Reid Jackson

4442 Assignment 2

February 29th 2020

250914839

1a)

$p(Water = warm \mid Play = yes) = 2/3$

$p(Water = warm \mid Play = no) = 1$

1b)

$p(Play = yes \mid Water = warm) = 2/3$

$p(Play = no \mid Water = warm) = 1/3$

1c)

$p(Play = yes \mid Forecast = same) = 1$

$p(Play = yes \mid Forecast = change) = 1/2$

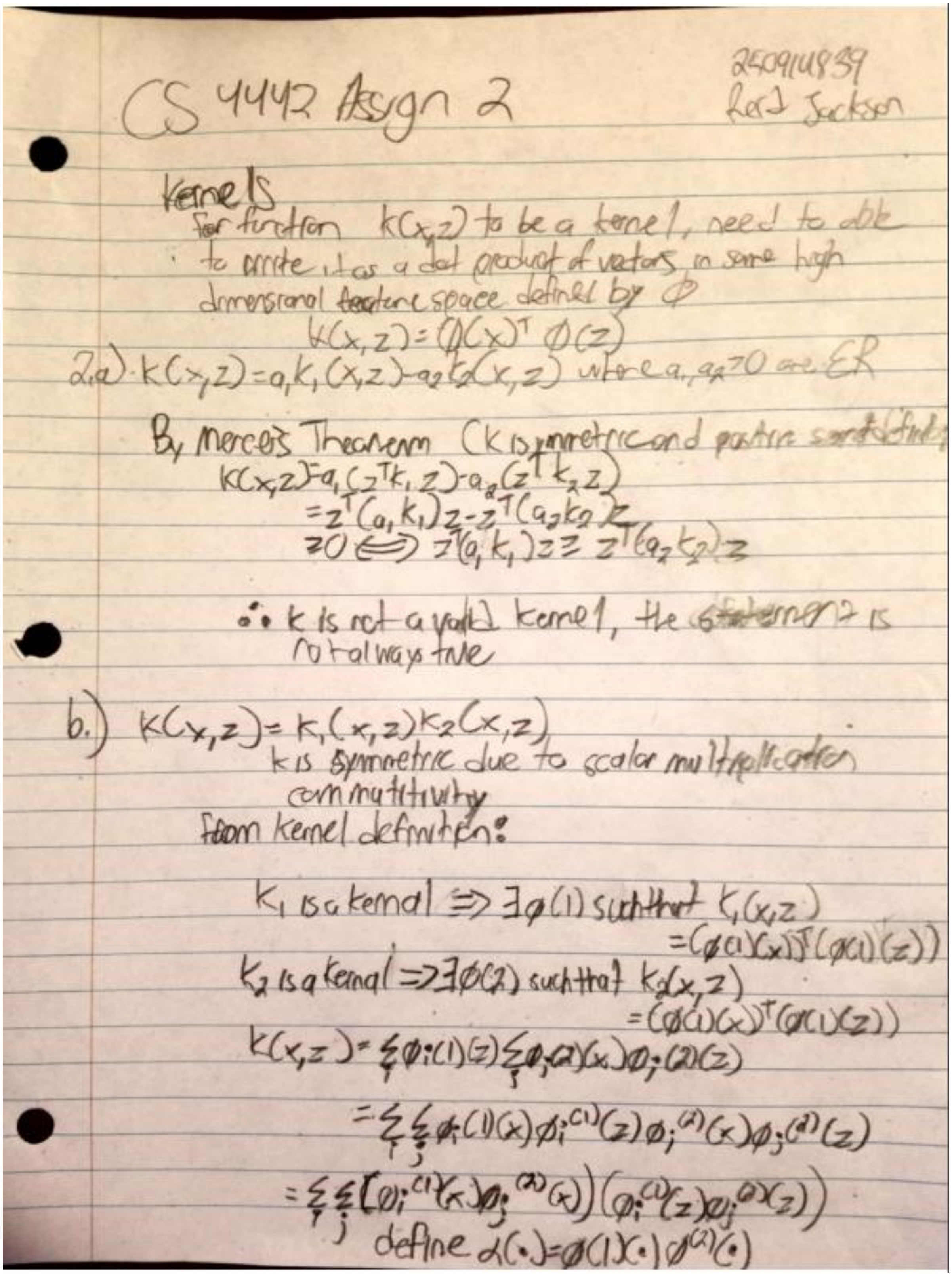
1d)

Laplace smoothing values: $a = 1$, $d = 2$

$p(Water = warm|Play = yes) = 3/5$

$p(Water = warm|Play = no) = 2/3$

2)



CS 4442 Assign 2

250914839
Reid Jackson

Kernels

For function $k(x,z)$ to be a kernel, need to able to write it as a dot product of vectors in some high dimensional feature space defined by Φ

$$k(x,z) = (\phi(x))^T \phi(z)$$

2.a) $k(x,z) = a_1 k_1(x,z) - a_2 k_2(x,z)$ where $a_1, a_2 > 0$ are $\in \mathbb{R}$

By mercer's Theorem (k is symmetric and positive semidefinite)

$$k(x,z) = a_1(z^T k_1 z) - a_2(z^T k_2 z)$$
$$= z^T(a_1 k_1)z - z^T(a_2 k_2)z$$
$$\geq 0 \iff z^T(a_1 k_1)z \geq z^T(a_2 k_2)z$$

∴ k is not a valid kernel, the statement is not always true

b.) $k(x,z) = k_1(x,z)k_2(x,z)$

k is symmetric due to scalar multiplication commutativity

from kernel definition:

k_1 is a kernal $\Rightarrow \exists \phi(1)$ such that $k_1(x,z)$
$= (\phi(1)(x))^T(\phi(1)(z))$

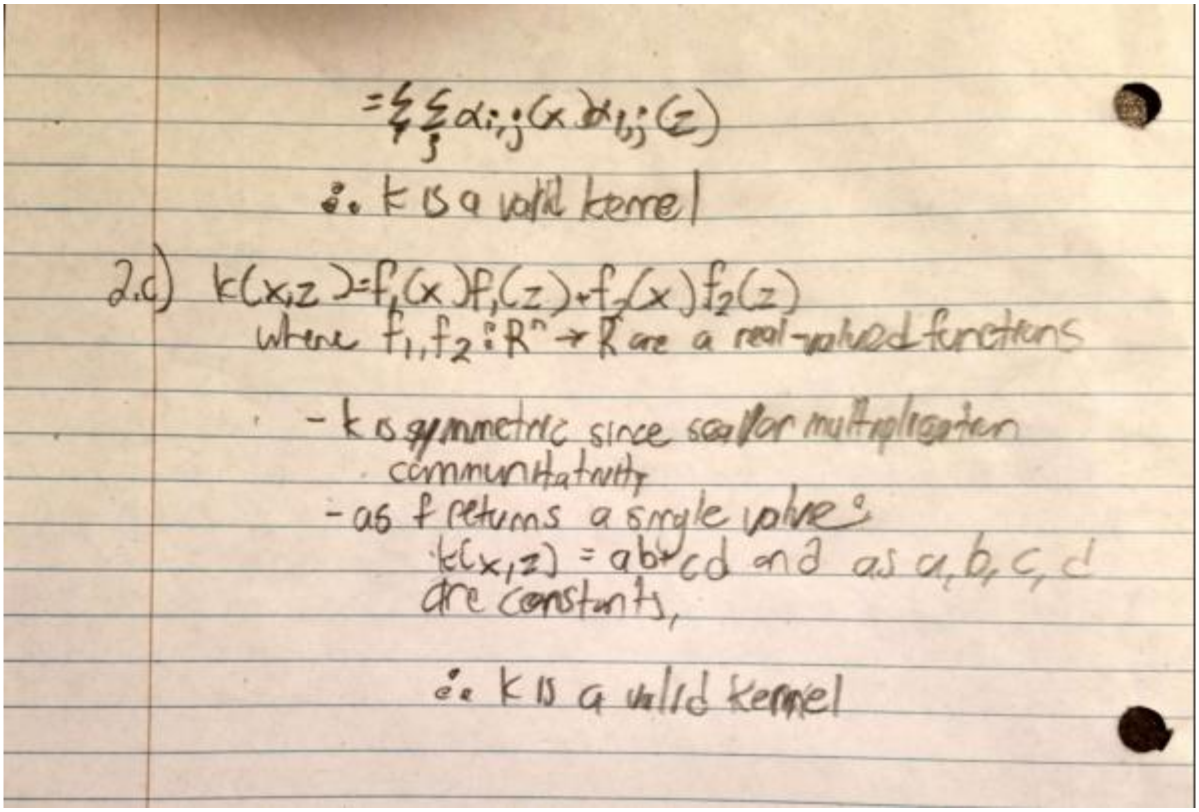
k_2 is a kernal $\Rightarrow \exists \phi(2)$ such that $k_2(x,z)$
$= (\phi(1)(x))^T(\phi(1)(z))$

$$k(x,z) = \sum_i \phi_i(1)(z) \sum_j \phi_j(2)(x)\phi_j(2)(z)$$

$$= \sum_i \sum_j \phi_i(1)(x)\phi_i^{(1)}(z)\phi_j^{(2)}(x)\phi_j^{(2)}(z)$$

$$= \sum_i \sum_j (\phi_i^{(1)}(x)\phi_j^{(2)}(x))(\phi_i^{(1)}(z)\phi_j^{(2)}(z))$$

define $\alpha(\cdot) = \phi(1)(\cdot)\phi^{(2)}(\cdot)$



$$= \sum_i \sum_j \alpha_{i,j}(x)\alpha_{i,j}(z)$$

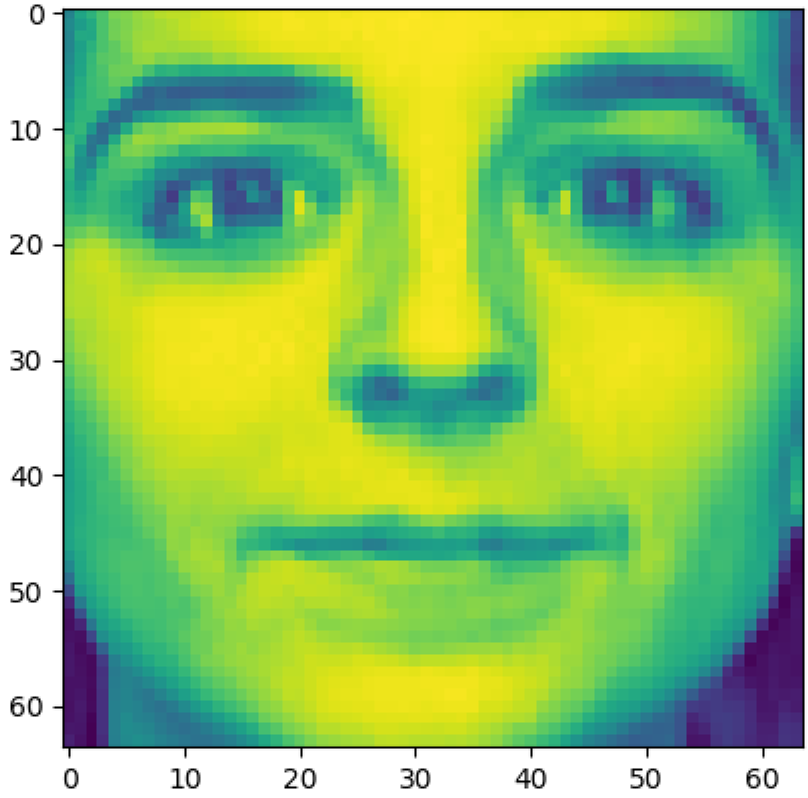
$\therefore$ k is a valid kernel

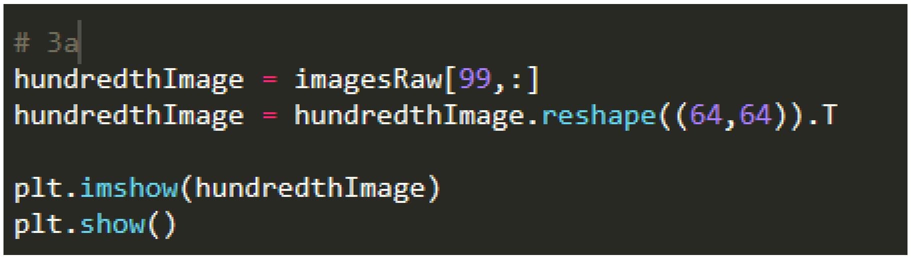
2.c) $k(x,z) = f_1(x)f_1(z) + f_2(x)f_2(z)$
where $f_1, f_2 : R^n \to R$ are a real-valued functions

- k is symmetric since scallar multiplication communitativity
- as f returns a single value:
  $k(x,z) = ab + cd$ and as a, b, c, d are constants,

$\therefore$ k is a valid kernel

3a)



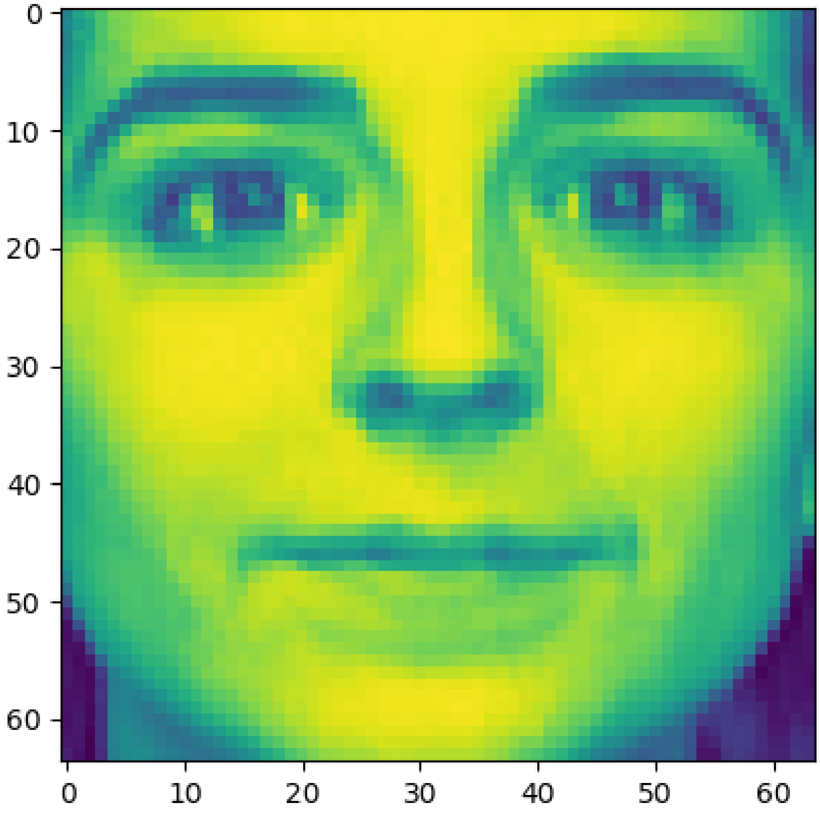


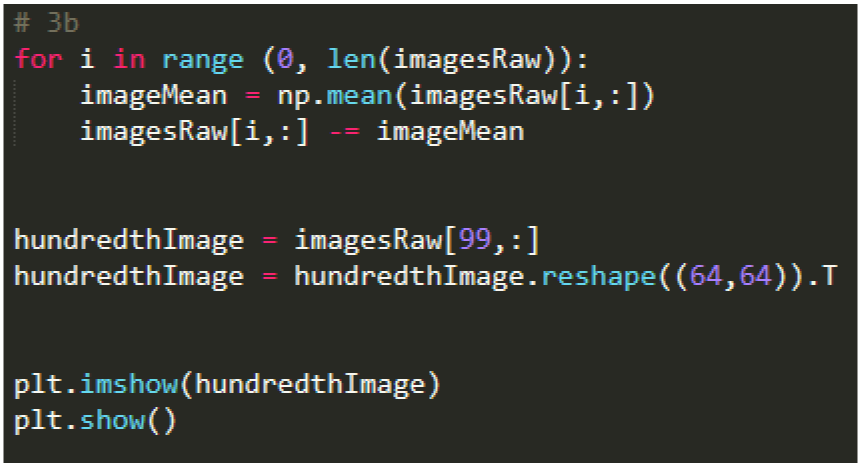
```
# 3a
hundredthImage = imagesRaw[99,:]
hundredthImage = hundredthImage.reshape((64,64)).T

plt.imshow(hundredthImage)
plt.show()
```

3b)

The images had a mean of: 132.38427856445313.

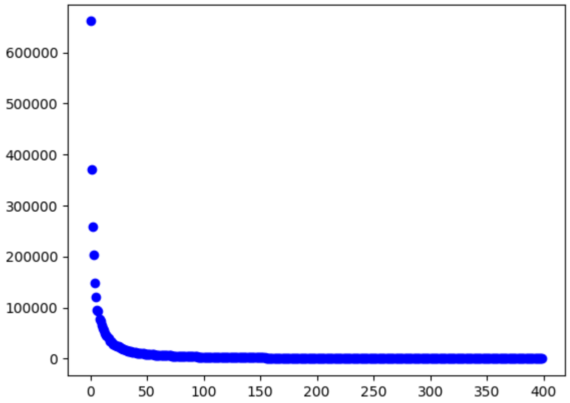
(This image is different but due to compressions the change in shading/sharpness have been lost)

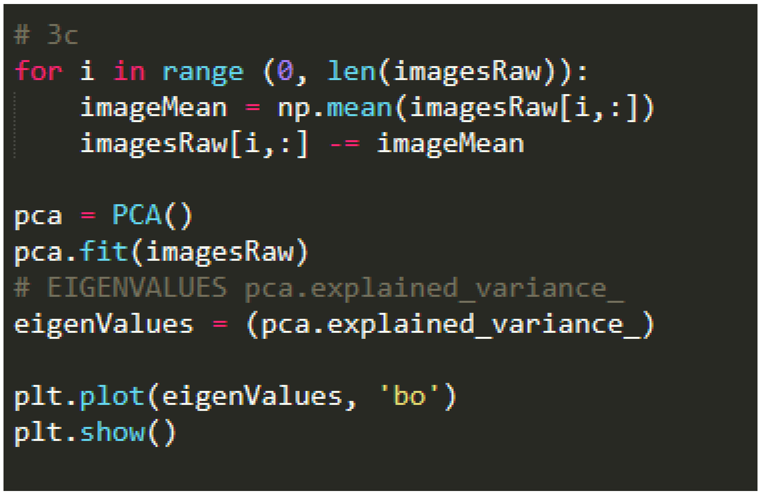




```
# 3b
for i in range (0, len(imagesRaw)):
    imageMean = np.mean(imagesRaw[i,:])
    imagesRaw[i,:] -= imageMean


hundredthImage = imagesRaw[99,:]
hundredthImage = hundredthImage.reshape((64,64)).T


plt.imshow(hundredthImage)
plt.show()
```

3c)





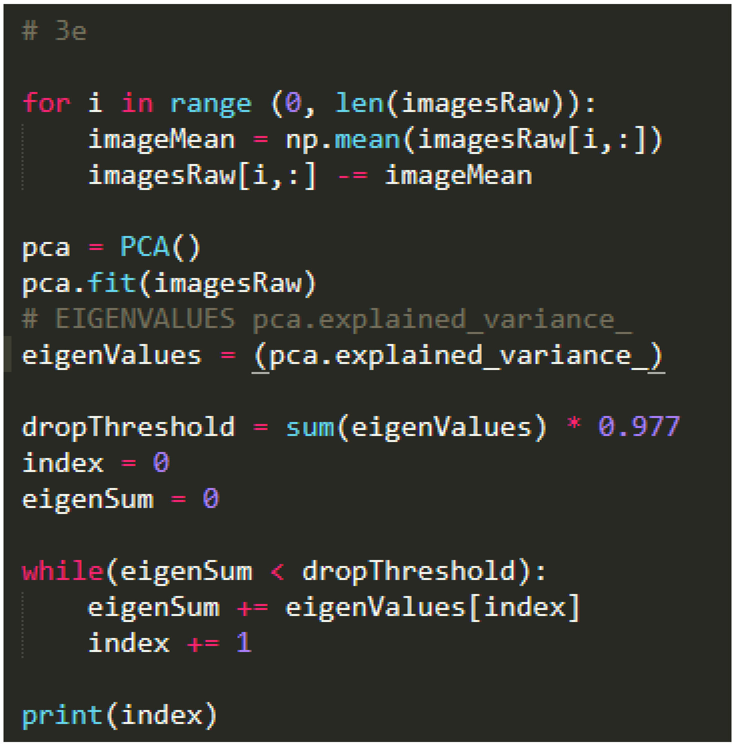
```
# 3c
for i in range (0, len(imagesRaw)):
    imageMean = np.mean(imagesRaw[i,:])
    imagesRaw[i,:] -= imageMean

pca = PCA()
pca.fit(imagesRaw)
# EIGENVALUES pca.explained_variance_
eigenValues = (pca.explained_variance_)

plt.plot(eigenValues, 'bo')
plt.show()
```

3d)

The last image, 400 reveals the least about the data after the previous 399 images have been analyzed and learned from. This eigenvalue shows that the rest of the images reveal most (99.9%+) of the variance between the images.

3e)

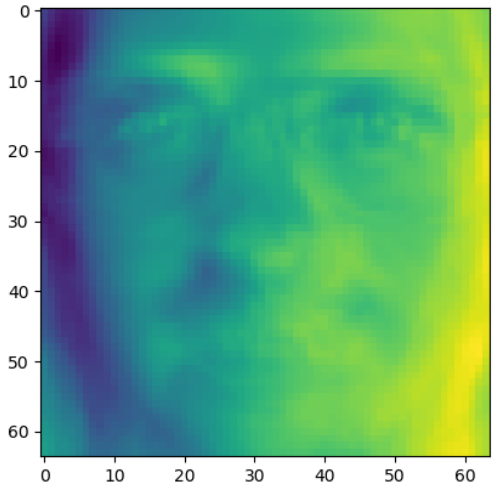
I used 97.7% for the cutoff point as this number is the value for 2 standard deviations above the mean for a stand distribution. With this cut off, *213* was the number found, so the first 213 components will be kept, while the other 187 will be dropped.

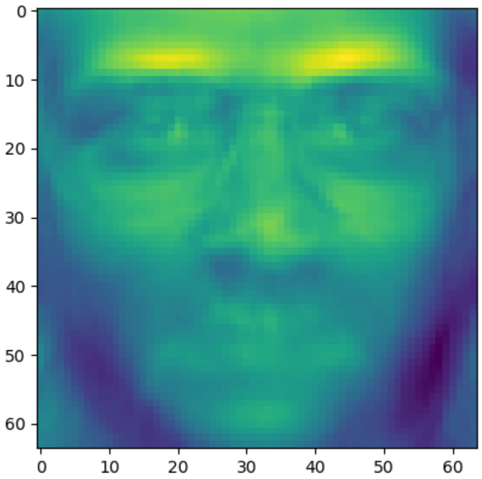


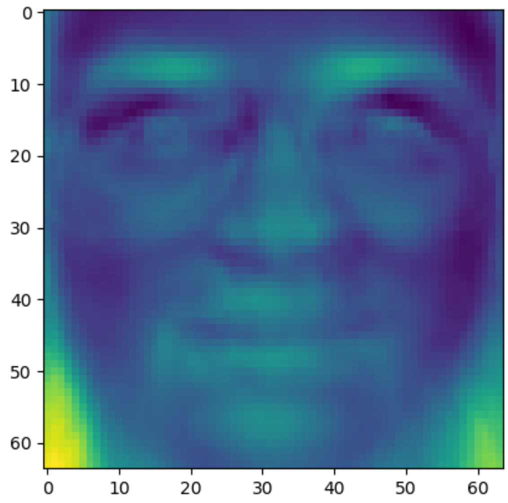
```
# 3e

for i in range (0, len(imagesRaw)):
    imageMean = np.mean(imagesRaw[i,:])
    imagesRaw[i,:] -= imageMean

pca = PCA()
pca.fit(imagesRaw)
# EIGENVALUES pca.explained_variance_
eigenValues = (pca.explained_variance_)

dropThreshold = sum(eigenValues) * 0.977
index = 0
eigenSum = 0

while(eigenSum < dropThreshold):
    eigenSum += eigenValues[index]
    index += 1

print(index)
```

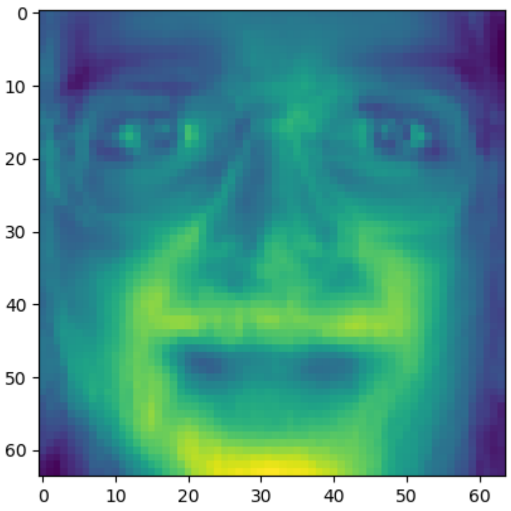
3f)

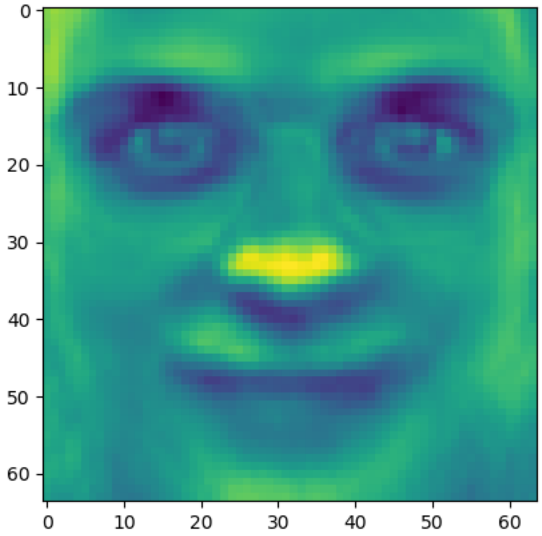
In order from top 1 to top 5 leading eigenvectors...

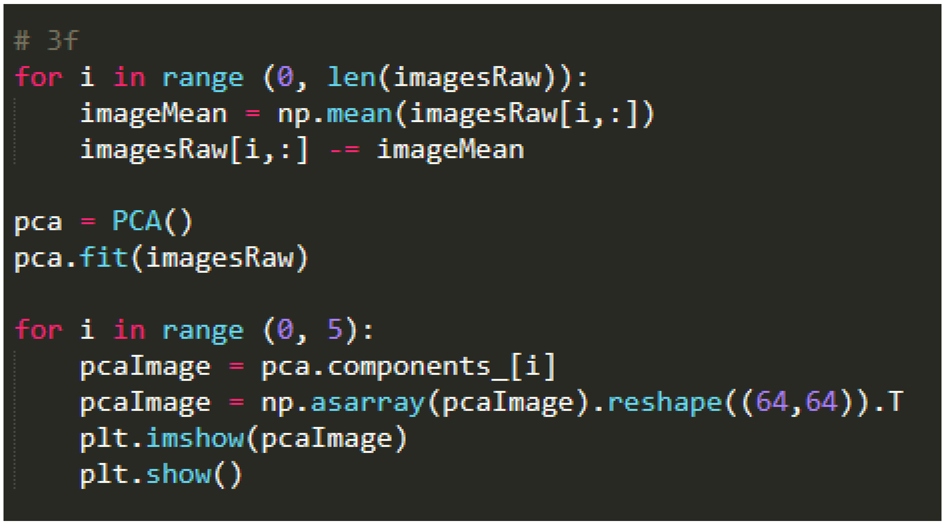








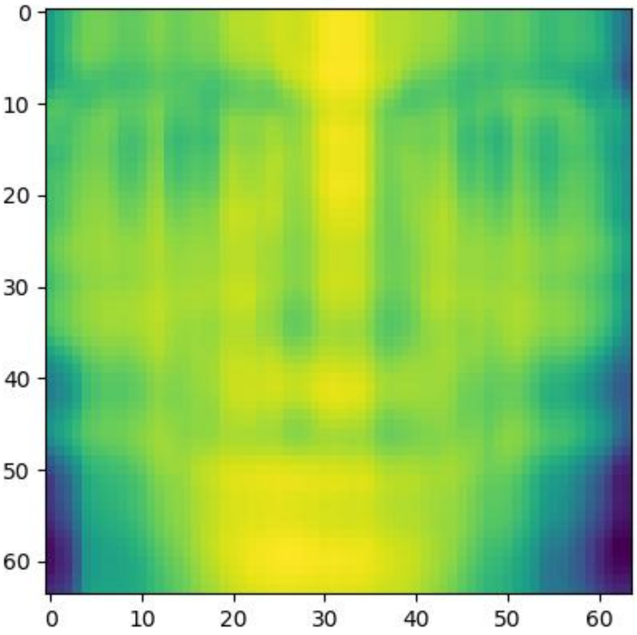




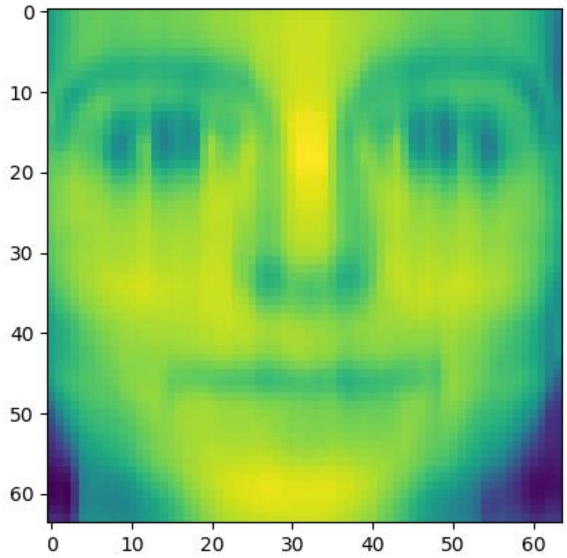
```
# 3f
for i in range (0, len(imagesRaw)):
    imageMean = np.mean(imagesRaw[i,:])
    imagesRaw[i,:] -= imageMean

pca = PCA()
pca.fit(imagesRaw)

for i in range (0, 5):
    pcaImage = pca.components_[i]
    pcaImage = np.asarray(pcaImage).reshape((64,64)).T
    plt.imshow(pcaImage)
    plt.show()
```

3g)

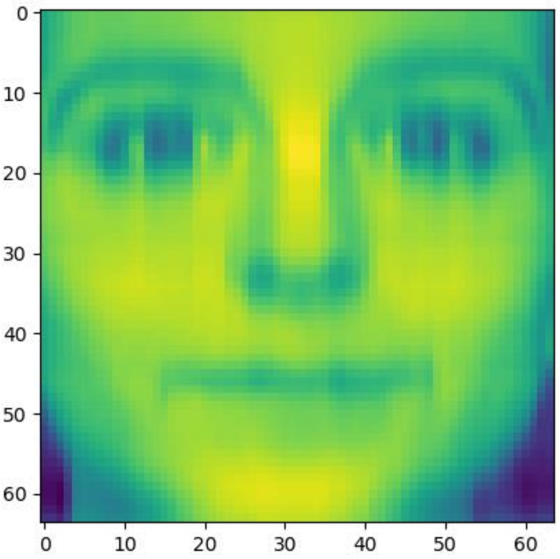
10 Components



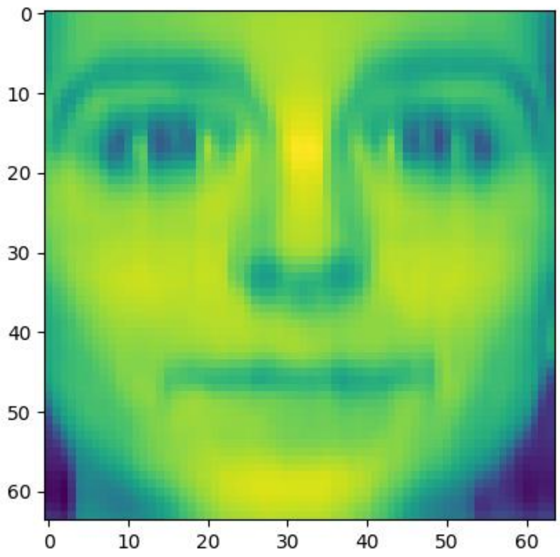
100 Components

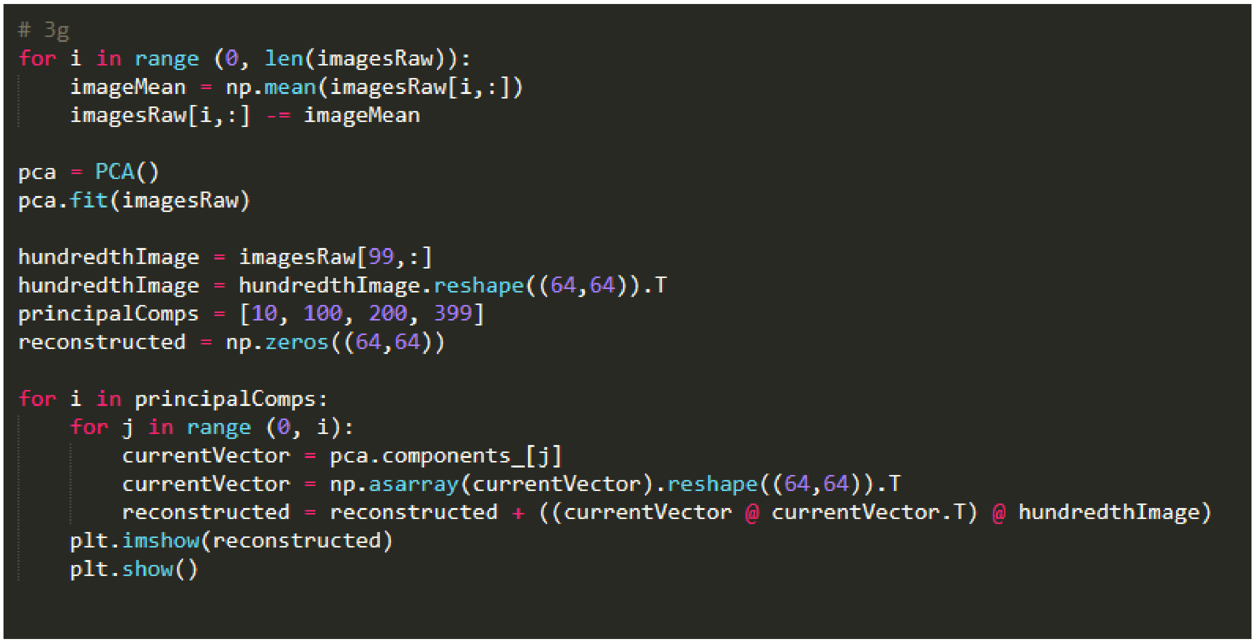


200 Components



399 Components





```
# 3g
for i in range (0, len(imagesRaw)):
    imageMean = np.mean(imagesRaw[i,:])
    imagesRaw[i,:] -= imageMean

pca = PCA()
pca.fit(imagesRaw)

hundredthImage = imagesRaw[99,:]
hundredthImage = hundredthImage.reshape((64,64)).T
principalComps = [10, 100, 200, 399]
reconstructed = np.zeros((64,64))

for i in principalComps:
    for j in range (0, i):
        currentVector = pca.components_[j]
        currentVector = np.asarray(currentVector).reshape((64,64)).T
        reconstructed = reconstructed + ((currentVector @ currentVector.T) @ hundredthImage)
    plt.imshow(reconstructed)
    plt.show()
```