[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

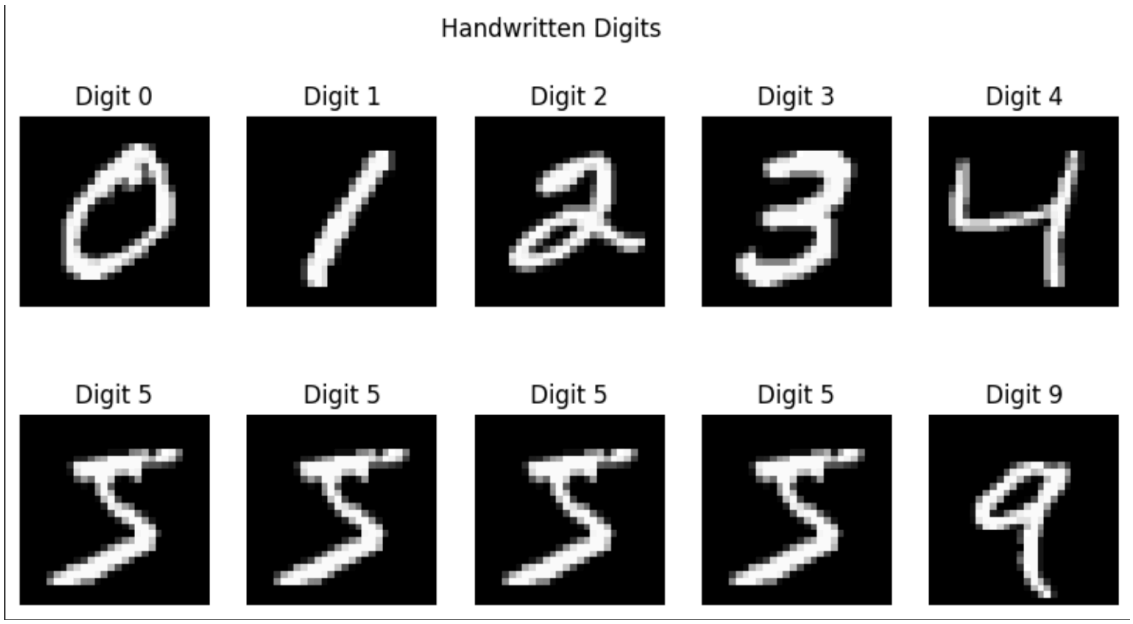
a.)

```python
#Problem a
from tensorflow.keras.datasets import mnist

#load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

#print info
print("Number of images in training set:", x_train.shape[0])
print("Number of images in testing set:", x_test.shape[0])
print("Image width:", x_train.shape[1])
print("Image height:", x_train.shape[2])
```

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

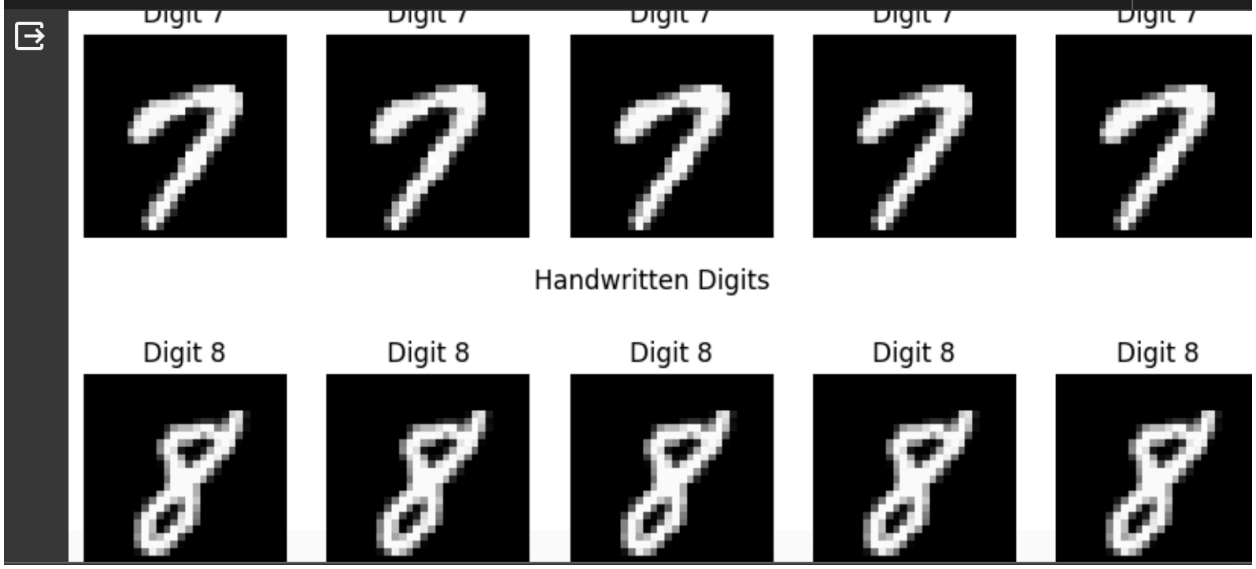
b.)

```python
#Problem b
import matplotlib.pyplot as plt
import numpy as np

def plot_digits(images, labels):
    #figure with 10 subplots
    fig, axes = plt.subplots(2, 5, figsize=(10, 5))
    fig.suptitle("Handwritten Digits")

    for i in range(10):
        #index of the first occurrence of each digit in labels
        index = np.argmax(labels == i)

        #image and label of the current digit
        digit_image = images[index]
        digit_label = labels[index]

        #digit in the corresponding subplot
        axes[i // 5, i % 5].imshow(digit_image, cmap='gray')
        axes[i // 5, i % 5].set_title(f"Digit {digit_label}")
        axes[i // 5, i % 5].axis('off')

    plt.show()

#usage with the first 10 images and labels from training set
plot_digits(x_train[:10], y_train[:10])
```



c.)

```python
#Problem c
#loop through digits 0-9 and plot images from training set
for digit in range(10):
    #images and labels for the current digit
    digit_images = x_train[y_train == digit][:5]  #first 5 images for each digit
    digit_labels = y_train[y_train == digit][:5]

    #display the images
    plot_digits(digit_images, digit_labels)
```



d.)

```python
#Problem d
#filter training set for 0 and 8 digits
x_train_01 = x_train[np.logical_or(y_train == 0, y_train == 8)]
y_train_01 = y_train[np.logical_or(y_train == 0, y_train == 8)]

#filter testing set for 0 and 8 digits
x_test_01 = x_test[np.logical_or(y_test == 0, y_test == 8)]
y_test_01 = y_test[np.logical_or(y_test == 0, y_test == 8)]
```

e.)

```
[7] #Problem e
    from sklearn.model_selection import train_test_split

    #combine x_train_01 and y_train_01 for splitting
    combined_train_01 = np.column_stack((x_train_01.reshape(x_train_01.shape[0], -1), y_train_01))

    #500 samples for validation set
    train_01, valid_01 = train_test_split(combined_train_01, test_size=500, stratify=combined_train_01[:, -1], random_state=42)

    #separate features and labels for training and validation sets
    x_train_01 = train_01[:, :-1].reshape(-1, 28, 28)
    y_train_01 = train_01[:, -1].astype(int)
    x_valid_01 = valid_01[:, :-1].reshape(-1, 28, 28)
    y_valid_01 = valid_01[:, -1].astype(int)
```

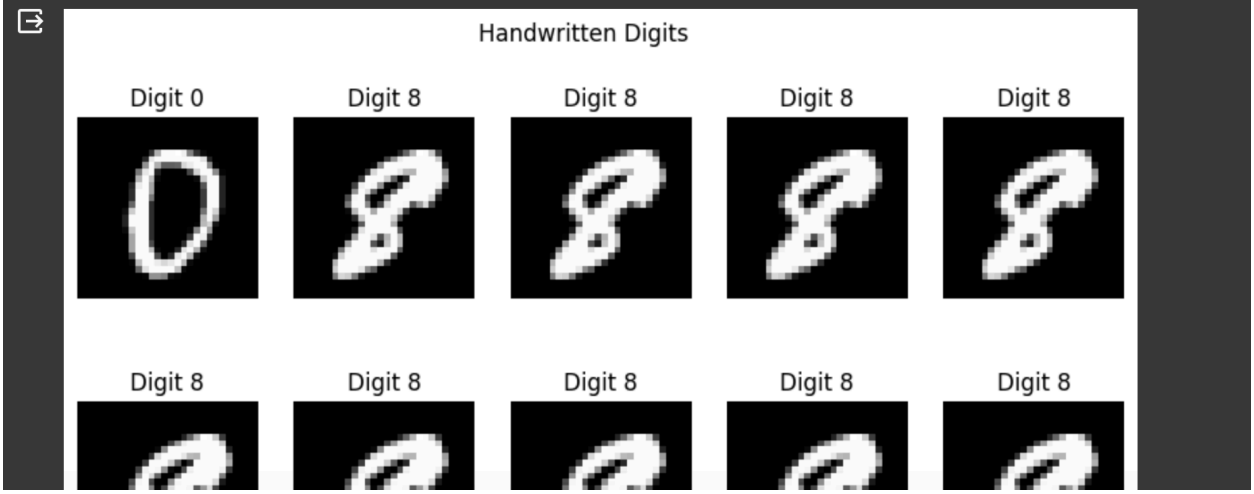
f.)

```
#Problem f
from sklearn.model_selection import train_test_split

#combine x_train_01 and y_train_01 for splitting
combined_train_01 = np.column_stack((x_train_01.reshape(x_train_01.shape[0], -1), y_train_01))

#500 samples for validation set
train_01, valid_01 = train_test_split(combined_train_01, test_size=500, stratify=combined_train_01[:, -1], random_state=42)

#separate features and labels for training and validation sets
x_train_01 = train_01[:, :-1].reshape(-1, 28, 28)
y_train_01 = train_01[:, -1].astype(int)
x_valid_01 = valid_01[:, :-1].reshape(-1, 28, 28)
y_valid_01 = valid_01[:, -1].astype(int)

print("Number of images in training set:", x_train_01.shape[0])
print("Number of images in validation set:", x_valid_01.shape[0])
print("Number of images in testing set:", x_test_01.shape[0])
```

```
Number of images in training set: 10774
Number of images in validation set: 500
Number of images in testing set: 1954
```

g.)

```python
#Problem g
#10 random images from the validation set
random_indices = np.random.choice(x_valid_01.shape[0], size=10, replace=False)
validation_images = x_valid_01[random_indices]
validation_labels = y_valid_01[random_indices]

#display the images
plot_digits(validation_images, validation_labels)
```

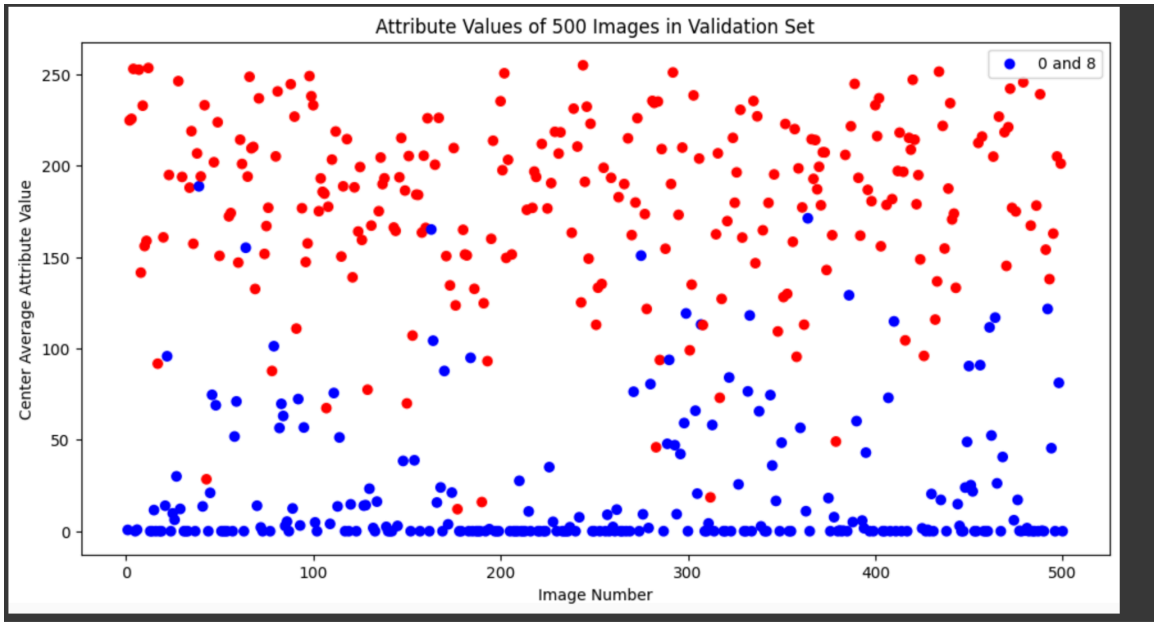


h.)

```python
#Problem h
#calculate the average of pixel values in the center 4x4 grid
def calculate_center_average(images):
    center_pixels = images[:, 12:16, 12:16]  # Extract the center 4x4 grid
    return np.mean(center_pixels, axis=(1, 2))  # Calculate the average of pixel values

#center average for training, validation, and testing sets
x_train_01_center_avg = calculate_center_average(x_train_01)
x_valid_01_center_avg = calculate_center_average(x_valid_01)
x_test_01_center_avg = calculate_center_average(x_test_01)
```

i.)

```python
[11] #Problem i
import matplotlib.pyplot as plt

#center average for the entire validation set
validation_center_avg = calculate_center_average(x_valid_01)

plt.figure(figsize=(12, 6))
plt.scatter(range(1, 501), validation_center_avg, c=['b' if label == 0 else 'r' for label in y_valid_01], marker='o', label='0 and 8')
plt.xlabel('Image Number')
plt.ylabel('Center Average Attribute Value')
plt.title('Attribute Values of 500 Images in Validation Set')
plt.legend(loc='upper right')
plt.show()
```



j.)

```python
#Promblem j
#visual estimation of threshold based on the plot
estimated_threshold = 170

#threshold to classify images
predicted_labels = np.where(validation_center_avg > estimated_threshold, 8, 0)

accuracy = np.mean(predicted_labels == y_valid_01)
print(f"Accuracy with the estimated threshold: {accuracy * 100:.2f}%")
```

```
Accuracy with the estimated threshold: 82.00%
```

k.)

```python
#Problem k
#threshold to classify images in the training set
train_predicted_labels = np.where(calculate_center_average(x_train_01) > estimated_threshold, 8, 0)

#threshold to classify images in the validation set
valid_predicted_labels = np.where(validation_center_avg > estimated_threshold, 8, 0)

#threshold to classify images in the testing set
test_predicted_labels = np.where(calculate_center_average(x_test_01) > estimated_threshold, 8, 0)

train_accuracy = np.mean(train_predicted_labels == y_train_01)
valid_accuracy = np.mean(valid_predicted_labels == y_valid_01)
test_accuracy = np.mean(test_predicted_labels == y_test_01)

print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Validation Accuracy: {valid_accuracy * 100:.2f}%")
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
```

```
Training Accuracy: 80.68%
Validation Accuracy: 82.00%
Testing Accuracy: 81.32%
```