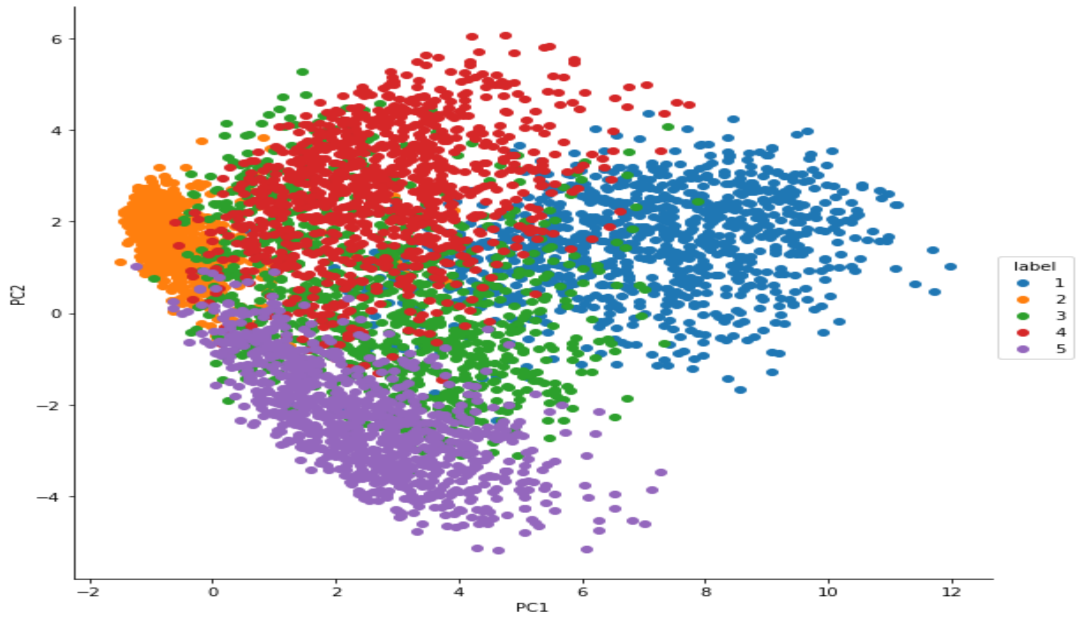
Machine Learning Homework 7
Dimension Reductions

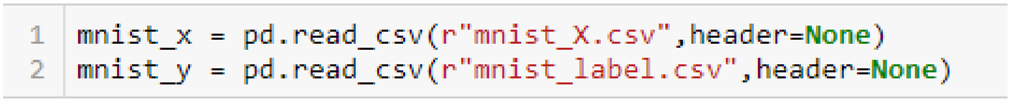
1. PRINCIPLE COMPONENT ANALYSIS (PCA)

- Visualization the first 2 principle components of mnist_X (5000) in 2D space



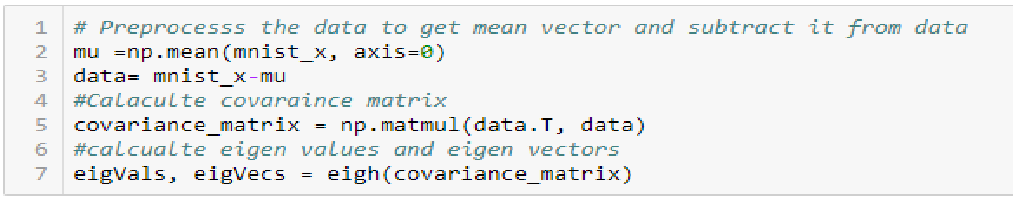
- PCA code explanations

✓ Load mnist_X and mnist_label

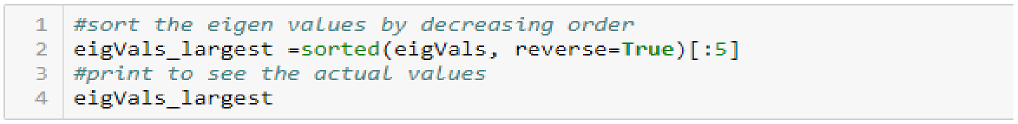


```
mnist_x = pd.read_csv(r"mnist_X.csv",header=None)
mnist_y = pd.read_csv(r"mnist_label.csv",header=None)
```

✓ Preprocess data, calculate covariance matrix, compute Eigen value and Eigen vectors

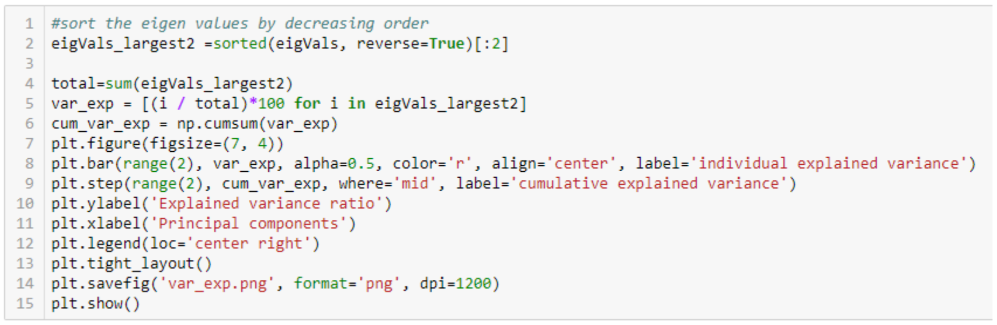


```
# Preprocesss the data to get mean vector and subtract it from data
mu =np.mean(mnist_x, axis=0)
data= mnist_x-mu
#Calaculte covaraince matrix
covariance_matrix = np.matmul(data.T, data)
#calcualte eigen values and eigen vectors
eigVals, eigVecs = eigh(covariance_matrix)
```



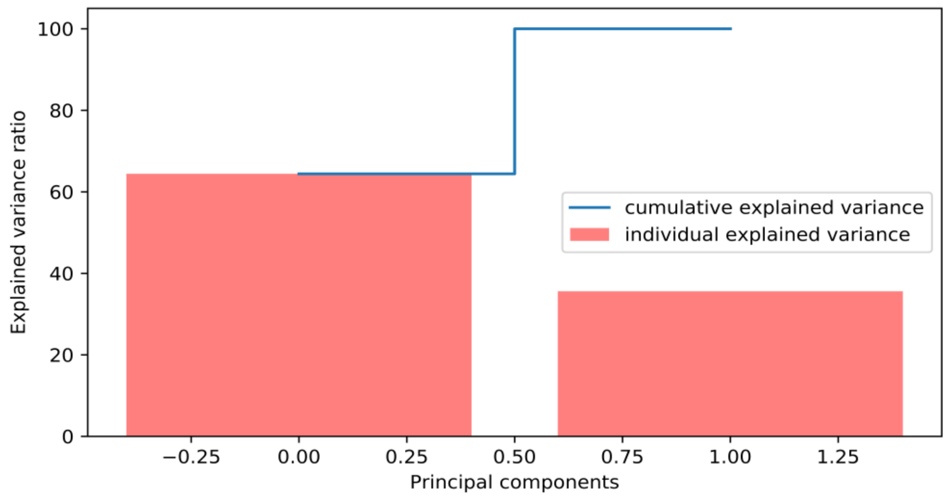
```
#sort the eigen values by decreasing order
eigVals_largest =sorted(eigVals, reverse=True)[:5]
#print to see the actual values
eigVals_largest
```

- ✓ The explained variance tells us how much information (variance) can be attributed to each of the principal components.

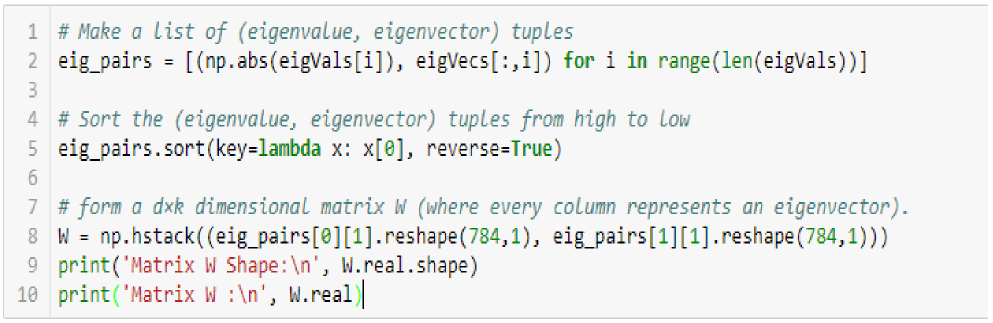


```python
#sort the eigen values by decreasing order
eigVals_largest2 =sorted(eigVals, reverse=True)[:2]

total=sum(eigVals_largest2)
var_exp = [(i / total)*100 for i in eigVals_largest2]
cum_var_exp = np.cumsum(var_exp)
plt.figure(figsize=(7, 4))
plt.bar(range(2), var_exp, alpha=0.5, color='r', align='center', label='individual explained variance')
plt.step(range(2), cum_var_exp, where='mid', label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='center right')
plt.tight_layout()
plt.savefig('var_exp.png', format='png', dpi=1200)
plt.show()
```

- ✓ The first 2 principle components corresponding to largest 2 Eigen values explain 99% of information in the given data.

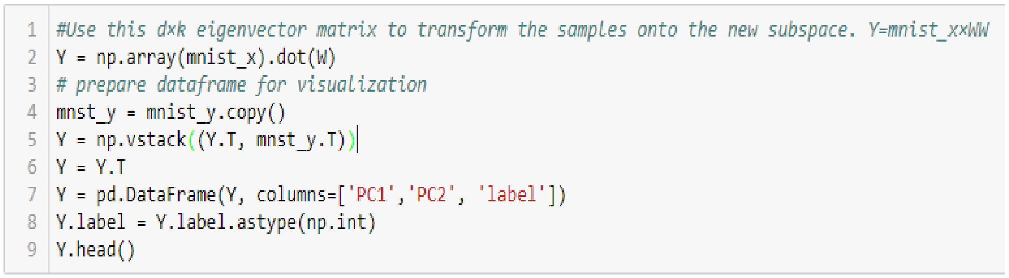


- ✓ Form a d×k dimensional matrix W (where every column represents an eigenvector)



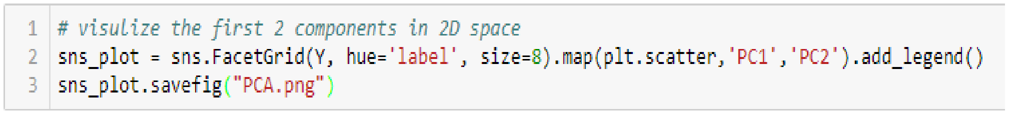
```python
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eigVals[i]), eigVecs[:,i]) for i in range(len(eigVals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# form a dxk dimensional matrix W (where every column represents an eigenvector).
W = np.hstack((eig_pairs[0][1].reshape(784,1), eig_pairs[1][1].reshape(784,1)))
print('Matrix W Shape:\n', W.real.shape)
print('Matrix W :\n', W.real)
```

✓ Use this d×k eigenvector matrix to transform the samples onto the new subspace. Y=mnist_x×W



```
#Use this dxk eigenvector matrix to transform the samples onto the new subspace. Y=mnist_x×W
Y = np.array(mnist_x).dot(W)
# prepare dataframe for visualization
mnst_y = mnist_y.copy()
Y = np.vstack((Y.T, mnst_y.T))
Y = Y.T
Y = pd.DataFrame(Y, columns=['PC1','PC2', 'label'])
Y.label = Y.label.astype(np.int)
Y.head()
```

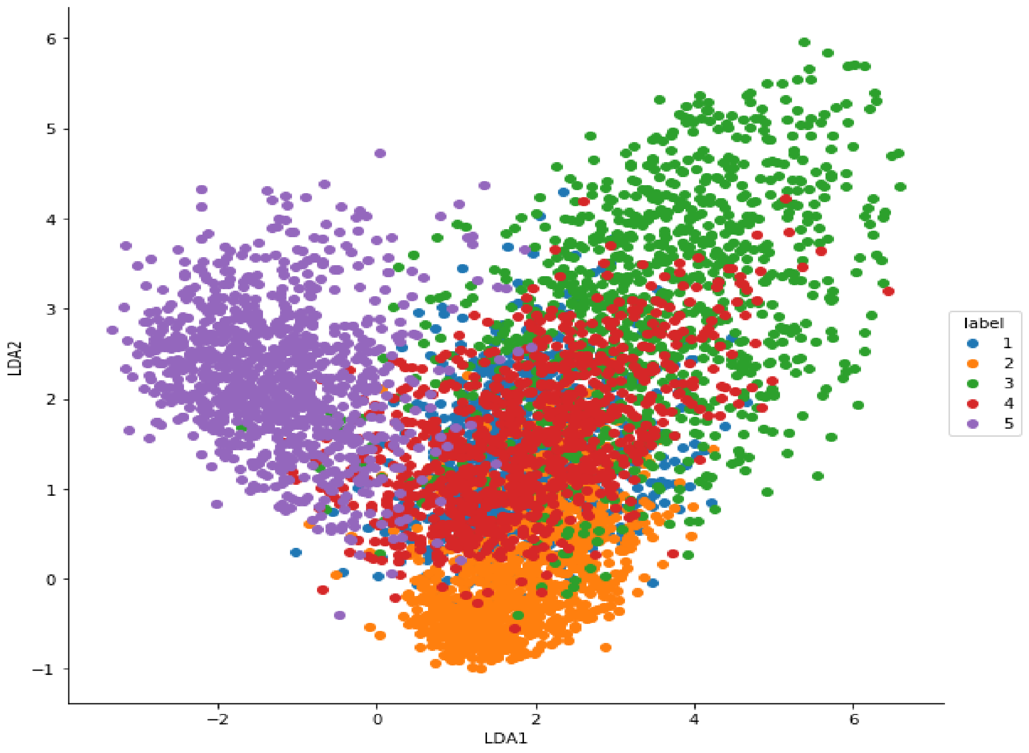
✓ Visualize the first 2 components in 2D space using seaborn.FacetGrid function



```
# visulize the first 2 components in 2D space
sns_plot = sns.FacetGrid(Y, hue='label', size=8).map(plt.scatter,'PC1','PC2').add_legend()
sns_plot.savefig("PCA.png")
```

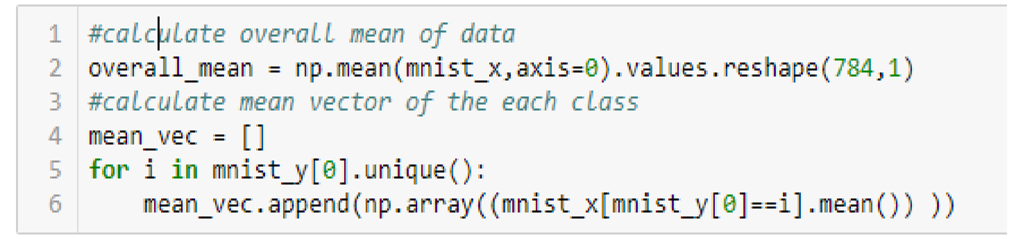
2. Linear Discriminant Analysis (LDA)

- Visualization of the first 2 LDA components of mnist_X (5000) in 2D space



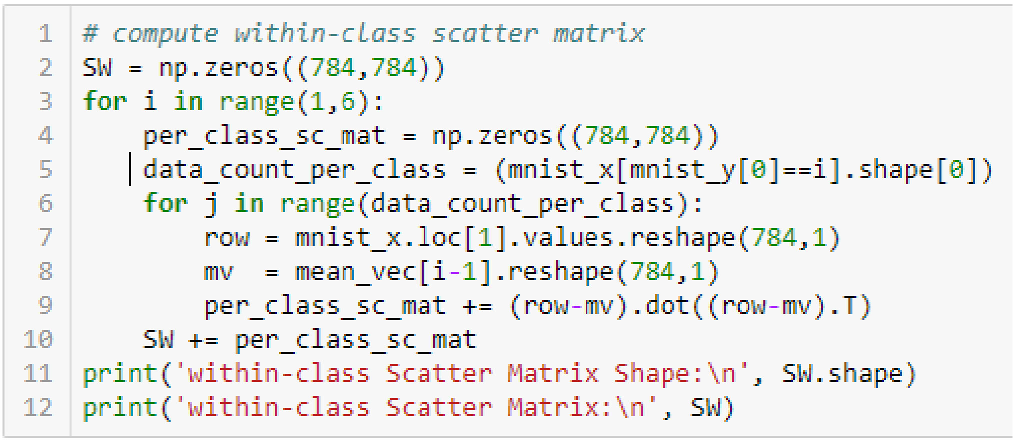
- LDA code explanations

  ✓ Calculate overall mean of data and calculate mean vector of each class



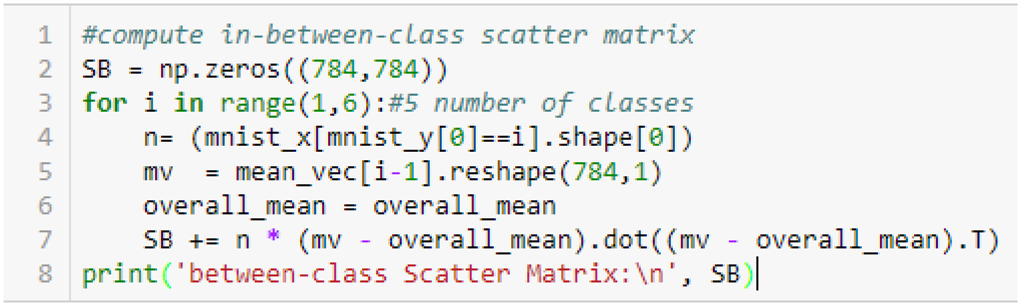
```python
#calculate overall mean of data
overall_mean = np.mean(mnist_x,axis=0).values.reshape(784,1)
#calculate mean vector of the each class
mean_vec = []
for i in mnist_y[0].unique():
    mean_vec.append(np.array((mnist_x[mnist_y[0]==i].mean()) ))
```

  ✓ Compute within-class scatter matrix



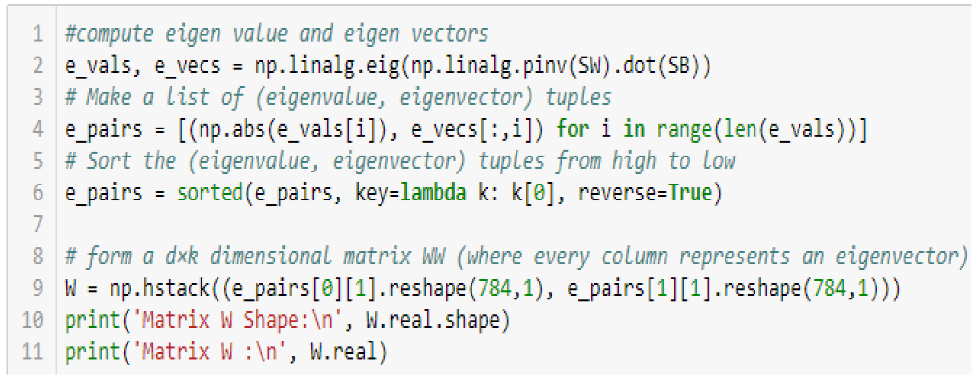
```python
# compute within-class scatter matrix
SW = np.zeros((784,784))
for i in range(1,6):
    per_class_sc_mat = np.zeros((784,784))
    data_count_per_class = (mnist_x[mnist_y[0]==i].shape[0])
    for j in range(data_count_per_class):
        row = mnist_x.loc[1].values.reshape(784,1)
        mv  = mean_vec[i-1].reshape(784,1)
        per_class_sc_mat += (row-mv).dot((row-mv).T)
    SW += per_class_sc_mat
print('within-class Scatter Matrix Shape:\n', SW.shape)
print('within-class Scatter Matrix:\n', SW)
```

  ✓ Compute in-between-class scatter matrix



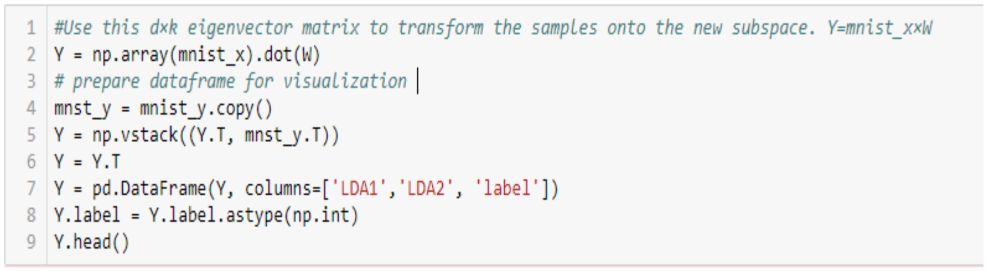
```python
#compute in-between-class scatter matrix
SB = np.zeros((784,784))
for i in range(1,6):#5 number of classes
    n= (mnist_x[mnist_y[0]==i].shape[0])
    mv  = mean_vec[i-1].reshape(784,1)
    overall_mean = overall_mean
    SB += n * (mv - overall_mean).dot((mv - overall_mean).T)
print('between-class Scatter Matrix:\n', SB)
```

- ✓ Compute Eigen value and Eigen vectors and pairs of Eigen value and Eigen vectors and form a d×k dimensional matrix W (where every column represents an eigenvector).



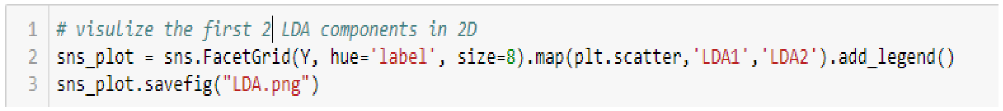
```python
#compute eigen value and eigen vectors
e_vals, e_vecs = np.linalg.eig(np.linalg.pinv(SW).dot(SB))
# Make a list of (eigenvalue, eigenvector) tuples
e_pairs = [(np.abs(e_vals[i]), e_vecs[:,i]) for i in range(len(e_vals))]
# Sort the (eigenvalue, eigenvector) tuples from high to low
e_pairs = sorted(e_pairs, key=lambda k: k[0], reverse=True)

# form a dxk dimensional matrix WW (where every column represents an eigenvector)
W = np.hstack((e_pairs[0][1].reshape(784,1), e_pairs[1][1].reshape(784,1)))
print('Matrix W Shape:\n', W.real.shape)
print('Matrix W :\n', W.real)
```

- ✓ Use this d×k eigenvector matrix to transform the samples onto the new subspace. Y=mnist_x×W and prepare dataframe for visualization



```python
#Use this dxk eigenvector matrix to transform the samples onto the new subspace. Y=mnist_xxW
Y = np.array(mnist_x).dot(W)
# prepare dataframe for visualization
mnst_y = mnist_y.copy()
Y = np.vstack((Y.T, mnst_y.T))
Y = Y.T
Y = pd.DataFrame(Y, columns=['LDA1','LDA2', 'label'])
Y.label = Y.label.astype(np.int)
Y.head()
```

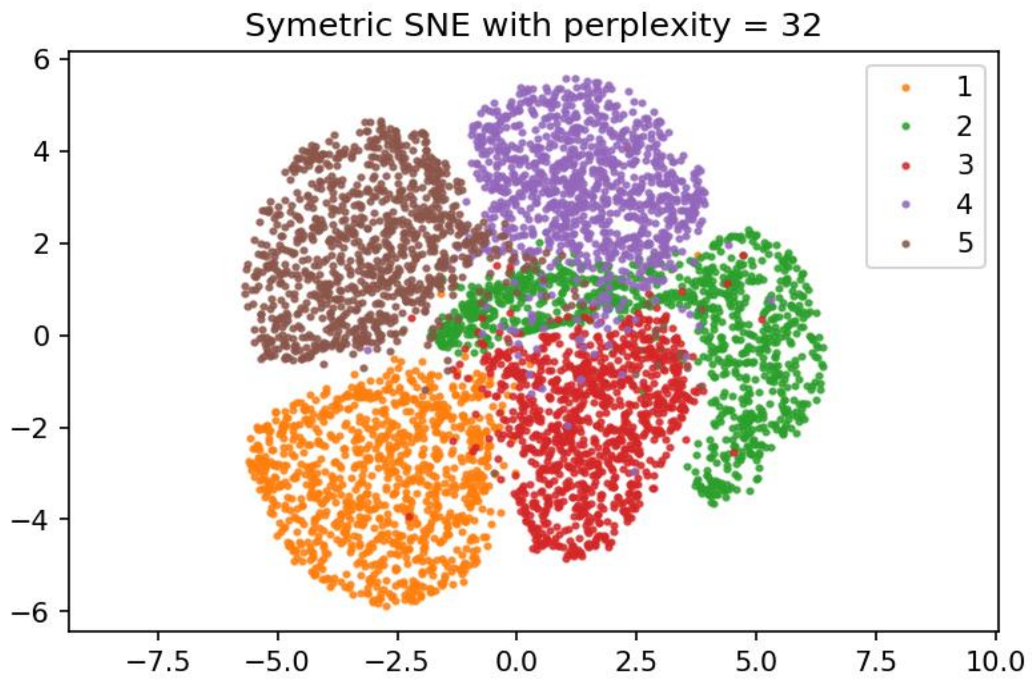
- ✓ Visualize the first 2 LDA components in 2D space using seaborn



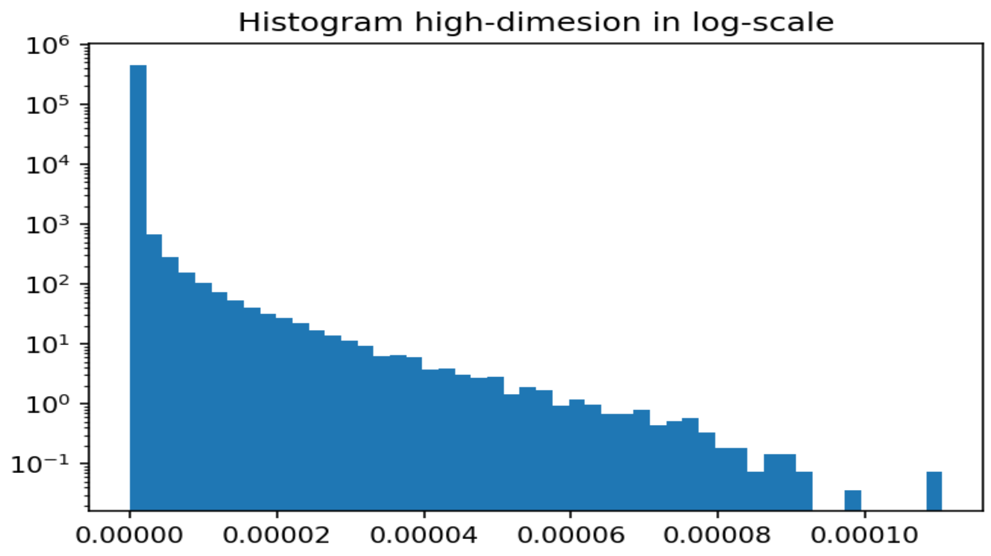
```python
# visulize the first 2 LDA components in 2D
sns_plot = sns.FacetGrid(Y, hue='label', size=8).map(plt.scatter,'LDA1','LDA2').add_legend()
sns_plot.savefig("LDA.png")
```

3. Symmetric SNE

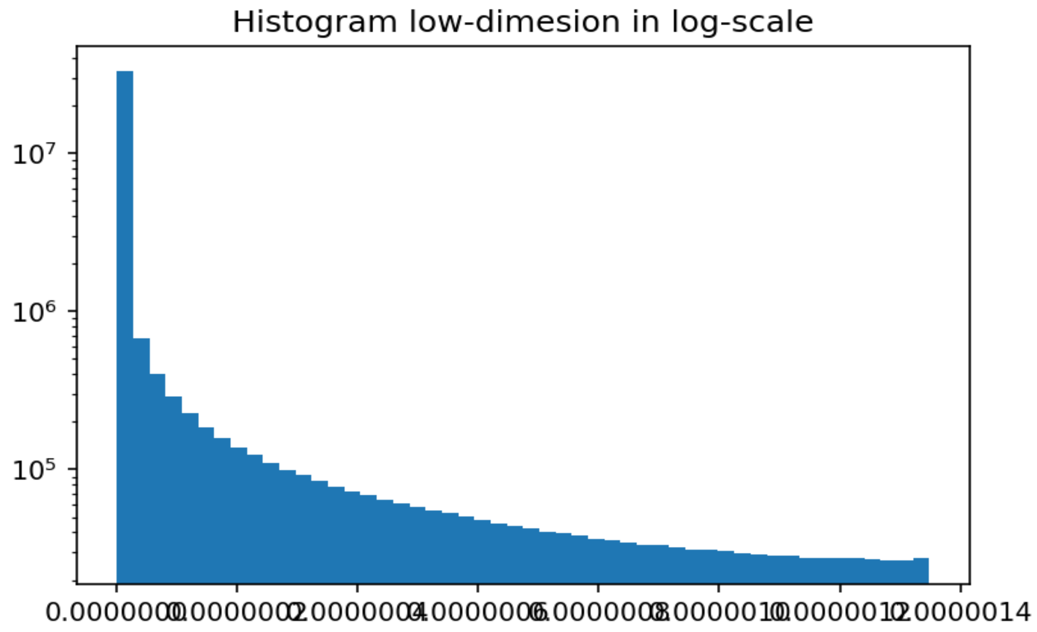
- Projection of data mnist_X.csv onto 2D space using symmetric SNE with perplexity=32



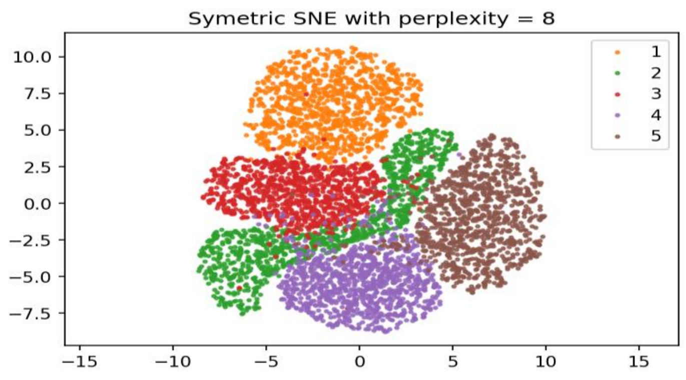
- Visualization of distribution of pairwise similarities in high-dimensional space, based on symmetric SNE

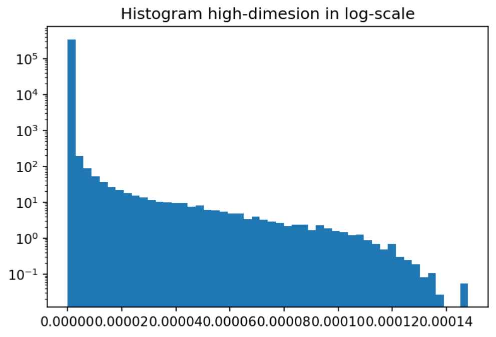


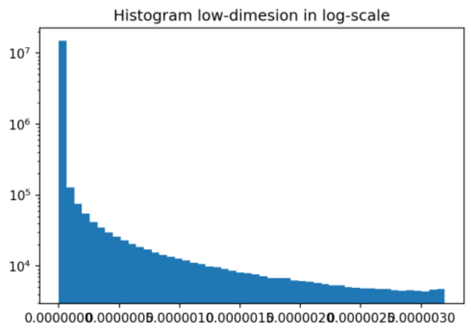
- Visualization of distribution of pairwise similarities in low-dimensional space, based on symmetric SNE

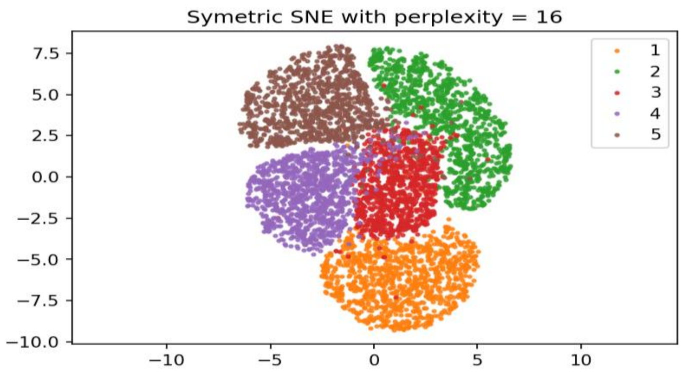


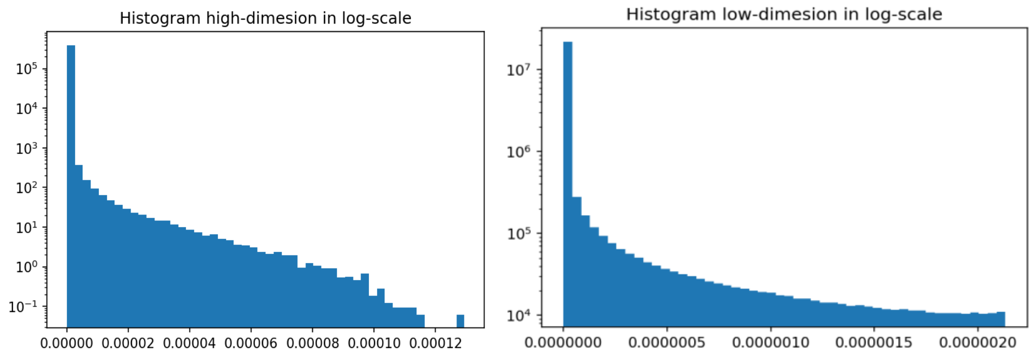
- I have also tried the other values of perplexity=8,16 for symmetric SNE





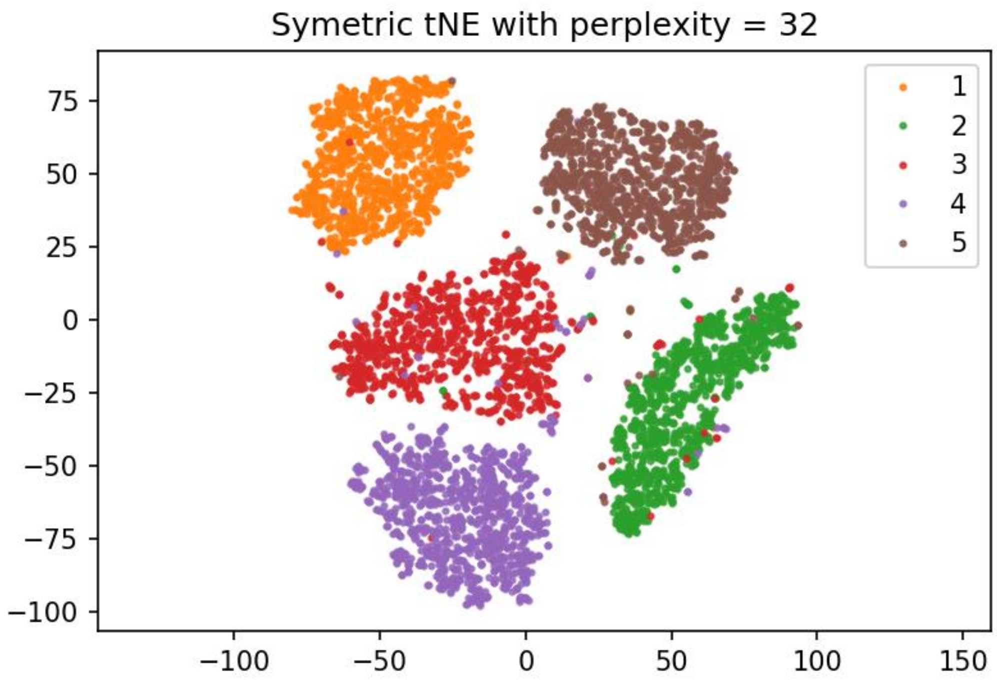




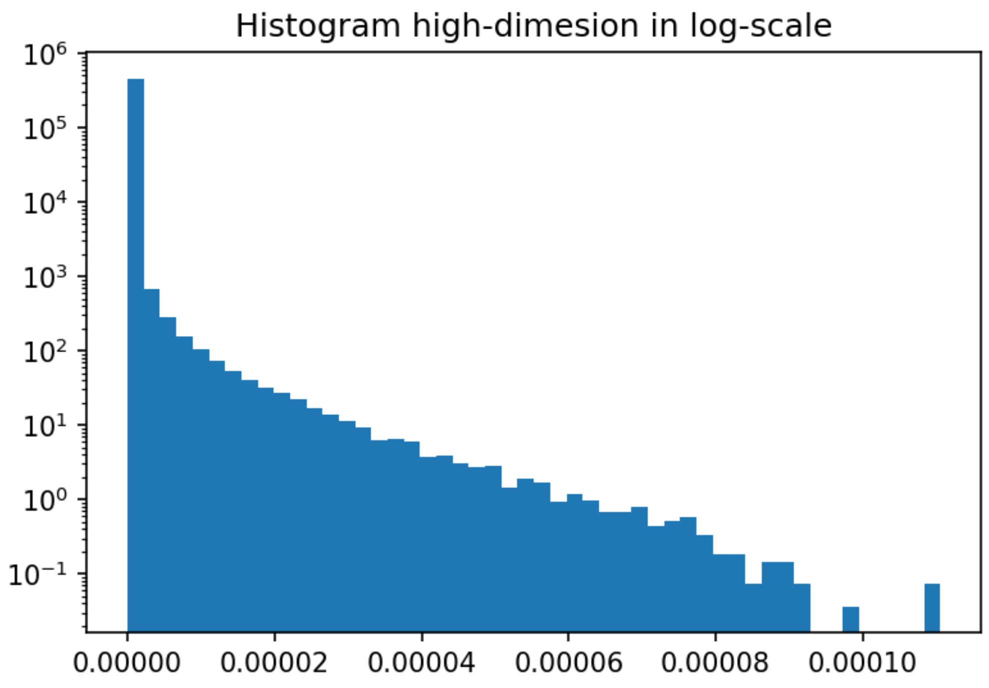


4. T-SNE

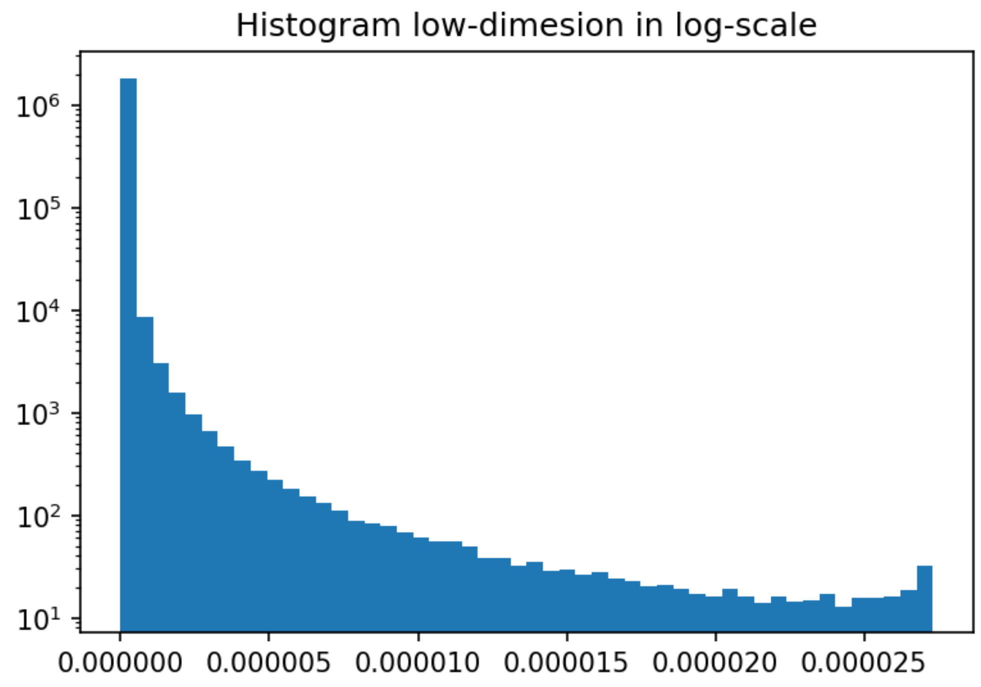
- Projection of data mnist_X.csv onto 2D space using T-SNE with perplexity=32



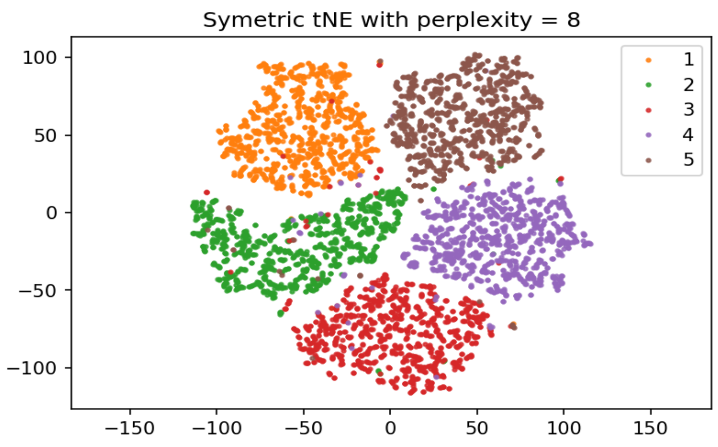
- Visualization of distribution of pairwise similarities in high-dimensional space, based on T-SNE

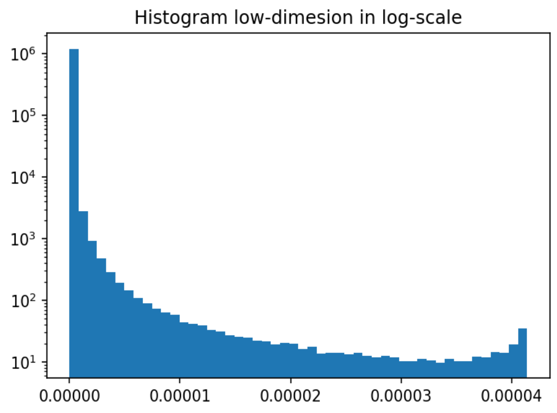


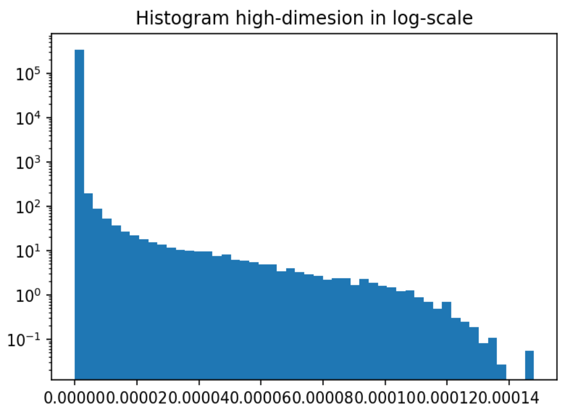
- Visualization of distribution of pairwise similarities in low-dimensional space, based on T-SNE

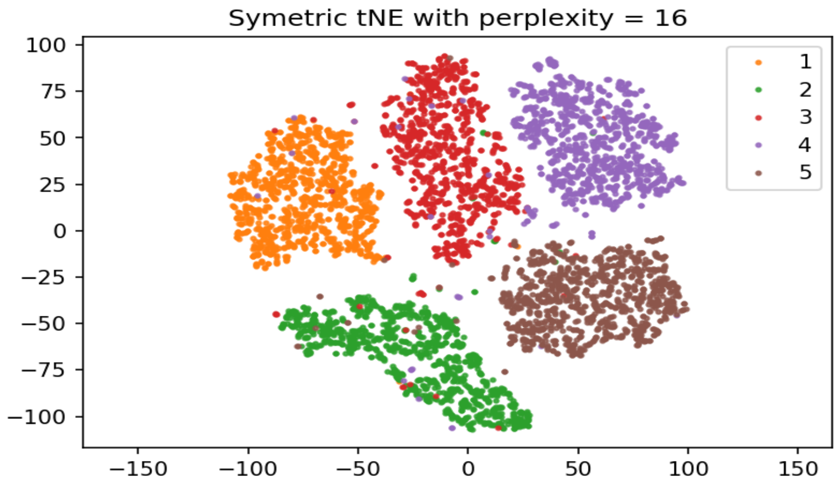


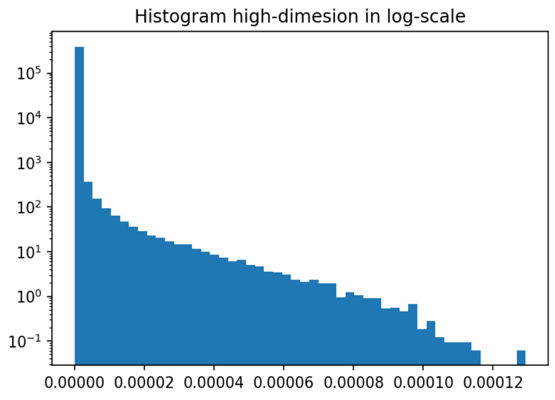
- I have also tried the other values of perplexity=8,16 for T-SNE

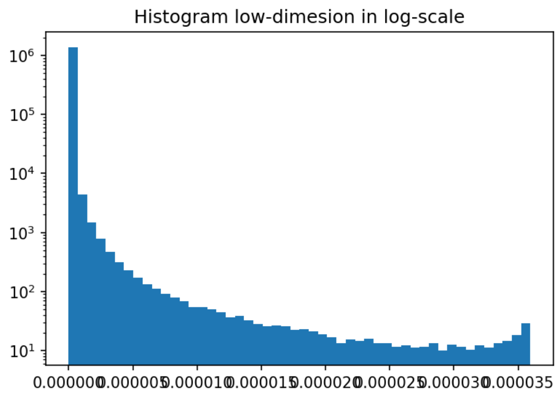












5. Differences between Symmetric SNE and T-SNE
    - The difference between t-SNE and symmetric SNE is the calculation of pairwise similarity in low-dimensional space. While t-SNE uses t-distribution (The letter t stands for stands for t-distribution), Symmetric SNE uses Gaussian distribution.

    - Pairwise similarity in low-dimensional space

        - T-SNE

$$q_{ij} = \frac{(1+ \| y_i - y_j \|^2)^{-1}}{\sum_{k \neq l}(1+ \| y_i - y_j \|^2)^{-1}}$$

        - Symmetric SNE

$$q_{ij} = \frac{\exp(- \| y_i - y_j \|^2)}{\sum_{k \neq l} \exp(- \| y_l - y_k \|^2)}$$

- The difference of the formula to compute the pairwise similarity in low-dimensional space leads to the different of the gradients.

    - T-SNE

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1+ \| y_i - y_j \|^2)^{-1}$$

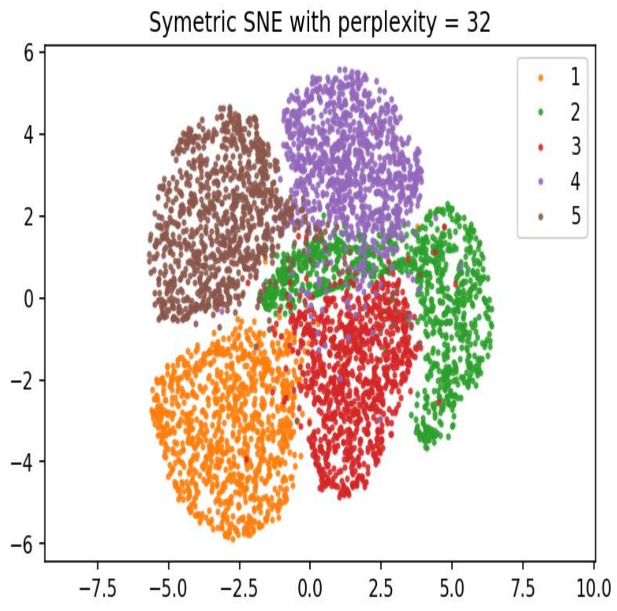
- Symmetric-SNE

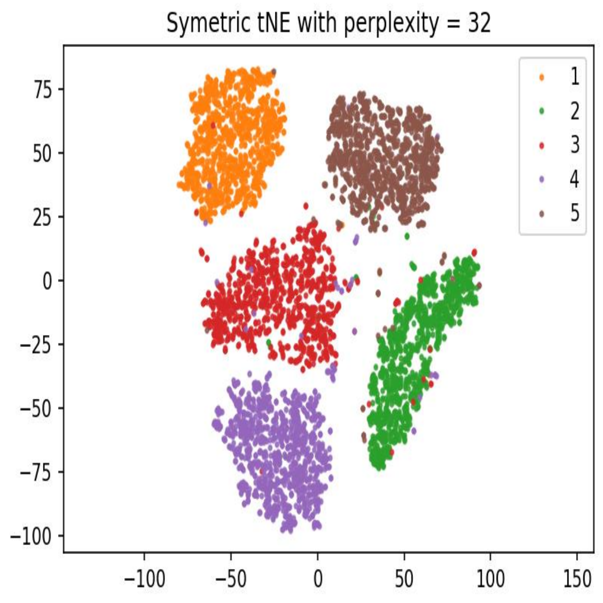
$$\frac{\partial C}{\partial y_i} = 2\sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

6. Discussion on the Projections of data using Symmetric SNE and t-SNE

- From the visualizations, we can see, there is an overlap between label “4”, “3”and “2” for Symmetric SNE but T-SNE solves this problem.
- The problem of symmetric SNE is that the data representation is crowded, while t-SNE solved this problem by using t-distribution.
- Comparing to PCA, Symmetric SNE is better in terms of visualization and separated the data. But however, t-SNE gives a better result than that of both SSNE and PCA.
- the gradient update of t-SNE has two advantages

    - For dissimilar points, using a smaller distance produces a larger gradient that repels these points
    - This rejection is not infinitely large (the denominator in the gradient), avoiding distances that are not similar that staying too far.

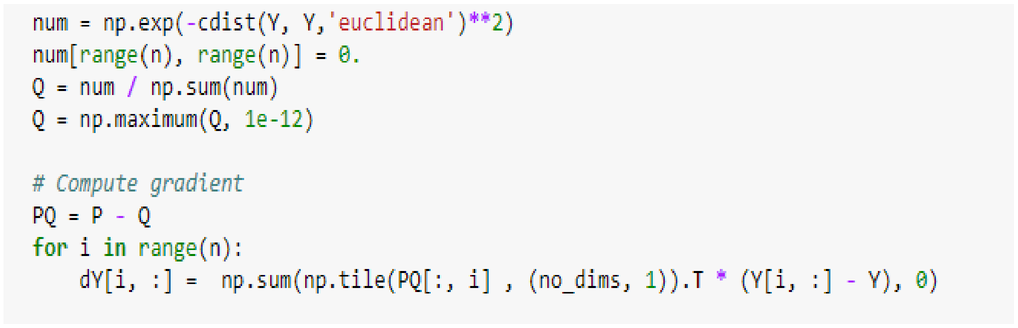




- Code Explanations for Symmetric SNE (**modifications for SSNE**)

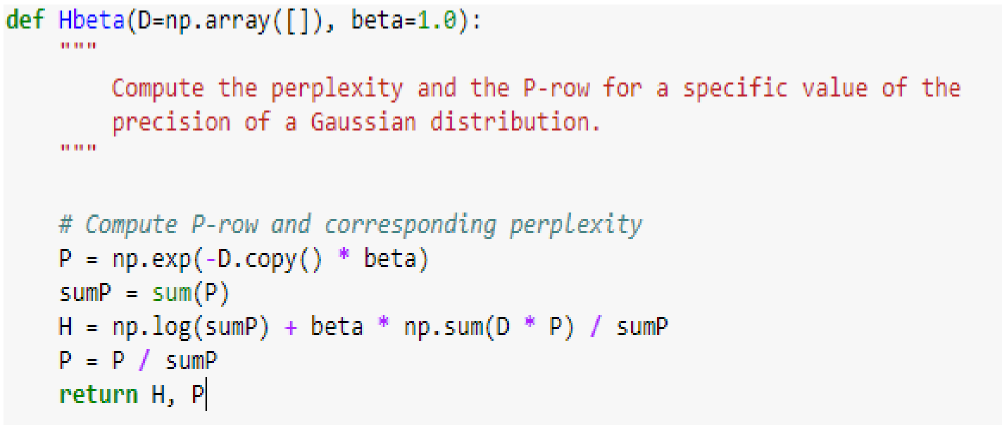
  ✓ Changing the code to compute Pairwise similarity in low-dimensional space in Symmetric SNE. Computing the Euclidean distance between the data in low-dimension to be Q value.

  ✓ Computing gradient with different formula



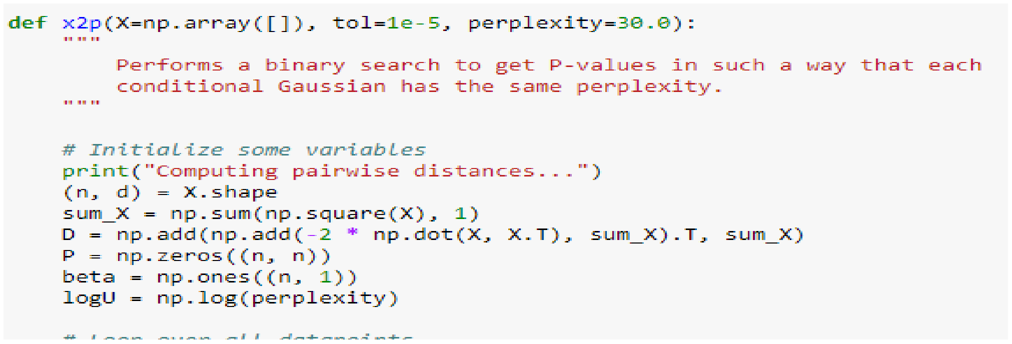
```python
num = np.exp(-cdist(Y, Y,'euclidean')**2)
num[range(n), range(n)] = 0.
Q = num / np.sum(num)
Q = np.maximum(Q, 1e-12)

# Compute gradient
PQ = P - Q
for i in range(n):
    dY[i, :] =  np.sum(np.tile(PQ[:, i] , (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

  ✓ Compute the perplexity and the P-row for a specific value of the precision of a Gaussian distribution.



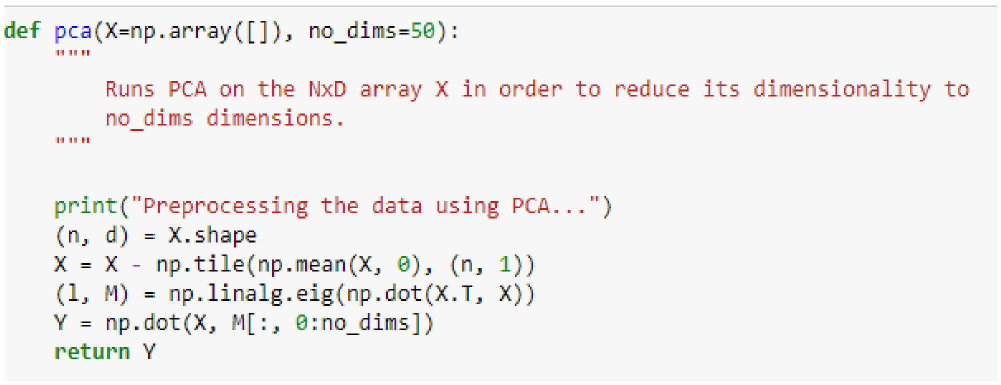
```python
def Hbeta(D=np.array([]), beta=1.0):
    """
        Compute the perplexity and the P-row for a specific value of the
        precision of a Gaussian distribution.
    """

    # Compute P-row and corresponding perplexity
    P = np.exp(-D.copy() * beta)
    sumP = sum(P)
    H = np.log(sumP) + beta * np.sum(D * P) / sumP
    P = P / sumP
    return H, P
```

  ✓ Performs a binary search to get P-values in such a way that each conditional Gaussian has the same perplexity.



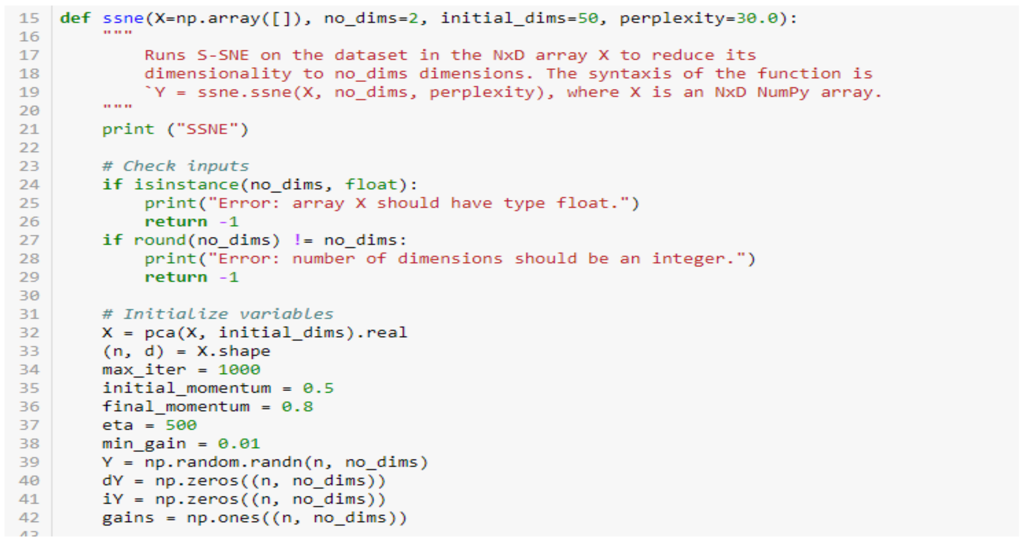
```python
def x2p(X=np.array([]), tol=1e-5, perplexity=30.0):
    """
        Performs a binary search to get P-values in such a way that each
        conditional Gaussian has the same perplexity.
    """

    # Initialize some variables
    print("Computing pairwise distances...")
    (n, d) = X.shape
    sum_X = np.sum(np.square(X), 1)
    D = np.add(np.add(-2 * np.dot(X, X.T), sum_X).T, sum_X)
    P = np.zeros((n, n))
    beta = np.ones((n, 1))
    logU = np.log(perplexity)

    # Loop over all datapoints
```

- ✓ Runs PCA on the NxD array X in order to reduce its dimensionality to no_dims dimensions.



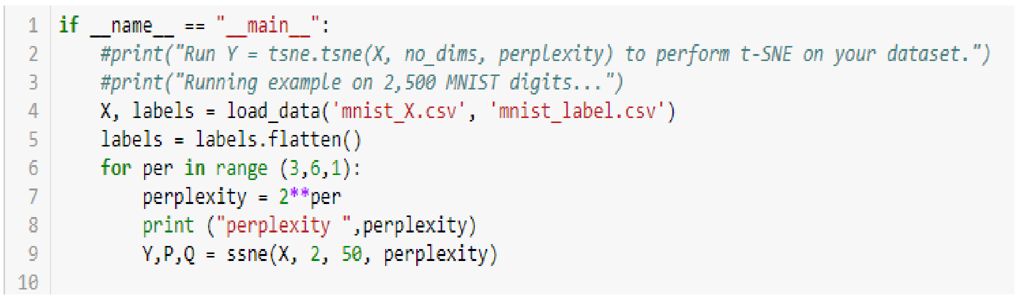
```python
def pca(X=np.array([]), no_dims=50):
    """
        Runs PCA on the NxD array X in order to reduce its dimensionality to
        no_dims dimensions.
    """

    print("Preprocessing the data using PCA...")
    (n, d) = X.shape
    X = X - np.tile(np.mean(X, 0), (n, 1))
    (l, M) = np.linalg.eig(np.dot(X.T, X))
    Y = np.dot(X, M[:, 0:no_dims])
    return Y
```

- ✓ Runs S-SNE on the dataset in the NxD array X to reduce its dimensionality to no_dims dimensions.



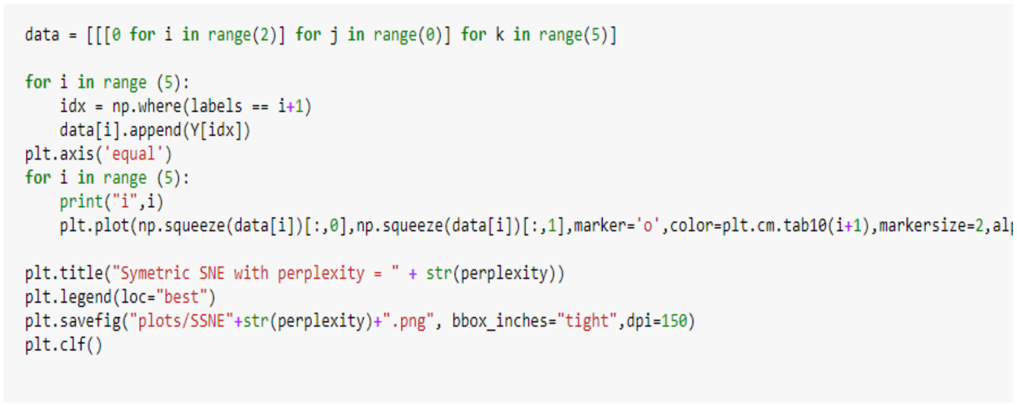
```python
def ssne(X=np.array([]), no_dims=2, initial_dims=50, perplexity=30.0):
    """
        Runs S-SNE on the dataset in the NxD array X to reduce its
        dimensionality to no_dims dimensions. The syntaxis of the function is
        `Y = ssne.ssne(X, no_dims, perplexity), where X is an NxD NumPy array.
    """
    print ("SSNE")

    # Check inputs
    if isinstance(no_dims, float):
        print("Error: array X should have type float.")
        return -1
    if round(no_dims) != no_dims:
        print("Error: number of dimensions should be an integer.")
        return -1

    # Initialize variables
    X = pca(X, initial_dims).real
    (n, d) = X.shape
    max_iter = 1000
    initial_momentum = 0.5
    final_momentum = 0.8
    eta = 500
    min_gain = 0.01
    Y = np.random.randn(n, no_dims)
    dY = np.zeros((n, no_dims))
    iY = np.zeros((n, no_dims))
    gains = np.ones((n, no_dims))
```

- ✓ Load data and call ssne function



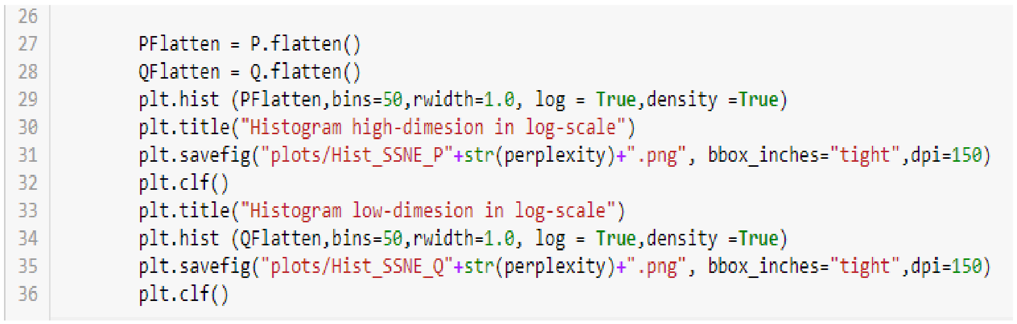
```python
if __name__ == "__main__":
    #print("Run Y = tsne.tsne(X, no_dims, perplexity) to perform t-SNE on your dataset.")
    #print("Running example on 2,500 MNIST digits...")
    X, labels = load_data('mnist_X.csv', 'mnist_label.csv')
    labels = labels.flatten()
    for per in range (3,6,1):
        perplexity = 2**per
        print ("perplexity ",perplexity)
        Y,P,Q = ssne(X, 2, 50, perplexity)

```

✓ Plot the projection in 2D for each value of perplexity



```
data = [[[0 for i in range(2)] for j in range(0)] for k in range(5)]

for i in range (5):
    idx = np.where(labels == i+1)
    data[i].append(Y[idx])
plt.axis('equal')
for i in range (5):
    print("i",i)
    plt.plot(np.squeeze(data[i])[:,0],np.squeeze(data[i])[:,1],marker='o',color=plt.cm.tab10(i+1),markersize=2,al

plt.title("Symetric SNE with perplexity = " + str(perplexity))
plt.legend(loc="best")
plt.savefig("plots/SSNE"+str(perplexity)+".png", bbox_inches="tight",dpi=150)
plt.clf()
```

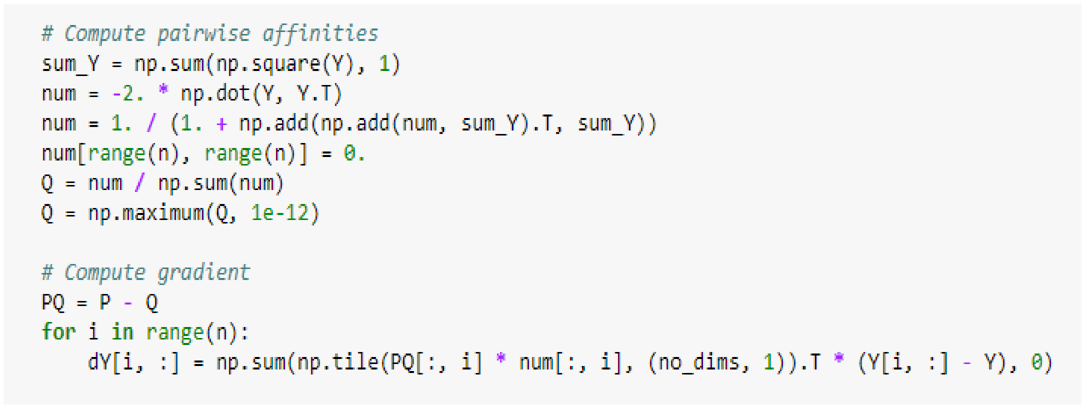
✓ Plot the distributions of data in both high-dimension and low-dimension



```

        PFlatten = P.flatten()
        QFlatten = Q.flatten()
        plt.hist (PFlatten,bins=50,rwidth=1.0, log = True,density =True)
        plt.title("Histogram high-dimesion in log-scale")
        plt.savefig("plots/Hist_SSNE_P"+str(perplexity)+".png", bbox_inches="tight",dpi=150)
        plt.clf()
        plt.title("Histogram low-dimesion in log-scale")
        plt.hist (QFlatten,bins=50,rwidth=1.0, log = True,density =True)
        plt.savefig("plots/Hist_SSNE_Q"+str(perplexity)+".png", bbox_inches="tight",dpi=150)
        plt.clf()
```

- Code Explanations for t-SNE

✓ T-SNE uses t-distribution for modelling the data points. Therefore, it calculates pair-wise similarity and gradient in low-dimension with different formula.

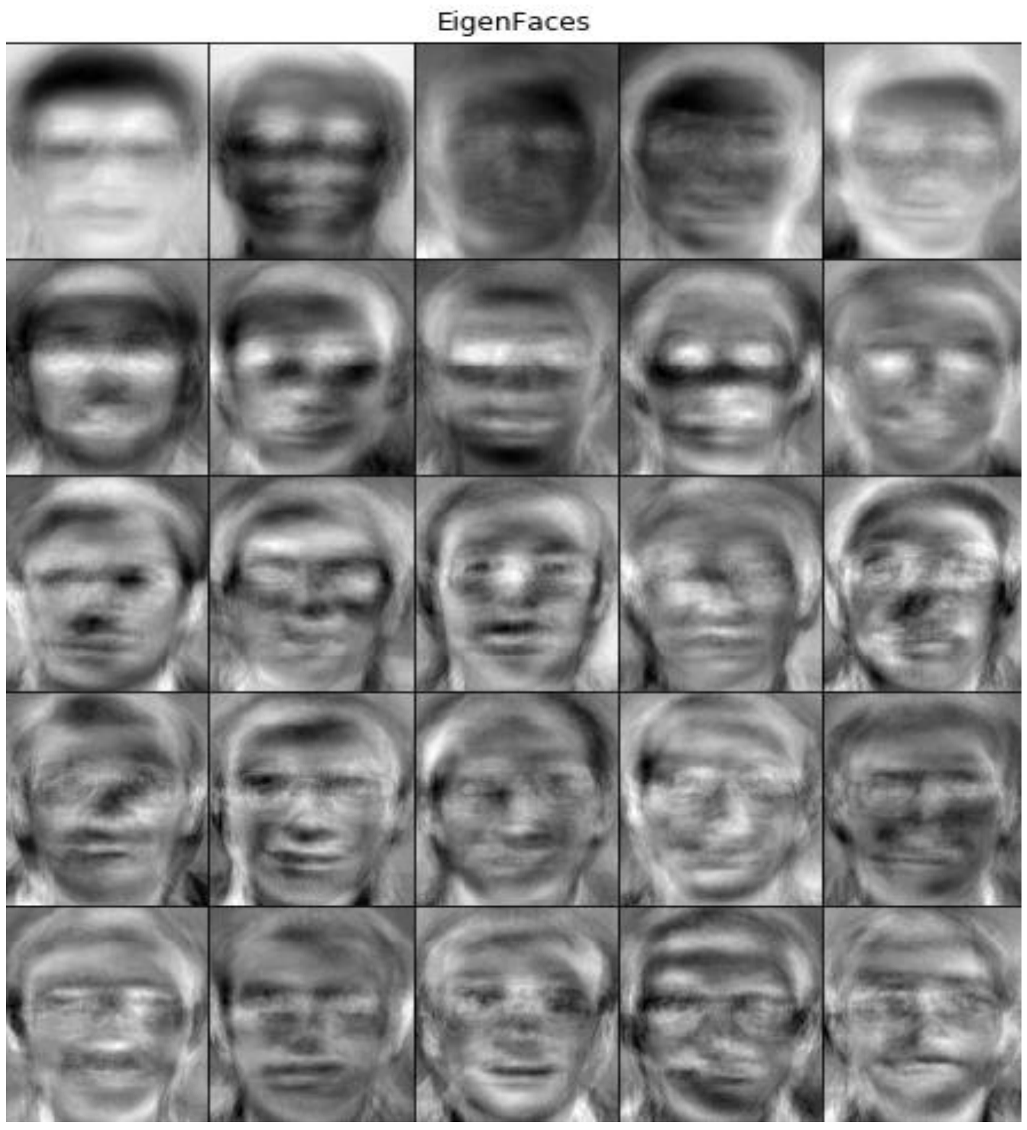


```
# Compute pairwise affinities
sum_Y = np.sum(np.square(Y), 1)
num = -2. * np.dot(Y, Y.T)
num = 1. / (1. + np.add(np.add(num, sum_Y).T, sum_Y))
num[range(n), range(n)] = 0.
Q = num / np.sum(num)
Q = np.maximum(Q, 1e-12)

# Compute gradient
PQ = P - Q
for i in range(n):
    dY[i, :] = np.sum(np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

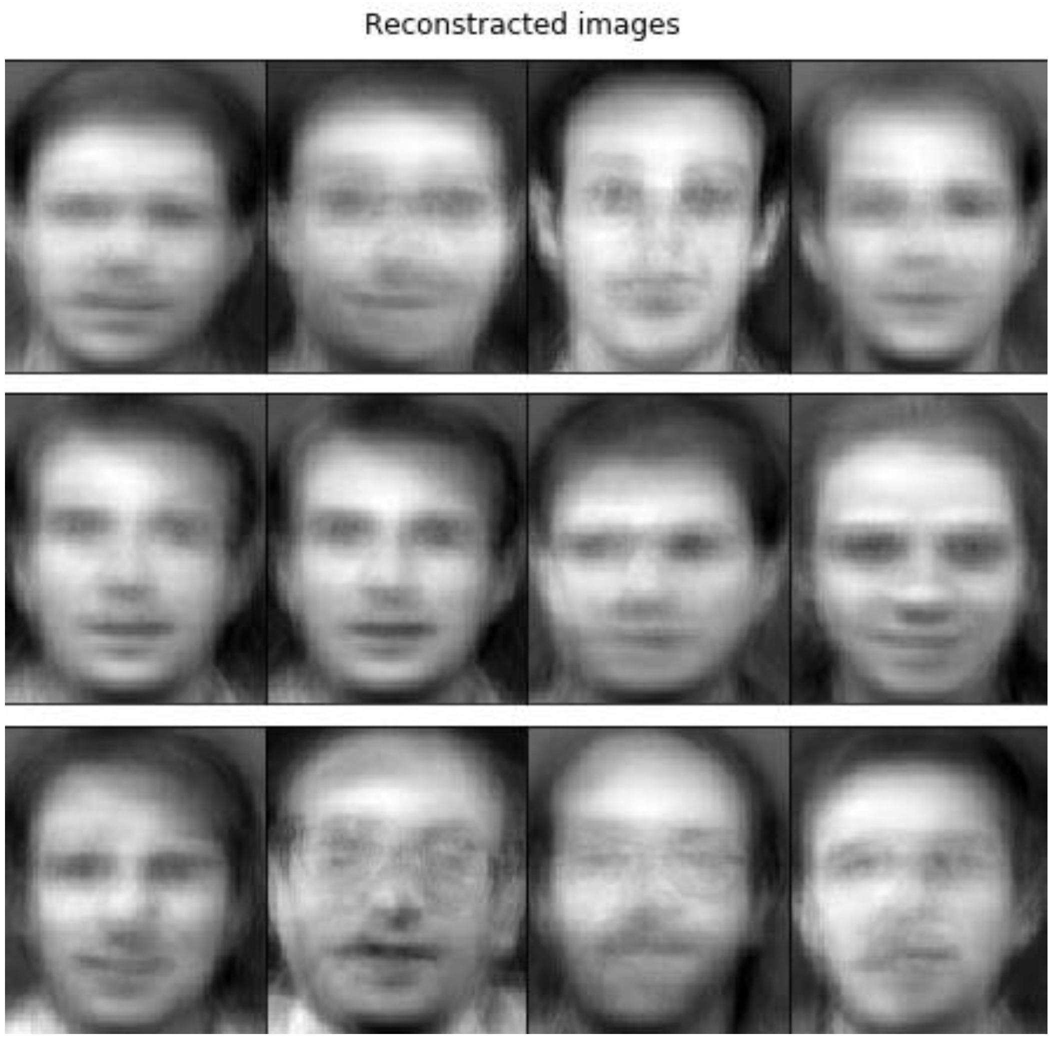
✓ The rest of the code used for t-SNE is similar to SSNE which explained above.

7. Eigenface

- The first 25 Eigenfaces using PCA on Face attributes dataset

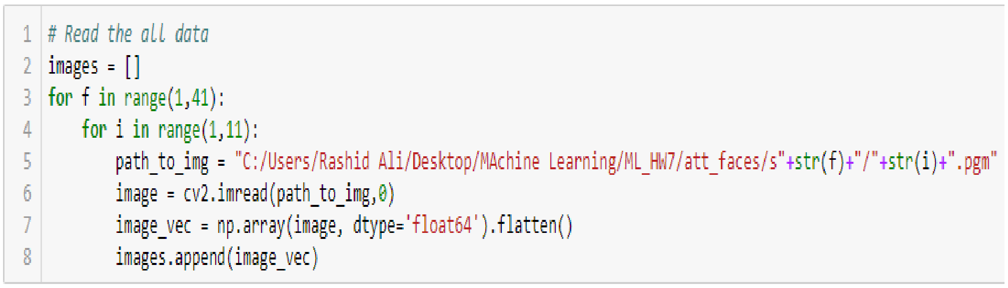


- Reconstruction of 10 images randomly picked



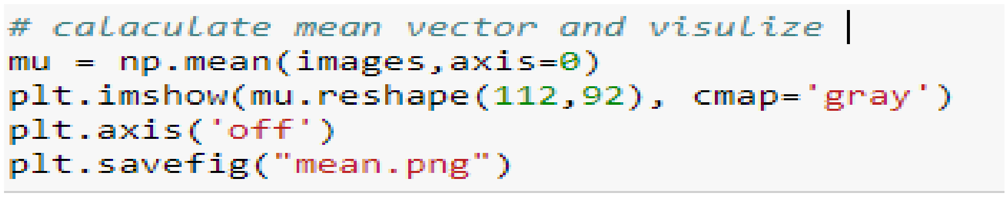
- Code Explanations for Eigenfaces

  ✓ Read images from each folder and append them images list



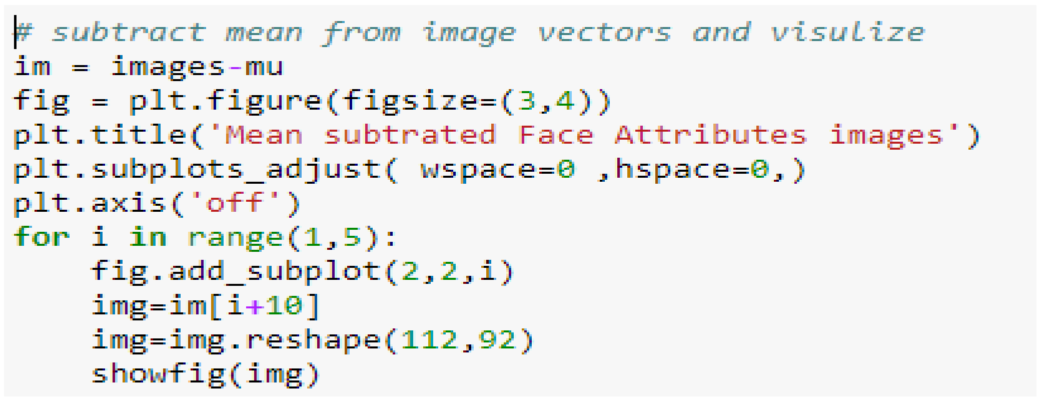
```python
# Read the all data
images = []
for f in range(1,41):
    for i in range(1,11):
        path_to_img = "C:/Users/Rashid Ali/Desktop/MAchine Learning/ML_HW7/att_faces/s"+str(f)+"/"+str(i)+".pgm"
        image = cv2.imread(path_to_img,0)
        image_vec = np.array(image, dtype='float64').flatten()
        images.append(image_vec)
```

- ✓ calculate mean vector of images and visualize



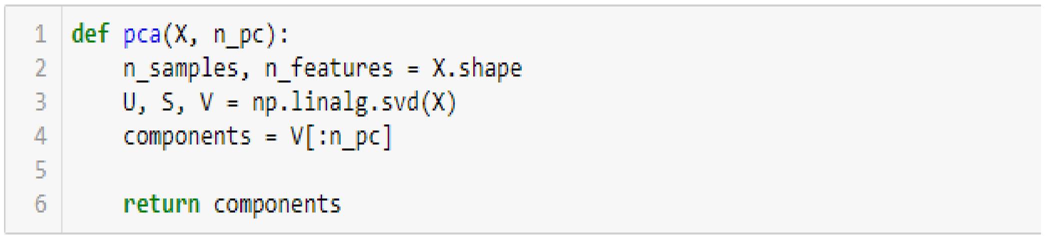
```python
# calaculate mean vector and visulize
mu = np.mean(images,axis=0)
plt.imshow(mu.reshape(112,92), cmap='gray')
plt.axis('off')
plt.savefig("mean.png")
```

- ✓ subtract mean vector from image vectors and visualize

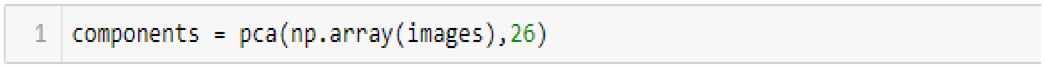


```python
# subtract mean from image vectors and visulize
im = images-mu
fig = plt.figure(figsize=(3,4))
plt.title('Mean subtrated Face Attributes images')
plt.subplots_adjust( wspace=0 ,hspace=0,)
plt.axis('off')
for i in range(1,5):
    fig.add_subplot(2,2,i)
    img=im[i+10]
    img=img.reshape(112,92)
    showfig(img)
```

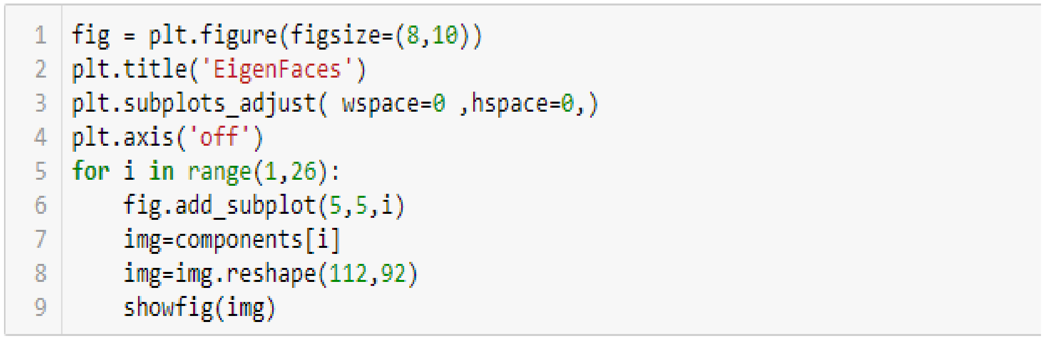
- ✓ perform PCA on images and return first 25 components and visualize them



```python
def pca(X, n_pc):
    n_samples, n_features = X.shape
    U, S, V = np.linalg.svd(X)
    components = V[:n_pc]

    return components
```

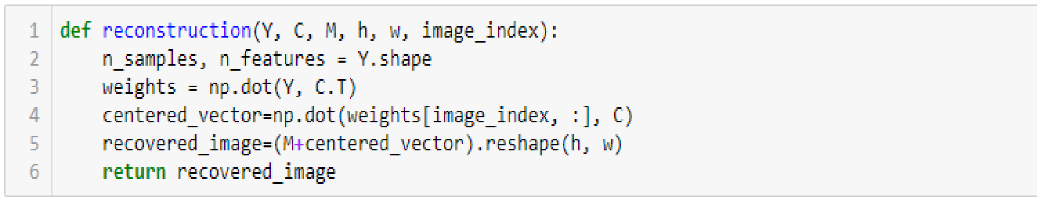


```python
components = pca(np.array(images),26)
```

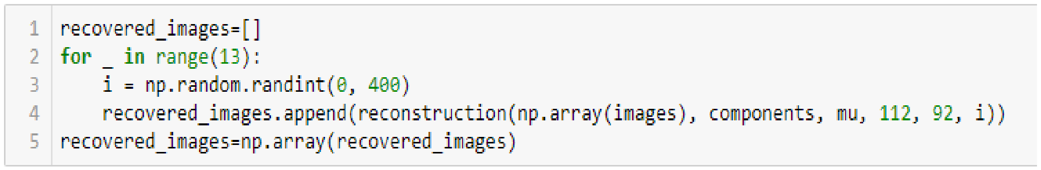


```python
fig = plt.figure(figsize=(8,10))
plt.title('EigenFaces')
plt.subplots_adjust( wspace=0 ,hspace=0,)
plt.axis('off')
for i in range(1,26):
    fig.add_subplot(5,5,i)
    img=components[i]
    img=img.reshape(112,92)
    showfig(img)
```

- ✓ randomly select the image from 400 images and perform reconstruction

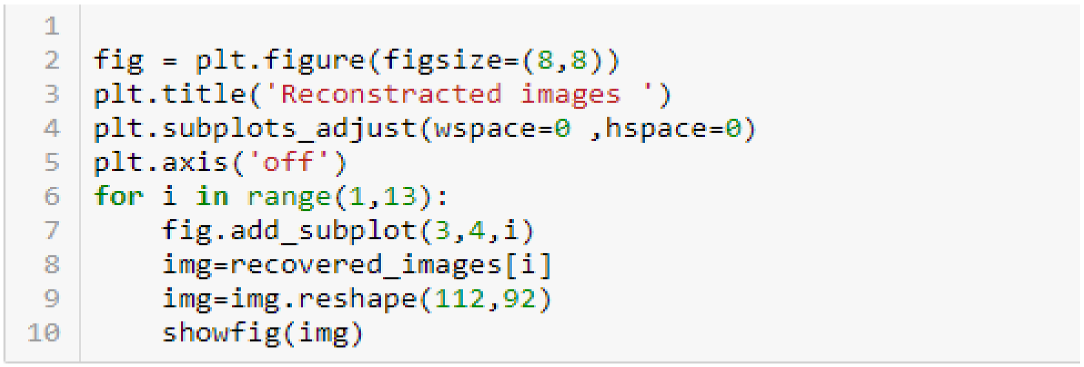


```python
def reconstruction(Y, C, M, h, w, image_index):
    n_samples, n_features = Y.shape
    weights = np.dot(Y, C.T)
    centered_vector=np.dot(weights[image_index, :], C)
    recovered_image=(M+centered_vector).reshape(h, w)
    return recovered_image
```



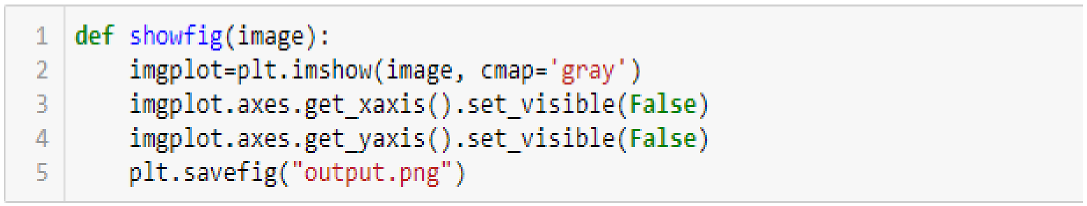
```python
recovered_images=[]
for _ in range(13):
    i = np.random.randint(0, 400)
    recovered_images.append(reconstruction(np.array(images), components, mu, 112, 92, i))
recovered_images=np.array(recovered_images)
```

- ✓ Visualize the reconstructed images



```python

fig = plt.figure(figsize=(8,8))
plt.title('Reconstructed images ')
plt.subplots_adjust(wspace=0 ,hspace=0)
plt.axis('off')
for i in range(1,13):
    fig.add_subplot(3,4,i)
    img=recovered_images[i]
    img=img.reshape(112,92)
    showfig(img)
```

- ✓ Show function for visualization



```python
def showfig(image):
    imgplot=plt.imshow(image, cmap='gray')
    imgplot.axes.get_xaxis().set_visible(False)
    imgplot.axes.get_yaxis().set_visible(False)
    plt.savefig("output.png")
```