```
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
```

a)

```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Print the number of images in each training and testing set
print(f"Number of training images: {x_train.shape[0]}")
print(f"Number of testing images: {x_test.shape[0]}")

# Print the image width and height
print(f"Image width: {x_train.shape[1]}")
print(f"Image height: {x_train.shape[2]}")
```

```
Number of training images: 60000
Number of testing images: 10000
Image width: 28
Image height: 28
```

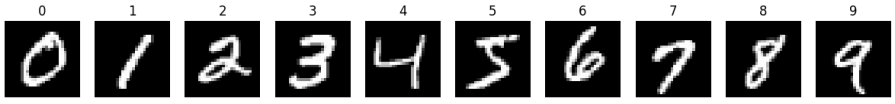
b)

```
def plot_digits(images, labels):
    # Create a figure with 10 subplots
    fig, axes = plt.subplots(1, 10, figsize=(15, 3))
    for i in range(10):
        # Find the first occurrence of each digit in the labels
        index = list(labels).index(i)

        # Plot the image to the digit label
        axes[i].imshow(images[index], cmap='gray')
        axes[i].set_title(str(i))
        axes[i].axis('off')
    plt.show()
```

c)

```
# Test
plot_digits(x_train, y_train)
```



d)

```
# Select the 0 and 8 digits from the training set
x_train_01 = x_train[(y_train == 0) | (y_train == 8)]
y_train_01 = y_train[(y_train == 0) | (y_train == 8)]

# Select the 0 and 8 digits from the testing set
x_test_01 = x_test[(y_test == 0) | (y_test == 8)]
y_test_01 = y_test[(y_test == 0) | (y_test == 8)]

print(f"Number of training images (0 and 8): {x_train_01.shape[0]}")
print(f"Number of testing images (0 and 8): {x_test_01.shape[0]}")
```

```
Number of training images (0 and 8): 11774
Number of testing images (0 and 8): 1954
```

e)

```
# Set a random seed for reproducibility
np.random.seed(50)

# Shuffle the training data
indices = np.arange(x_train_01.shape[0])
np.random.shuffle(indices)

# Select 500 images for the validation set
x_valid_01 = x_train_01[indices[:500]]
y_valid_01 = y_train_01[indices[:500]]

# The remaining images are for the training set
x_train_01 = x_train_01[indices[500:]]
y_train_01 = y_train_01[indices[500:]]

print(f"Number of training images (0 and 8): {x_train_01.shape[0]}")
print(f"Number of validation images (0 and 8): {x_valid_01.shape[0]}")
print(f"Number of testing images (0 and 8): {x_test_01.shape[0]}")
```

```
Number of training images (0 and 8): 11274
Number of validation images (0 and 8): 500
Number of testing images (0 and 8): 1954
```

h)

```
def calculate_center_average(images):
    # Initialize an empty list to store the average values
    averages = []

    # Define the center grid coordinates
    center_start = 12
    center_end = 16

    for image in images:
        # Extract the center 4x4 grid
        center_grid = image[center_start:center_end, center_start:center_end]

        # Calculate the average pixel value and append to the list
        averages.append(np.mean(center_grid))

    # Convert the list to a numpy array and return
    return np.array(averages)

# Calculate the average pixel values for training, validation, and testing sets
train_averages = calculate_center_average(x_train_01)
valid_averages = calculate_center_average(x_valid_01)
test_averages = calculate_center_average(x_test_01)
```

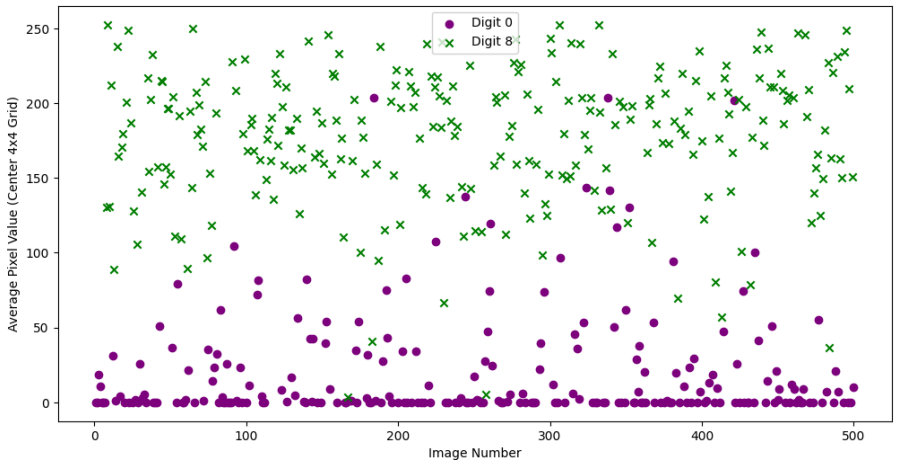
i)

```
def plot_validation_averages(valid_averages, y_valid):
    # Create a figure and axis
    plt.figure(figsize=(12, 6))

    # Plot the averages for digit 0
    plt.scatter(np.arange(1, 501)[y_valid == 0], valid_averages[y_valid == 0], color='purple', label='Digit 0', marker='o')

    # Plot the averages for digit 8
    plt.scatter(np.arange(1, 501)[y_valid == 8], valid_averages[y_valid == 8], color='green', label='Digit 8', marker='x')
    plt.xlabel('Image Number')
    plt.ylabel('Average Pixel Value (Center 4x4 Grid)')
    plt.legend()
    plt.show()

plot_validation_averages(valid_averages, y_valid_01)
```



j), k)

```
threshold = 100

def calculate_accuracy(averages, labels, threshold):
    # Predict labels based on the threshold
    predicted_labels = (averages > threshold).astype(int) * 8  # 0 if below threshold, 8 if above

    # Calculate accuracy
    accuracy = np.mean(predicted_labels == labels) * 100

    return accuracy

# Calculate accuracies
train_accuracy = calculate_accuracy(train_averages, y_train_01, threshold)
valid_accuracy = calculate_accuracy(valid_averages, y_valid_01, threshold)
test_accuracy = calculate_accuracy(test_averages, y_test_01, threshold)

print(f"Training accuracy: {train_accuracy:.2f}%")
print(f"Validation accuracy: {valid_accuracy:.2f}%")
print(f"Testing accuracy: {test_accuracy:.2f}%")
```

```
Training accuracy: 93.70%
Validation accuracy: 94.80%
Testing accuracy: 95.65%
```

Colab link: https://colab.research.google.com/drive/1qzxNdotBhyQybYvn098Y1WdhNZs2yLdl?usp=sharing