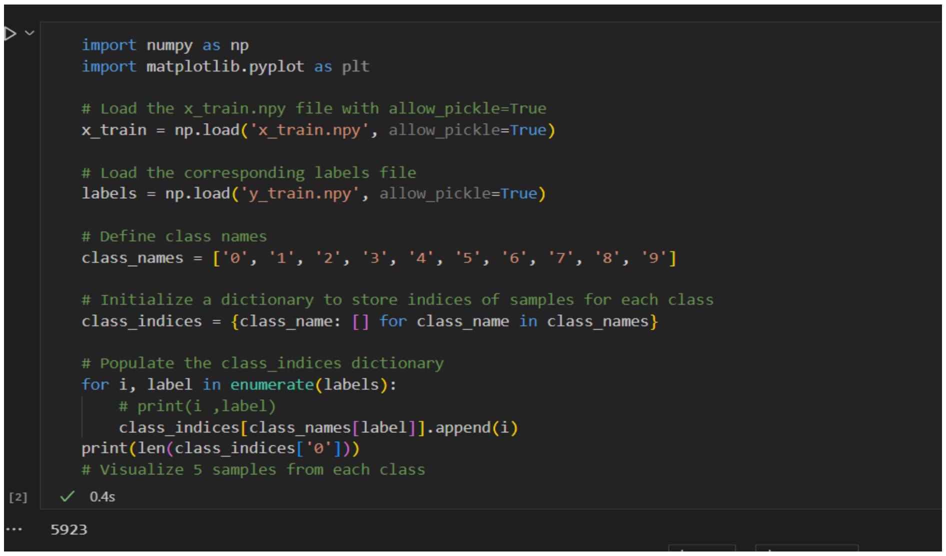
SML
Assignment 2
Vickey kumar
2021299

Question 1:
Use https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz MNIST dataset for this question and perform following tasks.
• It has all in all 60K train samples from 10 classes and 10K test samples.
10 classes are digits from 0-9. Labels or classes for all samples in train and test set is available.

According to the question , i make a class list of “0’ , “1” and so on to “9”
To store the data of y_train which is labels and x_tain data ,

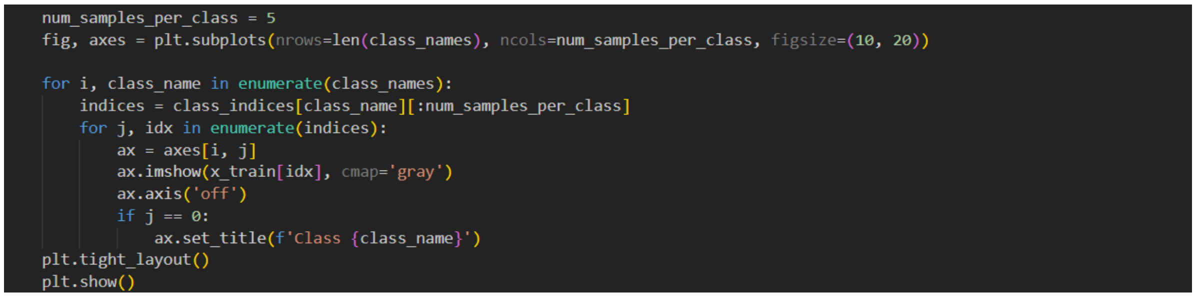


```
import numpy as np
import matplotlib.pyplot as plt

# Load the x_train.npy file with allow_pickle=True
x_train = np.load('x_train.npy', allow_pickle=True)

# Load the corresponding labels file
labels = np.load('y_train.npy', allow_pickle=True)

# Define class names
class_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

# Initialize a dictionary to store indices of samples for each class
class_indices = {class_name: [] for class_name in class_names}

# Populate the class_indices dictionary
for i, label in enumerate(labels):
    # print(i ,label)
    class_indices[class_names[label]].append(i)
print(len(class_indices['0']))
# Visualize 5 samples from each class
```

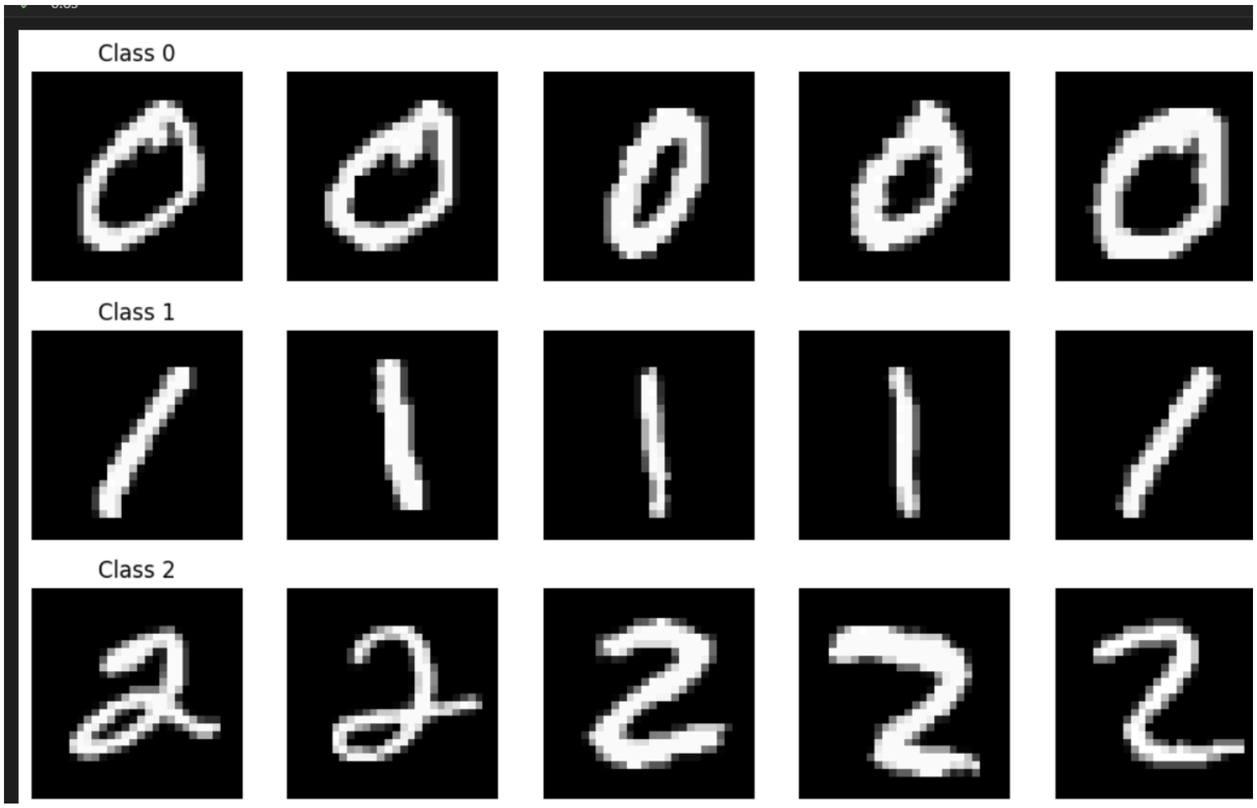
```
5923
```

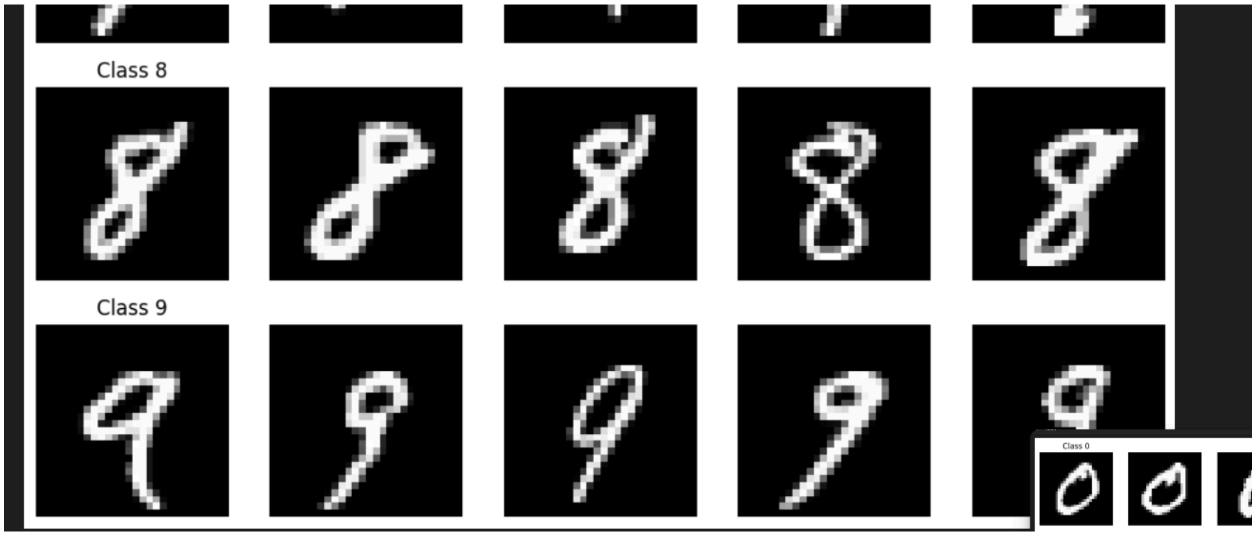
I check how many of class 0 has in y_train

- According to question 👍we visulated the first 5 image of each class

- 

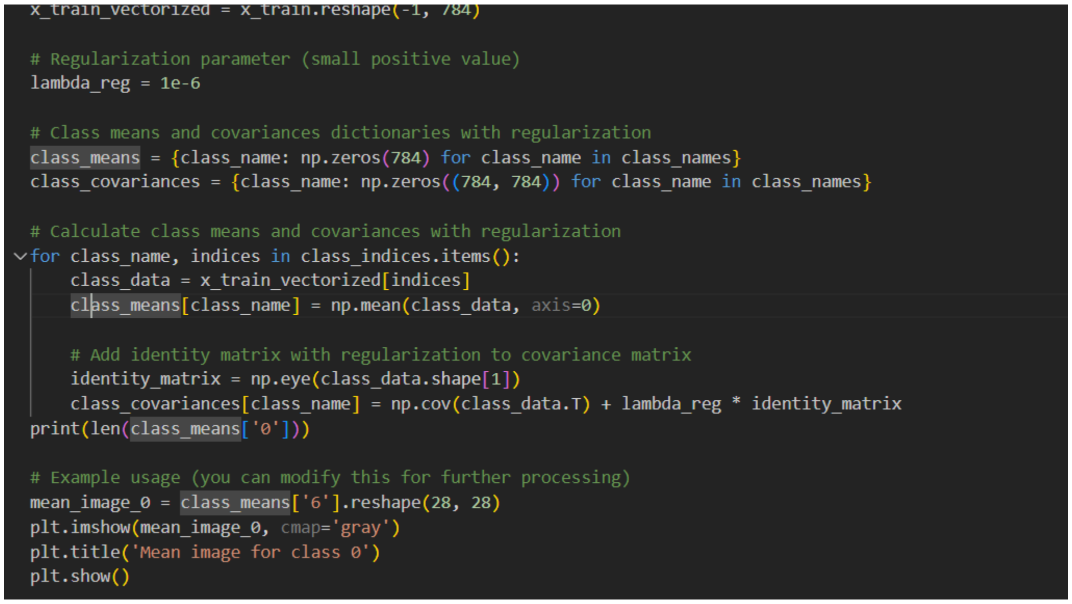
```python
num_samples_per_class = 5
fig, axes = plt.subplots(nrows=len(class_names), ncols=num_samples_per_class, figsize=(10, 20))

for i, class_name in enumerate(class_names):
    indices = class_indices[class_name][:num_samples_per_class]
    for j, idx in enumerate(indices):
        ax = axes[i, j]
        ax.imshow(x_train[idx], cmap='gray')
        ax.axis('off')
        if j == 0:
            ax.set_title(f'Class {class_name}')
plt.tight_layout()
plt.show()
```

- 



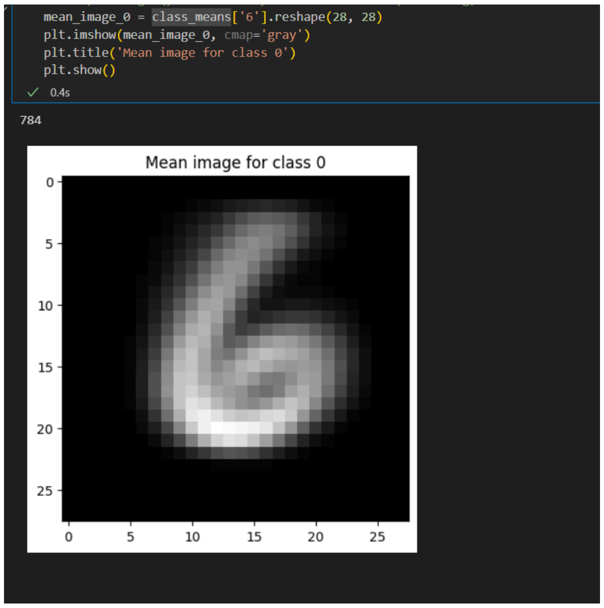
• Images are of size 28×28. Vectorize them to make it 784 dimensional.

Apply QDA on the given dataset. For each of the 10 classes you need to compute its mean vector and covariance vector. Use the QDA expression derived in the lecture. Your code should clearly have this expression. Note mean and covariance will be computed from train set only. Test set is not seen at this stage.



```python
x_train_vectorized = x_train.reshape(-1, 784)

# Regularization parameter (small positive value)
lambda_reg = 1e-6

# Class means and covariances dictionaries with regularization
class_means = {class_name: np.zeros(784) for class_name in class_names}
class_covariances = {class_name: np.zeros((784, 784)) for class_name in class_names}

# Calculate class means and covariances with regularization
for class_name, indices in class_indices.items():
    class_data = x_train_vectorized[indices]
    class_means[class_name] = np.mean(class_data, axis=0)

    # Add identity matrix with regularization to covariance matrix
    identity_matrix = np.eye(class_data.shape[1])
    class_covariances[class_name] = np.cov(class_data.T) + lambda_reg * identity_matrix
print(len(class_means['0']))

# Example usage (you can modify this for further processing)
mean_image_0 = class_means['6'].reshape(28, 28)
plt.imshow(mean_image_0, cmap='gray')
plt.title('Mean image for class 0')
plt.show()
```

Here i calculate the covariance and mean of the data



```python
mean_image_0 = class_means['6'].reshape(28, 28)
plt.imshow(mean_image_0, cmap='gray')
plt.title('Mean image for class 0')
plt.show()
```

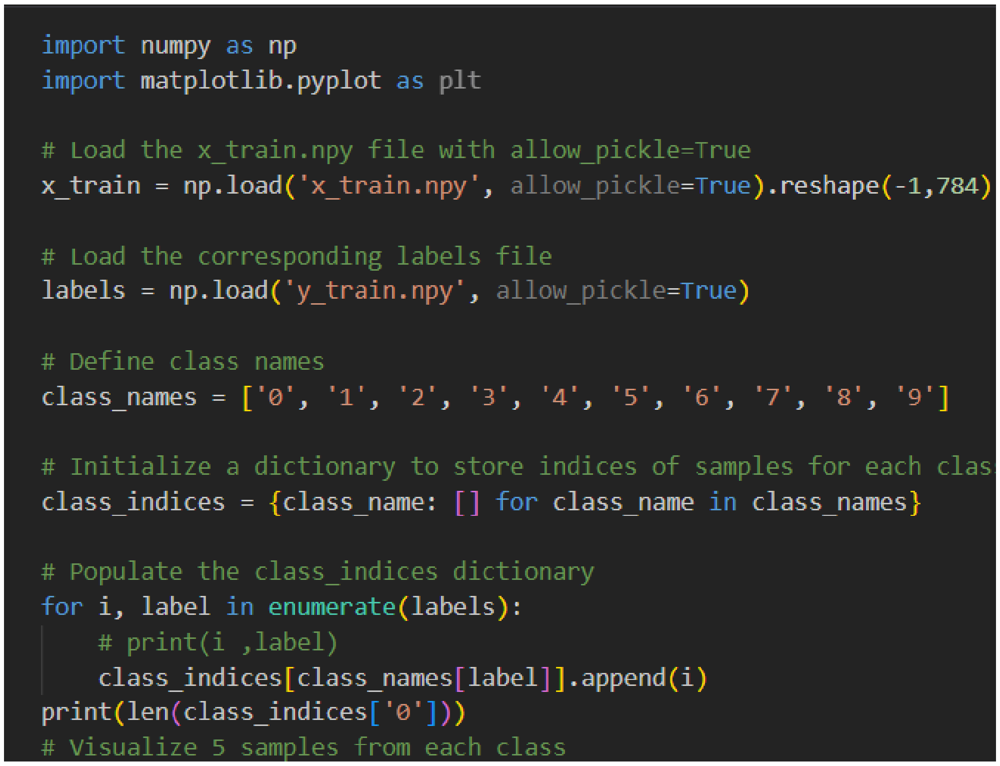
```
0.4s

784
```

Here i print the image by using the mean data of the matrix

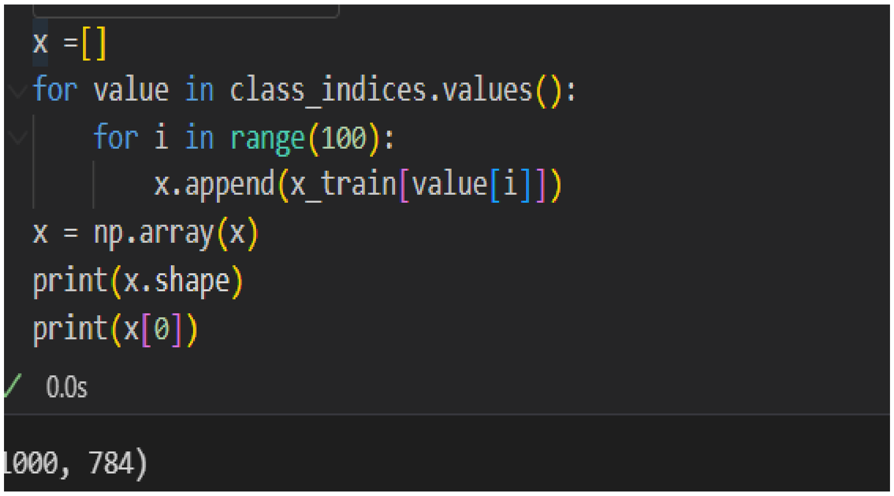
Find the class of all samples in test set. Report accuracy and class-wise accuracy for testing dataset. Accuracy is ratio of total number of samples correctly classified to the total number of samples tested. Total number of samples tested is 10K. Similarly, for each class report the accuracy. Note the labels or classes for each sample is given in the dataset.

Question 2:
Use same downloaded dataset from Question 1 and perform following tasks.
• Choose 100 samples from each class and create a 784×1000 data matrix. Let this be X.

I used the same distribution method to x_train and y_train data to make the X matrix by putting first class 0 100 vector matrix than next class matrix and so on , so X shape is 1000 *784

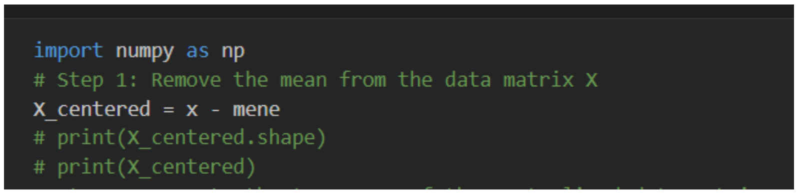


```
import numpy as np
import matplotlib.pyplot as plt

# Load the x_train.npy file with allow_pickle=True
x_train = np.load('x_train.npy', allow_pickle=True).reshape(-1,784)

# Load the corresponding labels file
labels = np.load('y_train.npy', allow_pickle=True)

# Define class names
class_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

# Initialize a dictionary to store indices of samples for each clas
class_indices = {class_name: [] for class_name in class_names}

# Populate the class_indices dictionary
for i, label in enumerate(labels):
    # print(i ,label)
    class_indices[class_names[label]].append(i)
print(len(class_indices['0']))
# Visualize 5 samples from each class
```



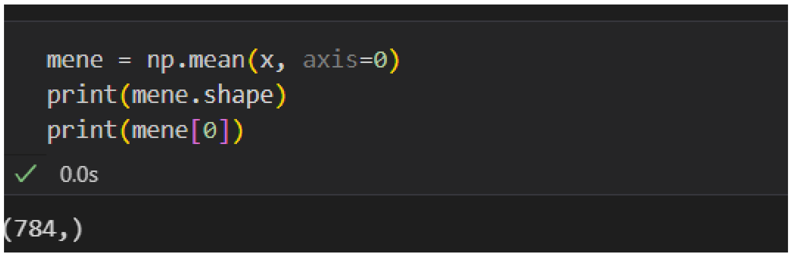
```
x =[]
for value in class_indices.values():
    for i in range(100):
        x.append(x_train[value[i]])
x = np.array(x)
print(x.shape)
print(x[0])
✓ 0.0s

000, 784)
```

- Then , Remove mean from X.



```
import numpy as np
# Step 1: Remove the mean from the data matrix X
X_centered = x - mene
# print(X_centered.shape)
# print(X_centered)
```

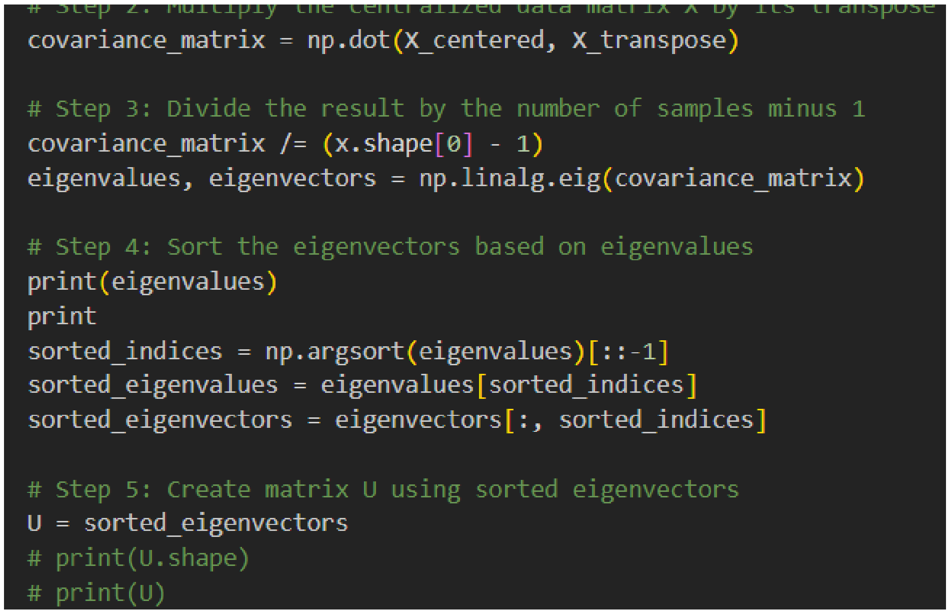
Mene is the mean by using np.mean along axis =0



```
mene = np.mean(x, axis=0)
print(mene.shape)
print(mene[0])
✓ 0.0s
(784,)
```

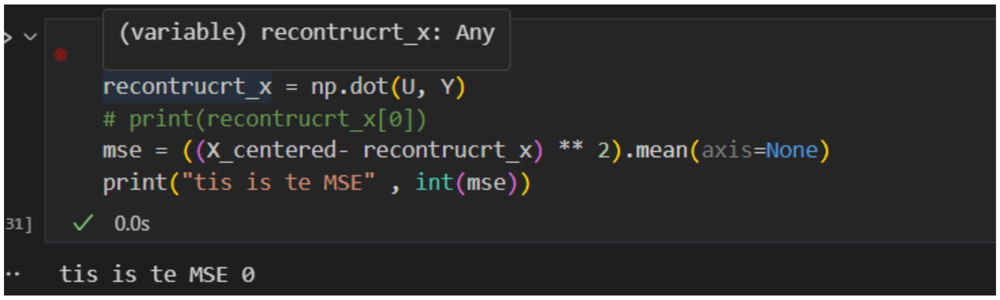
Apply PCA on the centralized X. You need to compute covariance S = XX>/999. The find its eigenvectors and eigenvalues. You can use any library for this. Sort them in descending order and create matrix U.

Here , we have to create the U matrix , we calculate it by using the decreasing the eigenvectors See in the code 👍



```
# Step 2: Multiply the centralized data matrix X by its transpose
covariance_matrix = np.dot(X_centered, X_transpose)

# Step 3: Divide the result by the number of samples minus 1
covariance_matrix /= (x.shape[0] - 1)
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# Step 4: Sort the eigenvectors based on eigenvalues
print(eigenvalues)
print
sorted_indices = np.argsort(eigenvalues)[::-1]
sorted_eigenvalues = eigenvalues[sorted_indices]
sorted_eigenvectors = eigenvectors[:, sorted_indices]

# Step 5: Create matrix U using sorted eigenvectors
U = sorted_eigenvectors
# print(U.shape)
# print(U)
```

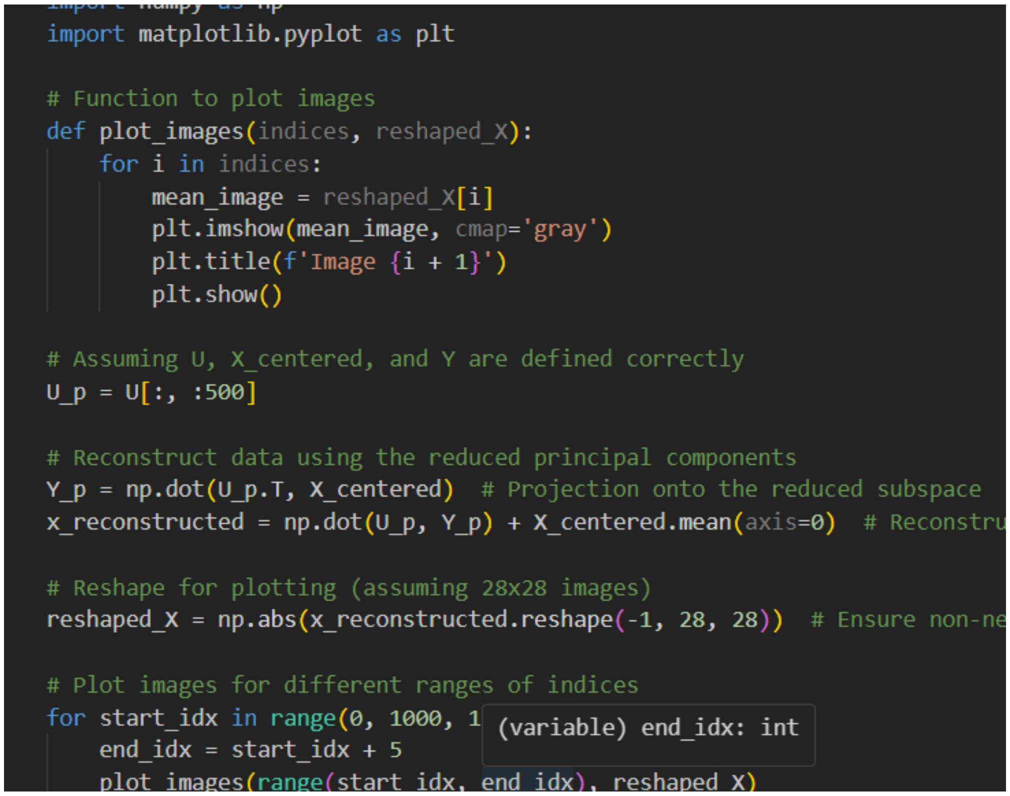
• Perform Y = U>X and reconstruct Xrecon = UY . Check the MSE be-
tween X and Xrecon. This should be close to 0. MSE = Pi,j (X(i, j) −Xrecon(i, j))2



```
(variable) recontrucrt_x: Any
recontrucrt_x = np.dot(U, Y)
# print(recontrucrt_x[0])
mse = ((X_centered- recontrucrt_x) ** 2).mean(axis=None)
print("tis is te MSE" , int(mse))
31] ✓ 0.0s
tis is te MSE 0
```

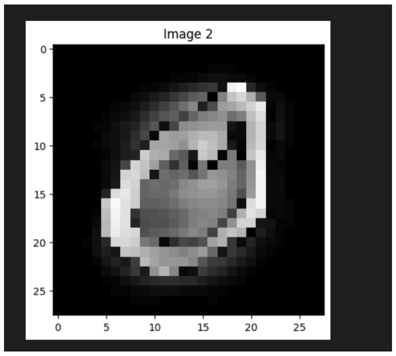
• Now chose p = 5, 10, 20 eigenvectors from U. For each p, obtain UpY , add mean that was removed from X, reshape each column to 28×28, and plot the image. You should see that as p increase the reconstructed images look more like their original counterparts. Plot 5 images from each class.

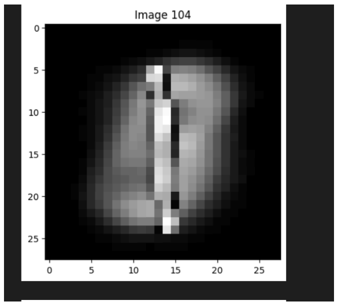
we put p and increasing it the accuracy of the image is correct and accurate you can see
Here we code it U_p for different p and Y_p and make X_reconstucted = U_p * Y_p

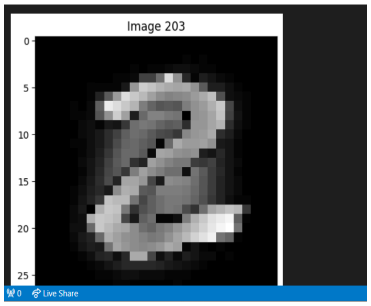


```
import matplotlib.pyplot as plt

# Function to plot images
def plot_images(indices, reshaped_X):
    for i in indices:
        mean_image = reshaped_X[i]
        plt.imshow(mean_image, cmap='gray')
        plt.title(f'Image {i + 1}')
        plt.show()

# Assuming U, X_centered, and Y are defined correctly
U_p = U[:, :500]

# Reconstruct data using the reduced principal components
Y_p = np.dot(U_p.T, X_centered)  # Projection onto the reduced subspace
x_reconstructed = np.dot(U_p, Y_p) + X_centered.mean(axis=0)  # Reconstru

# Reshape for plotting (assuming 28x28 images)
reshaped_X = np.abs(x_reconstructed.reshape(-1, 28, 28))  # Ensure non-ne

# Plot images for different ranges of indices
for start_idx in range(0, 1000, 1 (variable) end_idx: int
    end_idx = start_idx + 5
    plot_images(range(start_idx, end_idx), reshaped_X)
```

You can see as we increase the value of the p , here increase in accuracy in the

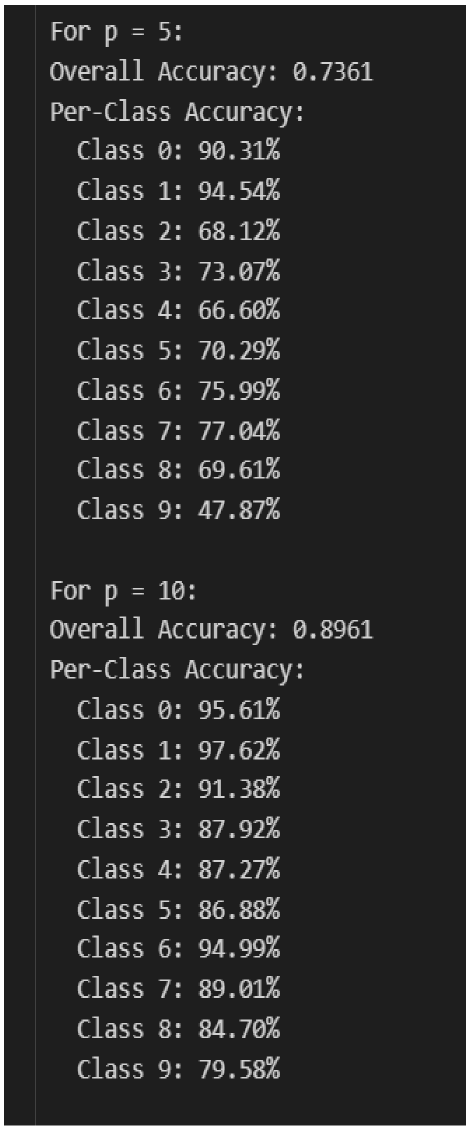




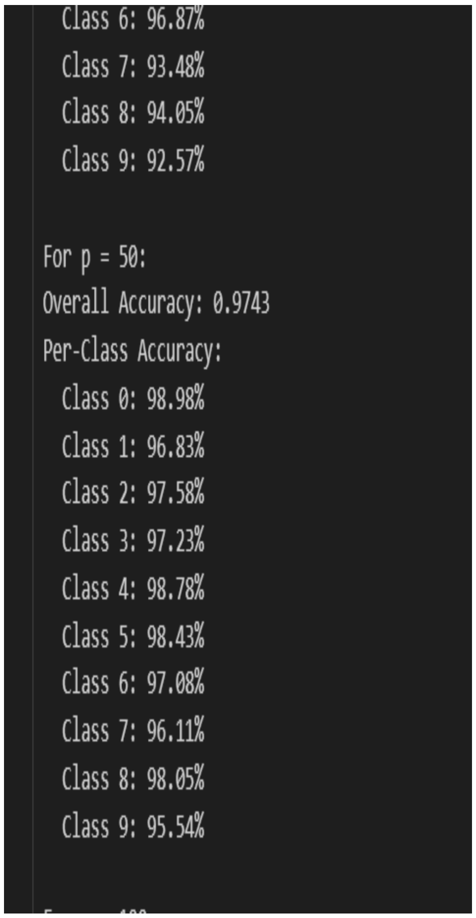


And so on to 9 class image form accurate .

Let test set be Xtest. Find Y = U>p Xtest. For each value of p find Y , and apply QDA from Q1 on Y. Obtain accuracy on test set as well as per class accuracy. As p inreases, accuracy shall increase.



```
For p = 5:
Overall Accuracy: 0.7361
Per-Class Accuracy:
  Class 0: 90.31%
  Class 1: 94.54%
  Class 2: 68.12%
  Class 3: 73.07%
  Class 4: 66.60%
  Class 5: 70.29%
  Class 6: 75.99%
  Class 7: 77.04%
  Class 8: 69.61%
  Class 9: 47.87%

For p = 10:
Overall Accuracy: 0.8961
Per-Class Accuracy:
  Class 0: 95.61%
  Class 1: 97.62%
  Class 2: 91.38%
  Class 3: 87.92%
  Class 4: 87.27%
  Class 5: 86.88%
  Class 6: 94.99%
  Class 7: 89.01%
  Class 8: 84.70%
  Class 9: 79.58%
```



```
  Class 6: 96.87%
  Class 7: 93.48%
  Class 8: 94.05%
  Class 9: 92.57%

For p = 50:
Overall Accuracy: 0.9743
Per-Class Accuracy:
  Class 0: 98.98%
  Class 1: 96.83%
  Class 2: 97.58%
  Class 3: 97.23%
  Class 4: 98.78%
  Class 5: 98.43%
  Class 6: 97.08%
  Class 7: 96.11%
  Class 8: 98.05%
  Class 9: 95.54%
```

As we observed that as we increase the p ,then the accuracy of the classification in increases