```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.linear_model
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from scipy.linalg import eigh
from keras.datasets import mnist
```

Introduction

In this notebook, we dive deep into the task of dimensionality reduction. Dimensionality reduction is useful for several purposes, including but not restricted to, visualization, storage, faster computation etc.

In part I, we will explore how PCA can be used for visualization.

In part II, we will see how explore how principal components can be used to reduce dimension for a classification task.

Part I: Visualization

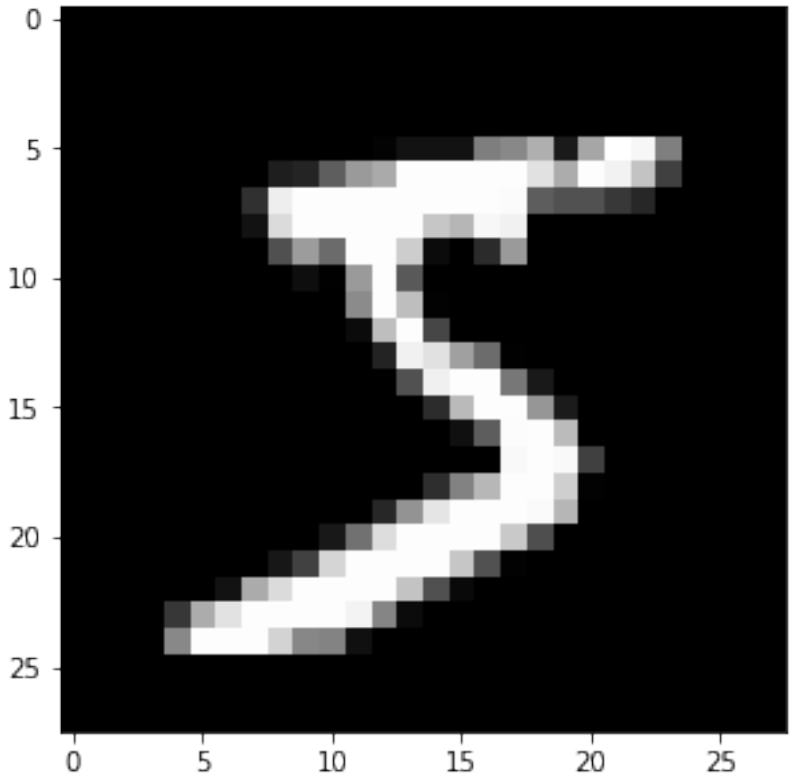
In this exercise, you need to implement the dimensionality reduction technique PCA on the MNIST dataset from scratch and visualize the results. In the MNIST dataset, you would find information on handwritten digits (0-9) stored in the form of a 60,000x28x28 array (60,000 samples of a 28x28 image), where each element of the array represents a single pixel (which varies from 0 to 1, where the black color is represented by 1 and white by 0 and middle values represent the shades of grey).

(a). Using the started code provided, find the first 2 principal components PCA_1 and PCA_2

```
# Load the MNIST Dataset
(X, y), _ = mnist.load_data()
print(X.shape)
print(y.shape)

nsamples, nx, ny = X.shape
X = X.reshape((nsamples,nx*ny))
```

```
(60000, 28, 28)
(60000,)
```

```
# Plotting a random sample data point
plt.figure(figsize = (20, 5))
grid_data = np.array(X[0]).reshape(28, 28)
plt.imshow(grid_data, interpolation = None, cmap = 'gray')
plt.show()
```




```
# Standardize the data and find the co-variance matrix

scaler = StandardScaler()
X_norm = scaler.fit_transform(X)
print(X_norm.shape)
```

```
(60000, 784)
```

```
# Find the covariance matrix

# TODO: Replace with correct answer
covar = np.matmul(np.transpose(X_norm), X_norm)
print(covar.shape)
```

```
(784, 784)
```

```
# Find the top two eigen-values and corresponding eigenvectors

# TODO: Replace with correct answer
values, vectors = eigh(covar, subset_by_index = [covar.shape[0]-2,
covar.shape[0]-1])
print(values.shape)
print(vectors.shape)
```

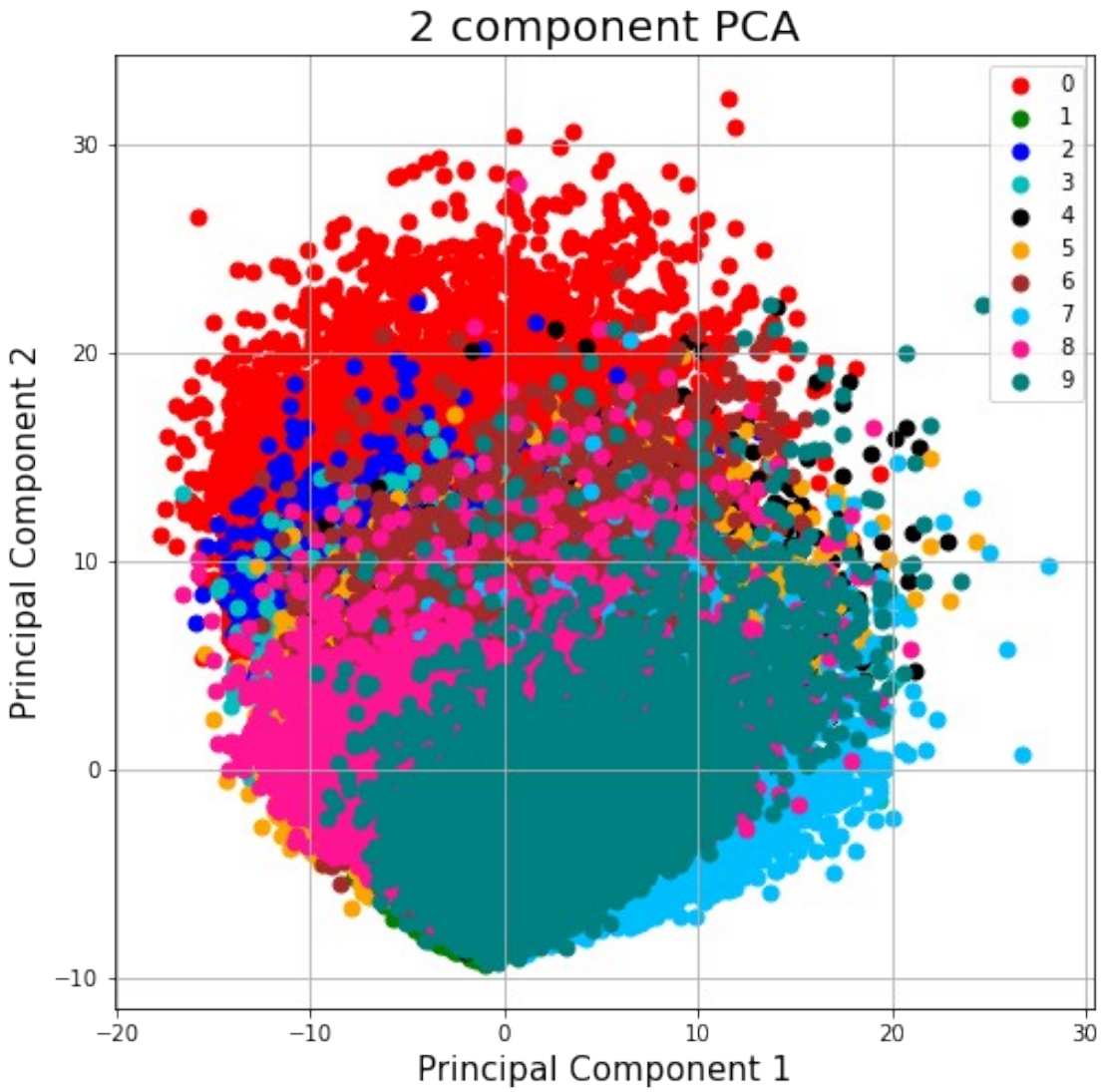
```
(2,)
(784, 2)
```

(b) Project the data onto these components and visualize it using a scatter plot

```
# Find the principal components PC1 and PC2

# TODO
projected_features = np.matmul(np.transpose(vectors),
np.transpose(X_norm))
print(projected_features.shape)
```

```
(2, 60000)
```

```
# Visulaize the obtained dataframe using scatterplot

# TODO
new_coordinates = np.transpose(np.vstack((projected_features, y)))
df = pd.DataFrame(data = new_coordinates, columns = ['principal
component 1', 'principal component 2', 'labels'])

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
colors = ['r', 'g', 'b', 'c', 'k', 'orange', 'brown', 'deepskyblue',
'deeppink', 'teal']
for label, color in zip(labels,colors):
    indicesToKeep = df['labels'] == label
    ax.scatter(df.loc[indicesToKeep, 'principal component 1']
               , df.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



Part II: Principal Component Regression and comparison with random projections

While you have already seen many properties of PCA so far, in this question we investigate if random projections are a good idea for dimensionality reduction. A few advantages of random projections over PCA can be:

1. PCA is expensive when the underlying dimension is high and the number of principal components is also large (however note that there are several very fast algorithms dedicated to doing PCA)
2. PCA requires you to have access to the feature matrix for performing computations. The second requirement of PCA is a bottle neck when you want to take only a low dimensional measurement of a very high dimensional data, e.g., in FMRI and in compressed sensing. In such cases, one needs to design a projection scheme before

seeing the data. We now turn to a concrete setting to study a few properties of PCA and random projections.

Suppose you are given n points $x_1, \ldots, x_n$ in R^D. Define the n x d matrix

$$X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_n^\top \end{bmatrix}$$

where each row of the matrix represents one of the given points. In this problem, we will consider a few low-dimensional linear embedding $\Psi : R^d \longrightarrow R^k$ that maps vectors from R^d to R^k.

Recall: PCA

Let $X = U \Sigma V^T$ denote the singular value decomposition of the matrix X. Assume that $n \geq d$ and let $\sigma_1, \sigma_2, \ldots, \sigma_d$ denote the singular values of X.Let $v_1, v_2, \ldots, v_d$ denote the columns of the matrix V.

Then the k-dimensional PCA embedding is expressed as: $\Psi_{\text{PCA}}(x) = \left(v_1^T x, \ldots, v_k^T x\right)^T$. Note that this embedding projects a d-dimensional vector on the linear span of the set $v_1, \ldots, v_k$ and that $v_i^T x$ denotes the i-th coordinate of the projected vector in the new space.

Random Projection

Consider the random matrix $J \in R^{k \times d}$ with each of its entry being i.i.d. $N(0,1)$ and consider the map $\psi_J : R^d \mapsto R^k$ such that $\psi_J(x) = \frac{1}{\sqrt{k}} J x$.

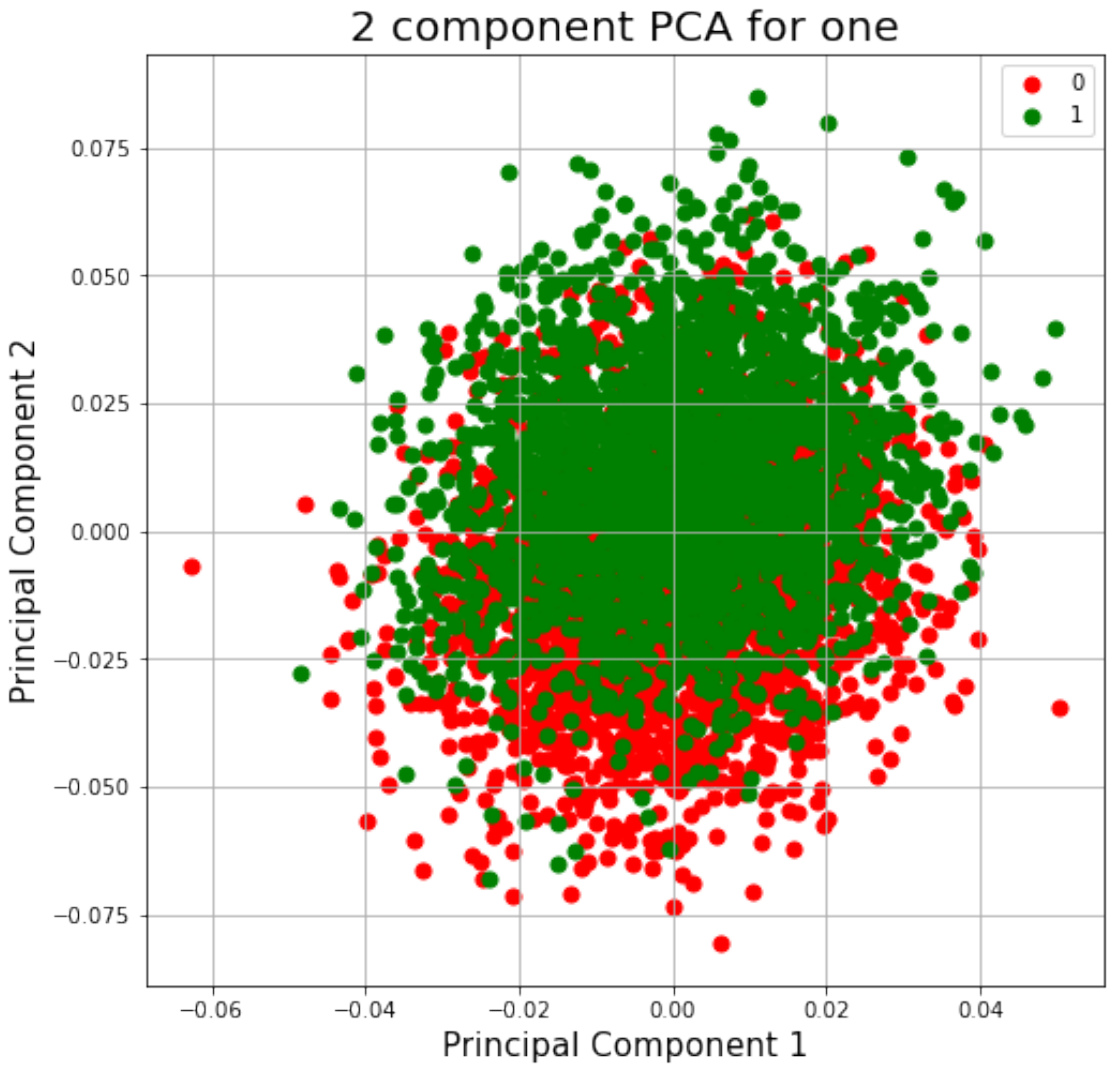
1. Use the starter code to load the three datasets one by one. Note that there are two unique values in y. **Visualize the features of X these datasets using (1) Top-2 PCA components, and (2) 2-dimensional random projections. Use the code to project the features to 2 dimensions and then scatter plot the 2 features with a different color for each class**. Note that you will obtain 2 plots for each dataset (total 6 plots for this part). **Do you observe a difference in PCA vs random projections? Do you see a trend in the three datasets?**

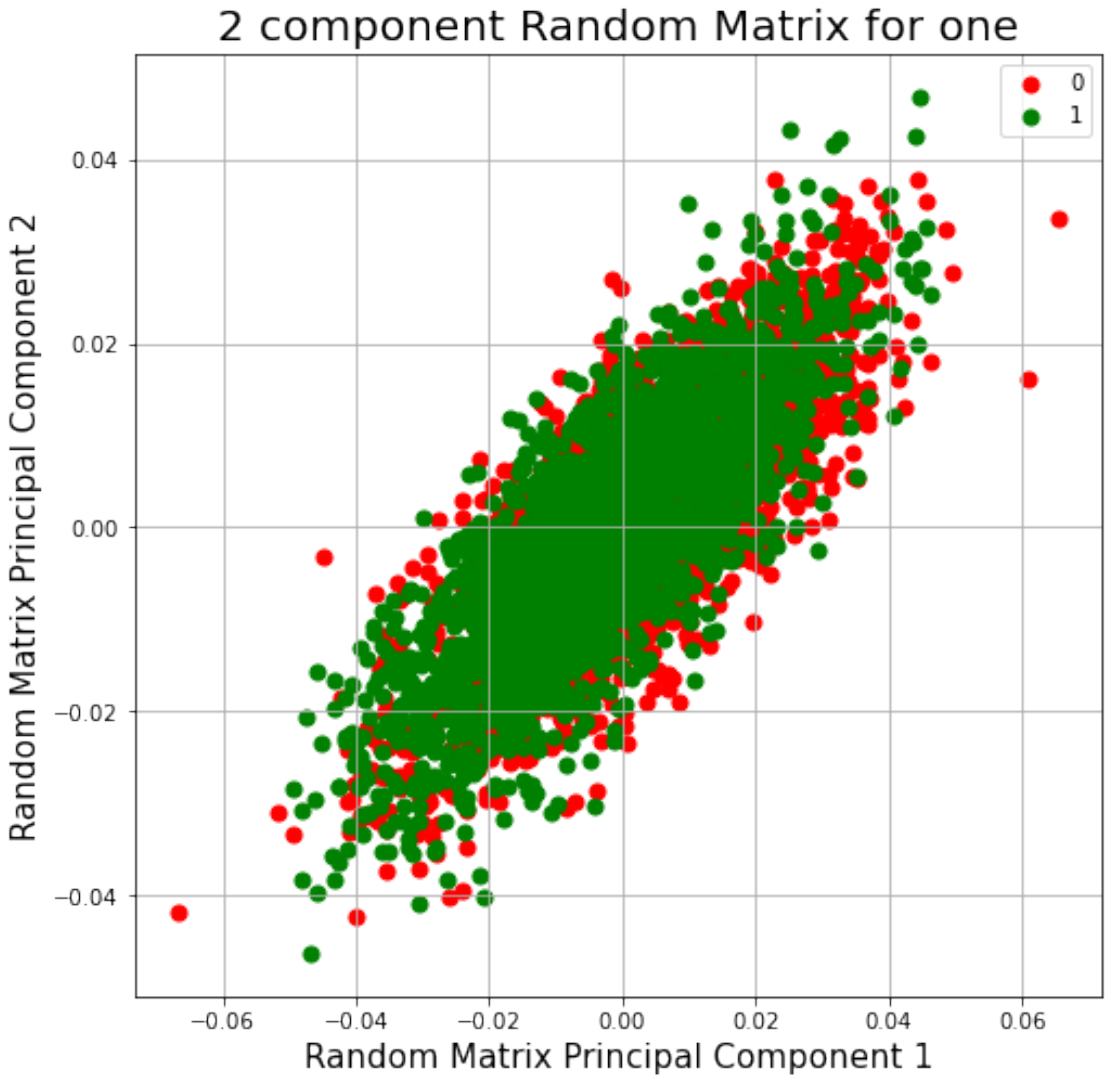
```
## Random Projections ##
def random_matrix(d, k):
    '''
    d = original dimension
    k = projected dimension
    '''
    #TODO
    J = np.random.rand(k, d)
    return J
```

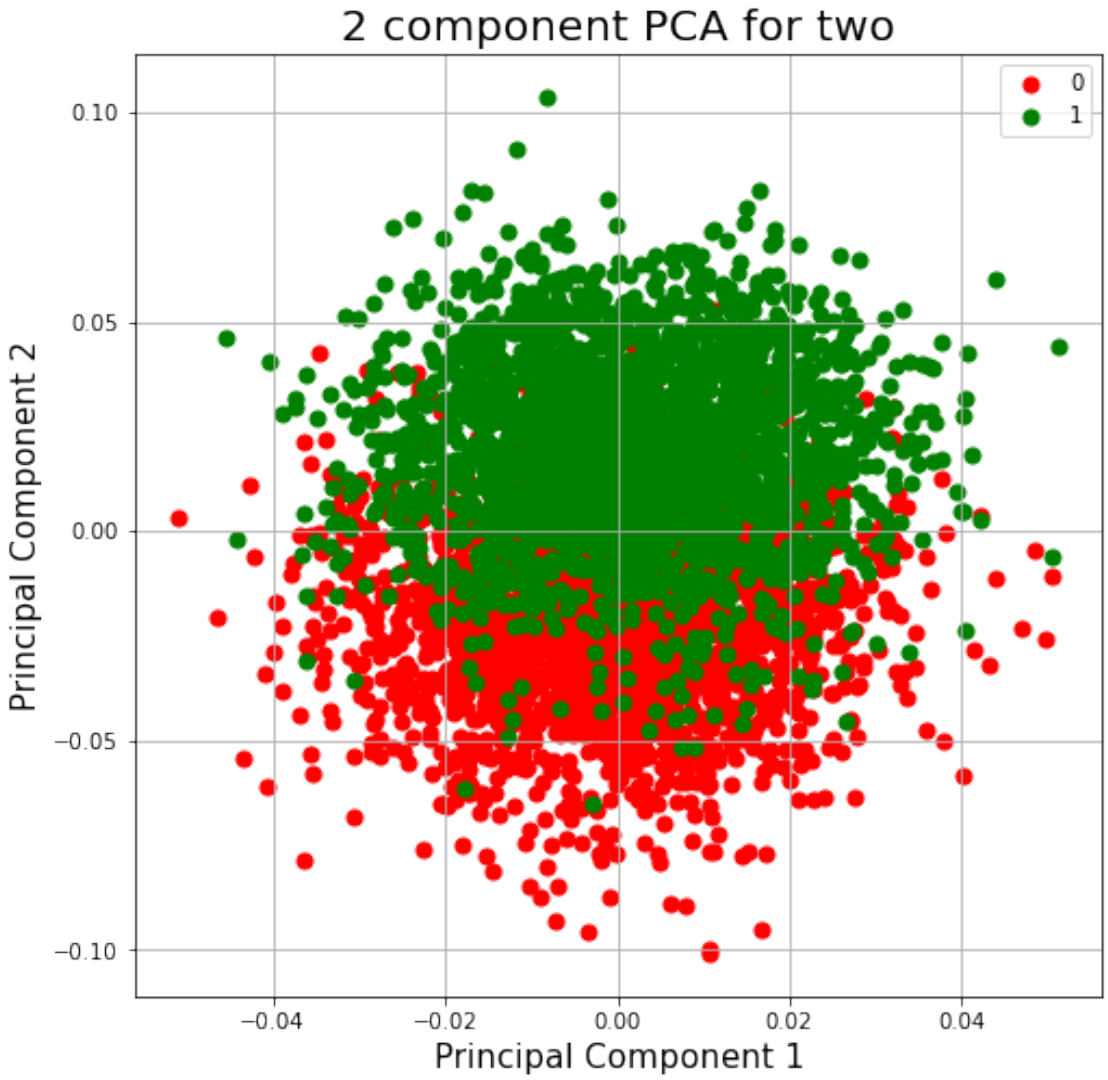
```
def random_proj(X, k):
    #TODO
    ws = []
    J = random_matrix(X.shape[1], k)
    for i in range(X.shape[0]):
        x = X[i]
        wj = 1/np.sqrt(k)*(np.matmul(J, x))
        ws.append(wj)
    return np.transpose(np.array(ws))

## PCA and projections ##
def my_pca(X, k):
    '''
    compute PCA components
    X = data matrix (each row as a sample)
    k = #principal components
    '''
    scaler = StandardScaler()
    X_norm = scaler.fit_transform(X)
    return pca_proj(X_norm, k)

def pca_proj(X, k):
    '''
    compute projection of matrix X
    along its first k principal components
    '''
    covar = np.matmul(np.transpose(X), X)
    klengthlist = (covar.shape[0]-k, covar.shape[0]-1)
    values, vectors = eigh(covar, subset_by_index = klengthlist)
    return np.transpose(vectors)

# to load the data:
data = np.load('./data1.npz')
X_one = data['X']
y_one = data['y']
n_one, d_one = X_one.shape

data = np.load('./data2.npz')
X_two = data['X']
y_two = data['y']
n_two, d_two = X_two.shape

data = np.load('./data3.npz')
X_three = data['X']
y_three = data['y']
n_three, d_three = X_three.shape

n_trials = 10  # to average for accuracies over random projections
```

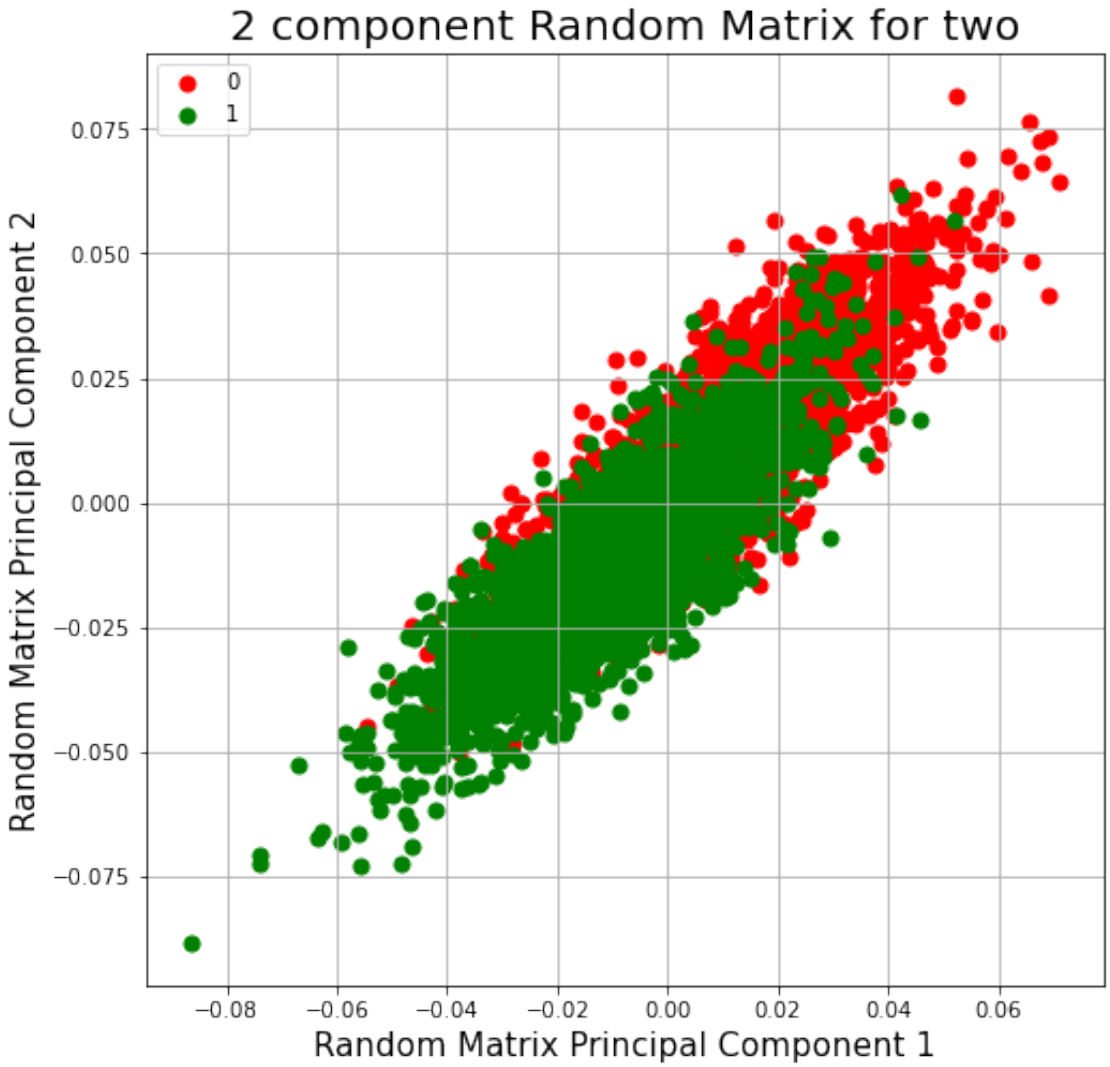
```
# Using PCA and Random Projection for:
# Visualizing the datasets
def visualize(X, y, datasetNumber):
    #pca
    pca = my_pca(X, 2)
    projected_features = np.matmul(pca, np.transpose(X))

    new_coordinates = np.transpose(np.vstack((projected_features, y)))
    df = pd.DataFrame(data = new_coordinates, columns = ['principal
component 1', 'principal component 2', 'labels'])

    fig = plt.figure(figsize = (8,8))
    ax = fig.add_subplot(1,1,1)
    ax.set_xlabel('Principal Component 1', fontsize = 15)
    ax.set_ylabel('Principal Component 2', fontsize = 15)
    ax.set_title('2 component PCA for ' + datasetNumber, fontsize =
20)
    labels = [1, -1]
    colors = ['r', 'g']
    for label, color in zip(labels,colors):
        indicesToKeep = df['labels'] == label
        ax.scatter(df.loc[indicesToKeep, 'principal component 1']
                   , df.loc[indicesToKeep, 'principal component 2']
                   , c = color
                   , s = 50)
    ax.legend(targets)
    ax.grid()

    #random
    rand = random_proj(X, 2)

    new_coordinates = np.transpose(np.vstack((rand, y)))
    df = pd.DataFrame(data = new_coordinates, columns = ['principal
component 1', 'principal component 2', 'labels'])

    fig = plt.figure(figsize = (8,8))
    ax = fig.add_subplot(1,1,1)
    ax.set_xlabel('Random Matrix Principal Component 1', fontsize =
15)
    ax.set_ylabel('Random Matrix Principal Component 2', fontsize =
15)
    ax.set_title('2 component Random Matrix for ' + datasetNumber,
fontsize = 20)
    labels = [1, -1]
    colors = ['r', 'g']
    for label, color in zip(labels,colors):
        indicesToKeep = df['labels'] == label
        ax.scatter(df.loc[indicesToKeep, 'principal component 1']
                   , df.loc[indicesToKeep, 'principal component 2']
                   , c = color
```

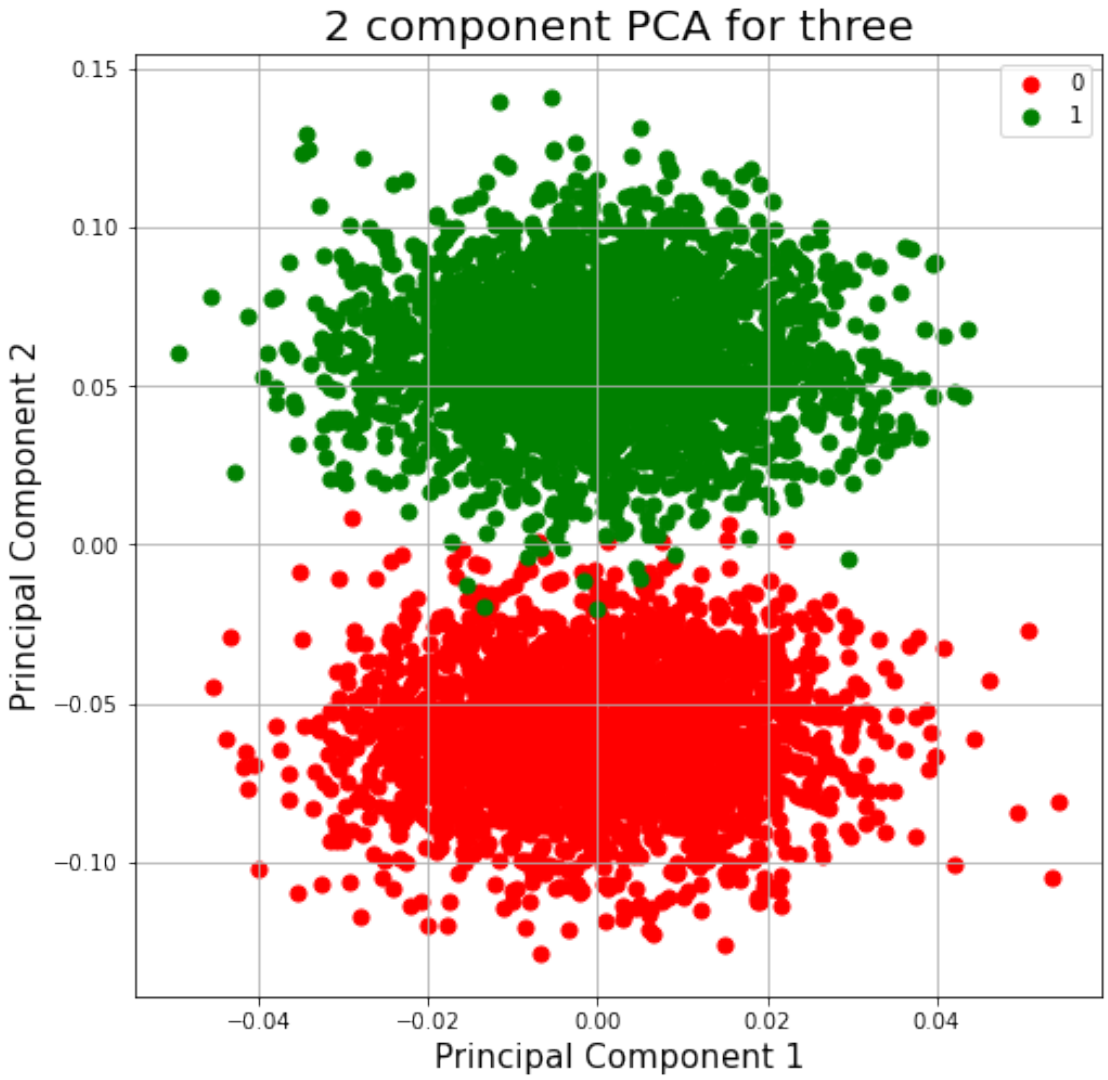
```
                    , s = 50)
    ax.legend(targets)
    ax.grid()

visualize(X_one, y_one, "one")
visualize(X_two, y_two, "two")
visualize(X_three, y_three, "three")

# plt.plot, plt.scatter would be useful for plotting
```

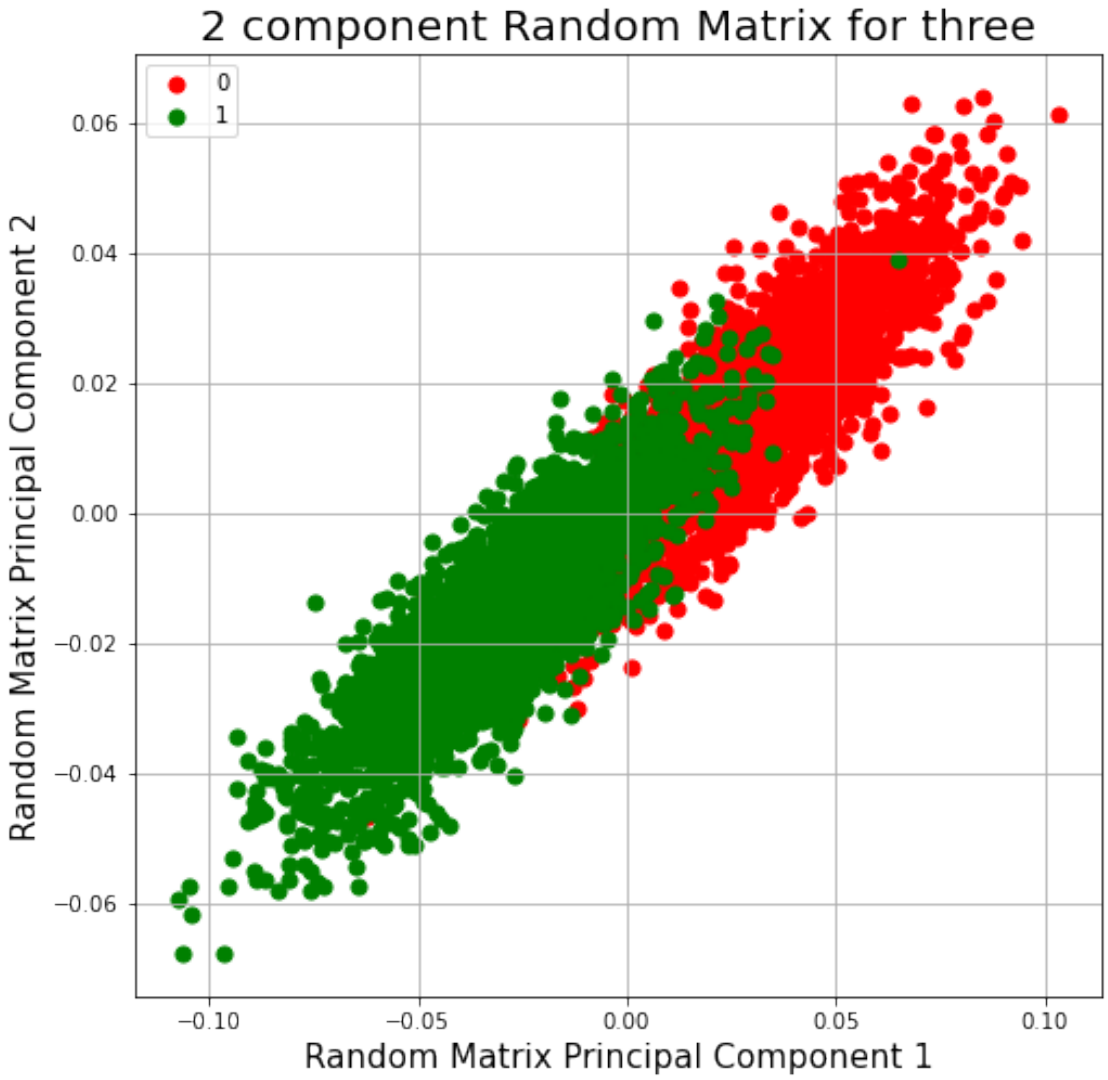



2 component Random Matrix for one
0
1
0.04
0.02
0.00
−0.02
−0.04
Random Matrix Principal Component 2
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
Random Matrix Principal Component 1


2 component PCA for two
0
1
0.10
0.05
0.00
−0.05
−0.10
Principal Component 2
−0.04
−0.02
0.00
0.02
0.04
Principal Component 1


2 component Random Matrix for two
0
1
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.08
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
Random Matrix Principal Component 2
Random Matrix Principal Component 1


2 component PCA for three
Principal Component 2
Principal Component 1
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.04
−0.02
0.00
0.02
0.04
0
1



There is a clearly observable difference in PCA vs random because while the PCA is more centered, with both components gravitating towards the origin and forming a ring-like shape around it, the random matrix is more linear. However, we can see a trend in the data. For the two PCA principal components (let us call them A and B), if A and B are very similar and overlap heavily, then the linearity will be very similar for the random matrix. If A is distinctly above B (Component 2 is where there is a distinct difference), then the random matrix will show this difference in linearity with a horizontal gap between the two groups. This is clearly seen in dataset 3.

1. For each dataset, we will now fit a linear model on different projections of features to perform classification. The code for fitting a linear model with projected features and predicting a label for a given feature, is given to you. Use these functions and write a code that does prediction in the following way: (1) Use top k-PCA features to obtain one set of results, and (2) Use k-dimensional random projection to obtain the second set of results (take average accuracy over 10 random projections for smooth curves). Use the projection functions given in the starter code to select these

features. You should vary k from 1 to d where d is the dimension of each feature xi. **Plot the accuracy for PCA and Random projection as a function of k. Comment on the observations on these accuracies as a function of k and across different datasets.**

```
######### LINEAR MODEL FITTING ############

def rand_proj_accuracy_split(X, y, k):
    '''
    Fitting a k dimensional feature set obtained
    from random projection of X, versus y
    for binary classification for y in {-1, 1}
    '''
    # test train split
    _, d = X.shape
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)

    # random projection
    J = np.transpose(random_matrix(d, k))
    rand_proj_X = X_train.dot(J)

    # fit a linear model
    line = sklearn.linear_model.LinearRegression(fit_intercept=False)
    line.fit(rand_proj_X, y_train)

    # predict y
    y_pred=line.predict(X_test.dot(J))

    # return the test error
    return 1-np.mean(np.sign(y_pred)!= y_test)

def pca_proj_accuracy(X, y, k):
    '''
    Fitting a k dimensional feature set obtained
    from PCA projection of X, versus y
    for binary classification for y in {-1, 1}
    '''
    # test-train split
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20)

    # pca projection
    P = np.transpose(my_pca(X_train, k))
    pca_proj_X = X_train.dot(P)

    # fit a linear model
    line = sklearn.linear_model.LinearRegression(fit_intercept=False)
    line.fit(pca_proj_X, y_train)
```

```
     # predict y
    y_pred=line.predict(X_test.dot(P))

    # return the test error
    return 1-np.mean(np.sign(y_pred)!= y_test)

# Computing the accuracies over different datasets.

def plotAccuracy(X, y, d, datasetNumber):
    #TODO
    pca_acc = []
    rand_acc = []
    for i in range(1, d):
        randacc = 0
        for j in range(10):
            randacc += rand_proj_accuracy_split(X, y, i)
        randacc = randacc/10
        rand_acc.append(randacc)
        pca_acc.append(pca_proj_accuracy(X, y, i))

    fig = plt.figure(figsize = (8,8))
    ax = fig.add_subplot(2,2,1)
    ax.set_xlabel('k', fontsize = 15)
    ax.set_ylabel('Accuracy', fontsize = 15)
    ax.set_title('Accuracy of PCA Matrix for ' + datasetNumber,
fontsize = 20)
    k = np.arange(1, d)
    ax.plot(k, pca_acc)

    ax2 = fig.add_subplot(2,2,2)
    ax2.set_xlabel('k', fontsize = 15)
    ax2.set_ylabel('Accuracy', fontsize = 15)
    ax2.set_title('Accuracy of Random Matrix for ' + datasetNumber,
fontsize = 20)
    k = np.arange(1, d)
    ax2.plot(k, rand_acc)
    ax.grid()
    ax2.grid()
    plt.subplots_adjust(left=-0.6, right=0.6)

plotAccuracy(X_one, y_one, d_one, "one")
plotAccuracy(X_two, y_two, d_two, "two")
plotAccuracy(X_three, y_three, d_three, "three")

# Don't forget to average the accuracy for multiple
# random projections to get a smooth curve.
```

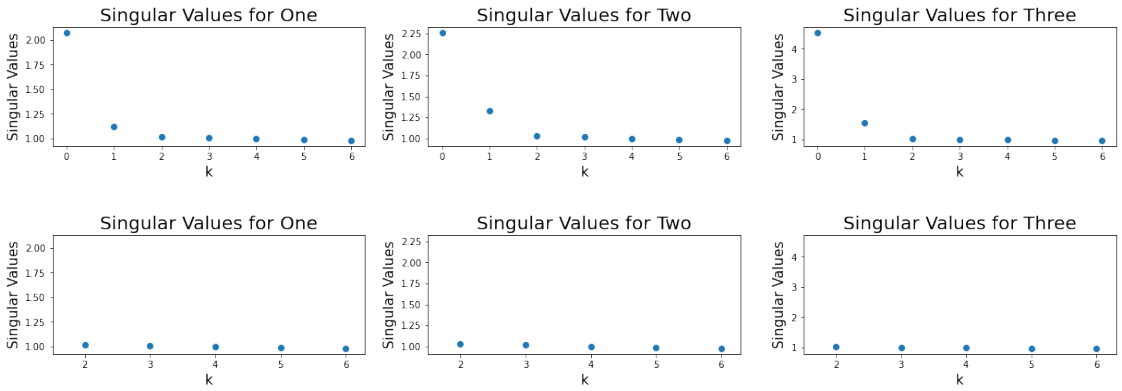
For each dataset, we can say that there is a specific value of k that will result in the highest possible accuracy. For one, that is 3; for two that is 4; and for three that is 5. The random matrix results in a consistent increase in accuracy as k increases, while the PCA is more sporadic.

1. Now plot the singular values for the feature matrices of the three datasets. **Do you observe a pattern across the three datasets? How does it help to explain the performance of PCA observed in the previous parts?**

```
#TODO
oneu, ones, onv = np.linalg.svd(X_one)
twou, twos, twov = np.linalg.svd(X_two)
threeu, threes, threev = np.linalg.svd(X_three)

fig = plt.figure(figsize = (8,8))
```

```
ax = fig.add_subplot(3,3,1)
ax.set_xlabel('k', fontsize = 15)
ax.set_ylabel('Singular Values', fontsize = 15)
ax.set_title('Singular Values for One', fontsize = 20)
k = np.arange(len(ones))
ax.scatter(k, ones)

ax2 = fig.add_subplot(3,3,2)
ax2.set_xlabel('k', fontsize = 15)
ax2.set_ylabel('Singular Values', fontsize = 15)
ax2.set_title('Singular Values for Two', fontsize = 20)
k = np.arange(len(ones))
ax2.scatter(k, twos)

ax3 = fig.add_subplot(3,3,3)
ax3.set_xlabel('k', fontsize = 15)
ax3.set_ylabel('Singular Values', fontsize = 15)
ax3.set_title('Singular Values for Three', fontsize = 20)
k = np.arange(len(ones))
ax3.scatter(k, threes)
plt.subplots_adjust(left=-1, right=1)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(3,3,1)
ax.set_xlabel('k', fontsize = 15)
ax.set_ylabel('Singular Values', fontsize = 15)
ax.set_title('Singular Values for One', fontsize = 20)
k = np.arange(len(ones))
ax.scatter(k, ones)
plt.xlim(left=1.5)

ax2 = fig.add_subplot(3,3,2)
ax2.set_xlabel('k', fontsize = 15)
ax2.set_ylabel('Singular Values', fontsize = 15)
ax2.set_title('Singular Values for Two', fontsize = 20)
k = np.arange(len(ones))
ax2.scatter(k, twos)
plt.xlim(left=1.5)

ax3 = fig.add_subplot(3,3,3)
ax3.set_xlabel('k', fontsize = 15)
ax3.set_ylabel('Singular Values', fontsize = 15)
ax3.set_title('Singular Values for Three', fontsize = 20)
k = np.arange(len(ones))
ax3.scatter(k, threes)
plt.subplots_adjust(left=-1, right=1)
plt.xlim(left=1.5)
```

```
(1.5, 6.3)
```



There are three graphs above, one for each dataset. Below that the same graph is shown, but only for singular values corresponding to k>1. We can see a clear decrease in the eigenvalues which aligns with the immediate spikes for k=1. Afterwards, we see a slight fluctuation for the singular values mapping from k=2 to k=6. This maps to the PCA accuracy graph still, such as when we see the trend of increasing/decreasing singular values equal the increasing/decreasing accuracy.