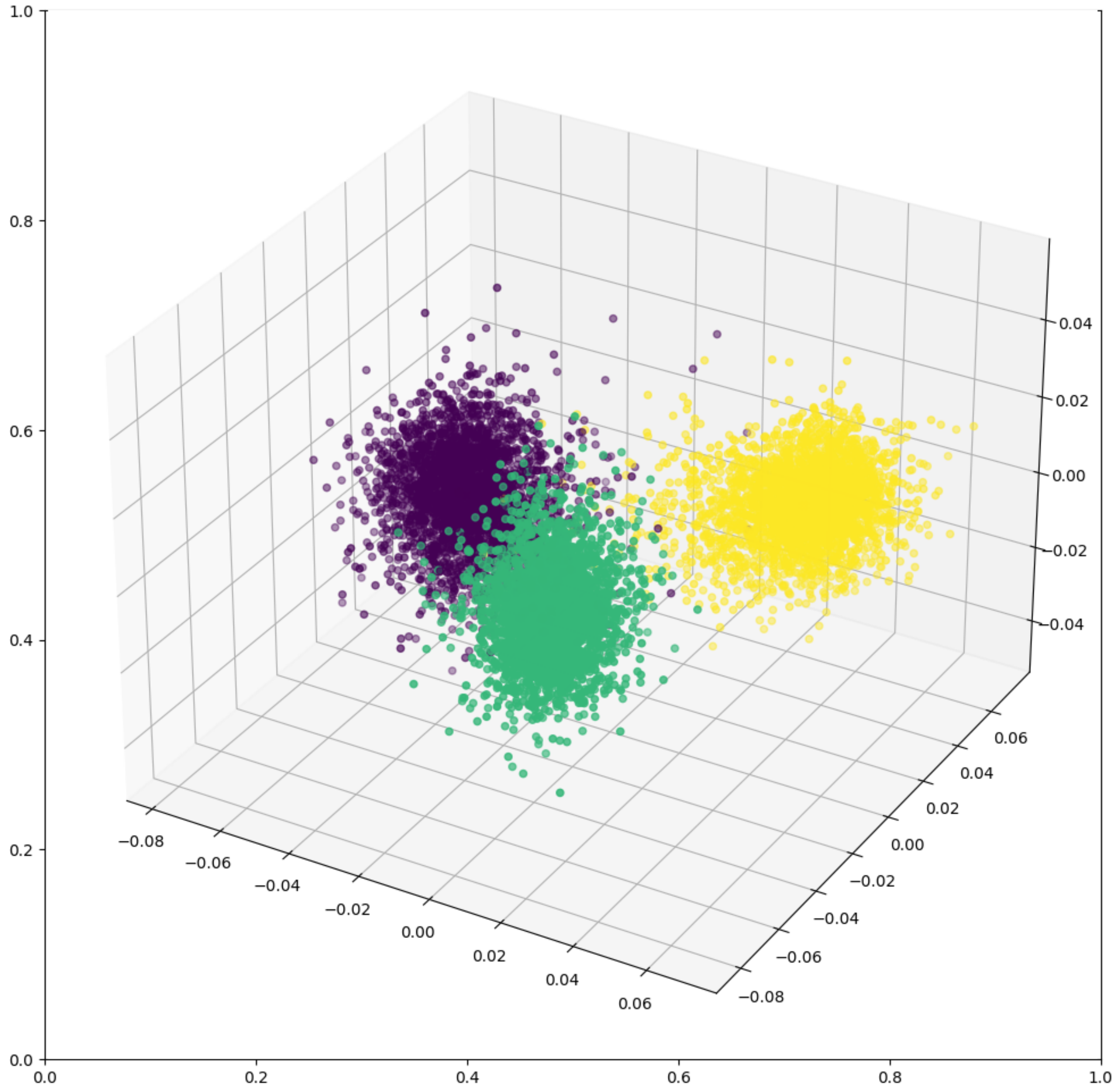
```
In [ ]: my_plot(Z, Y_join, c_dist=True)
```

```
Average distance from centroid for digit 0 = 0.0039405826363087945
Average distance from centroid for digit 6 = 0.003823206249144394
Average distance from centroid for digit 9 = 0.004906788729796304
```



```
In [ ]: my_plot(Z_3, Y_join_3, three_dim=True, c_dist=True)
```

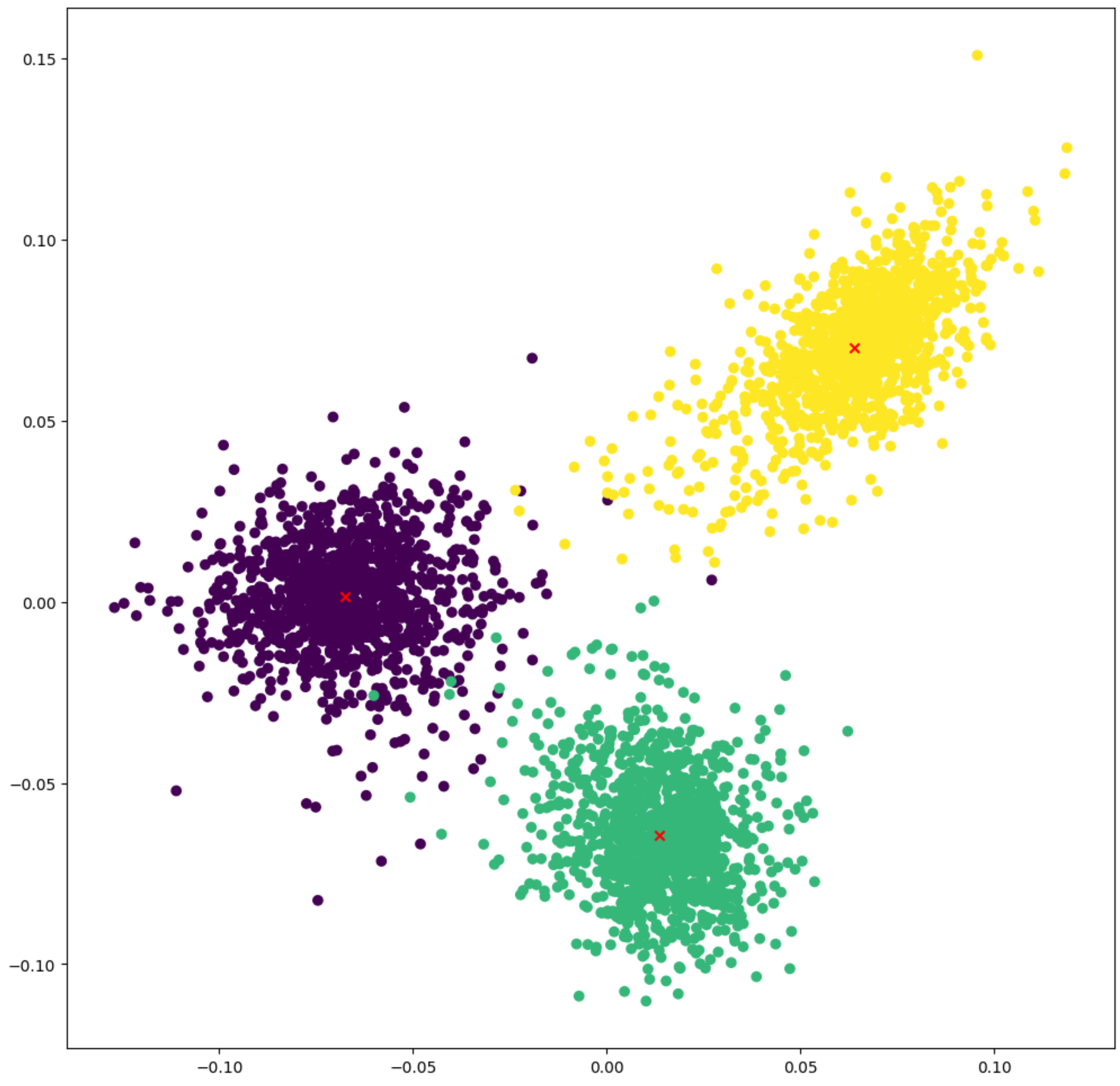
```
Average distance from centroid for digit 0 = 0.004053137395919538
Average distance from centroid for digit 6 = 0.003935487088943355
Average distance from centroid for digit 9 = 0.0050202725129837485
```



```
In [ ]: Z_test, Y_join_test, _ = LDA(Xtest, Ytest, k=2)
```

```
In [ ]: my_plot(Z_test, Y_join_test, c_dist=True)
```

```
Average distance from centroid for digit 0 = 0.011625032083517884
Average distance from centroid for digit 6 = 0.011057444206316425
Average distance from centroid for digit 9 = 0.01469317563937877
```



Classification

```
In [ ]: choosen_numbers = [0, 6, 9]
        indeces = [index for index, elem in enumerate(Ytest) if elem in choosen_n
        Xtest_fixed = Xtest[:, indeces]
        Ytest_fixed = Ytest[indeces]

        random_index = random.randint(0, len(indeces))
        new_x = Xtest_fixed[:, random_index]
        true_digit = Ytest_fixed[random_index]
        print(true_digit)
```

```
6
```

In [ ]:
```
def classify(Z, Y, P, x):
    z = P @ x
    c_0 = c_k(Z, Y_join, 0)
    c_6 = c_k(Z, Y_join, 6)
    c_9 = c_k(Z, Y_join, 9)

    d_0 = np.linalg.norm((z - c_0))
    d_6 = np.linalg.norm((z - c_6))
    d_9 = np.linalg.norm((z - c_9))

    min = d_0
    if d_6 < min:
        if d_6 < d_9:
            return  choosen_numbers[1]
        else:
            return choosen_numbers[2]
    elif d_9 < min:
        if d_9 < d_6:
            return choosen_numbers[2]
        else:
            return choosen_numbers[1]
    else:
        return choosen_numbers[0]
```

In [ ]:
```
Z_pca, Y_pca, P_pca = PCA(Xtrain, Ytrain, k=2)
Z_pca_3, Y_pca_3, P_pca_3 = PCA(Xtrain, Ytrain, k=3)
```

In [ ]:
```
Z_lda, Y_lda, P_lda = LDA(Xtrain, Ytrain, k=2)
Z_lda_3, Y_lda_3, P_lda_3 = LDA(Xtrain, Ytrain, k=3)
```

In [ ]:
```
hitting = {"pca": 0, "pca_3": 0, "lda": 0, "lda_3": 0}
for index, elem in enumerate(Xtest_fixed.T):
    true_cluster = Ytest_fixed[index]
    if classify(Z_pca, Y_pca, P_pca, elem) == true_cluster:
        hitting["pca"]+=1
    if classify(Z_pca_3, Y_pca_3, P_pca_3, elem) == true_cluster:
        hitting["pca_3"]+=1
    if classify(Z_lda, Y_lda, P_lda, elem) == true_cluster:
        hitting["lda"]+=1
    if classify(Z_lda_3, Y_lda_3, P_lda_3, elem) == true_cluster:
        hitting["lda_3"]+=1
```

In [ ]:
```
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(Xtest_fixed.T)*100
```

In [ ]:
```
print(hitting)
print(accuracy)
```

```
{'pca': 1192, 'pca_3': 993, 'lda': 3541, 'lda_3': 3535}
{'pca': 33.046853340726365, 'pca_3': 27.52980316052121, 'lda': 98.1702245
6334904, 'lda_3': 98.00388134183532}
```

Visualizing DYAD

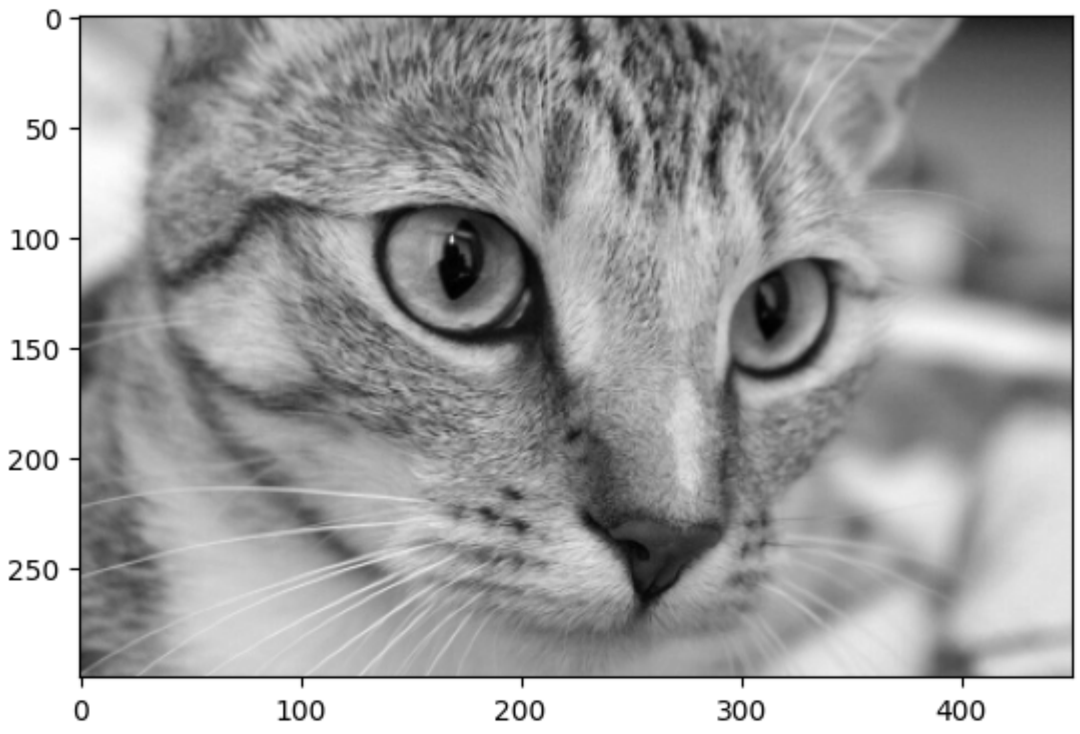
```
In [ ]: import skimage
        import numpy as np

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

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

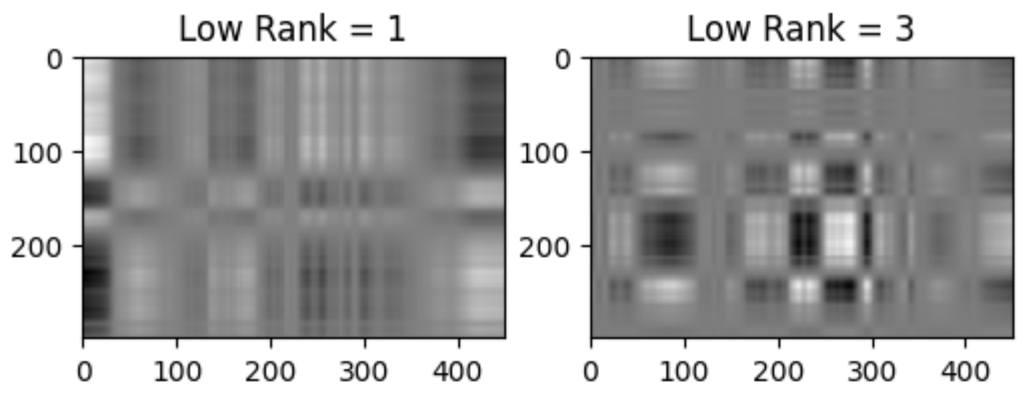
```
In [ ]: plt.imshow(img, cmap="gray")
```

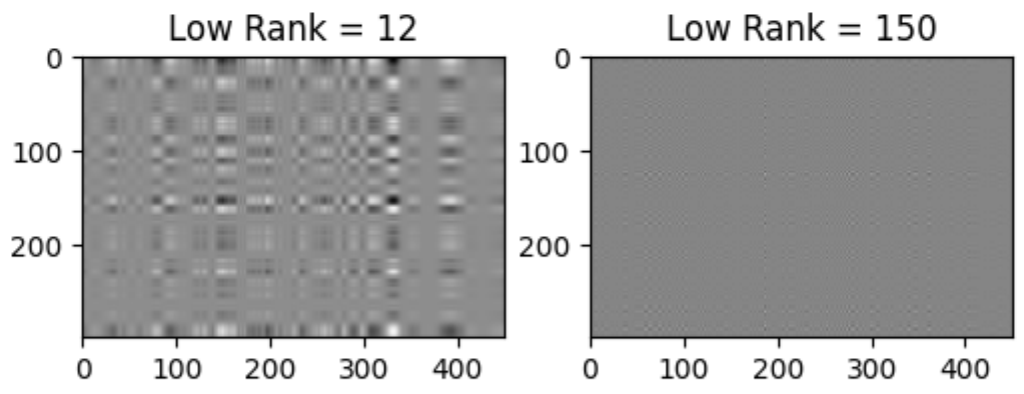
```
Out[ ]: <matplotlib.image.AxesImage at 0x7f0f38ae40a0>
```



```
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("Low Rank = " + str(i))
            j+=1
            plt.imshow(M_tmp, cmap="gray")
        plt.show()
```





```
In [ ]: print([sv for sv in s])
```

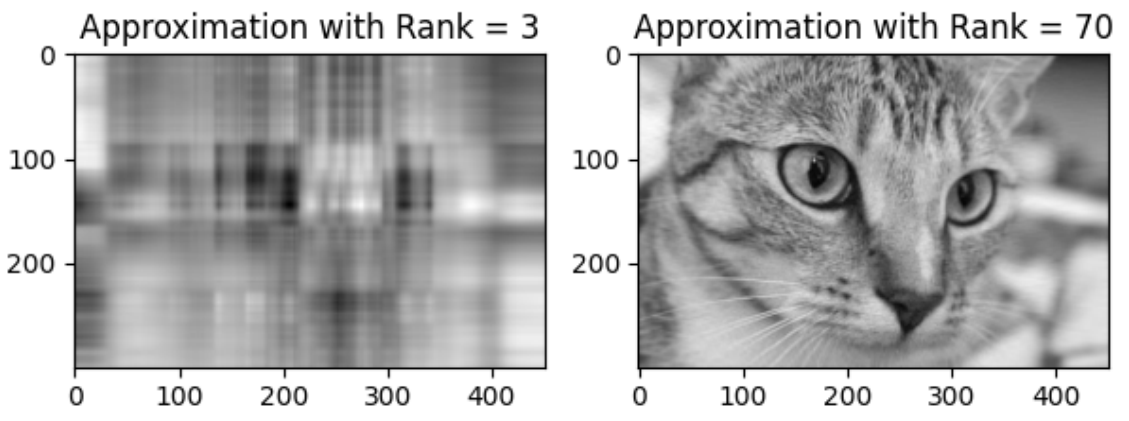
```
[41308.42858057524, 5576.383506047912, 4354.477252545095, 3199.7095132191
22, 2955.0902593304445, 2619.6891998433034, 2301.5715287749267, 1878.3572
250006262, 1760.6777916492438, 1714.0428596724305, 1444.936284240833, 134
8.6353581060416, 1280.163729882838, 1256.4396519172283, 1123.38753145191
2, 1080.9241585010502, 991.317043374143, 964.8891009452345, 892.268969408
108, 854.5629953875243, 820.9061626276884, 771.3347793160051, 749.0899473
868152, 704.2772068951741, 633.7187230919791, 619.8969622622641, 602.5023
078458329, 578.5759617117941, 560.4374295218408, 550.3061870224382, 527.6
476924602771, 505.3788521700549, 497.46720510466986, 481.4336035399547, 4
75.60071285341434, 457.4702326825267, 443.3287721889408, 436.379619835211
26, 425.46360411749026, 420.7215432031017, 416.4528219001583, 403.9376865
2718273, 395.15429178895073, 383.184588241315, 379.9314916639518, 360.68
764978638836, 353.3722491913094, 346.72991767575996, 341.201327653448, 3
35.93592717602576, 327.05790342233985, 320.6557929823241, 315.13424273192
885, 310.401388445748, 306.79945250185375, 301.57530528089666, 296.175839
61490465, 293.0309119033324, 284.91767007835284, 278.0779394882805, 268.6
722247018213, 263.78681238306757, 261.3828023914314, 258.7465789163530з,
254.51066694435548, 248.056514008994, 246.7534400392861, 242.676563758770
24, 240.5558208233636, 236.09938842748554, 232.39038042117417, 229.196030
7597998, 227.8503076251745, 224.5956252602344, 219.82322688866708, 215.86
079704465456, 207.8995565228711, 207.29665278456952, 205.79719758730474,
203.48920468203025, 203.05024032417649, 196.51927347734423, 191.202754189
48214, 188.55573880799034, 185.93802949575414, 182.18717607871548, 179.39
592659523944, 178.60191707685914, 176.46804610573008, 175.95833843631885,
173.30961461353712, 171.48749695811875, 169.79442247853888, 169.164889008
47315, 166.71015054760173, 163.55778512498776, 162.5765263802391, 157.878
2069342471, 155.83910106636256, 154.8227991693882, 153.66938828727382, 15
0.75144586017194, 149.49831717085118, 147.92370423692532, 146.21291066074
693, 144.89796244504984, 144.19603665803447, 141.82516852875395, 138.9953
6047982912, 137.57610829996497, 134.59156684512675, 133.7335048114341, 13
2.90441534589615, 131.64482899304662, 129.07390660442007, 127.50671702034
678, 125.12813512255772, 124.85776547659303, 123.59005665514792, 122.1057
3118195146, 121.84317331927703, 119.49691873949051, 118.13269750243249, 1
15.4897901406234, 114.04218854255708, 113.39493058476356, 110.71953737698
446, 109.96810456304435, 109.03790372423639, 108.13343251327248, 106.0802
1124014026, 104.67387835665448, 104.0670662294345, 102.54320897276699, 10
1.05085993517791, 99.75946651969873, 98.69850718076917, 97.7771583306395
2, 96.01082706362781, 95.12008918590895, 93.8958548540033, 93.22863350581
544, 92.97076204288163, 91.7404807422285, 90.72114643059028, 89.217083516
88417, 87.78495144530676, 85.86998277132793, 85.57682737595542, 84.599861
83915804, 83.74111719260621, 83.01226525627627, 82.26629422110587, 81.466
11814259141, 81.00274826280342, 78.57079407703633, 77.553900384645, 76.13
597647313183, 75.79905024080365, 73.7217319446857, 73.58282041909676, 73.
00673177924335, 71.94286009095526, 71.34745161813869, 70.47905206337389,
69.8910601799077, 69.17185198398097, 67.97896520689363, 67.5357139722420
1, 67.0280734022932, 66.14867906294309, 65.50506122634712, 64.06991560847
052, 63.51522778529213, 62.86049870203887, 61.45000687635181, 60.81063135
296813, 60.30021047494613, 59.23797501353465, 58.381427663058055, 57.849
00231009429, 57.29937792236252, 56.80139356070489, 56.3537616270895, 55.0
2178195190745, 53.93160389953905, 53.381904179483726, 53.04810882990343,
52.89569749882746, 51.94791947245796, 51.18615575336753, 50.7314304452589
4, 49.65071662506061, 49.37833175795316, 48.473508332223666, 47.918667775
78507, 47.321125891549194, 46.57103179646882, 46.061112103863486, 45.1768
3194540637, 44.72771015596878б, 43.68181621810799, 43.63376860393961, 42.
8075093544132, 42.70460770710221, 42.48644243789599, 42.12080857284756, 4
1.3757066269З008, 40.882892409292054, 40.568739787010706, 39.999812284666
305, 39.103429091618, 38.69365725105472, 38.60393072200382, 37.4480804464
94515, 37.32799988520659, 36.73916110971898, 36.281220894274945, 35.74845
4068706, 35.24265580764886, 34.68288474155794, 34.37991492412546, 33.3935
4731299269, 33.125269642185664, 32.71366115338737, 32.39373760395129, 31.
89592425313421, 31.285430529161452, 30.649968229548655, 30.2399447607710
3, 29.91196170319954, 29.254316972741258, 28.88805947484022, 28.395464542
73874, 28.154562236302148, 27.78205435105319, 27.243089842501657, 27.1504
27072219287, 26.292383750466765, 25.915412499028616, 25.58210026180289, 2
5.09699243624865, 24.547820956478297, 24.1560407828741, 24.05355605523712
4, 23.982015345833933, 23.285412520858916, 23.25356664331792, 22.22343838
355383, 21.966065151647765, 21.836335889102088, 21.468274691481778, 21.28
4877940880946, 20.841253474211683, 20.342573122899815, 20.21176143652857
2, 19.868829801480786, 19.343713752474425, 19.033540330520907, 18.8872140
17251246, 18.270435149199493, 17.709645468428945, 17.419456301746113, 16.
998533180028613, 16.805226253864888, 16.62189475638226, 15.9624712707715,
15.939269836049716, 15.709809440430067, 15.39650420295212, 14.83090185112
1364, 14.752470753367259, 14.203469388716911, 13.897621645246977, 13.4913
34236084548, 13.343475647649758, 12.949047685585489, 12.702355333568946,
12.475009468307178, 12.083665631621166, 12.053625771242555, 11.4016500423
96193, 11.167262163938494, 10.974927460031704, 10.77905246030566, 10.4348
16342018413, 10.402937679399049, 10.158256464740141, 9.677476651296594,
9.626862548922787, 9.224510007062175, 8.698190961957446, 8.48927232076837
2, 8.240076493284354, 8.078050496471295, 7.834352343047375, 7.19996813207
2684, 6.687496949988073, 6.222831550021724, 6.033755018119725]
```

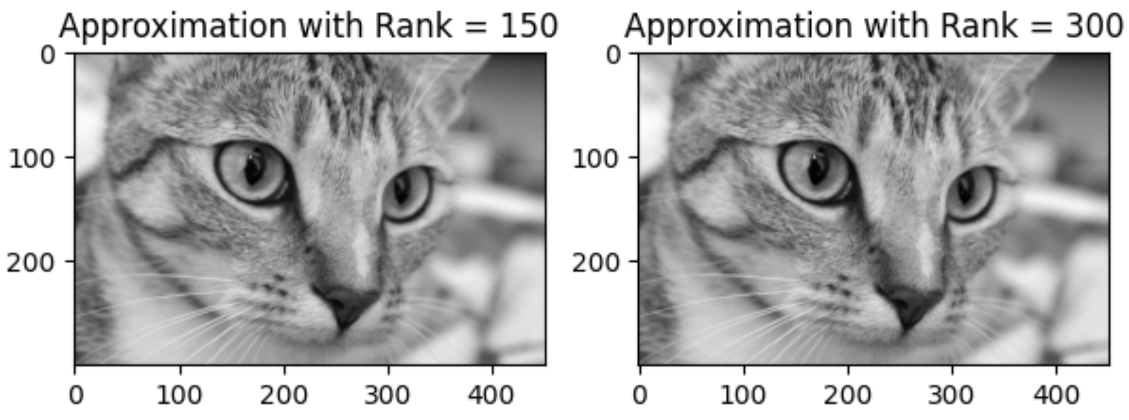
In [ ]:
```python
def rank_approximation(U, s, VT, A = None, ks = [3, 15, 30, 170], toshow
    if len(ks) > 4 and toshow:
        raise Exception(f"A maximum number of 4 trials of approximation m
    if toshow:
        fig = plt.figure(figsize = (4*img.shape[0]/170, 4*img.shape[1]/17
        rows = 2
        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 wi
            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 [ ]:
```python
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
```



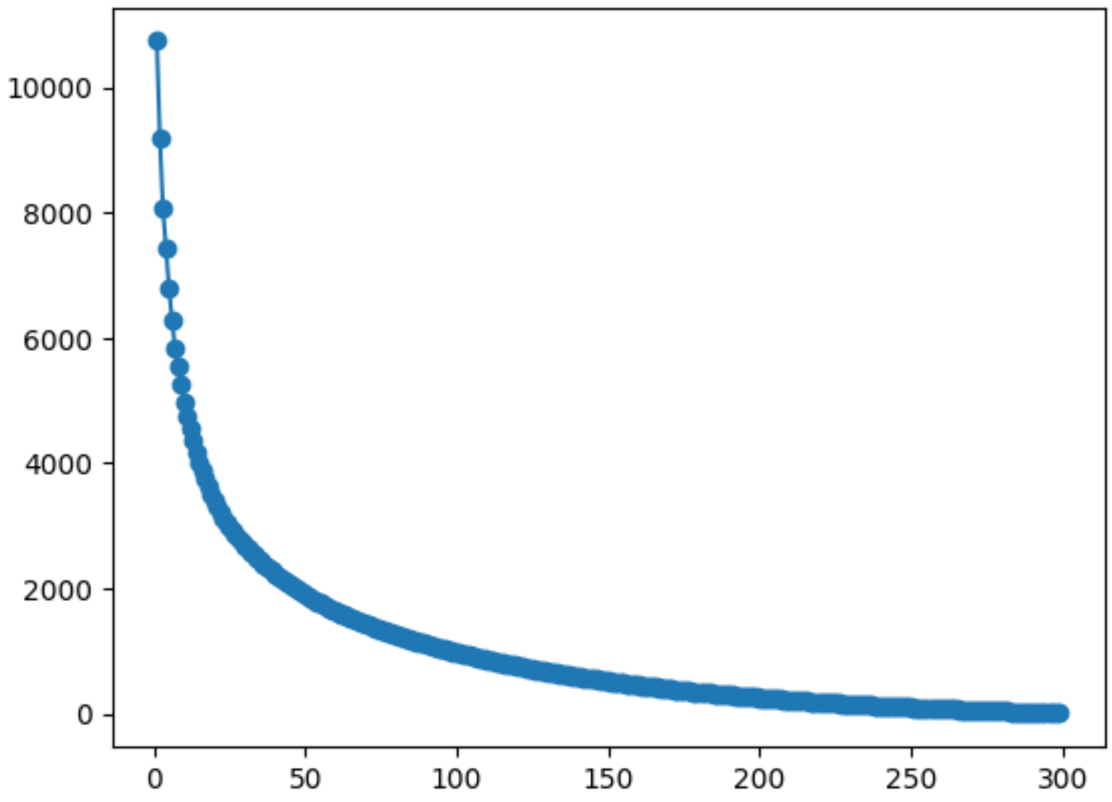


```
In [ ]: for item in zip(ks, errs):
            print(f"Error of {item[1]} with {item[0]}-rank approximation.")
```

```
Error of 8081.498441134487 with 3-rank approximation.
Error of 1432.2311363729689 with 70-rank approximation.
Error of 520.8644647073337 with 150-rank approximation.
Error of 1.1576830509210408e-10 with 300-rank approximation.
```

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

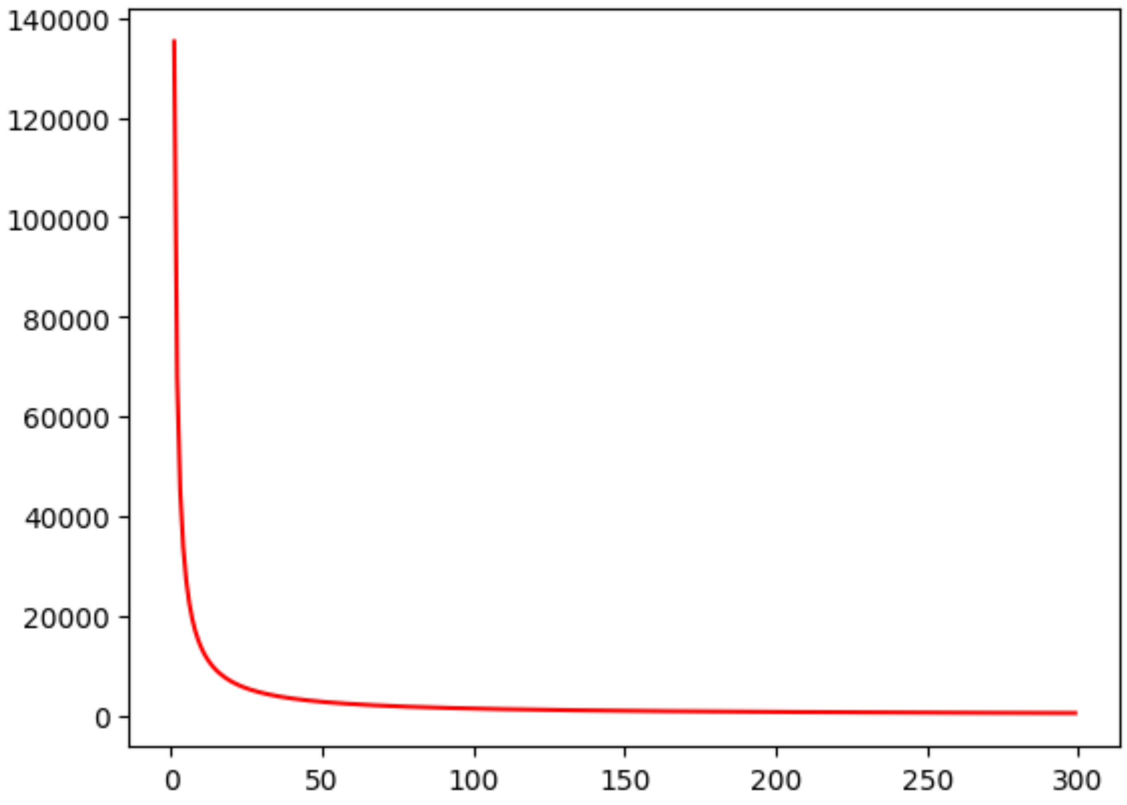
```
Out[ ]: [<matplotlib.lines.Line2D at 0x7f0f384abd90>]
```



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

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

```
Out[ ]: [<matplotlib.lines.Line2D at 0x7f0f38416760>]
```


140000
120000
100000
80000
60000
40000
20000
0
0
50
100
150
200
250
300