

# MLPy Workshop 2

Student 1, Student 2

January 26, 2025

## 1 Week 2: Principal Component Analysis

In this workshop, we will work through a set of problems on dimensionality reduction – a canonical form of unsupervised learning. Within the machine learning pipeline, dimensionality reduction is an important tool, which can be used in EDA to understand patterns in the data, feature engineering to create a low-dimensional representation of the inputs, and/or in the final phase when you are presenting and visualizing your solution.

As usual, the worksheets will be completed in teams of 2-3, using **pair programming**, and we have provided cues to switch roles between driver and navigator. When completing worksheets:

- You will have tasks tagged by (CORE) and (EXTRA).
- Your primary aim is to complete the (CORE) components during the WS session, afterwards you can try to complete the (EXTRA) tasks for your self-learning process.
- Look for the as cue to switch roles between driver and navigator.
- In some Exercises, you will see some beneficial hints at the bottom of questions.

Instructions for submitting your workshops can be found at the end of worksheet. As a reminder, you must submit a pdf of your notebook on Learn by 16:00 PM on the Friday of the week the workshop was given.

As you work through the problems it will help to refer to your lecture notes (navigator). The exercises here are designed to reinforce the topics covered this week. Please discuss with the tutors if you get stuck, even early on!

### 1.1 Outline

1. Problem Definition and Setup
2. **Principal Component Analysis**
  - a. Examining the Basis Vectors and Scores
  - b. Selecting the Number of Components
  - c. Other Digits

## 2 Problem Definition and Setup

### 2.1 Packages

First, let's load in some packages to get us started.

```
[1]: import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

### 2.2 Data

Our dataset will be the famous [MNIST](#) dataset of handwritten digits, which we will download from sklearn. The dataset consists of a set of grayscale images of the numbers 0-9 and corresponding labels. Usually the goal is to train a classifier (i.e. given an image, what digit does it correspond to?). Here we will throw away the labels and focus on the images themselves. Specifically, we will use dimensionality reduction to explore the images and underlying patterns and find a low-dimensional representation.

First, load the data:

```
[2]: # mac location:
df = pd.read_csv('/Users/abhi/mnist_784.csv')

# windows location:
# df = pd.read_csv(r'C:\Users\abhi\Documents\Code\mnist_784.csv')

[3]: # the class value is the actual digit, all of the values before that are the
    ↪ pixel values:
X = df.drop('class', axis=1).to_numpy()
y = df['class'].to_numpy()
```

#### 2.2.1 Exercise 1 (CORE)

What is stored in X and y in the command above? What is the shape/datatype etc if an array?

```
[4]: print(X.shape, y.shape)
```

```
(70000, 784) (70000,)
```

X contains the image data, and y contains what integer the pixels represent.

Now, let's create a dictionary, with the digit classes (0-9) as keys, where the corresponding values are the set of all images corresponding to that particular label.

```
[5]: digits_dict = {}
X_ = X.copy()
count = 0
```

```

for label in y:
    if label in digits_dict:
        digits_dict[label] += [X_[count]]
    else:
        digits_dict[label] = [X_[count]]
    count += 1

```

Next let's visualize some of the images. We will start by picking a label and plotting a few images from within the dictionary. Note that each image contains a total of 784 pixels (28 by 28) and we will need to reshape the image to plot with `imshow(..., cmap='gray_r')`. Try also changing the label to view different digits.

```
[6]: X_.shape
```

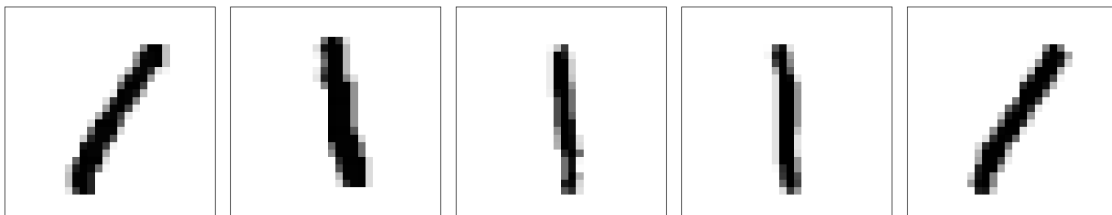
```
[6]: (70000, 784)
```

```

[7]: mylabel = 1
     n_images_per_label = 5

     fig = plt.figure(figsize=(4*n_images_per_label, 4))
     for j in range(n_images_per_label):
         ax_number = 1 + j
         ax = fig.add_subplot(1, n_images_per_label, ax_number)
         ax.imshow(digits_dict[mylabel][j].reshape((28,28)), cmap='gray_r')
         ax.set_xticks([])
         ax.set_yticks([])
     fig.tight_layout()

```



### 2.2.2 Exercise 2 (EXTRA)

Edit the code above to plot a few images for multiple labels.

Hint

Create a vector of labels and add additional for loop in the code above.

```

[8]: mult_labels = [1, 3, 7]

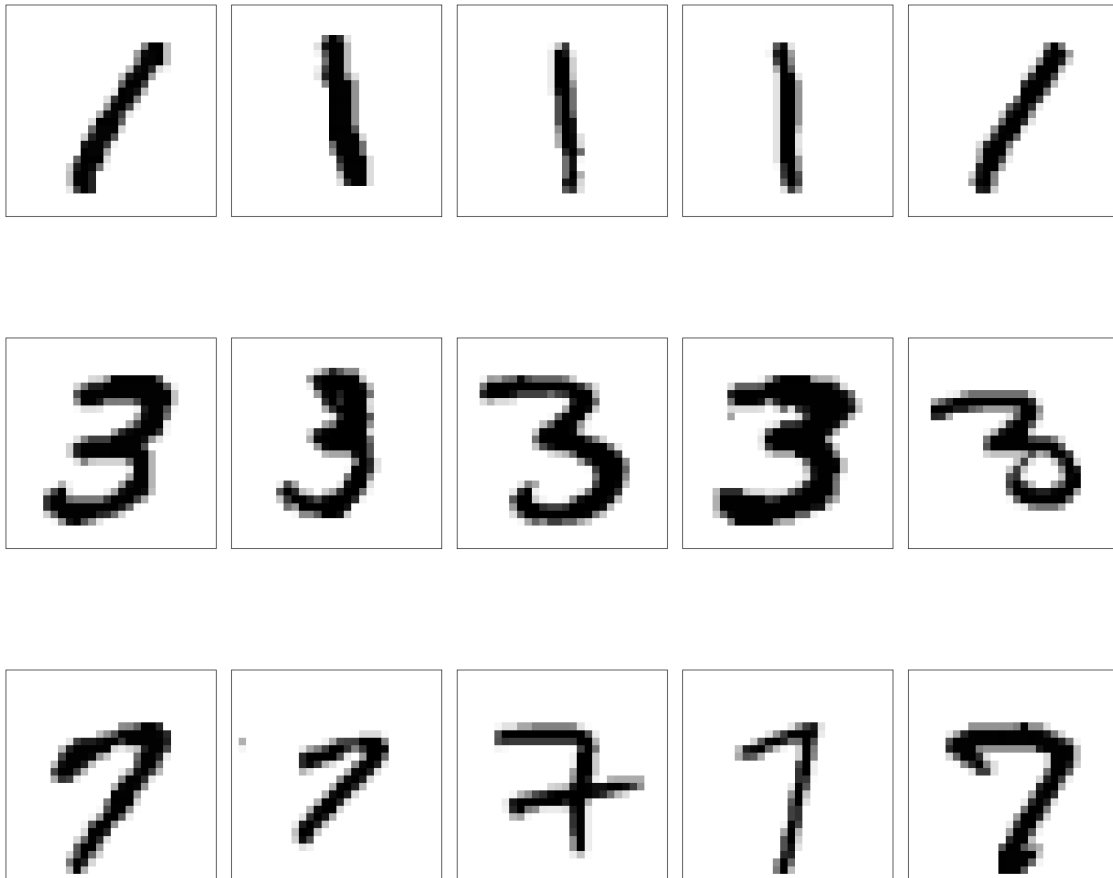
     for mylabel in mult_labels:

```

```

fig = plt.figure(figsize=(4*n_images_per_label, 4))
for j in range(n_images_per_label):
    ax_number = 1 + j
    ax = fig.add_subplot(1, n_images_per_label, ax_number)
    ax.imshow(digits_dict[mylabel][j].reshape((28,28)), cmap='gray_r')
    ax.set_xticks([])
    ax.set_yticks([])
fig.tight_layout()

```



### 2.2.3 Exercise 3 (CORE)

Now focus on the 3s only and create a data matrix called `X_threes`. Define also `N` (# datapoints) and `D` (# features).

What are the features in this problem? How many features and data points are there?

```
[9]: X_threes = np.array(digits_dict[3])
```

```
[10]: X_threes.shape
```

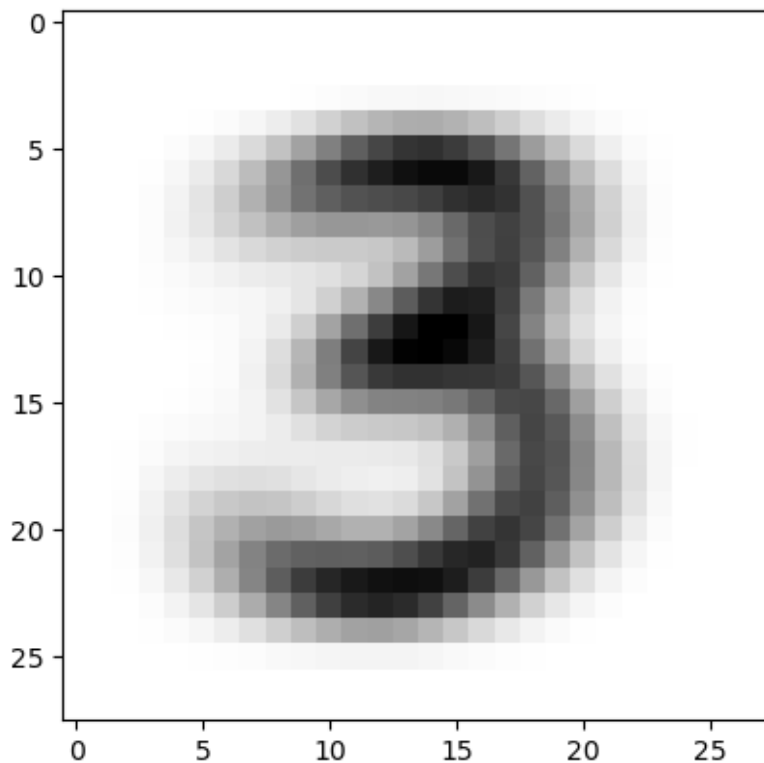
```
[10]: (7141, 784)
```

There are 7141 images i.e. datapoints, and each data point has 784 pixels i.e. the features.

#### 2.2.4 Exercise 4 (CORE)

Now compute and plot the mean image of three.

```
[11]: X_three_mean = np.sum(X_threes, axis=0) / len(X_threes)
plt.imshow(X_three_mean.reshape((28,28)), cmap='gray_r')
plt.show()
```



Run the following code to first create a new data matrix that centers the data by subtracting the mean image, and then visualise some of the images and compare to the original data. Note: you will need to replace `X_three_mean` with the name you gave the mean image in the computation above.

```
[12]: X_three_centred = X_threes - X_three_mean

n_images = 5

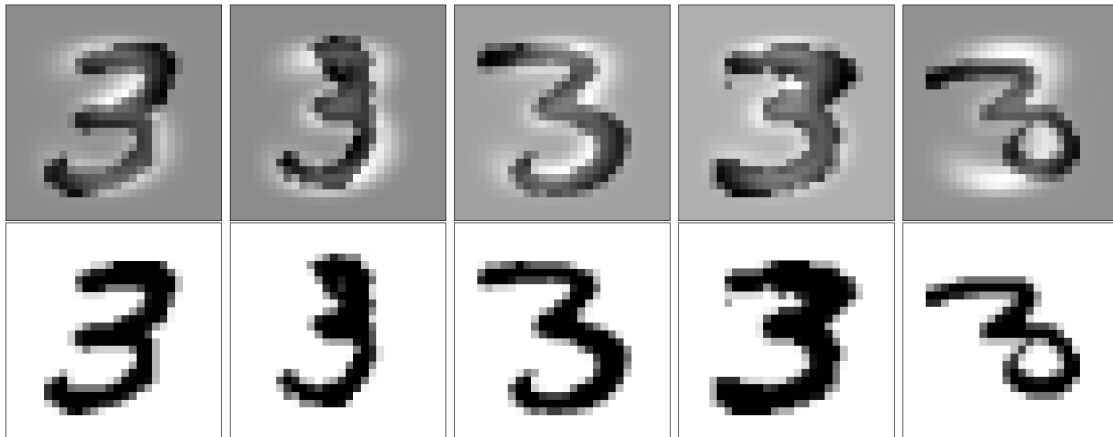
fig = plt.figure(figsize=(4*n_images, 4*2))
for j in range(n_images):
```

```

ax = fig.add_subplot(2, n_images, j+1)
ax.imshow(X_three_centred[j,:].reshape((28,28)), cmap='gray_r')
ax.set_xticks([])
ax.set_yticks([])

ax = fig.add_subplot(2, n_images, j+1+n_images)
ax.imshow(X_threes[j,:].reshape((28,28)), cmap='gray_r')
ax.set_xticks([])
ax.set_yticks([])
fig.tight_layout()

```



### 2.2.5 Exercise 5 (CORE)

Comment on whether or not the images need to be standardized before using PCA

Yes we must standardize the data using PCA.

**Now, is a good point to switch driver and navigator**

## 3 PCA

Now, we will perform PCA to summarize the main patterns in the images. We will use the `PCA()` transform from the `sklearn.decomposition` package:

- We can specify the number of components with the option `n_components`. If omitted, all components are kept.
- Note that by default the `PCA()` transform centers the variables to have zero mean (but does not scale them). After fitting, we can access the mean through the attribute `mean_`. If we also want to standardize to have not only zero mean but also unit variance, we can set `whiten=True`.
- We can access the basis vectors (principal components) through the `components_` attribute.

- We can call `fit()` to fit the model, followed by `transform` to obtain the low-dimensional representation (or also `fit_transform`).

First, let's create the PCA transform and call `fit()`:

```
[13]: pca_threes = PCA(n_components = 200)
      pca_threes.fit(X_threes)
```

```
[13]: PCA(n_components=200)
```

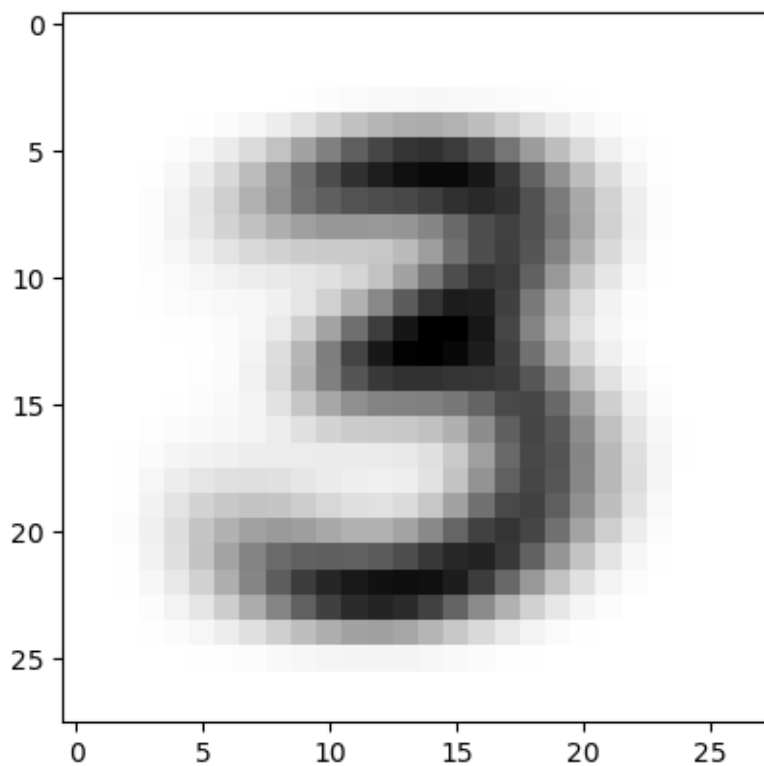
### 3.1 Examining the Basis Vectors and Scores

#### 3.1.1 Exercise 6 (EXTRA)

Plot the mean image by accessing the `mean_` attribute and check that it is the same as above.

```
[14]: plt.imshow(pca_threes.mean_.reshape((28,28)), cmap='gray_r')
```

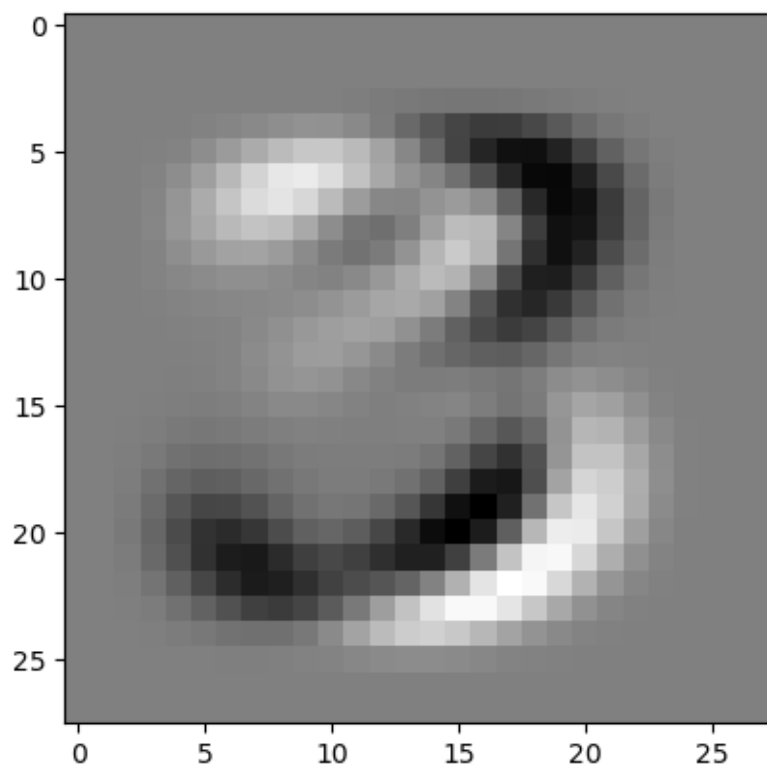
```
[14]: <matplotlib.image.AxesImage at 0x15f746e40>
```



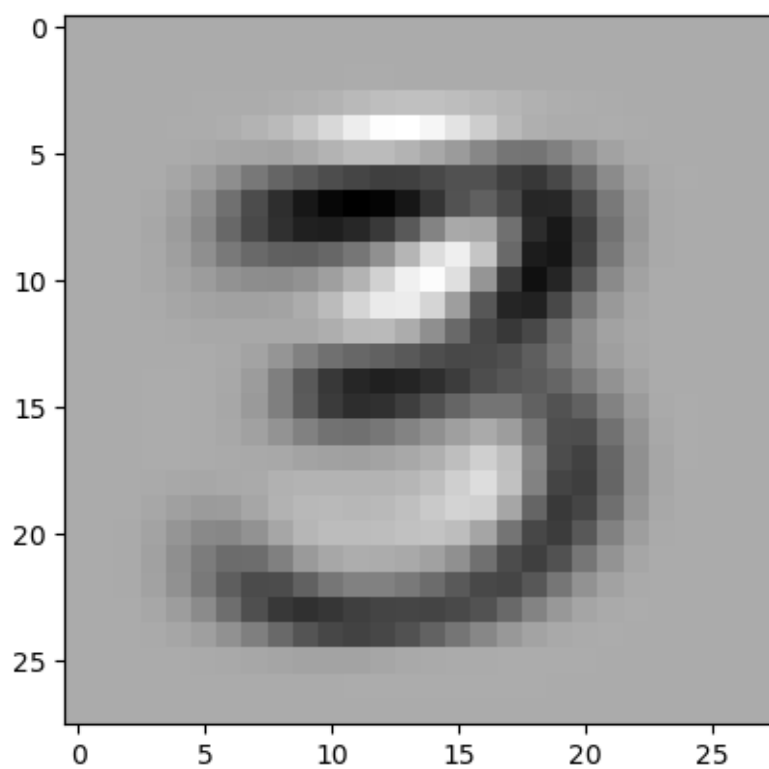
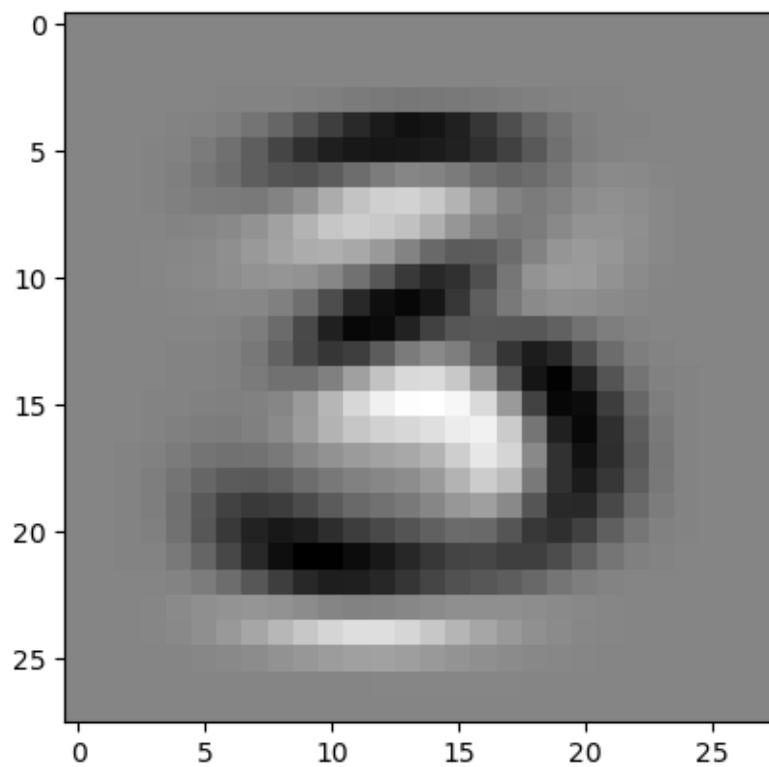
#### 3.1.2 Exercise 7 (CORE)

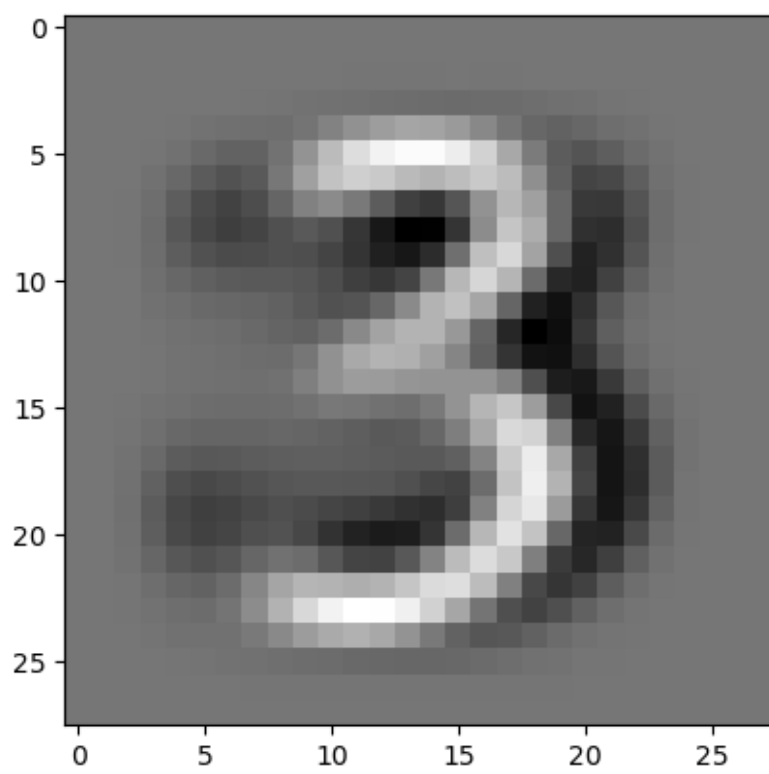
Plot the the first ten basis vectors as images by accessing the `components_` attribute. Overall, what patterns do they seem describe?

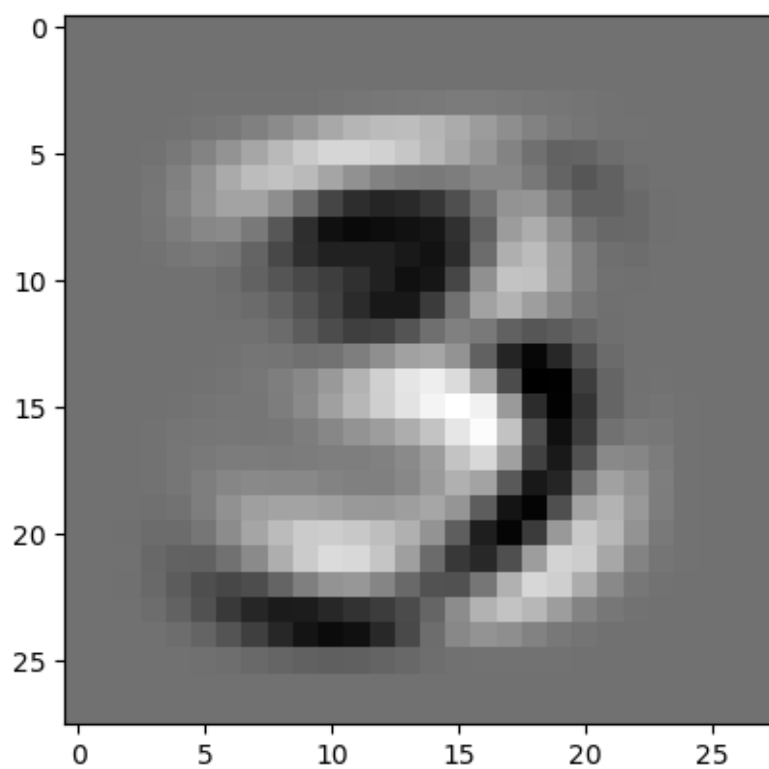
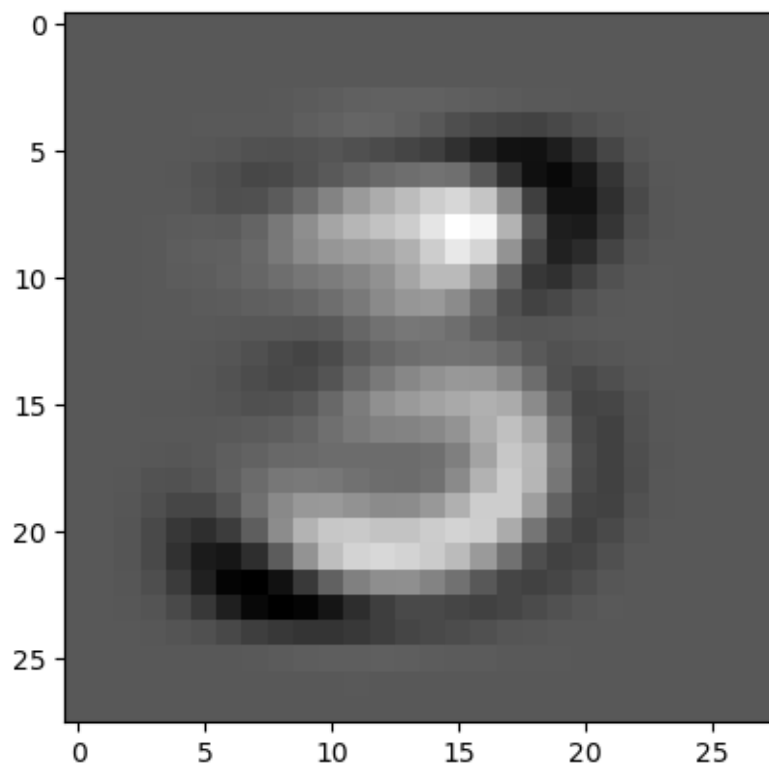
```
[15]: for i in range(10):  
      plt.imshow(pca_threes.components_[i].reshape((28,28)), cmap='gray_r')  
      plt.show()
```

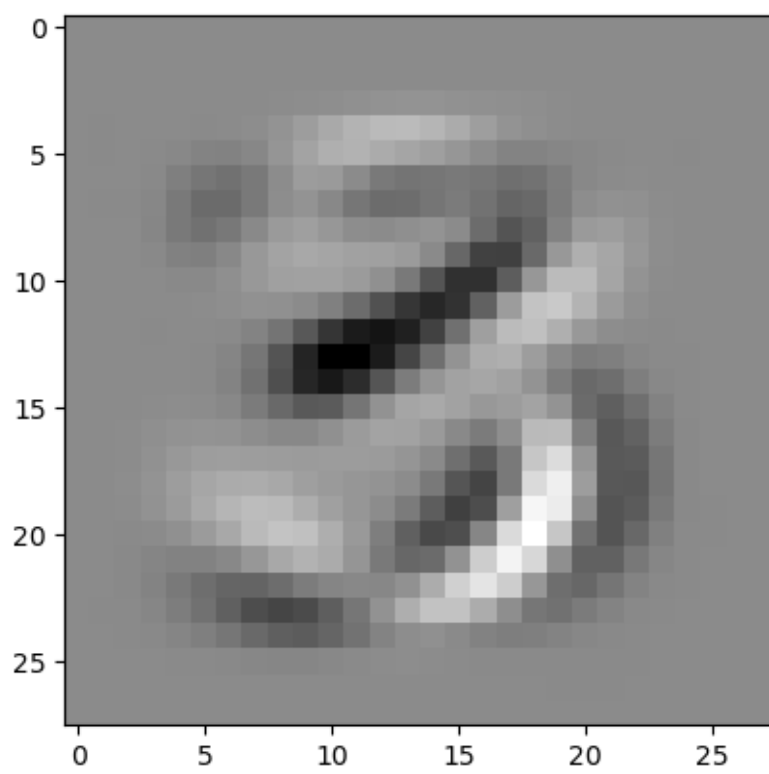


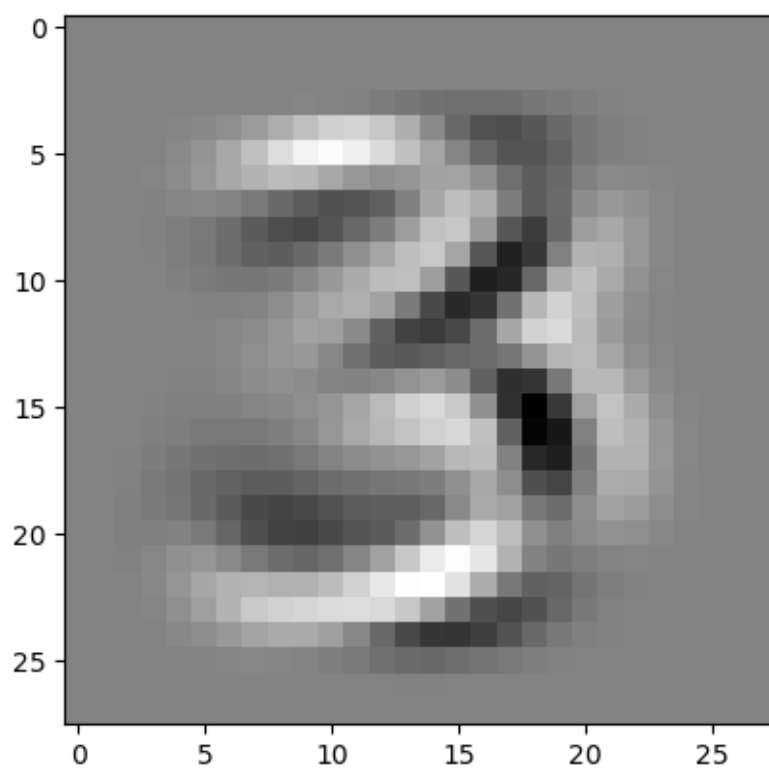
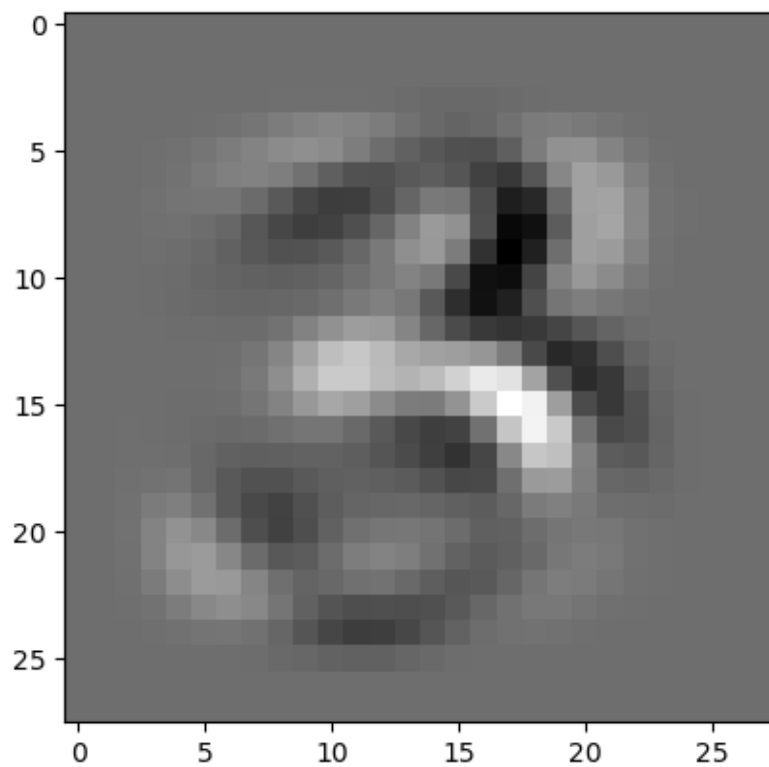


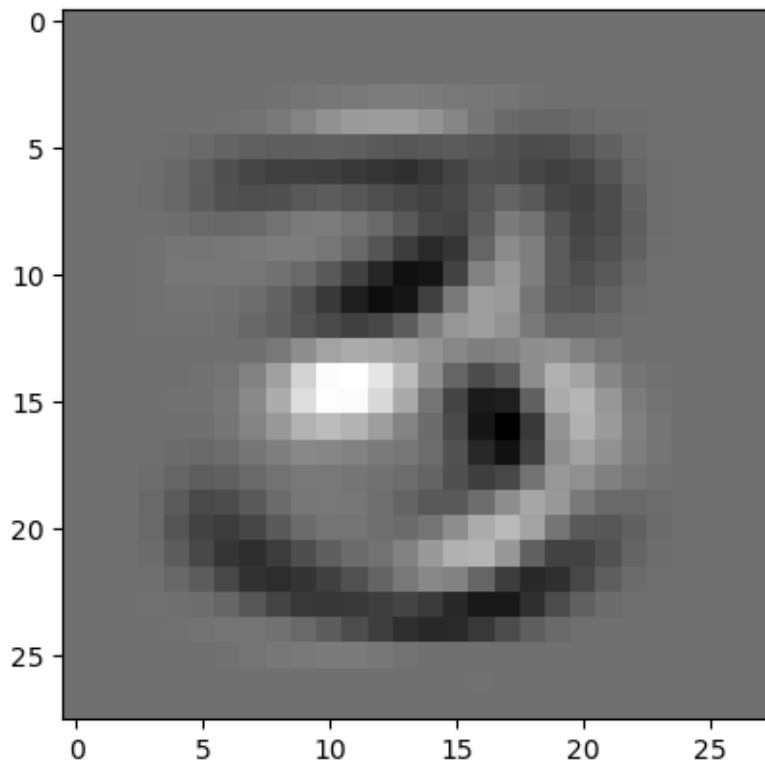












The main pattern they describe are the two curves forming the shape of the 3

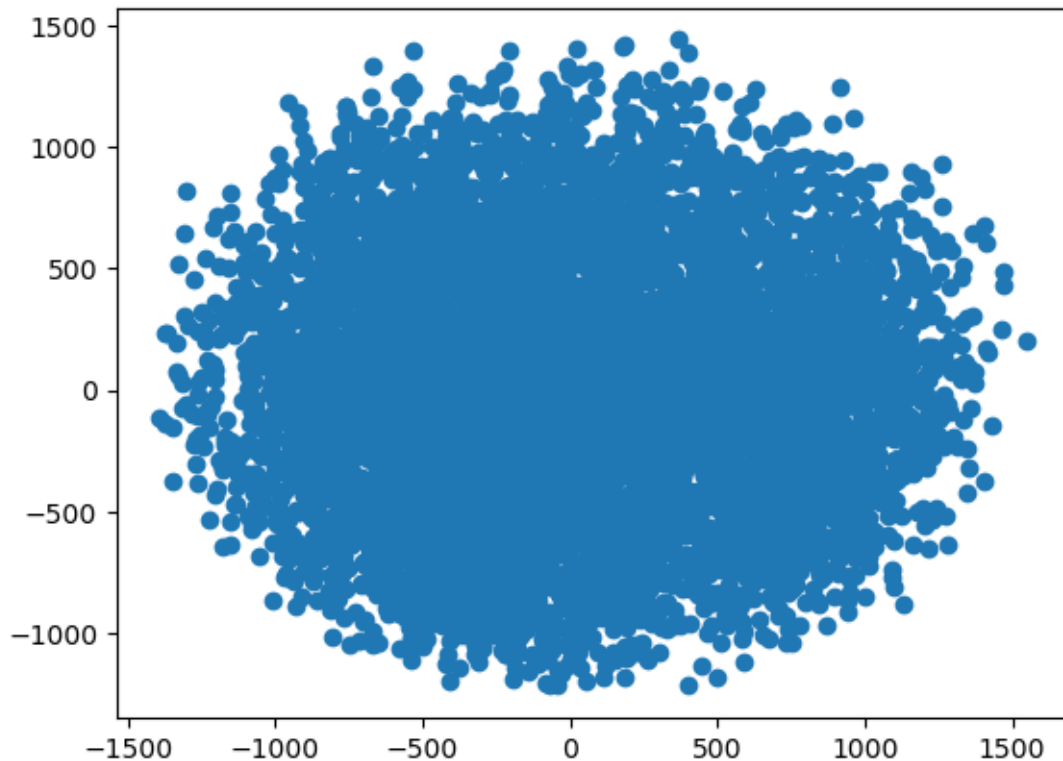
### 3.1.3 Exercise 8 (CORE)

- a) Use the `transform()` method to compute the PCA scores and save them in an object called `scores`. Then, plot the data points in the low-dimensional space spanned by the first two principal components.

```
[24]: scores = pca_threes.transform(X_threes)

# plot the first two principal components
plt.scatter(scores[:,0], scores[:,1])

N, D = X_threes.shape
```



To better interpret the latent dimensions, let's look at some projected points along each dimension and the corresponding images. Specifically, run the following code to:

- first compute the 5, 25, 50, 75, 95% quantiles of the scores for the first two dimensions
- then find the data point whose projection is closest to each combination of quantiles.

```
[25]: s1q = np.quantile(scores[:,0],[.05,.25,.5,.75,.95])
s2q = np.quantile(scores[:,1],[.05,.25,.5,.75,.95])

idx = np.zeros([len(s1q),len(s2q)])

for i in range(len(s1q)):
    for j in range(len(s2q)):
        aux = ((scores[:,0] - s1q[i])**2 + (scores[:,1] - s2q[j])**2).
        ↪reshape(N,1)
        idx[i,j] = np.where(aux == min(aux))[0][0]

idx = idx.astype(int)
```

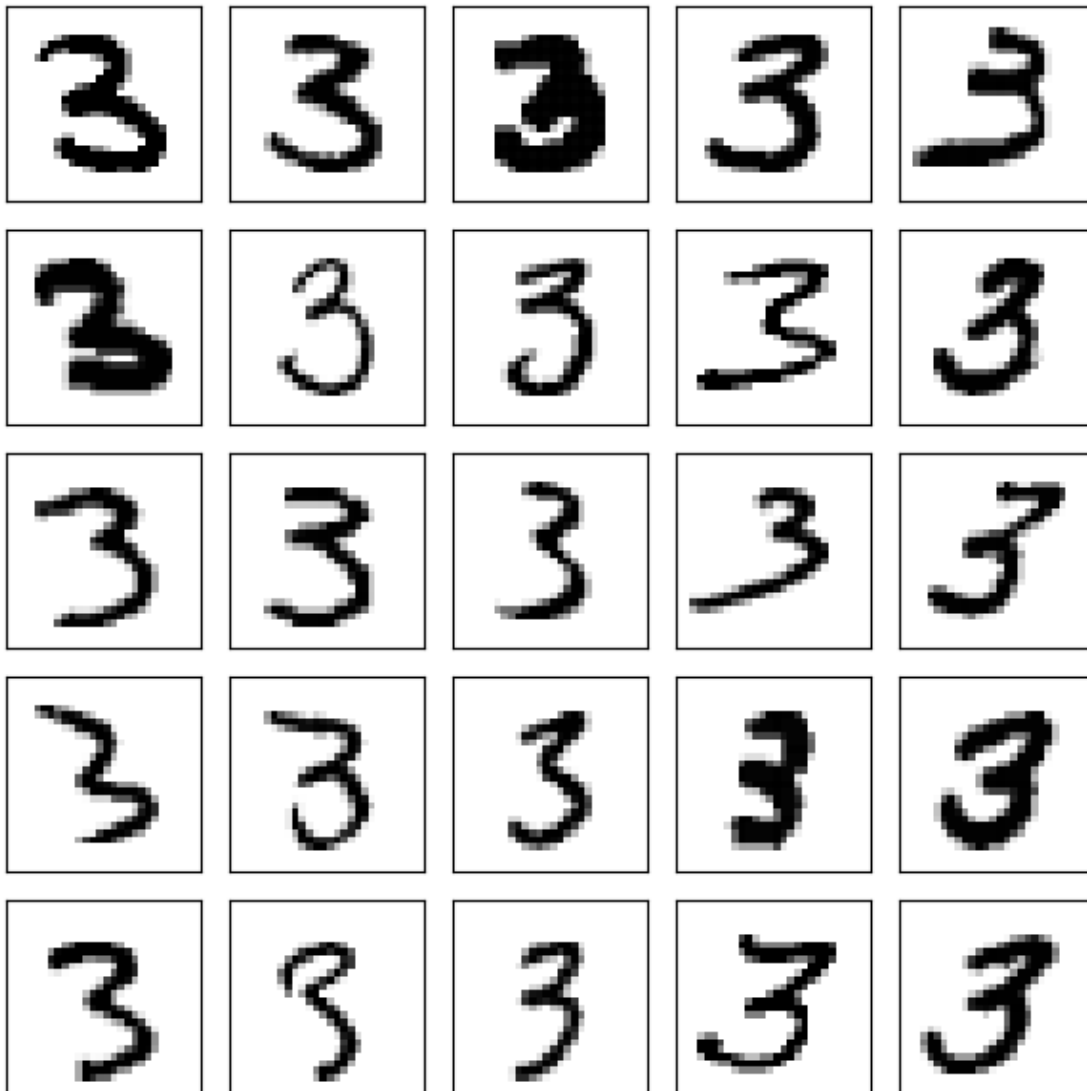
b) Now, add these points in red to your plot above in.

```
[26]: idx
```

```
[26]: array([[6396, 4562, 6506, 6077, 452],
           [ 584, 4811, 5928, 612, 1001],
           [1620, 6261, 589, 5318, 861],
           [2480, 896, 4300, 4990, 7020],
           [5834, 3379, 5868, 6891, 5516]])
```

c) Run the following code to plot the images corresponding to this grid of points. Describe the general pattern of the first (left to right) and second (down to up) principal component.

```
[27]: fig, ax = plt.subplots(len(s1q),len(s2q),figsize=(6,6))
      for i in range(len(s1q)):
          for j in range(len(s2q)):
              ax[len(s2q)-1-j,i].imshow(X_threes[idx[i,j],:].reshape((28,28)),
              cmap='gray_r')
      plt.setp(ax, xticks=[], yticks=[])
      fig.tight_layout()
```





The stroke on the 3s on the first row are thicker compared to the rest.

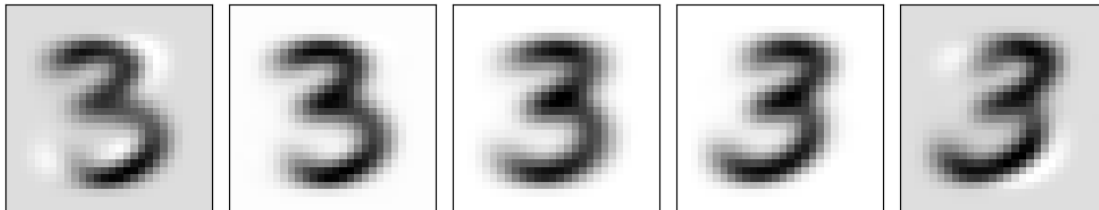
You can also try to create some artificial images, by fixing different values of the weights. This can also help to interpret the latent dimensions.

```
[28]: weight1 = [-1000,-500,0,500,1000]
weight2 = 0

images_pc1 = np.zeros([len(weight1),D])

count = 0
for w in weight1:
    images_pc1[count,:] = (pca_threes.mean_ + pca_threes.components_[0,:
↪]*w+pca_threes.components_[1,:]*weight2)
    count += 1

fig, ax = plt.subplots(1,len(weight1),figsize=(10,6))
for i in range(len(weight1)):
    ax[i].imshow(images_pc1[i,:].reshape((28,28)), cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```

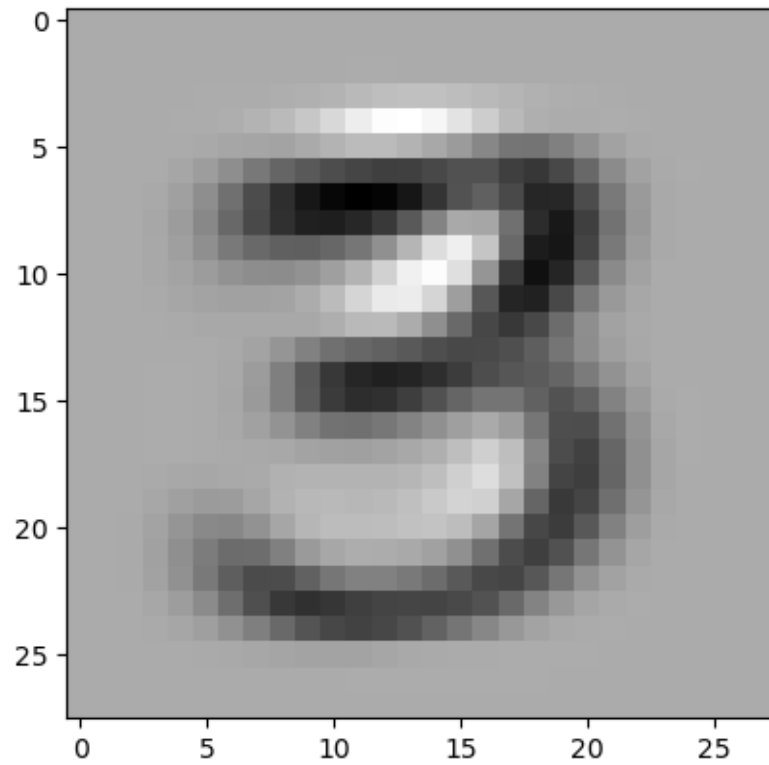


### 3.1.4 Exercise 9 (CORE)

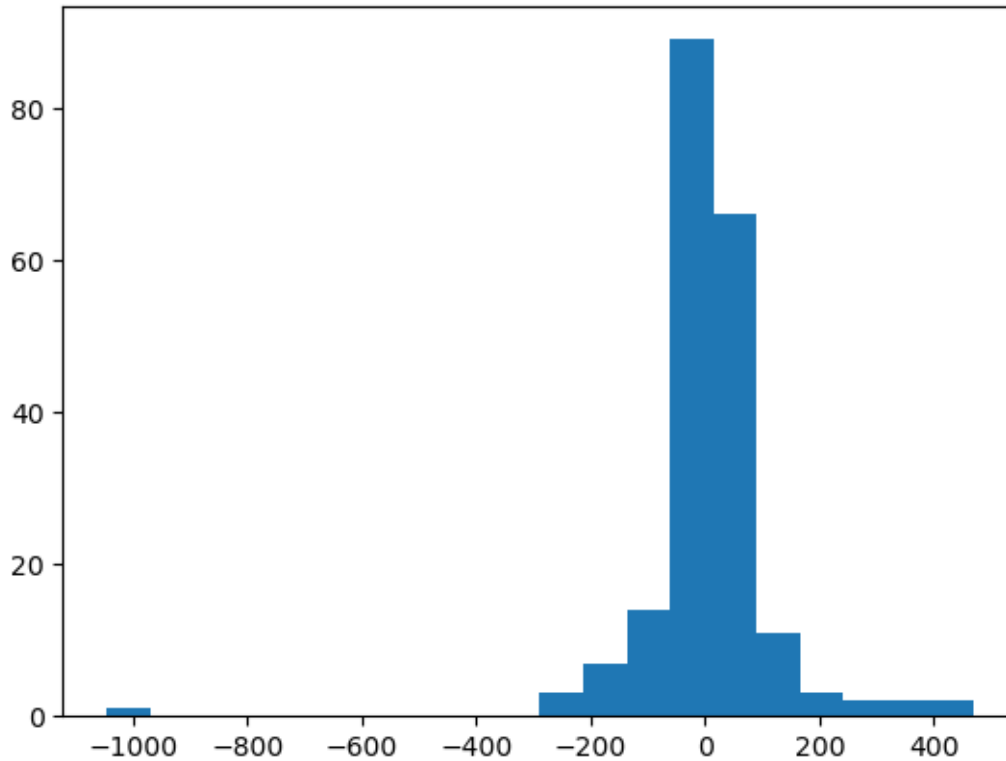
Repeat this to describe the third principal component. Look at the histogram of its scores to decide what values of weights to use.

```
[29]: # First we plot the 3rd principal component.
plt.imshow(pca_threes.components_[2].reshape((28,28)), cmap='gray_r')
```

```
[29]: <matplotlib.image.AxesImage at 0x15e8f8230>
```



```
[30]: # next we plot the histogram of scores to identify weight values:  
plt.hist(scores[2], bins=20)  
plt.show()
```



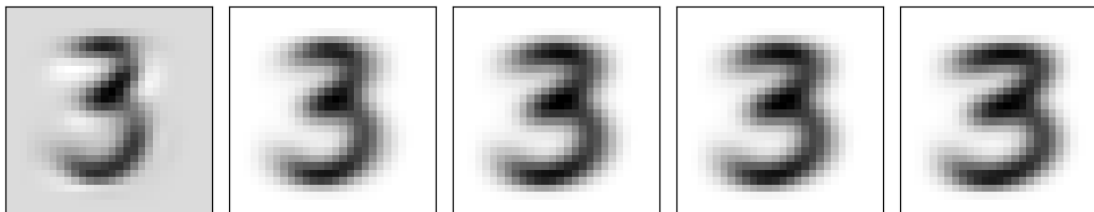
```
[31]: # define our weights using the scores:
weight1 = [-1000, -300, 0, 50, 400]

# run the same codeblock as before:
images_pc1 = np.zeros([len(weight1),D])

count = 0
for w in weight1:

    #set to be the third principal component and remove the w2 weightage on
    ↪ other components.
    images_pc1[count,:] =(pca_threes.mean_ + pca_threes.components_[2,:]*w)
    count += 1

fig, ax = plt.subplots(1,len(weight1),figsize=(10,6))
for i in range(len(weight1)):
    ax[i].imshow(images_pc1[i,:].reshape((28,28)), cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```



We don't observe much change when the weights are above -300 but at -1000 we do see a difference in image.

### 3.1.5 Exercise 10 (EXTRA)

In lecture, we saw that we can also compute the basis vectors from an SVD decomposition of the data matrix. Use the `svd` function in `scipy.linalg` to compute the first three basis vectors and verify that they are the same (up to a change in sign – note that the signs may be flipped because each principal component specifies a direction in the  $D$ -dimensional space and flipping the sign has no effect as the direction does not change).

Does `PCA()` perform principal component analysis using an eigendecomposition of the empirical covariance matrix or using a SVD decomposition of the data matrix?

[ ]:

Now, is a good point to switch driver and navigator

## 3.2 Selecting the Number of Components

### 3.2.1 Exercise 11 (CORE)

Next, let's investigate how many components are needed by considering how much variance is explained by each component.

Note that the `pca_threes` object has an attribute `explained_variance_` (variance of each component) and `explained_variance_ratio_` (proportion of variance explained by each component).

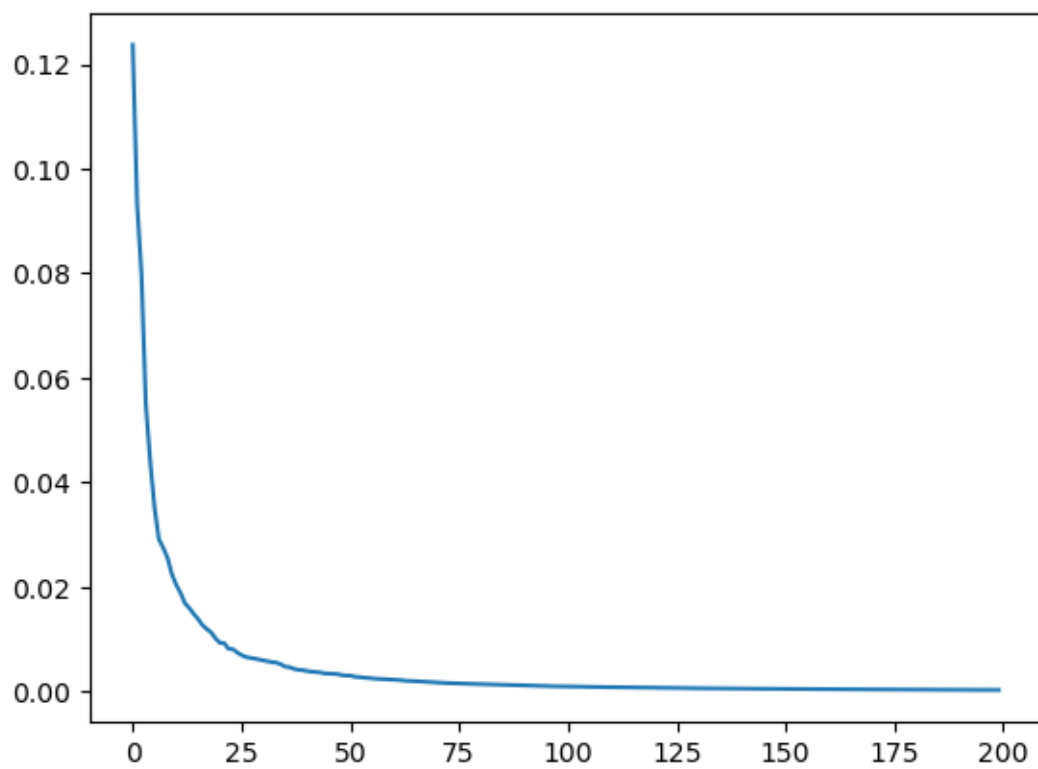
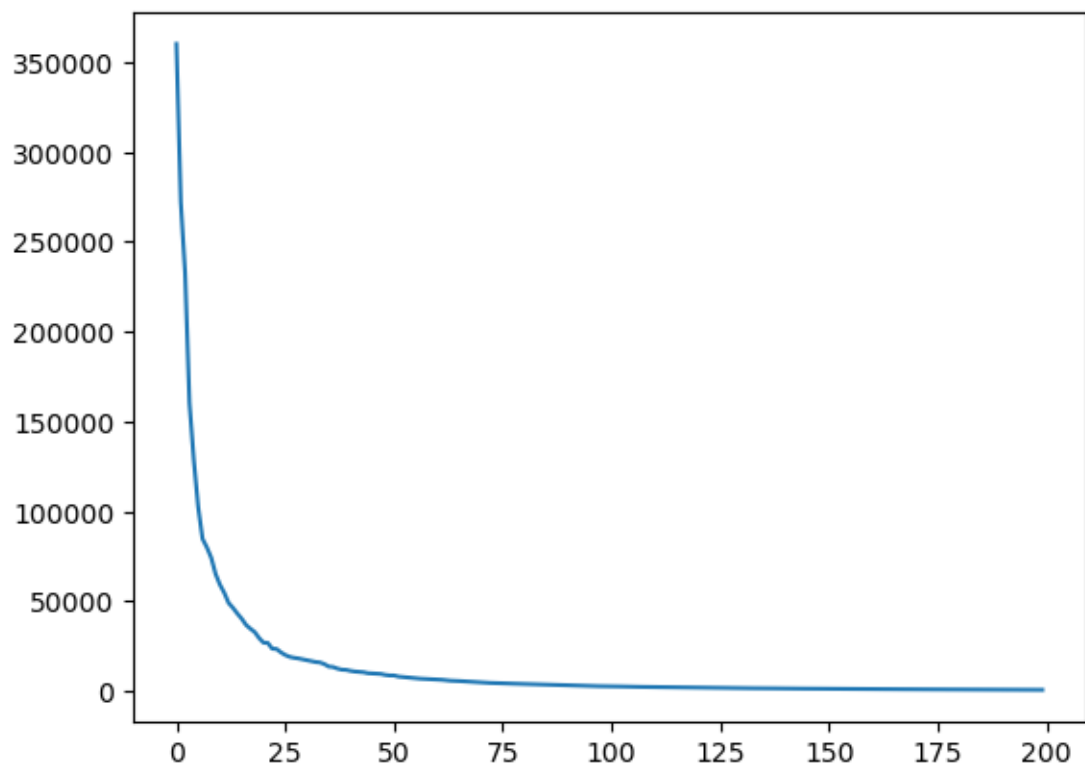
Plot both the proportion of variance explained and the cumulative proportion of variance explained. Provide a suggestion of how many components to use. How much variance is explained by the suggest number of components? Comment on why we may be able to use this number of components in relation to the total number of features.

Hint

You can use `cumsum()` to compute the cumulative sum of the elements in a vector.

```
[32]: plt.plot(pca_threes.explained_variance_)
      plt.show()

      plt.plot(pca_threes.explained_variance_ratio_)
      plt.show()
```



We see the ‘elbow’ around component 25. As such we could potentially use 25 components.

### 3.2.2 Exercise 12 (CORE)

For your selected number of components, compute the reconstructed images. Plot the reconstruction for a few images and compare with the original images. Comment on the results.

Hint

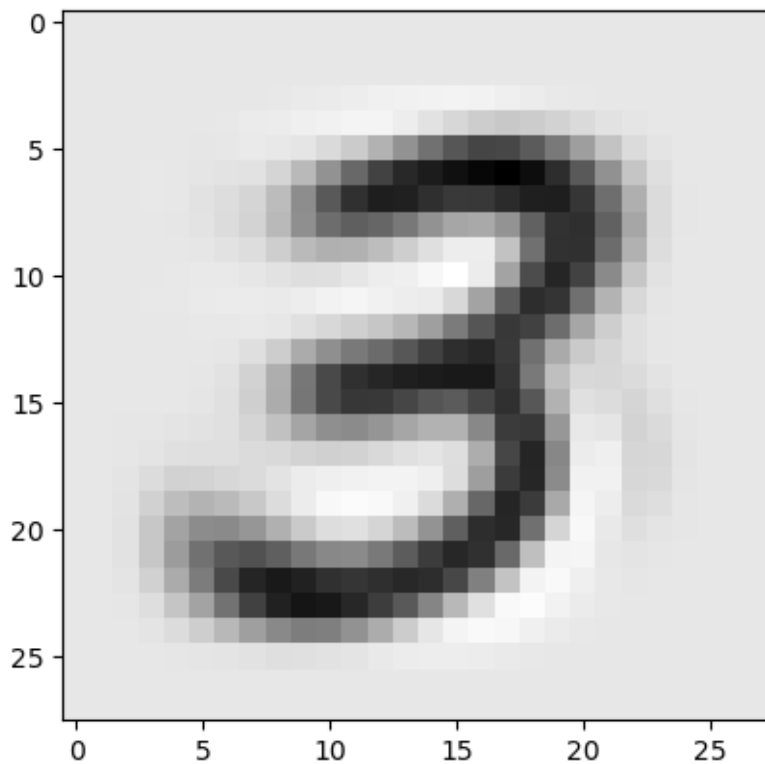
You can use `inverse_transform()` to decode the scores.

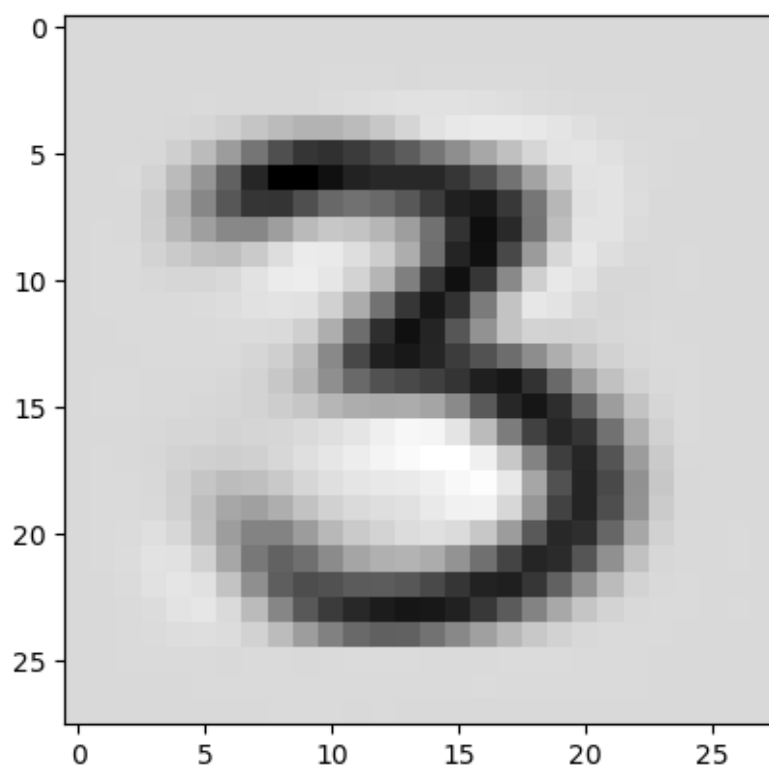
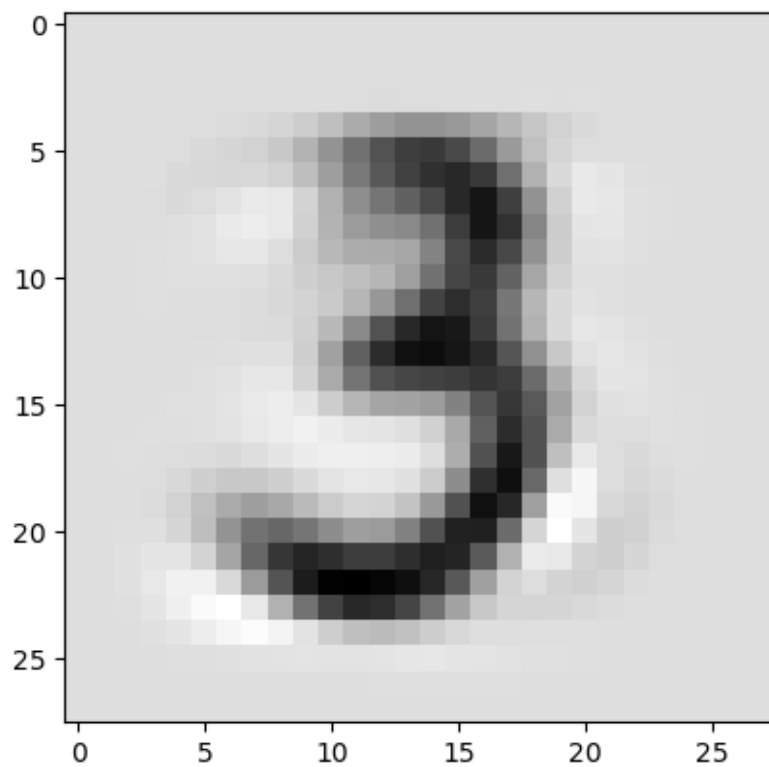
```
[39]: pca_25 = PCA(n_components = 25)
      pca_25.fit(X_threes)

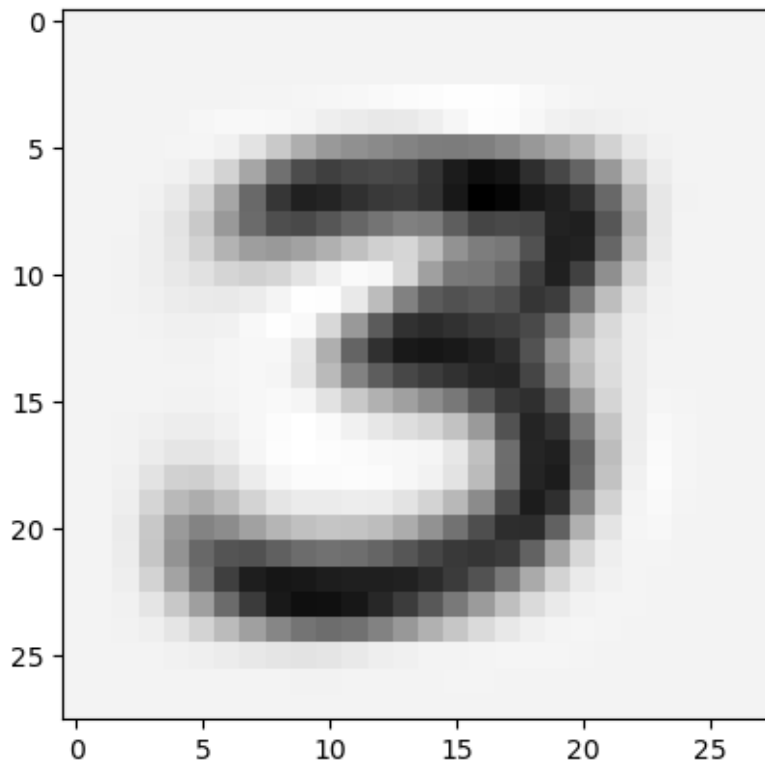
      # plot reconstructed images:

      X_reconstructed = pca_25.inverse_transform(pca_25.transform(X_threes))

      for i in range(4):
          plt.imshow(X_reconstructed[i].reshape((28,28)), cmap='gray_r')
          plt.show()
```







I am unsure if this is what we were supposed to do.

Now, is a good point to switch driver and navigator

### 3.3 Other Digits

Now, let's consider another digit.

#### 3.3.1 Exercise 13 (CORE)

Perform PCA for another choice of digit. What do the first two components describe? Do some digits have better approximations than others? Comment on why this may be.

```
[40]: X_sevens = np.array(digits_dict[7])

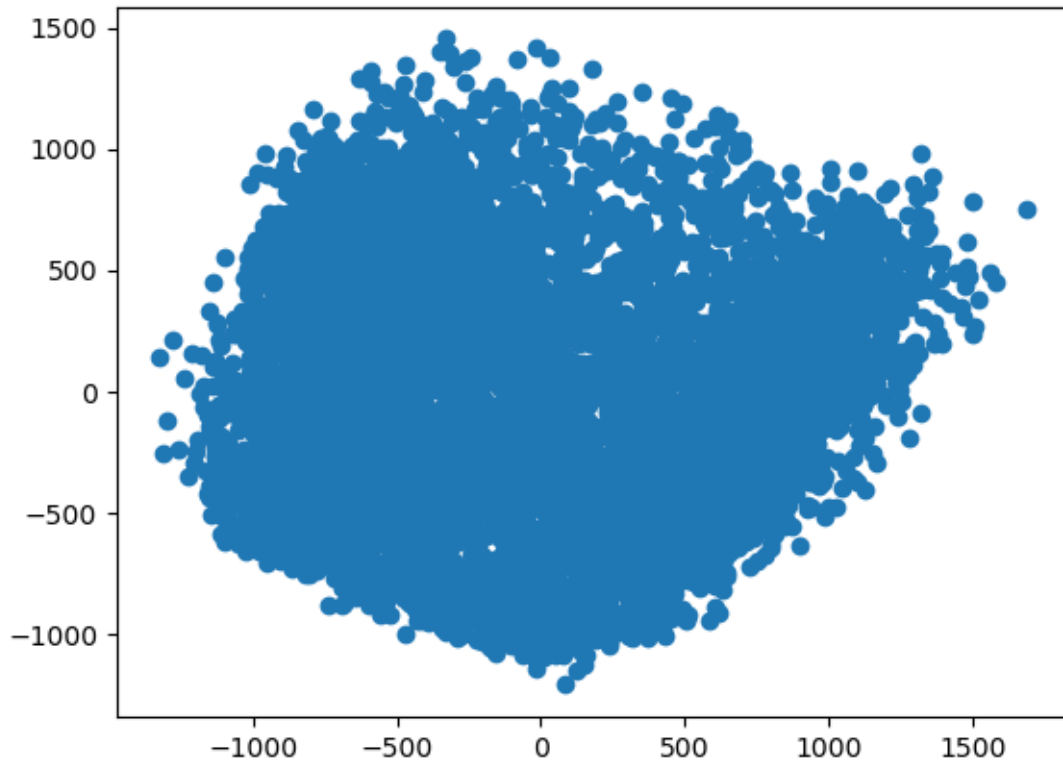
pca_25_sevens = PCA(n_components = 25)
pca_25_sevens.fit(X_sevens)

scores_25_sevens = pca_25_sevens.transform(X_sevens)

plt.scatter(scores_25_sevens[:,0], scores_25_sevens[:,1])
```



```
[40]: <matplotlib.collections.PathCollection at 0x15e9f5430>
```



Yes it would seem that some digits have better approximations. Compared to the 3s the 7s basis plot looks different. There is some linearity that can be observed in the plot above but very barely.

### 3.3.2 Exercise 14 (EXTRA)

Finally, consider now two digits of your choice (edit the code below if you wish to pick different digits).

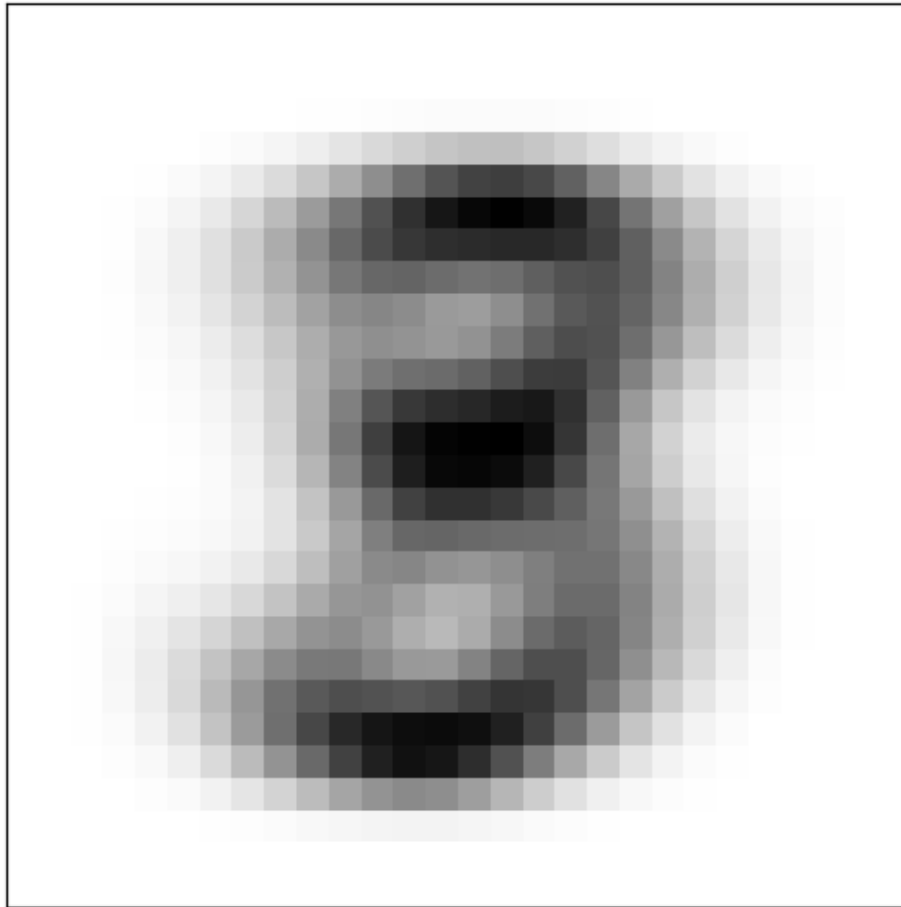
```
[42]: # Extract data
X_twodigits = np.concatenate((digits_dict[3], digits_dict[8]))
N, D = X_twodigits.shape
```

Run the following code to compute and plot the mean and some of the principle components for this dataset.

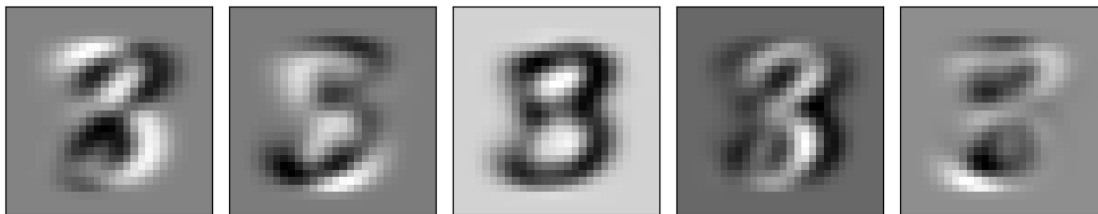
```
[43]: # Fit PCA
pca_digits = PCA(n_components = 50)
pca_digits.fit(X_twodigits)
```

```
[43]: PCA(n_components=50)
```

```
[44]: # Plot the mean image
fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(111)
ax.imshow(pca_digits.mean_.reshape(28, 28), cmap='gray_r')
ax.set_xticks([])
ax.set_yticks([])
fig.tight_layout()
```



```
[45]: # Plot basis vectors
n_plot = 5
fig, ax = plt.subplots(1,5,figsize=(10,4))
for n in range(n_plot):
    ax[n].imshow(pca_digits.components_[n,:].reshape((28,28)), cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```

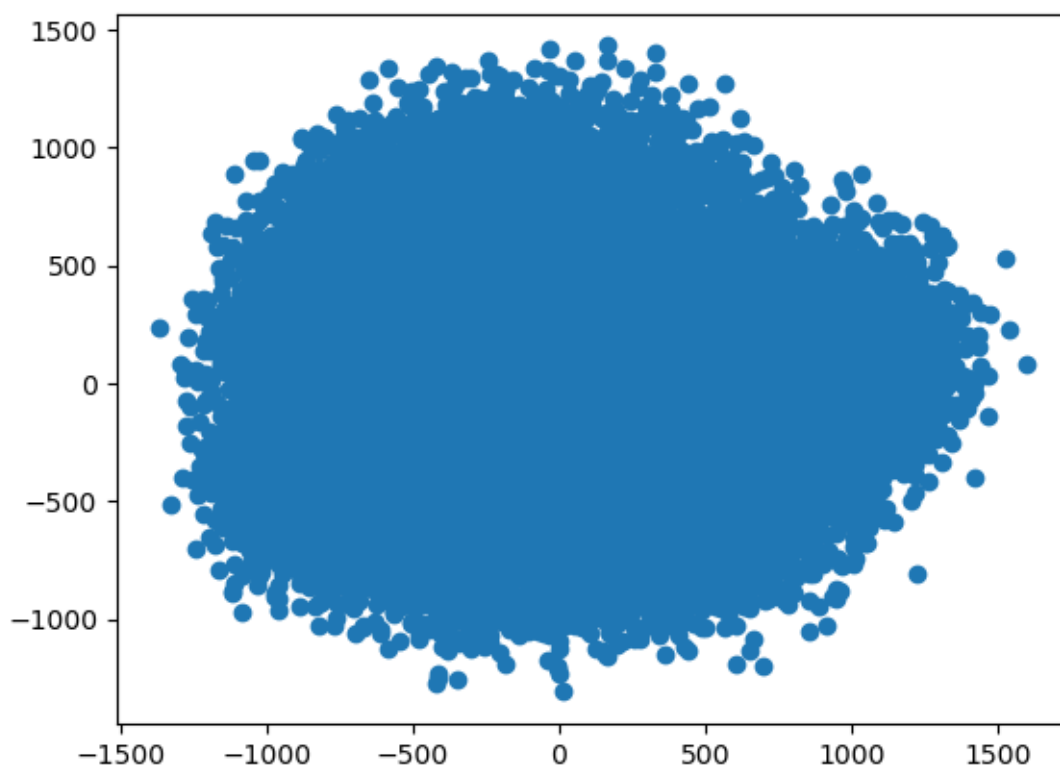


Plot the projection of the data in the latent space and color the data by the labels. What do you observe?

```
[ ]: # plot project of data in the latent space and color data by the labels:
scores = pca_digits.transform(X_twodigits)

# unsure how to label the colors:
plt.scatter(scores[:,0], scores[:,1])
```

```
[ ]: <matplotlib.collections.PathCollection at 0x164a9bb60>
```



Try also to generate artificial images and describe how images change along the PCs.

[ ]:

## 4 Competing the Worksheet

At this point you have hopefully been able to complete all the CORE exercises and attempted the EXTRA ones. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Before generating the PDF, please go to Edit -> Edit Notebook Metadata and change 'Student 1' and 'Student 2' in the **name** attribute to include your name.

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF.

[ ]: `!jupyter nbconvert --to pdf mlp_week02.ipynb`