Worksheet 10

Name: Jeong Yong Yang, Junho Son UID: U95912941, U64222022

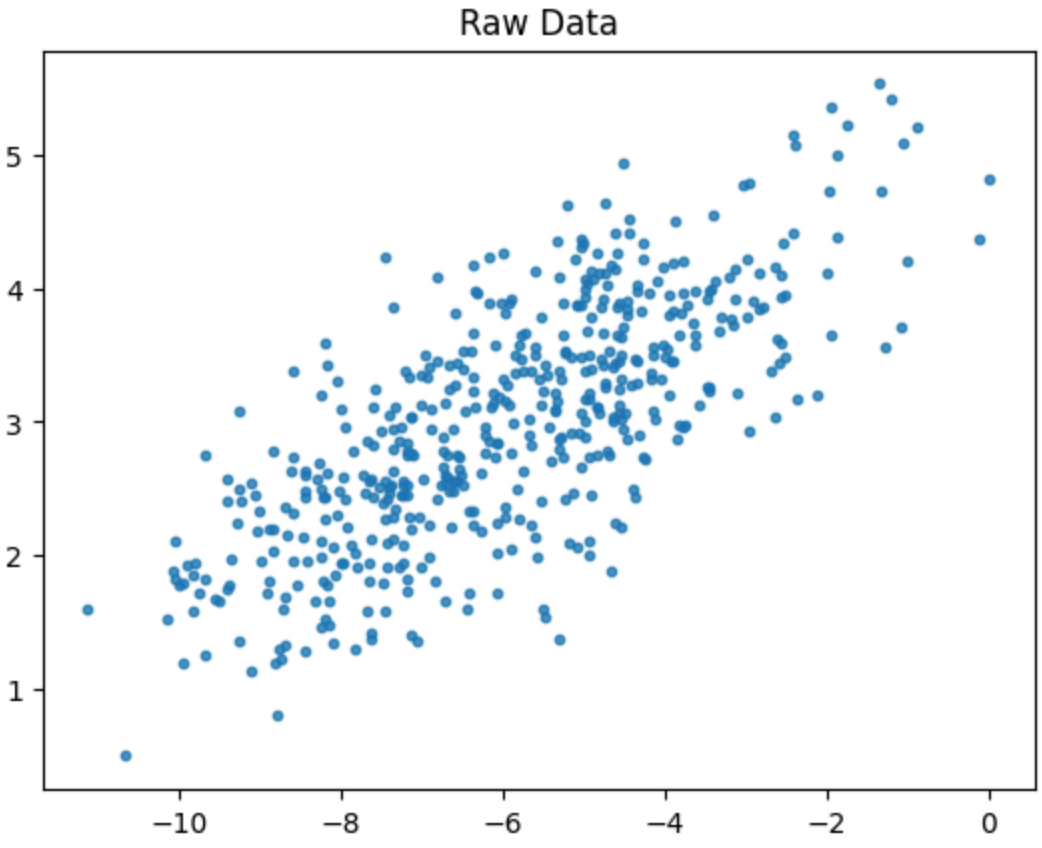
Topics

- Singular Value Decomposition

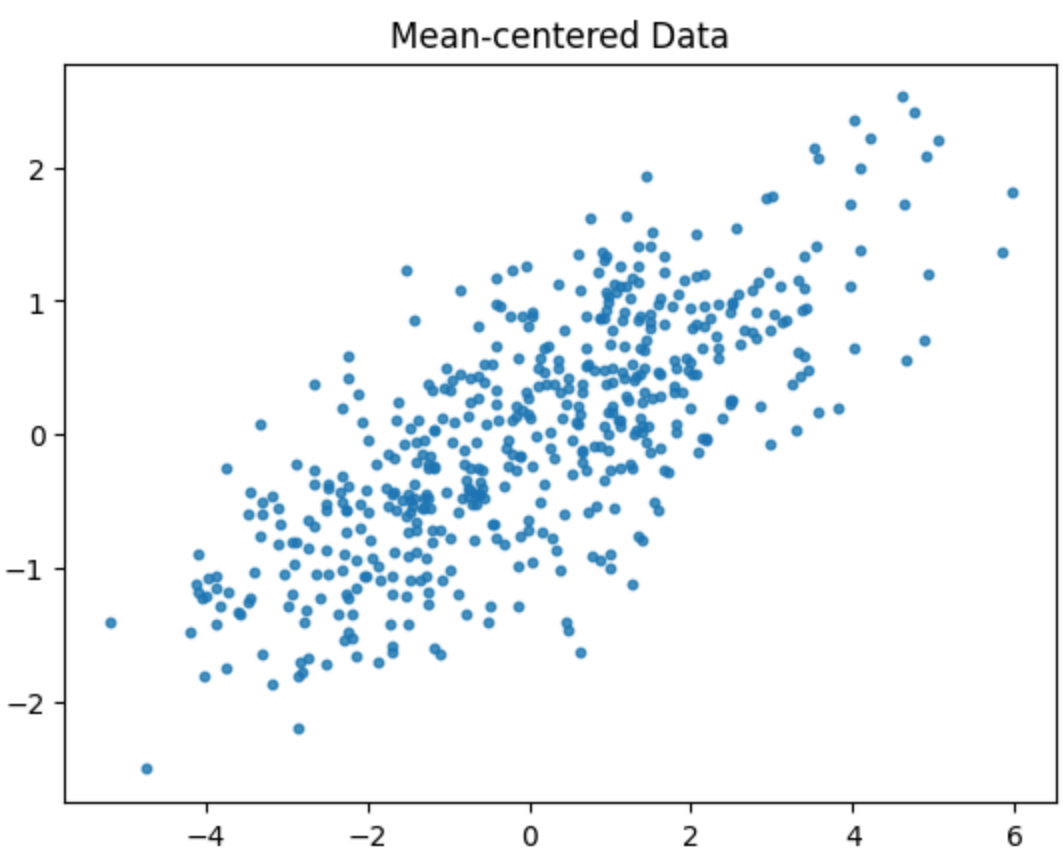
Feature Extraction

SVD finds features that are orthogonal. The Singular Values correspond to the importance of the feature or how much variance in the data it captures.

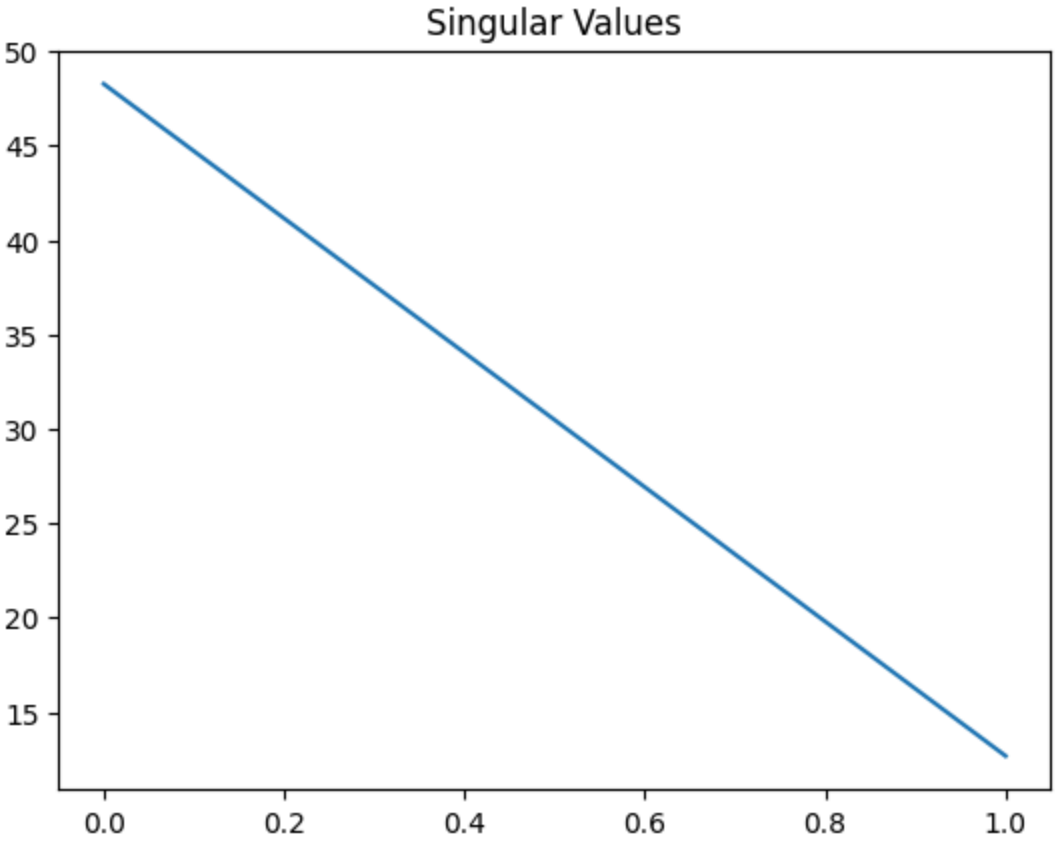
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

n_samples = 500
C = np.array([[0.1, 0.6], [2., .6]])
X = np.random.randn(n_samples, 2) @ C + np.array([-6, 3])
plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.8)
plt.title("Raw Data")
plt.show()
```



In [ ]:
```
X = X - np.mean(X, axis=0)
plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.8)
plt.title("Mean-centered Data")
plt.show()
```

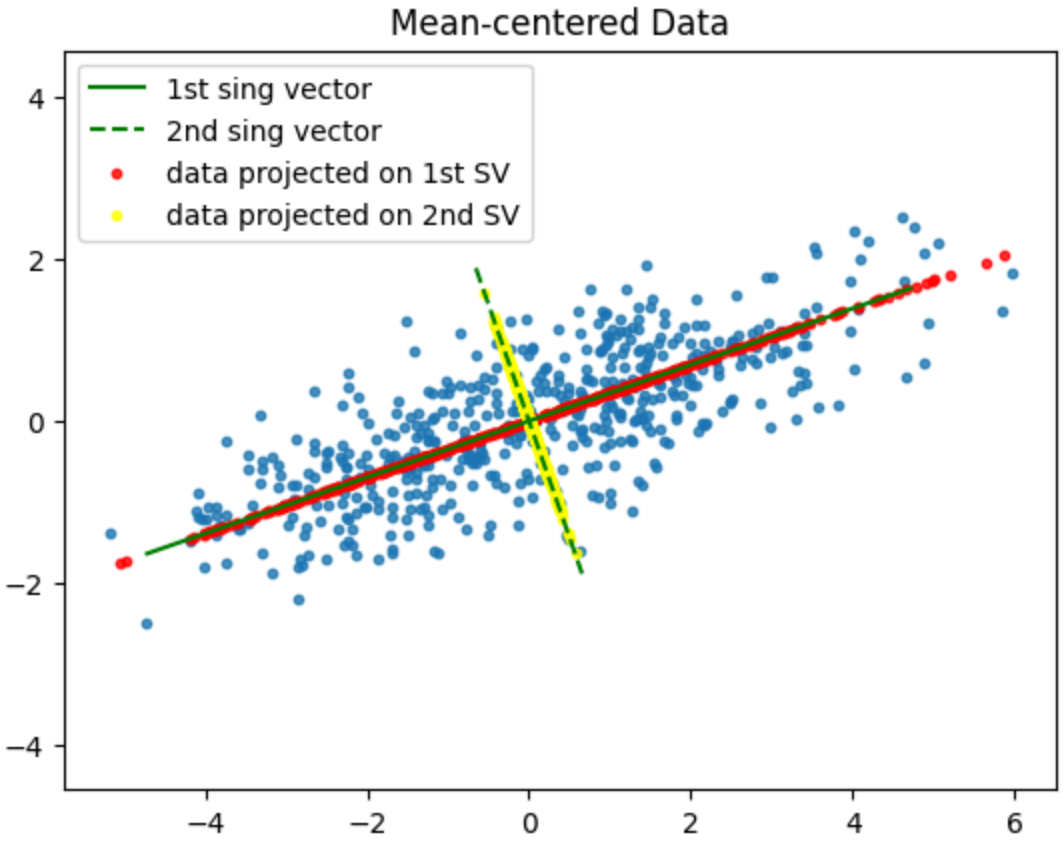


In [ ]:
```
u,s,vt=np.linalg.svd(X, full_matrices=False)
plt.plot(s) # only 2 singular values
plt.title("Singular Values")
plt.show()
```

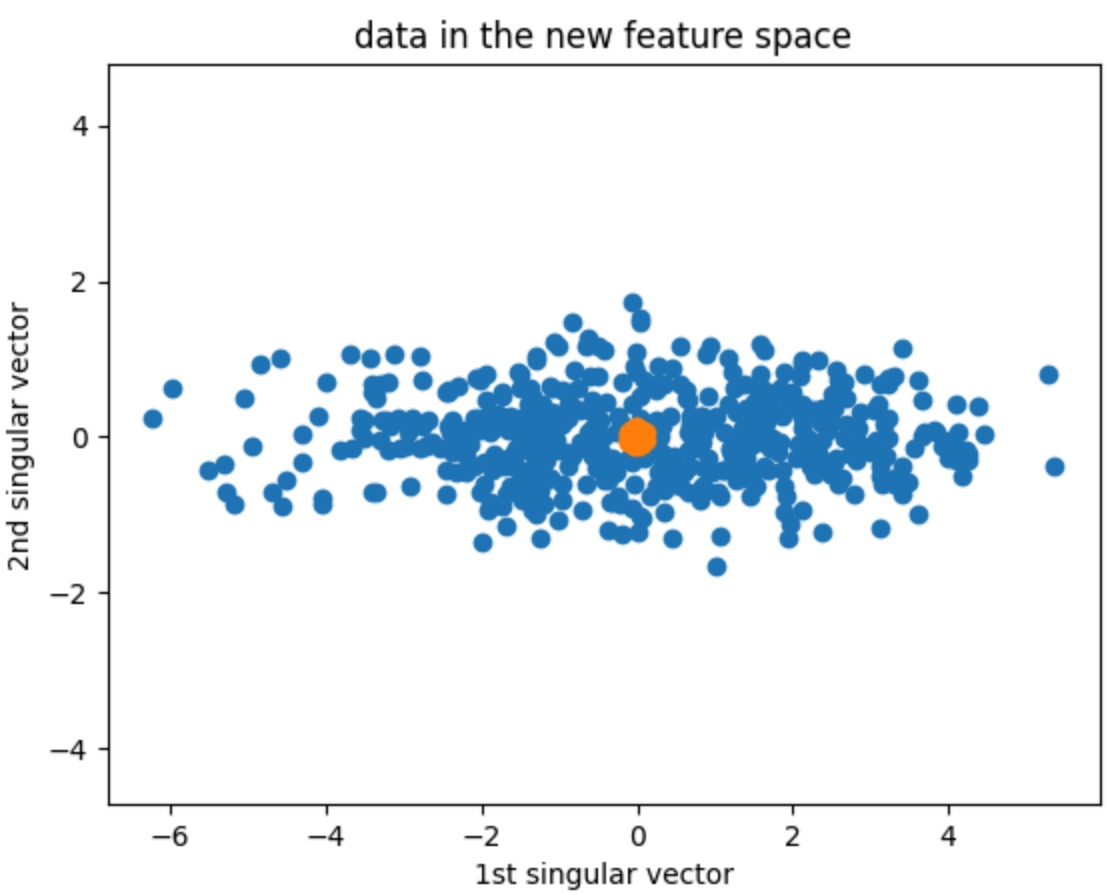


In [ ]:
```
scopy0 = s.copy()
scopy1 = s.copy()
scopy0[1:] = 0.0
```

```
scopy1[:1] = 0.0
approx0 = u.dot(np.diag(scopy0)).dot(vt)
approx1 = u.dot(np.diag(scopy1)).dot(vt)
plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.8)
sv1 = np.array([[-5],[5]]) @ vt[[0],:]
sv2 = np.array([[-2],[2]]) @ vt[[1],:]
plt.plot(sv1[:,0], sv1[:,1], 'g-', label="1st sing vector")
plt.plot(sv2[:,0], sv2[:,1], 'g--', label="2nd sing vector")
plt.scatter(approx0[:, 0] , approx0[:, 1], s=10, alpha=0.8, color="red", label="data pro
plt.scatter(approx1[:, 0] , approx1[:, 1], s=10, alpha=0.8, color="yellow", label="data
plt.axis('equal')
plt.legend()
plt.title("Mean-centered Data")
plt.show()
```

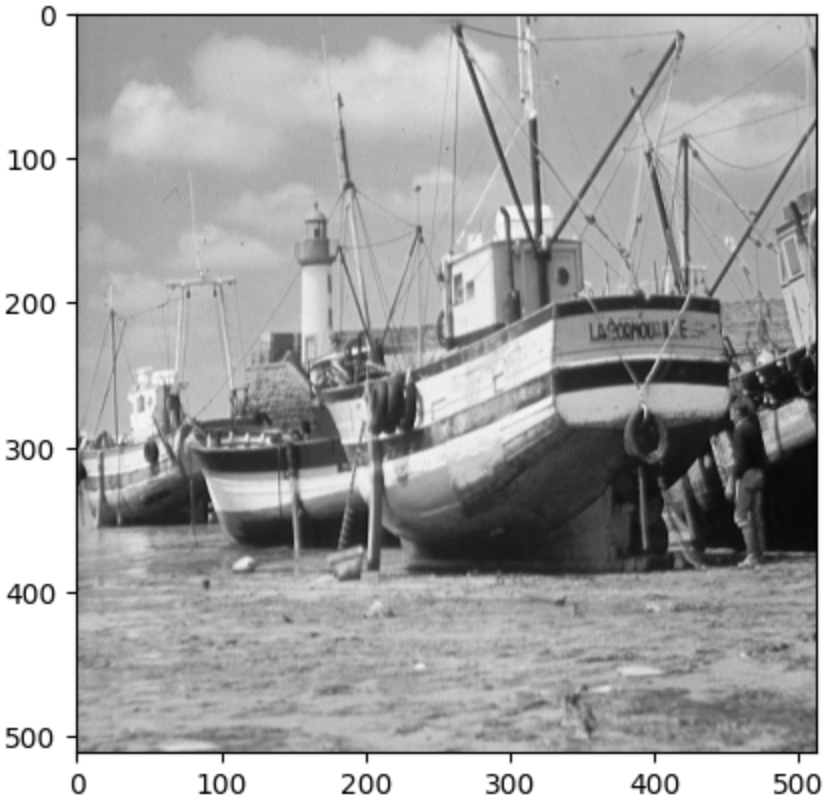


In [ ]:
```
# show ouput from svd is the same
orthonormal_X = u
shifted_X = u.dot(np.diag(s))
plt.axis('equal')
plt.scatter(shifted_X[:,0], shifted_X[:,1])
plt.scatter(orthonormal_X[:,0], orthonormal_X[:,1])
plt.xlabel("1st singular vector")
plt.ylabel("2nd singular vector")
plt.title("data in the new feature space")
plt.show()
```



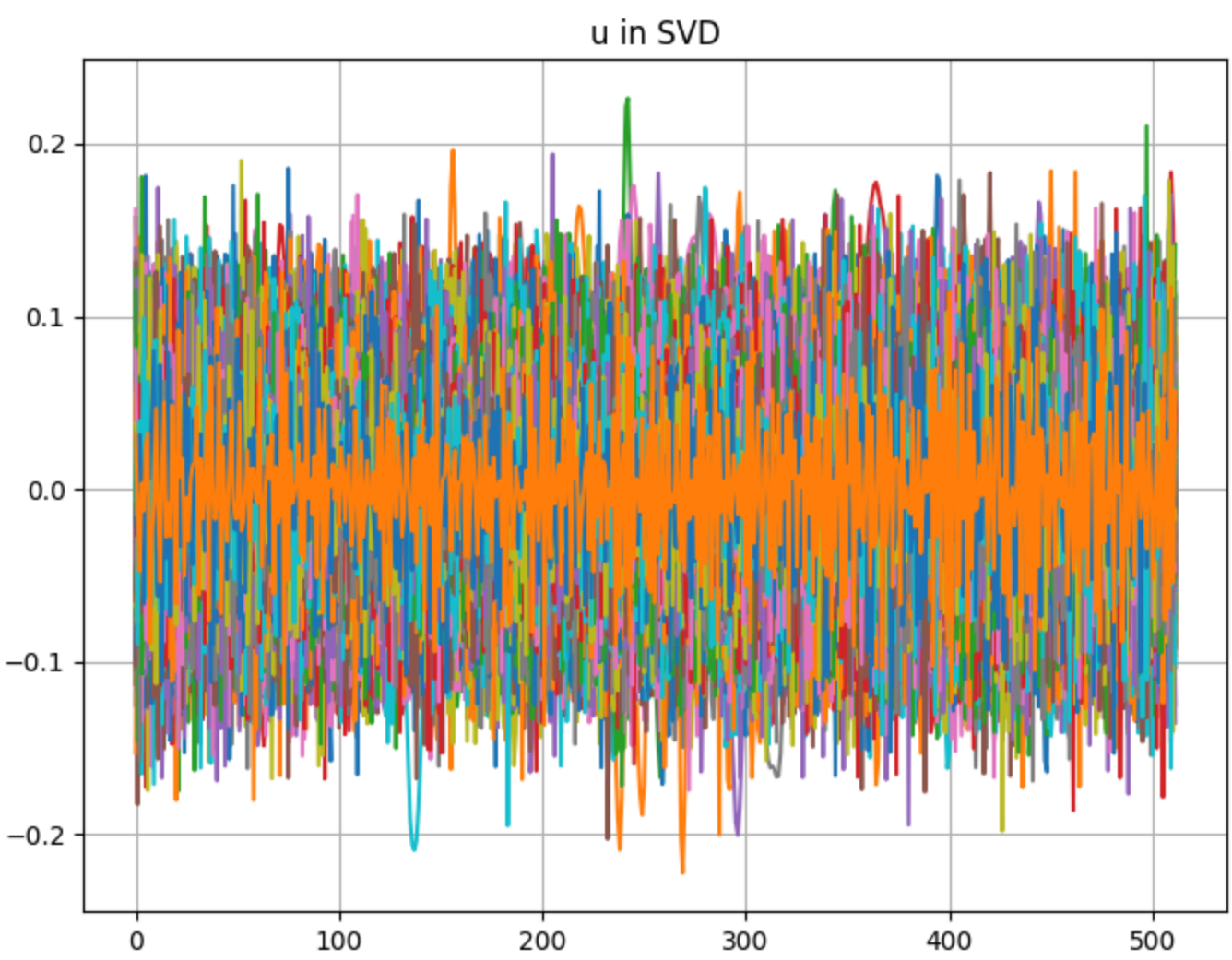
In [ ]:
```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

boat = np.loadtxt('./boat.dat')
plt.figure()
plt.imshow(boat, cmap = cm.Greys_r)
```

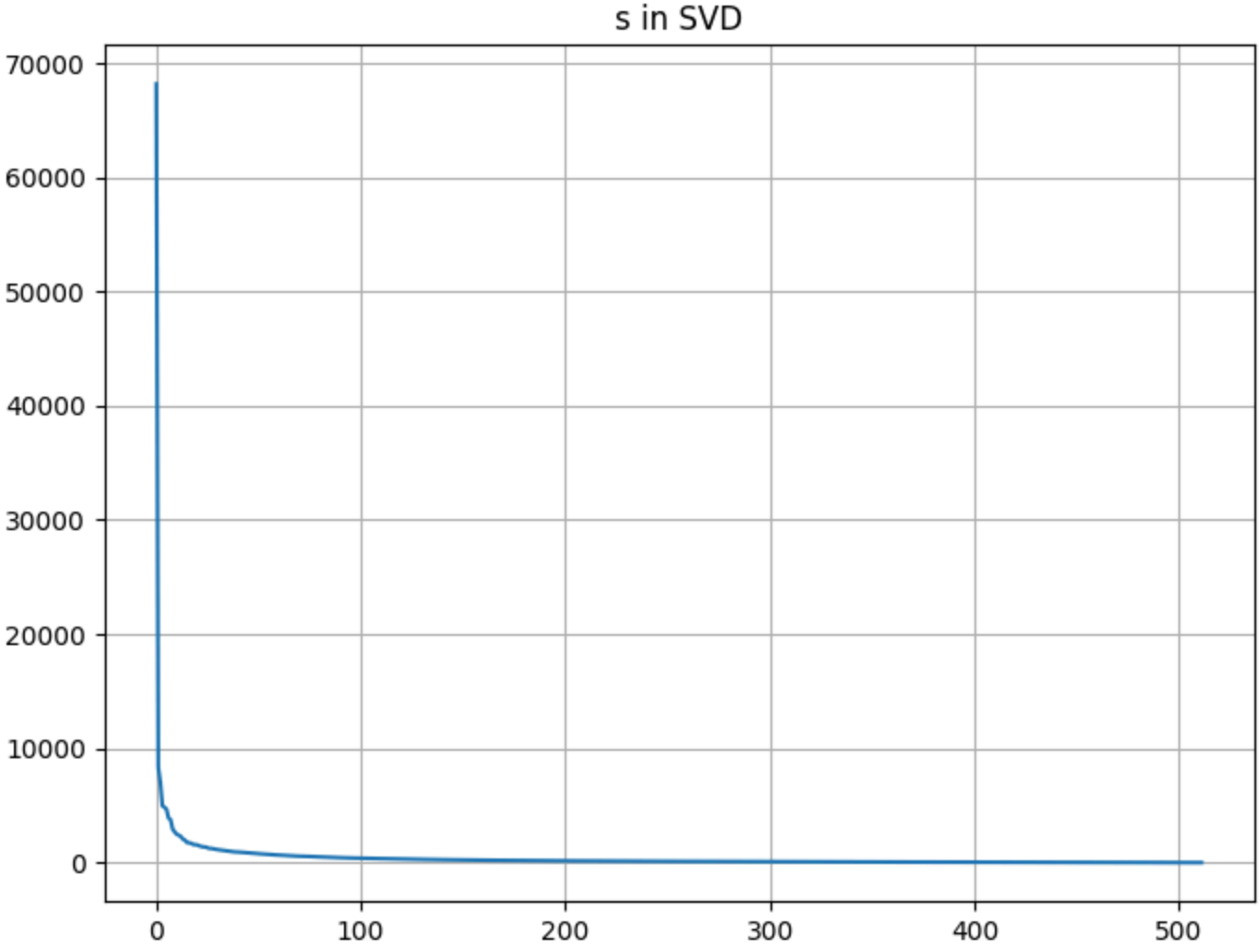
Out[ ]:
```
<matplotlib.image.AxesImage at 0x7b825efcb1f0>
```

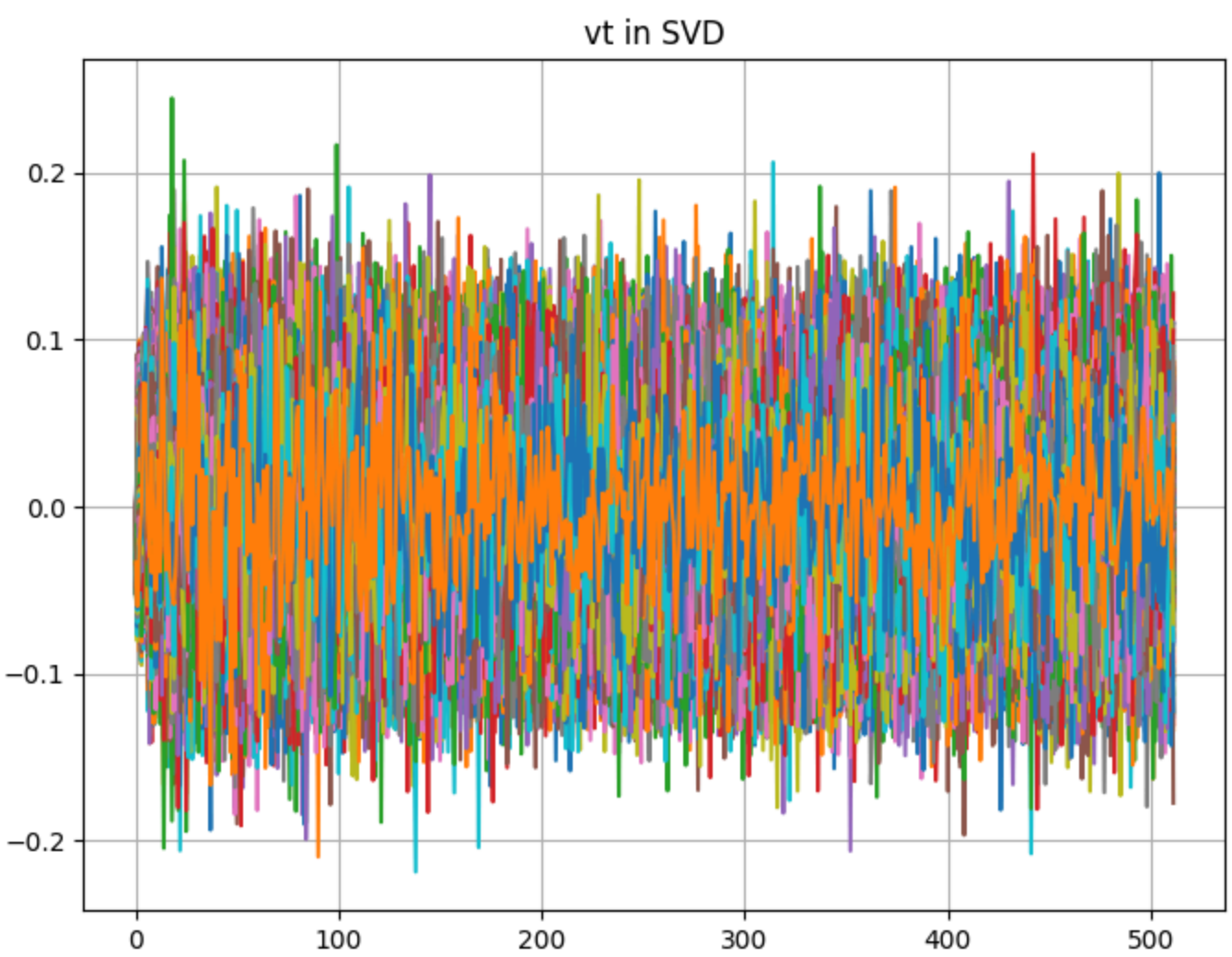


a) Plot the singular values of the image above (note: a gray scale image is just a matrix).

In [ ]:
```
u,s,vt=np.linalg.svd(boat,full_matrices=False)
plt.figure(figsize = (8, 6))
plt.plot(u)
plt.title('u in SVD')
plt.grid(True)
plt.show()

plt.figure(figsize = (8, 6))
plt.plot(s)
plt.title('s in SVD')
plt.grid(True)
plt.show()

plt.figure(figsize = (8, 6))
plt.plot(vt)
plt.title('vt in SVD')
plt.grid(True)
plt.show()
```


u in SVD
0.2
0.1
0.0
−0.1
−0.2
0
100
200
300
400
500

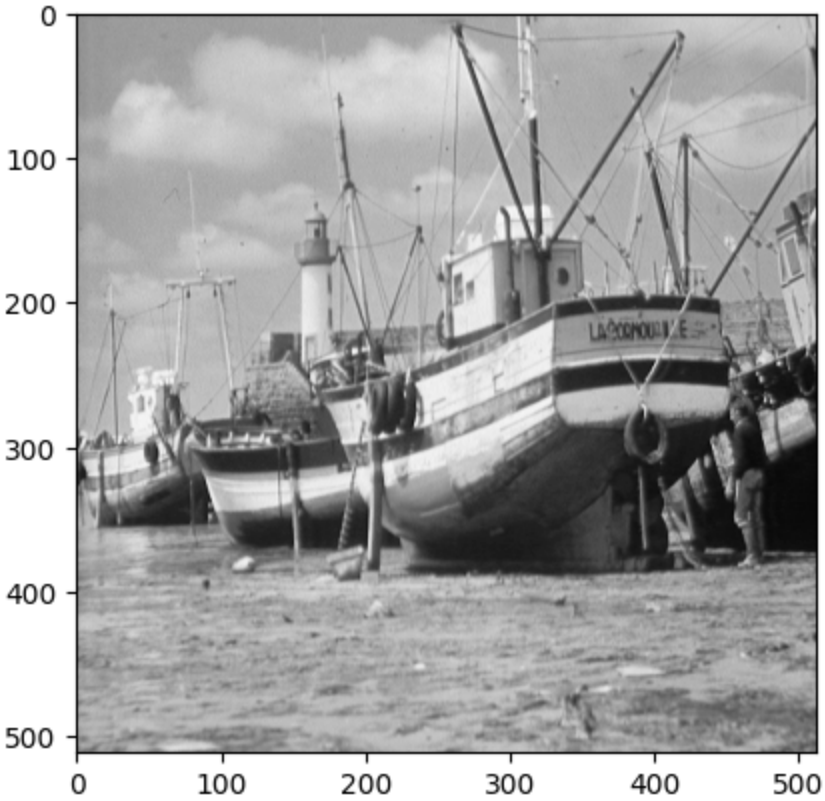

s in SVD
70000
60000
50000
40000
30000
20000
10000
0
0
100
200
300
400
500



Notice you can get the image back by multiplying the matrices back together:

In [ ]:
```
boat_copy = u.dot(np.diag(s)).dot(vt)
plt.figure()
plt.imshow(boat_copy, cmap = cm.Greys_r)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7b825dbc31c0>
```

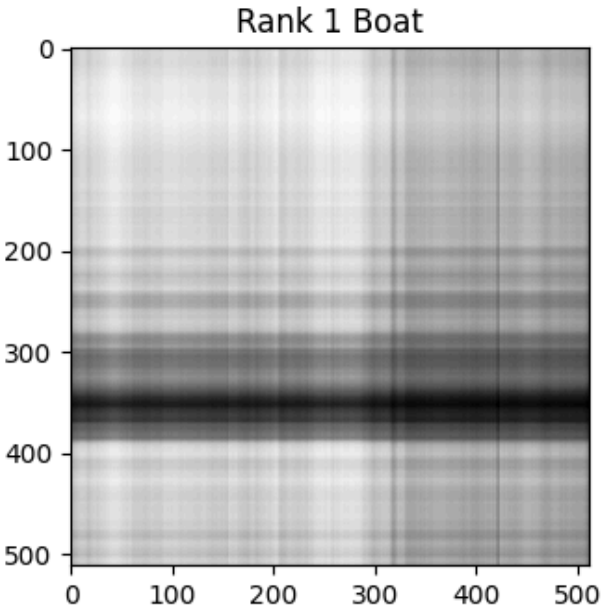


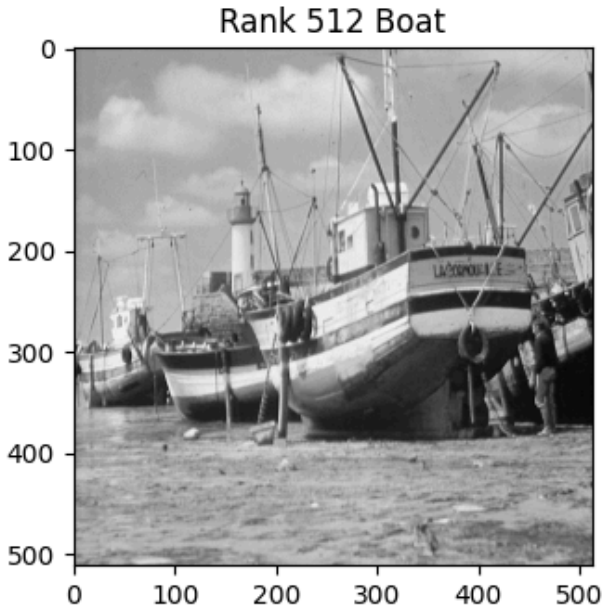
b) Create a new matrix `scopy` which is a copy of `s` with all but the first singular value set to 0.

In [ ]:
```
scopy = s.copy()
scopy[1:] = 0.0
```

c) Create an approximation of the boat image by multiplying `u`, `scopy`, and `v` transpose. Plot them side by side.

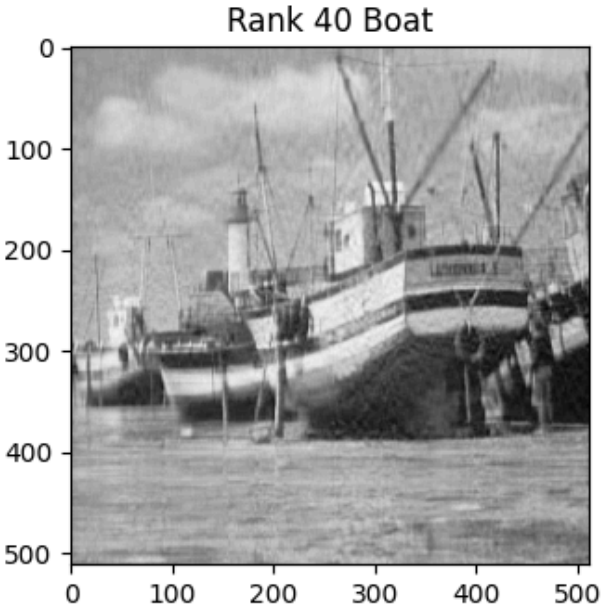
In [ ]:
```
boat_app = u.dot(np.diag(scopy)).dot(vt)

plt.figure(figsize=(9,6))
plt.subplot(1,2,1)
plt.imshow(boat_app, cmap = cm.Greys_r)
plt.title('Rank 1 Boat')
plt.subplot(1,2,2)
plt.imshow(boat, cmap = cm.Greys_r)
plt.title('Rank 512 Boat')
_ = plt.subplots_adjust(wspace=0.5)
plt.show()
```

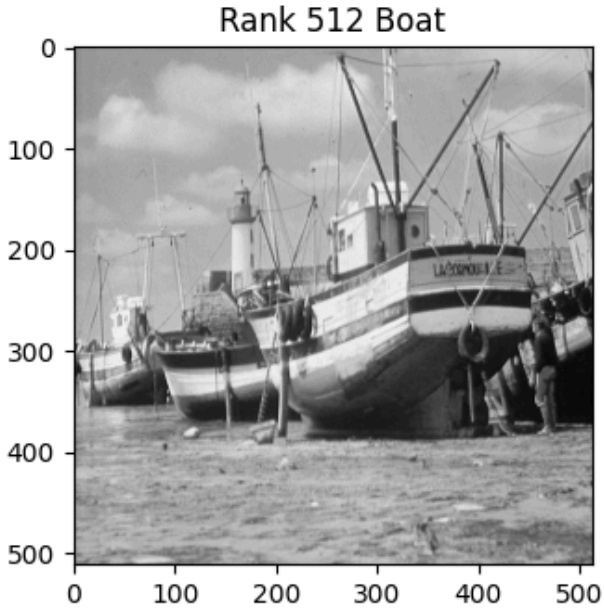




d) Repeat c) with 40 singular values instead of just 1.

In [ ]:
```
scopy = s.copy()
scopy[40:] = 0.0

boat_app = u.dot(np.diag(scopy)).dot(vt)

plt.figure(figsize=(9,6))
plt.subplot(1,2,1)
plt.imshow(boat_app, cmap = cm.Greys_r)
plt.title('Rank 40 Boat')
plt.subplot(1,2,2)
plt.imshow(boat, cmap = cm.Greys_r)
plt.title('Rank 512 Boat')
_ = plt.subplots_adjust(wspace=0.5)
plt.show()
```





Why you should care

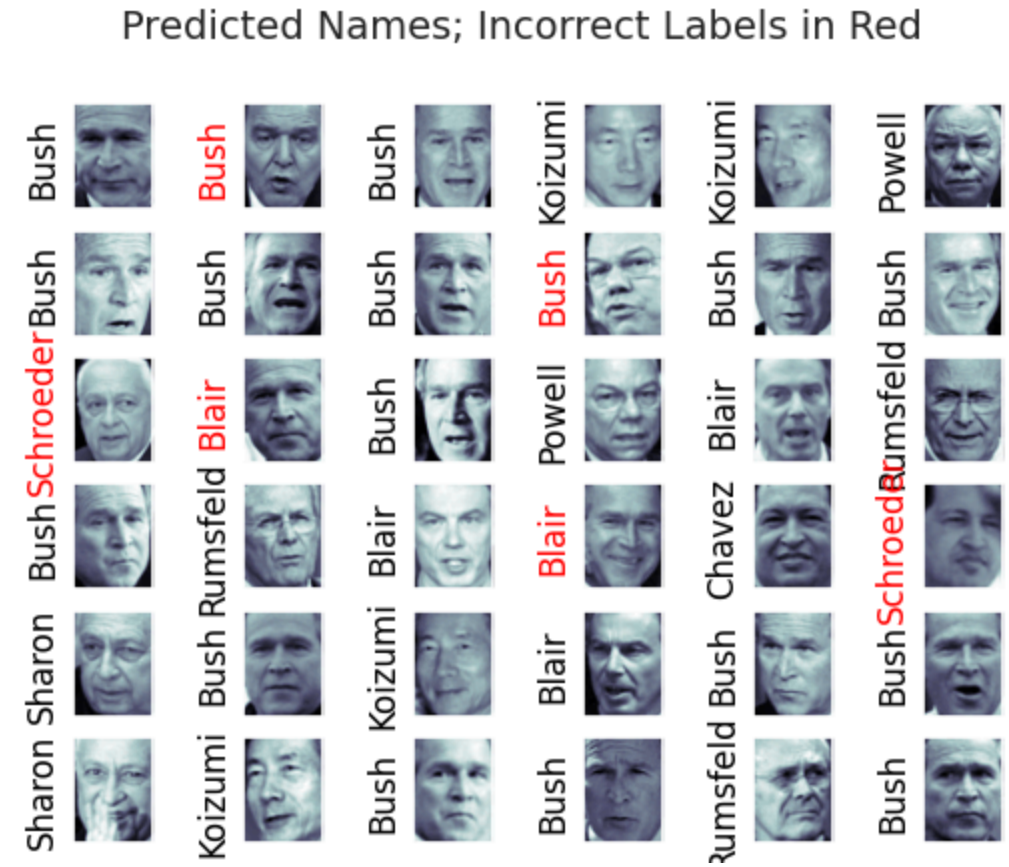
a) By using an approximation of the data, you can improve the performance of classification tasks since:

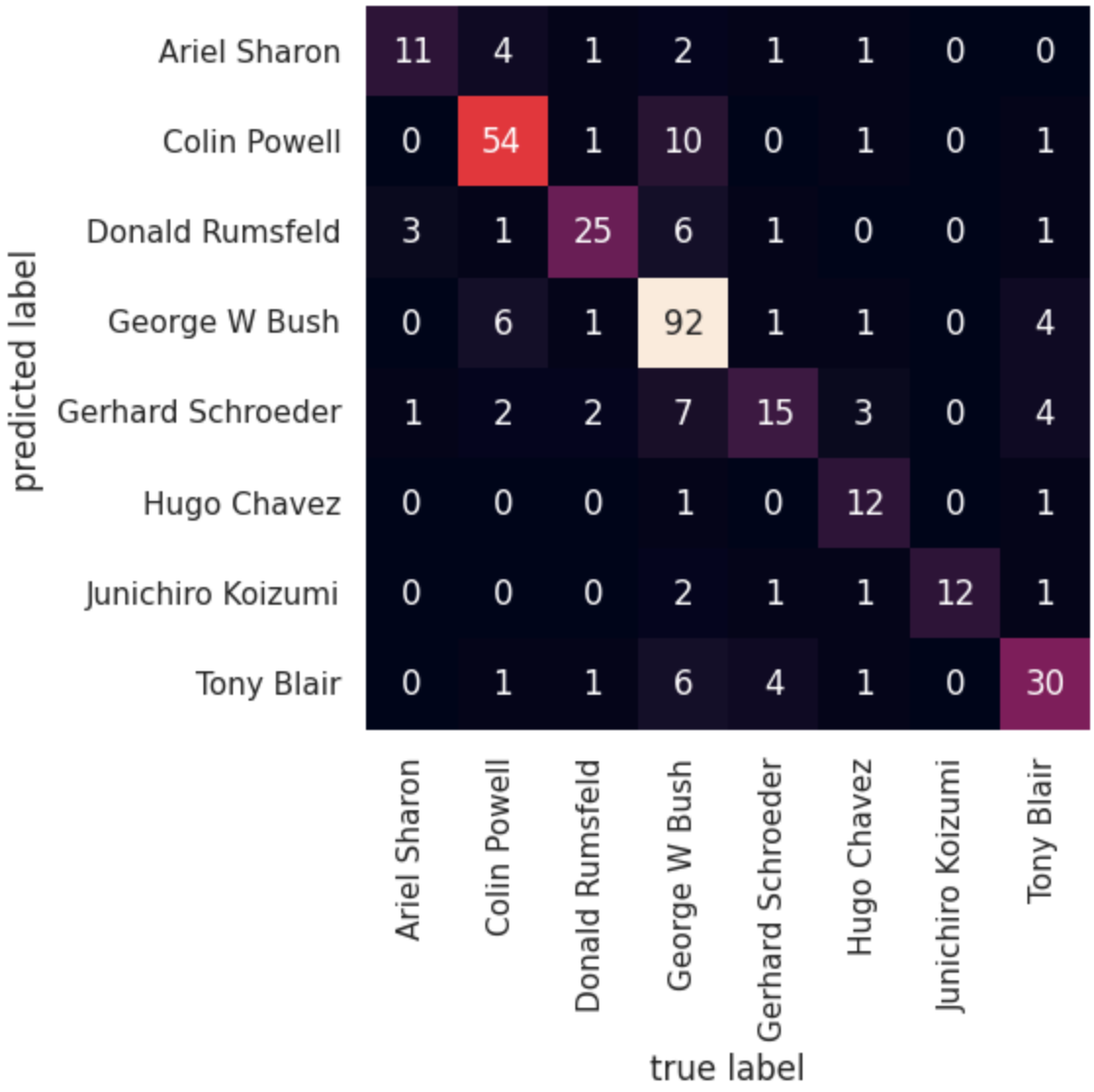
1. there is less noise interfering with classification
2. no relationship between features after SVD
3. the algorithm is sped up when reducing the dimension of the dataset

Below is some code to perform facial recognition on a dataset. Notice that, applied blindly, it does not perform well:

In [ ]:
```
import numpy as np
from PIL import Image
import seaborn as sns
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.datasets import fetch_lfw_people
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV, train_test_split

sns.set()

# Get face data
faces = fetch_lfw_people(min_faces_per_person=60)

# plot face data
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel=faces.target_names[faces.target[i]])
plt.show()

# split train test set
Xtrain, Xtest, ytrain, ytest = train_test_split(faces.data, faces.target, random_state=4

# blindly fit svm
svc = SVC(kernel='rbf', class_weight='balanced', C=5, gamma=0.001)

# fit model
model = svc.fit(Xtrain, ytrain)
yfit = model.predict(Xtest)

fig, ax = plt.subplots(6, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14)
plt.show()

mat = confusion_matrix(ytest, yfit)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('true label')
```

```
plt.ylabel('predicted label')
plt.show()

print("Accuracy = ", accuracy_score(ytest, yfit))
```



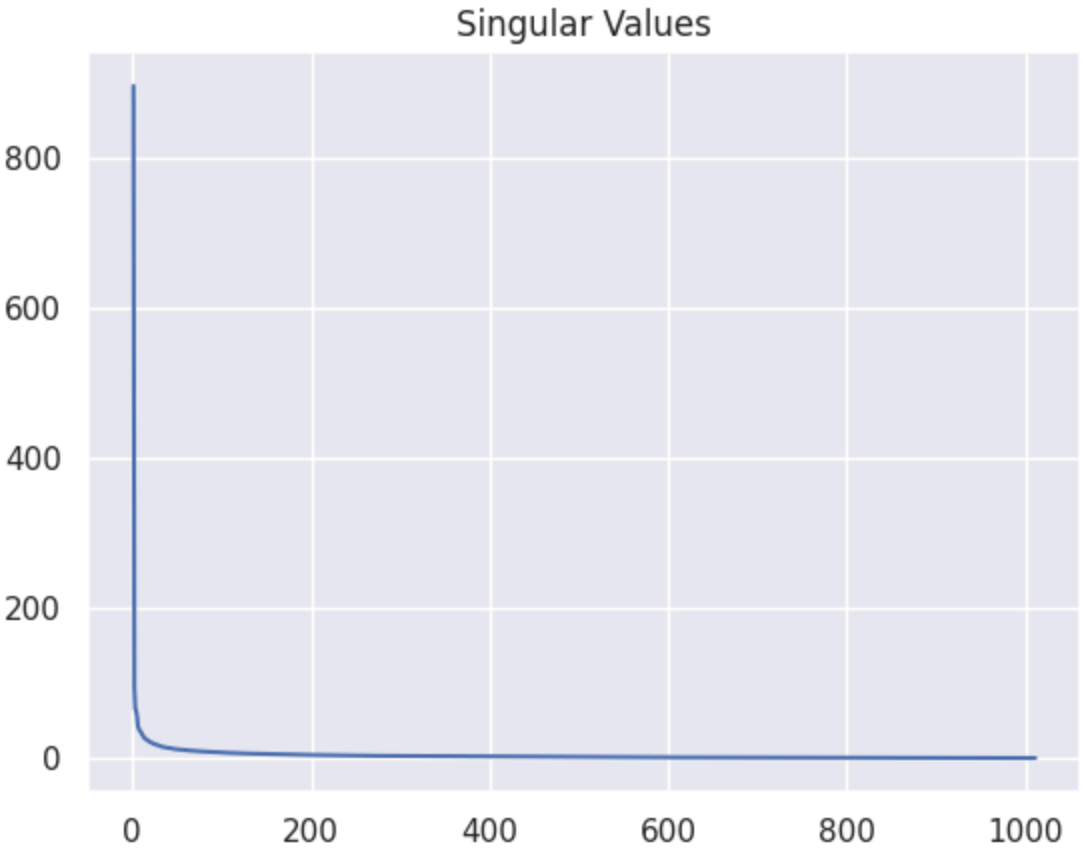


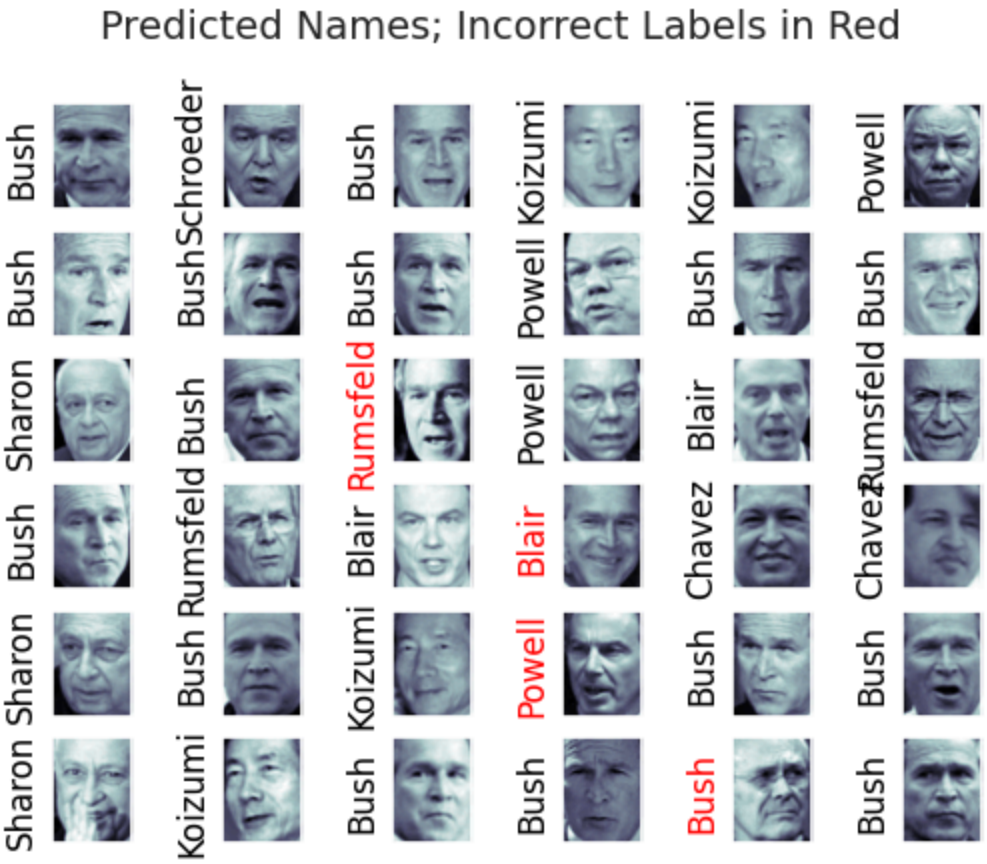

```
Accuracy =  0.744807121661721
```

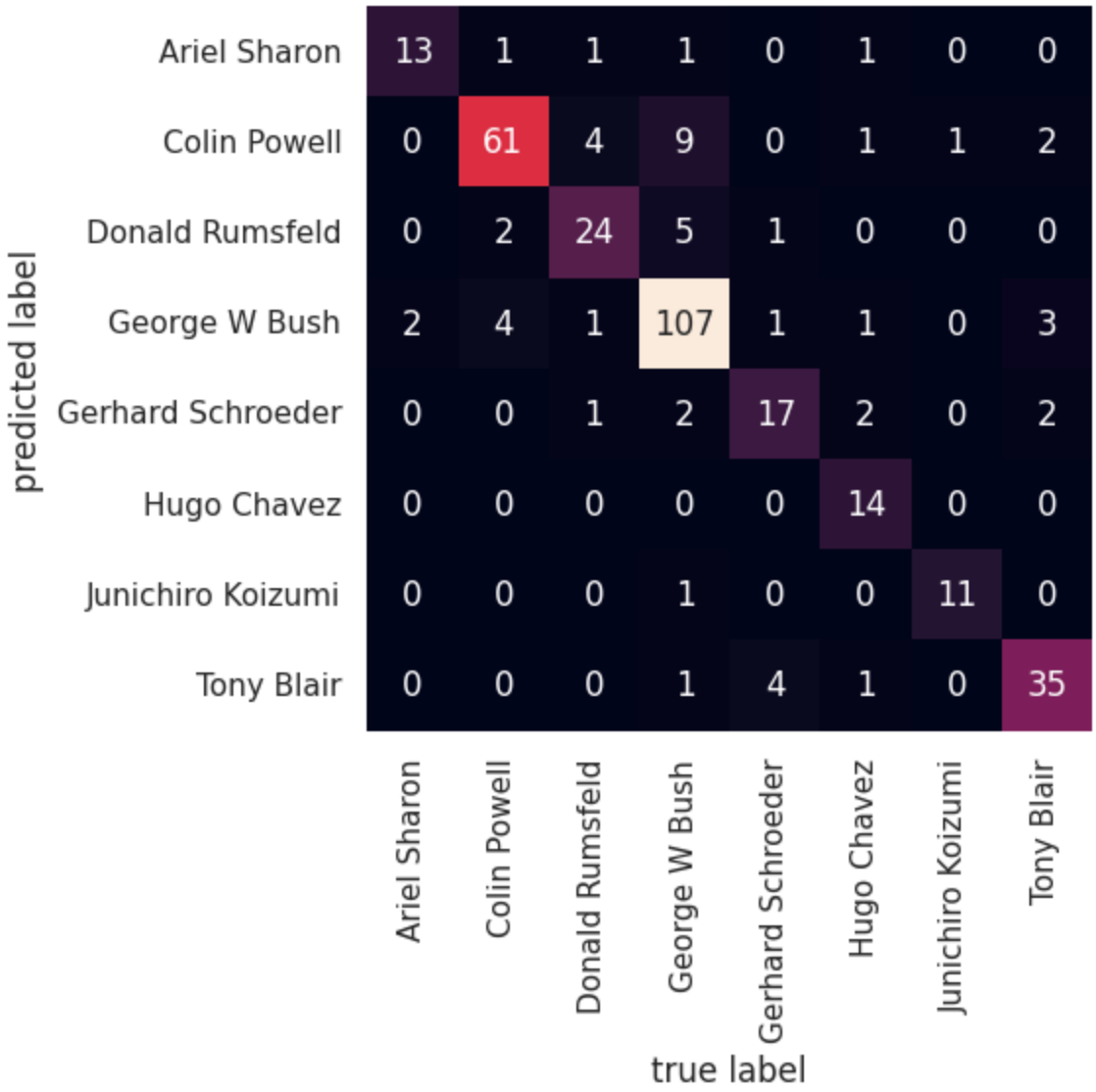
By performing SVD before applying the classification tool, we can reduce the dimension of the dataset.

In [ ]:
```
# look at singular values
_, s, _ = np.linalg.svd(Xtrain, full_matrices=False)
plt.plot(range(1,len(s)+1),s)
plt.title("Singular Values")
plt.show()

# extract principal components
pca = PCA(n_components= 250, whiten=True)
svc = SVC(kernel='rbf', class_weight='balanced', C=5, gamma=0.001)
svcpca = make_pipeline(pca, svc)
model = svcpca.fit(Xtrain, ytrain)
yfit = model.predict(Xtest)

fig, ax = plt.subplots(6, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14)
plt.show()

mat = confusion_matrix(ytest, yfit)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
```

```
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()

print("Accuracy = ", accuracy_score(ytest, yfit))
```







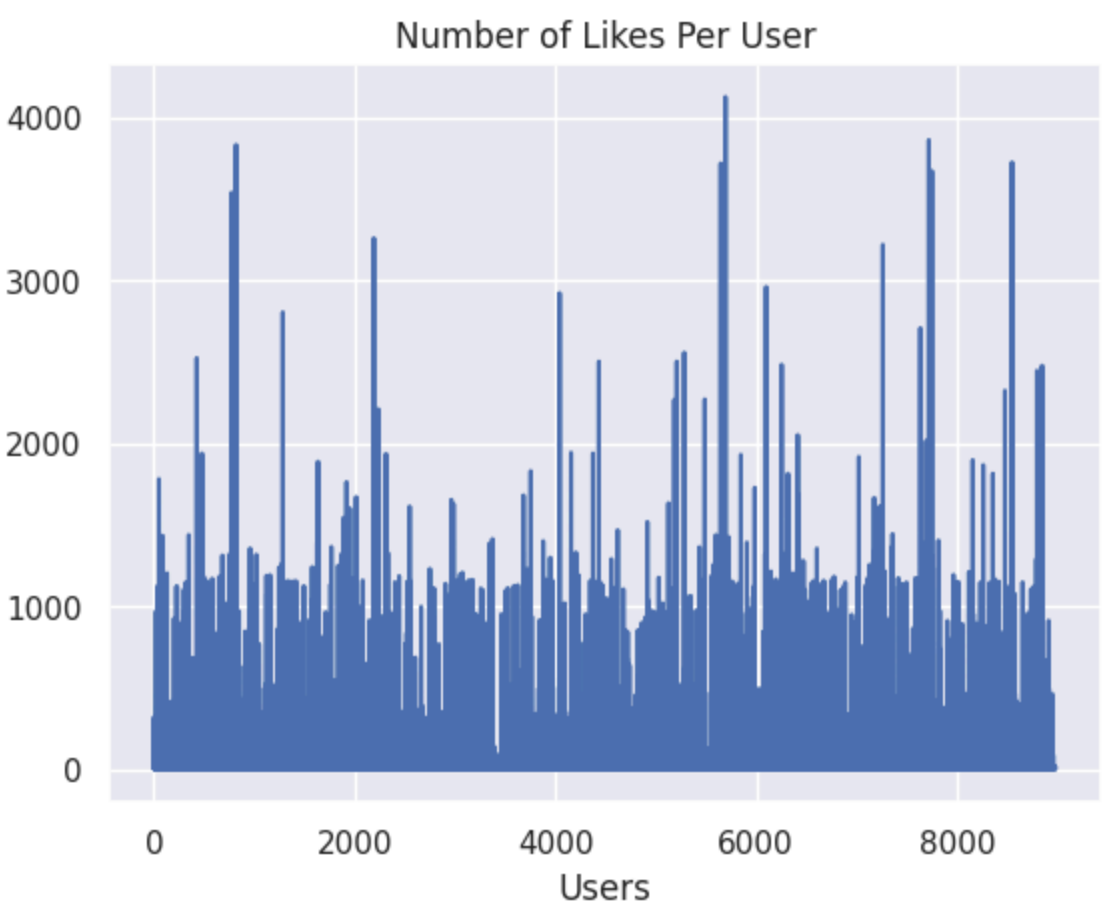

```
Accuracy =  0.8367952522255193
```

Similar to finding k in K-means, we're trying to find the point of diminishing returns when picking the number of singular vectors (also called principal components).

b) SVD can be used for anomaly detection.

The data below consists of the number of 'Likes' during a six month period, for each of 9000 users across the 210 content categories that Facebook assigns to pages.

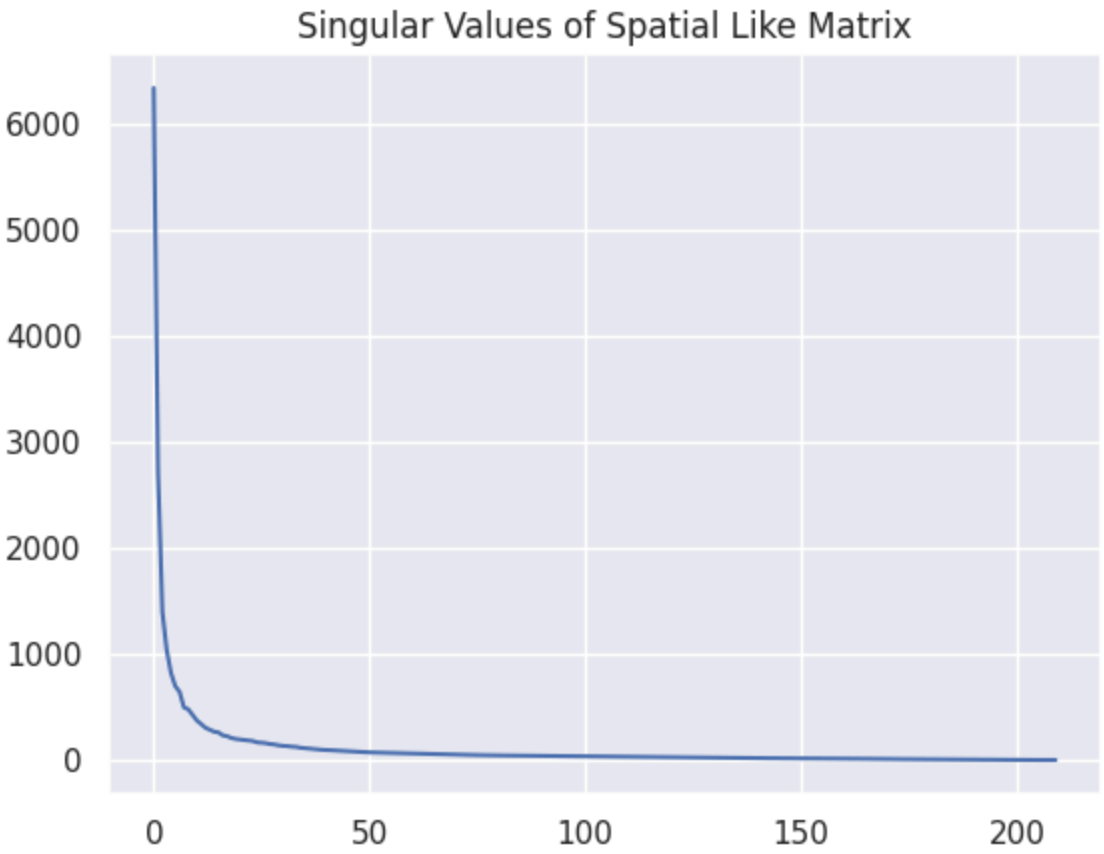
In [ ]:
```
data = np.loadtxt('spatial_data.txt')

FBSpatial = data[:,1:]
FBSnorm = np.linalg.norm(FBSpatial,axis=1,ord=1)
plt.plot(FBSnorm)
plt.title('Number of Likes Per User')
_ = plt.xlabel('Users')
plt.show()
```

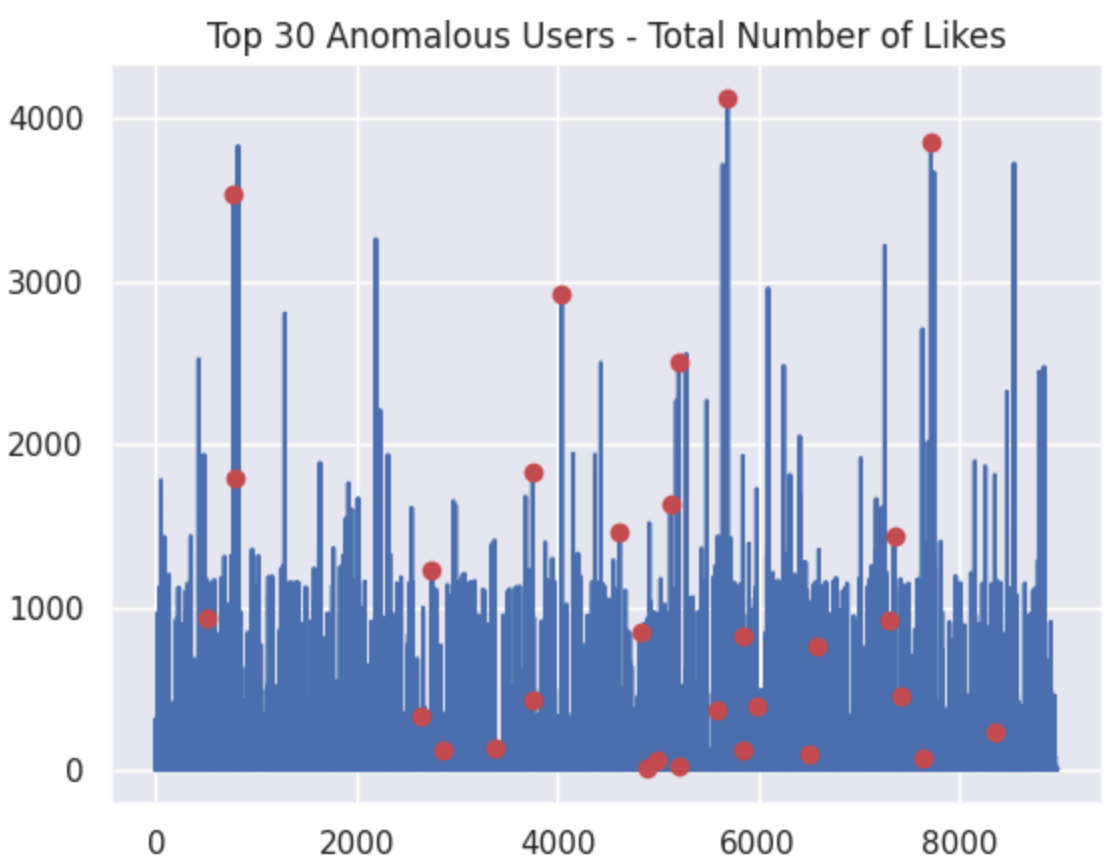


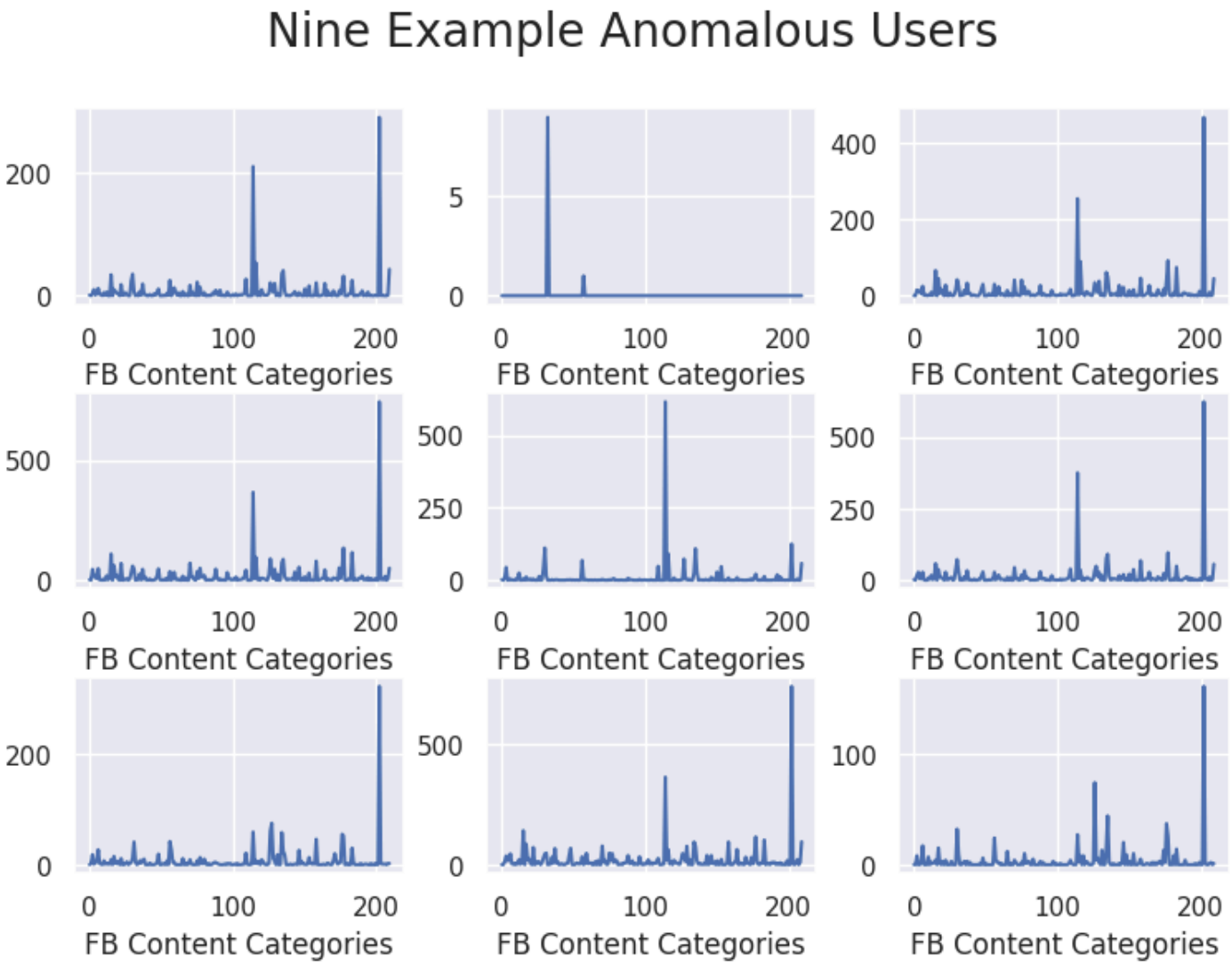
How users distribute likes across categories follows a general pattern that most users follow. This behavior can be captured using few singular vectors. And anomalous users can be easily identified.

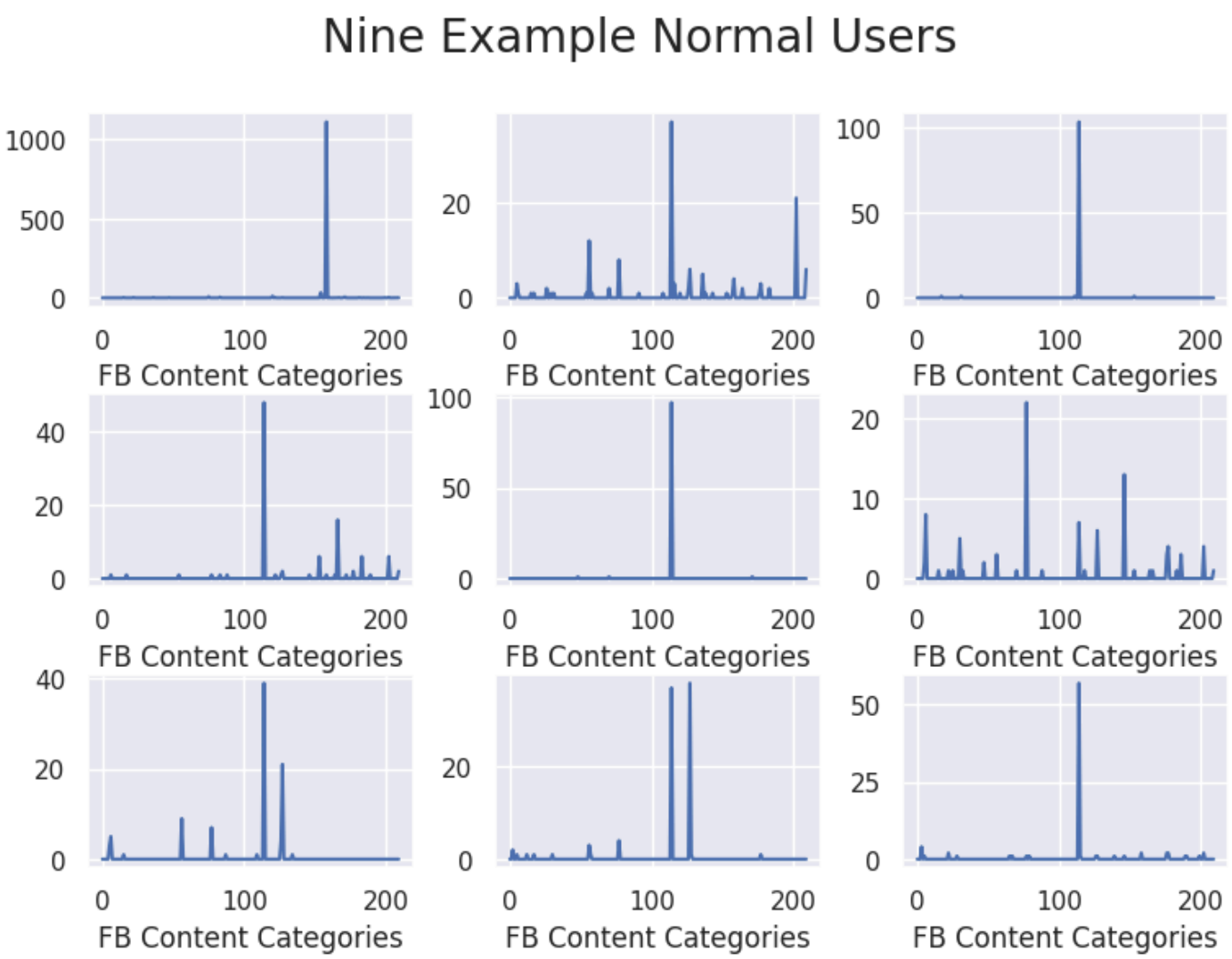
In [ ]:
```
u,s,vt = np.linalg.svd(FBSpatial,full_matrices=False)
plt.plot(s)
_ = plt.title('Singular Values of Spatial Like Matrix')
plt.show()

RANK = 100
scopy = s.copy()
scopy[RANK:] = 0.
N = u @ np.diag(scopy) @ vt
O = FBSpatial - N
Onorm = np.linalg.norm(O, axis=1)
anomSet = np.argsort(Onorm)[-30:]
# plt.plot(Onorm)
# plt.plot(anomSet, Onorm[anomSet],'ro')
# _ = plt.title('Norm of Residual (rows of O)')
# plt.show()

plt.plot(FBSnorm)
plt.plot(anomSet, FBSnorm[anomSet],'ro')
_ = plt.title('Top 30 Anomalous Users - Total Number of Likes')
plt.show()

# anomalous users
plt.figure(figsize=(9,6))
for i in range(1,10):
    ax = plt.subplot(3,3,i)
    plt.plot(FBSpatial[anomSet[i-1],:])
    plt.xlabel('FB Content Categories')
plt.subplots_adjust(wspace=0.25,hspace=0.45)
_ = plt.suptitle('Nine Example Anomalous Users',size=20)
```

```
plt.show()

# normal users
set = np.argsort(Onorm)[0:7000]
# that have high overall volume
max = np.argsort(FBSnorm[set])[::-1]
plt.figure(figsize=(9,6))
for i in range(1,10):
    ax = plt.subplot(3,3,i)
    plt.plot(FBSpatial[set[max[i-1]],:])
    plt.xlabel('FB Content Categories')
plt.subplots_adjust(wspace=0.25,hspace=0.45)
_ = plt.suptitle('Nine Example Normal Users',size=20)
plt.show()
```




Top 30 Anomalous Users - Total Number of Likes
4000
3000
2000
1000
0
0
2000
4000
6000
8000


Nine Example Anomalous Users
FB Content Categories
FB Content Categories
FB Content Categories
FB Content Categories
FB Content Categories
FB Content Categories
FB Content Categories
FB Content Categories
FB Content Categories

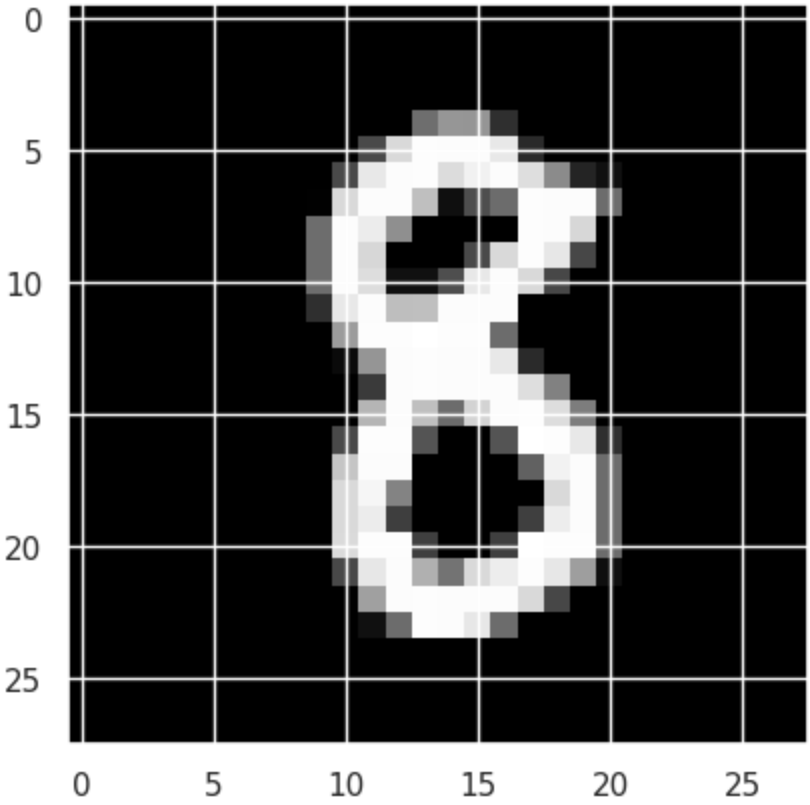


Challenge Problem

a) Fetch the "mnist_784" data. Pick an image of a digit at random and plot it.

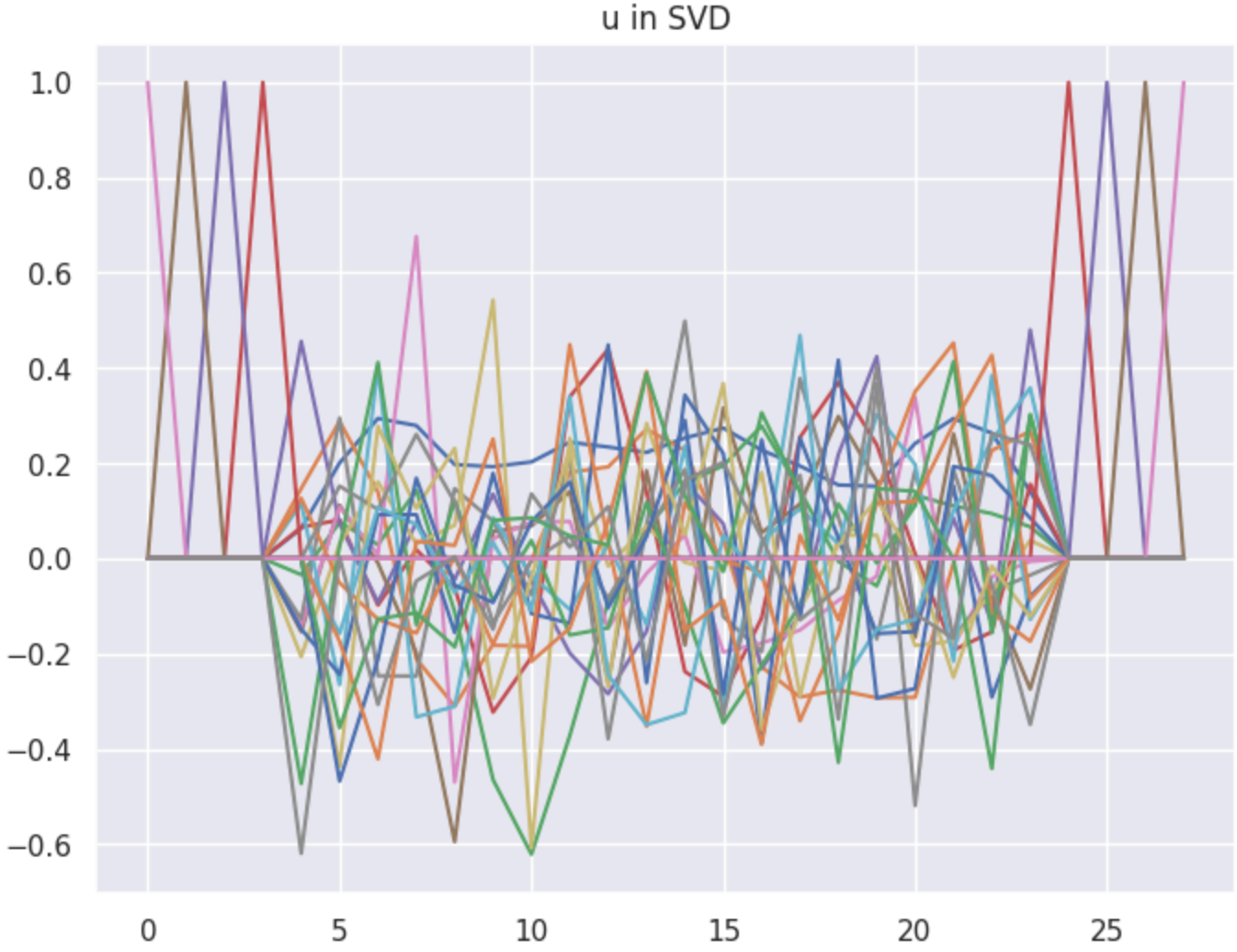
In [20]:
```
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml

X, y = fetch_openml(name="mnist_784", version=1, return_X_y=True, as_frame=False)
```

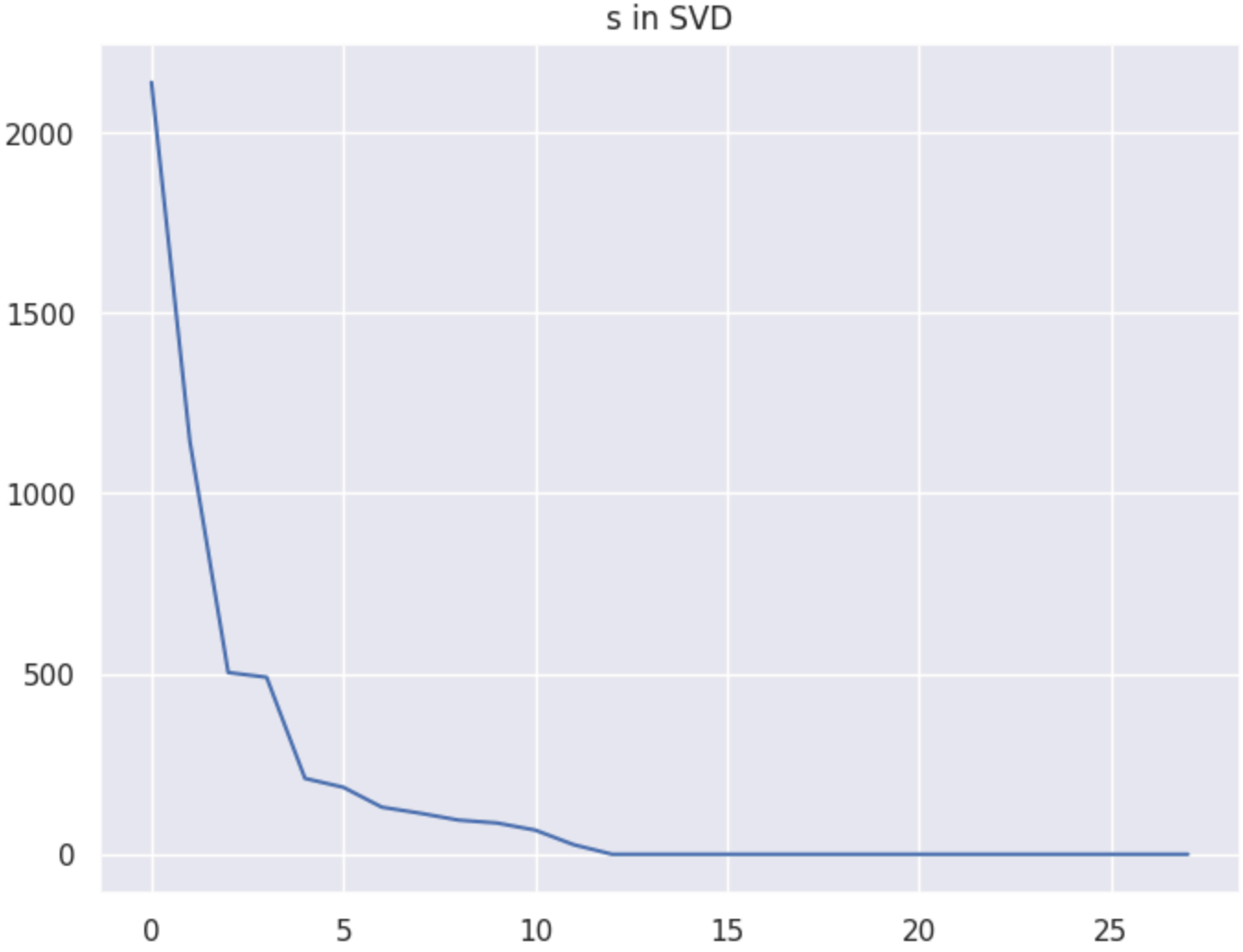
In [21]:
```
randomNumber = np.random.randint(0, X.shape[0])
randomDigit = X[randomNumber].reshape(28, 28)

plt.imshow(randomDigit, cmap='gray')
plt.show()
```



b) Plot its singular value plot.

In [22]:
```
u,s,vt=np.linalg.svd(randomDigit,full_matrices=False)
plt.figure(figsize = (8, 6))
plt.plot(u)
plt.title('u in SVD')
plt.grid(True)
plt.show()

plt.figure(figsize = (8, 6))
plt.plot(s)
plt.title('s in SVD')
plt.grid(True)
plt.show()

plt.figure(figsize = (8, 6))
plt.plot(vt)
plt.title('vt in SVD')
plt.grid(True)
plt.show()
```

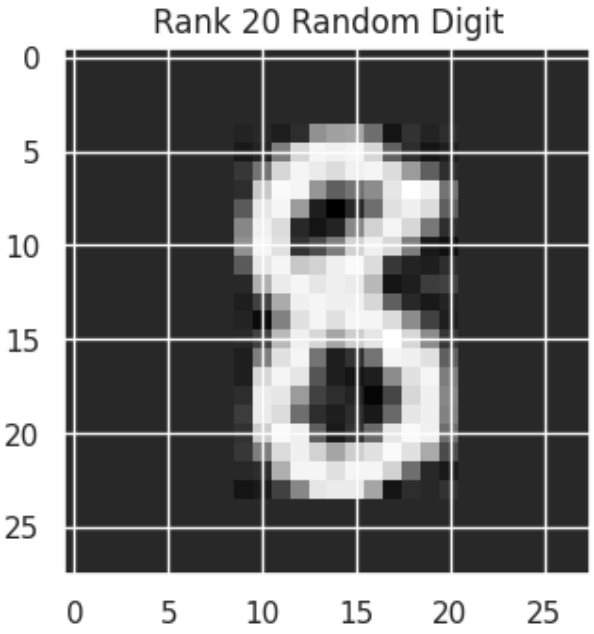

u in SVD
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
5
10
15
20
25

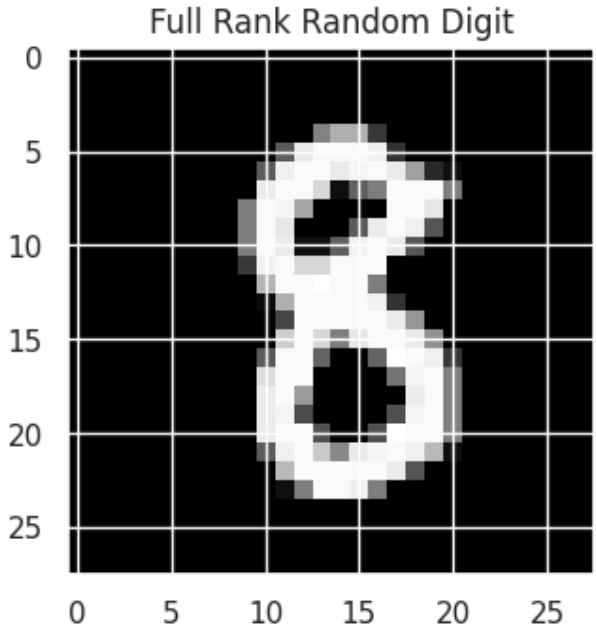

s in SVD
2000
1500
1000
500
0
0
5
10
15
20
25

c) By setting some singular values to 0, plot the approximation of the image next to the original image

In [26]:

```
scopy = s.copy()
scopy[5:] = 0.0

digit_app = u.dot(np.diag(scopy)).dot(vt)

plt.figure(figsize=(9,6))
plt.subplot(1,2,1)
plt.imshow(digit_app, cmap = cm.Greys_r)
plt.title('Rank 20 Random Digit')
plt.subplot(1,2,2)
plt.imshow(randomDigit, cmap = cm.Greys_r)
plt.title('Full Rank Random Digit')
_ = plt.subplots_adjust(wspace=0.5)
plt.show()
```



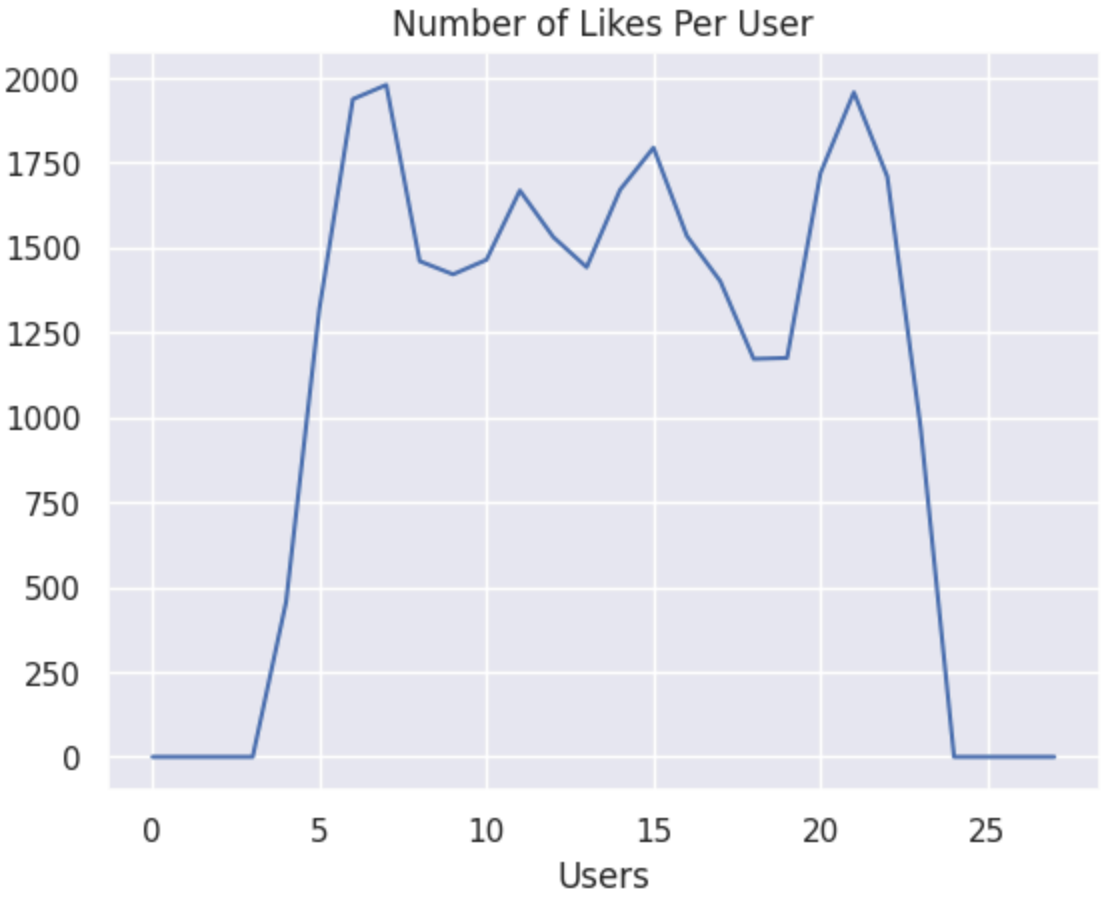


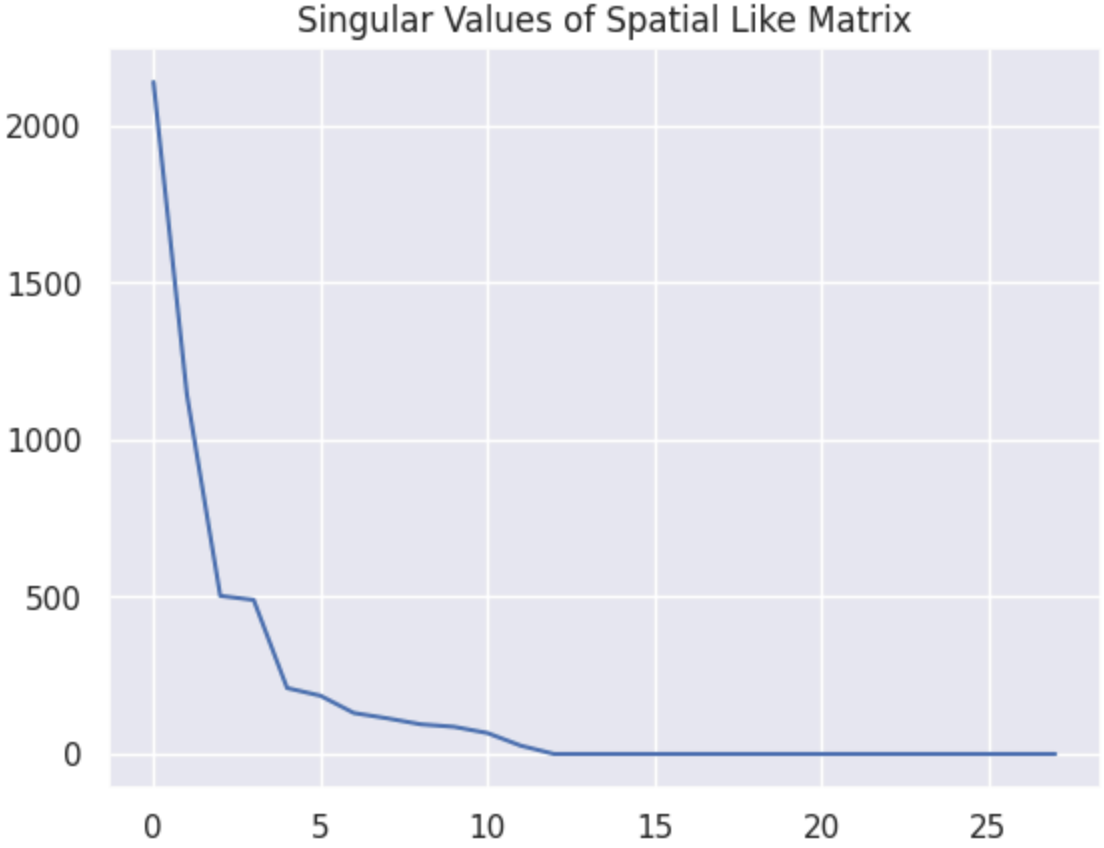
d) Consider the entire dataset as a matrix. Perform SVD and explain why / how you chose a particular rank. Note: you may not be able to run this on the entire dataset in a reasonable amount of time so you may take a small random sample for this and the following questions.

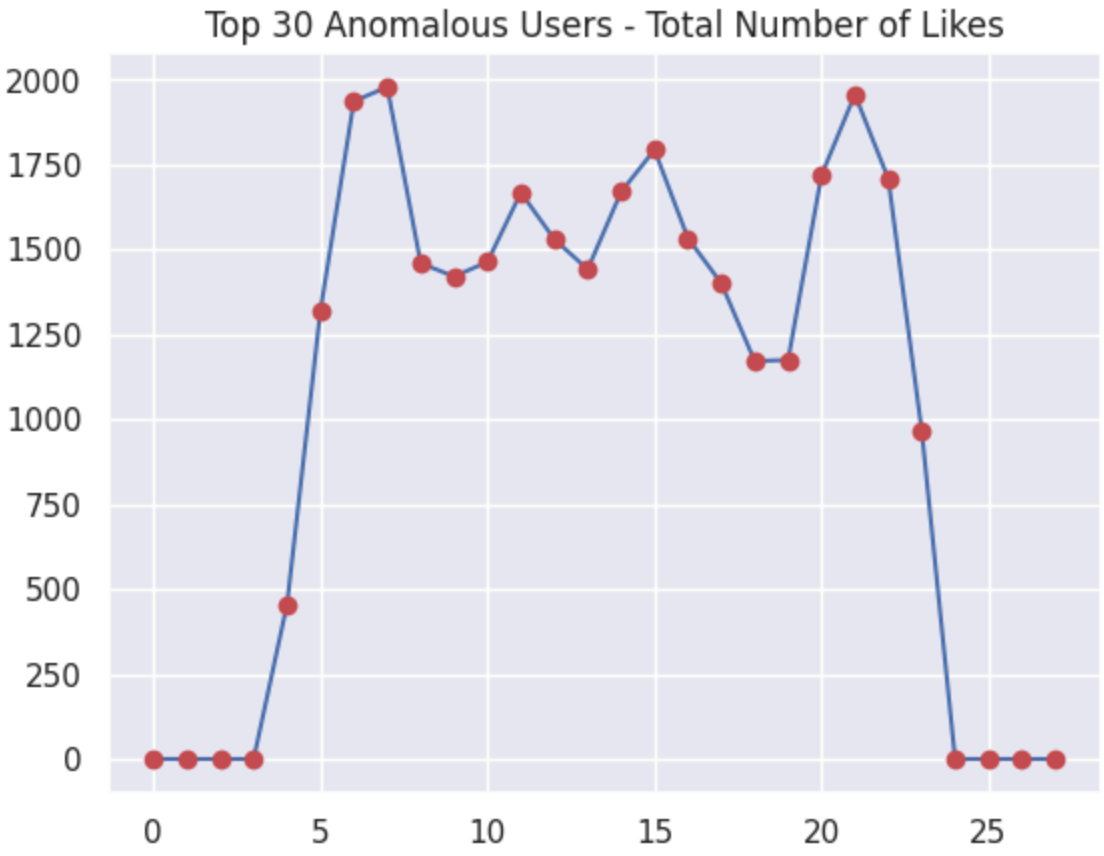
In [33]:

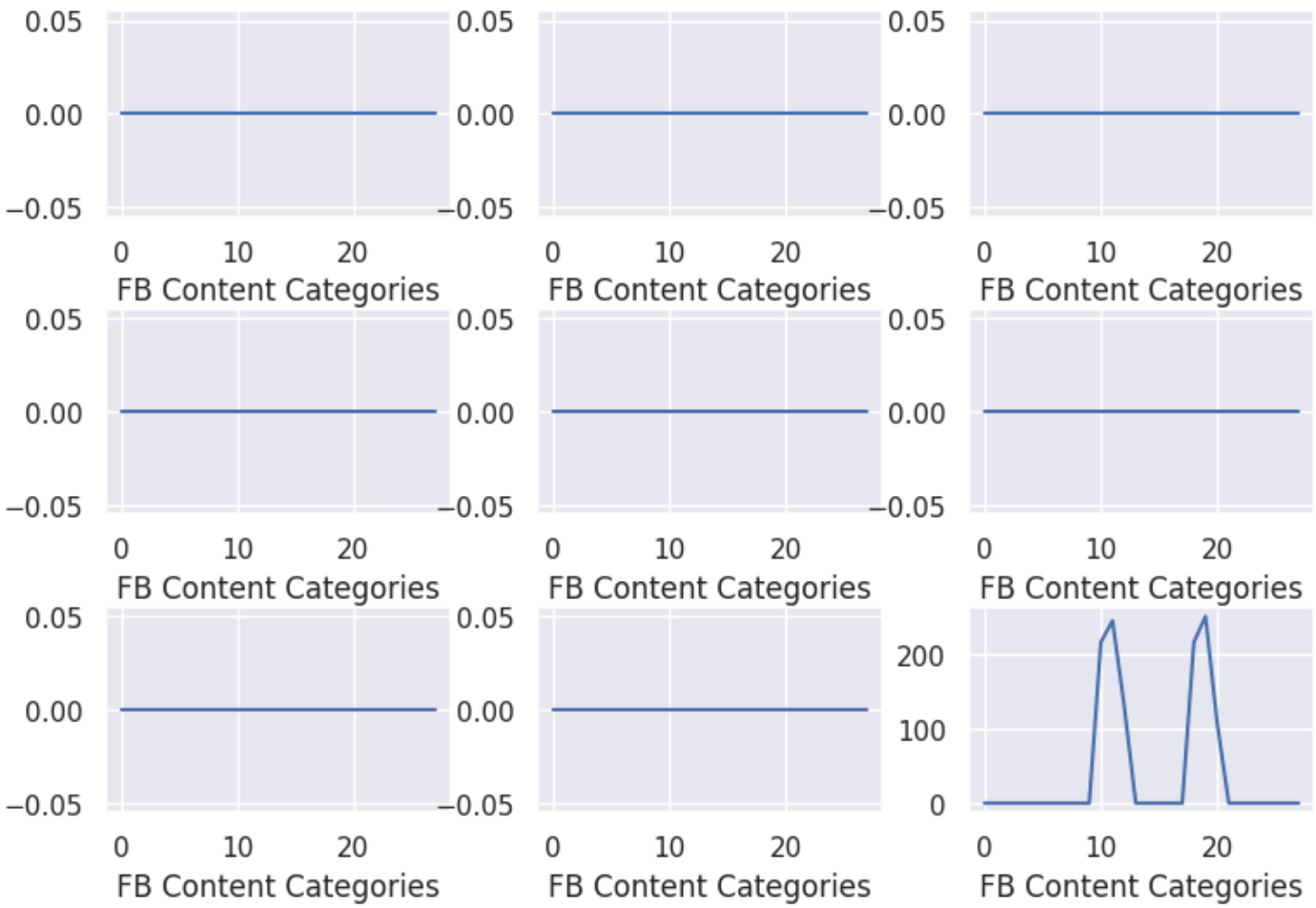
```
FBSpatial = randomDigit[:,:]
FBSnorm = np.linalg.norm(FBSpatial,axis=1,ord=1)
plt.plot(FBSnorm)
plt.title('Number of Likes Per User')
_ = plt.xlabel('Users')
plt.show()

u,s,vt = np.linalg.svd(randomDigit,full_matrices=False)
plt.plot(s)
_ = plt.title('Singular Values of Spatial Like Matrix')
plt.show()

RANK = 10
scopy = s.copy()
scopy[RANK:] = 0.
N = u @ np.diag(scopy) @ vt
O = FBSpatial - N
Onorm = np.linalg.norm(O, axis=1)
anomSet = np.argsort(Onorm)[-30:]
# plt.plot(Onorm)
# plt.plot(anomSet, Onorm[anomSet],'ro')
# _ = plt.title('Norm of Residual (rows of O)')
# plt.show()

plt.plot(FBSnorm)
plt.plot(anomSet, FBSnorm[anomSet],'ro')
_ = plt.title('Top 30 Anomalous Users - Total Number of Likes')
plt.show()

# anomalous users
plt.figure(figsize=(9,6))
for i in range(1,10):
    ax = plt.subplot(3,3,i)
    plt.plot(FBSpatial[anomSet[i-1],:])
    plt.xlabel('FB Content Categories')
plt.subplots_adjust(wspace=0.25,hspace=0.45)
```

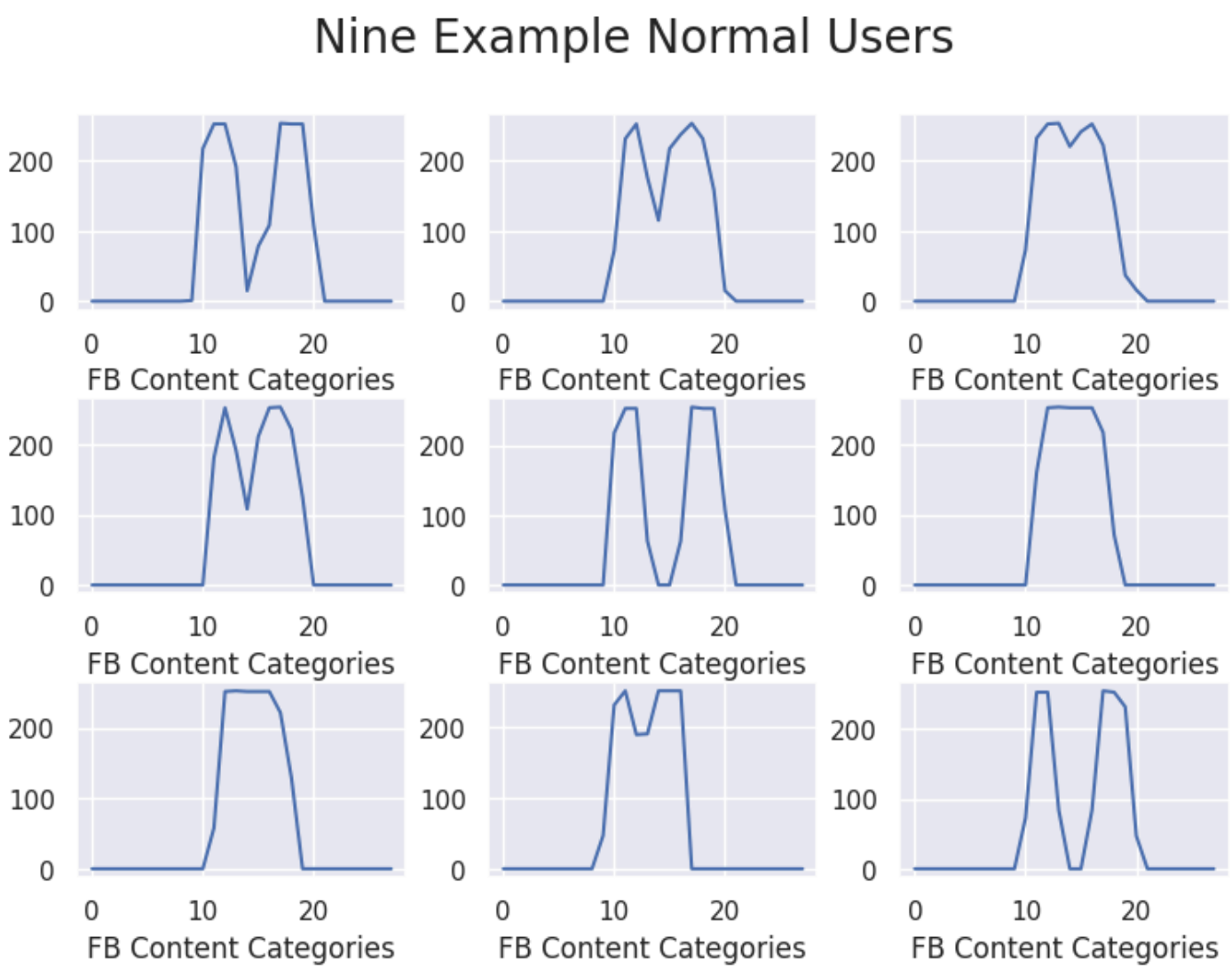
```
_ = plt.suptitle('Nine Example Anomalous Users',size=20)
plt.show()

# normal users
set = np.argsort(Onorm)[0:7000]
# that have high overall volume
max = np.argsort(FBSnorm[set])[::-1]
plt.figure(figsize=(9,6))
for i in range(1,10):
    ax = plt.subplot(3,3,i)
    plt.plot(FBSpatial[set[max[i-1]],:])
    plt.xlabel('FB Content Categories')
plt.subplots_adjust(wspace=0.25,hspace=0.45)
_ = plt.suptitle('Nine Example Normal Users',size=20)
plt.show()
```




Singular Values of Spatial Like Matrix
2000
1500
1000
500
0
0
5
10
15
20
25


Top 30 Anomalous Users - Total Number of Likes
2000
1750
1500
1250
1000
750
500
250
0
0
5
10
15
20
25


Nine Example Anomalous Users
0.05
0.00
−0.05
0
10
20
FB Content Categories
200
100
0

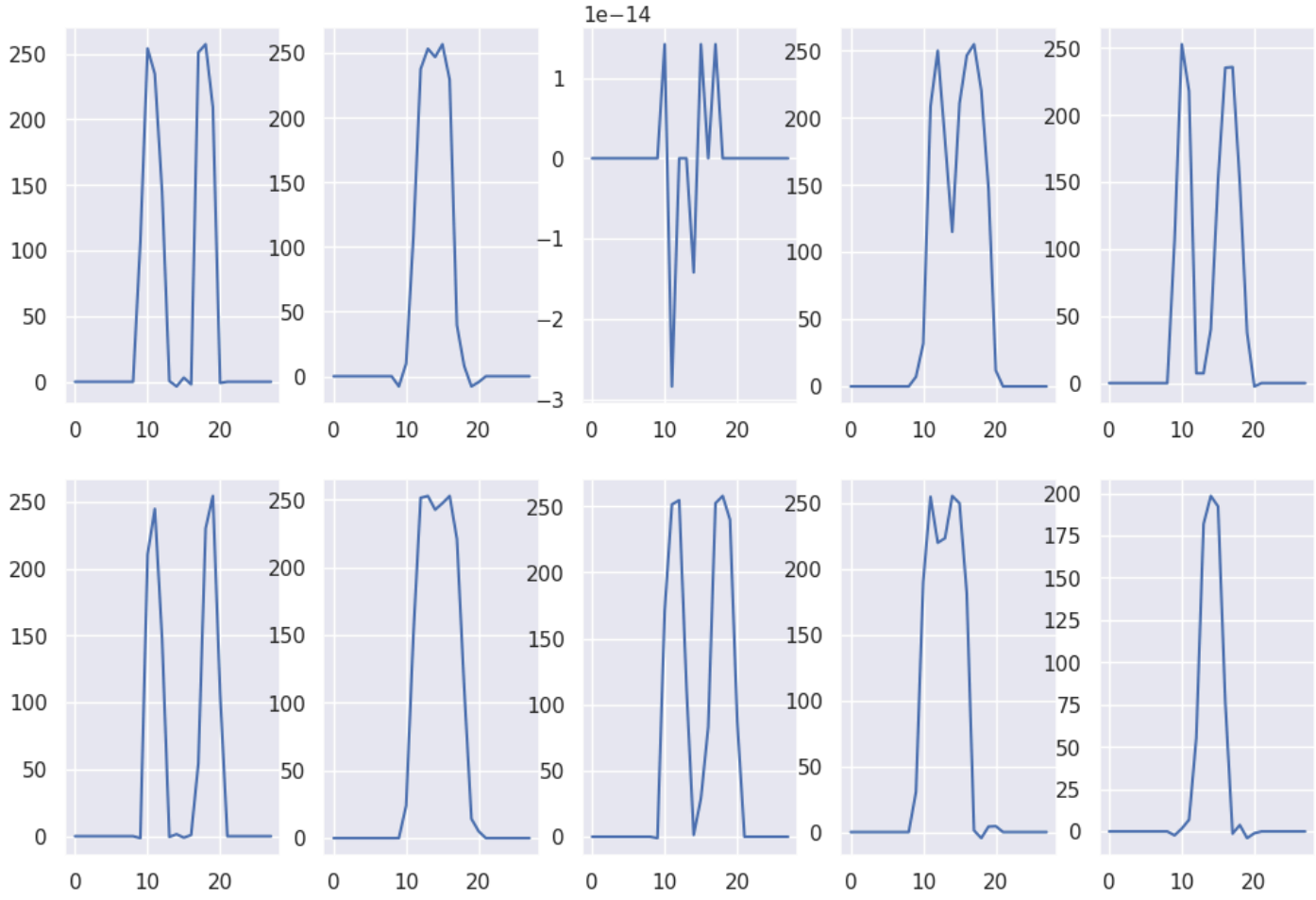


I chose rank 3 because when looking at the graph of s, 3 was located when the graph was decreasing. From 1 - 5, choosing the rank from 1 - 3 was the most ideal choice, but 1 and 2 might lead to a significant loss of information, allowing me to choose 3 as the rank.

e) Using Kmeans on this new dataset, cluster the images from d) using 10 clusters and plot the centroid of each cluster. Note: the centroids should be represented as images.

In [48]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=10, init='k-means++')
kmeans.fit_predict(N)


centroids = kmeans.cluster_centers_
fig, axes = plt.subplots(2, 5, figsize=(12, 8))
for i, ax in enumerate(axes.flatten()):
    ax.plot(centroids[i])
plt.show()
```

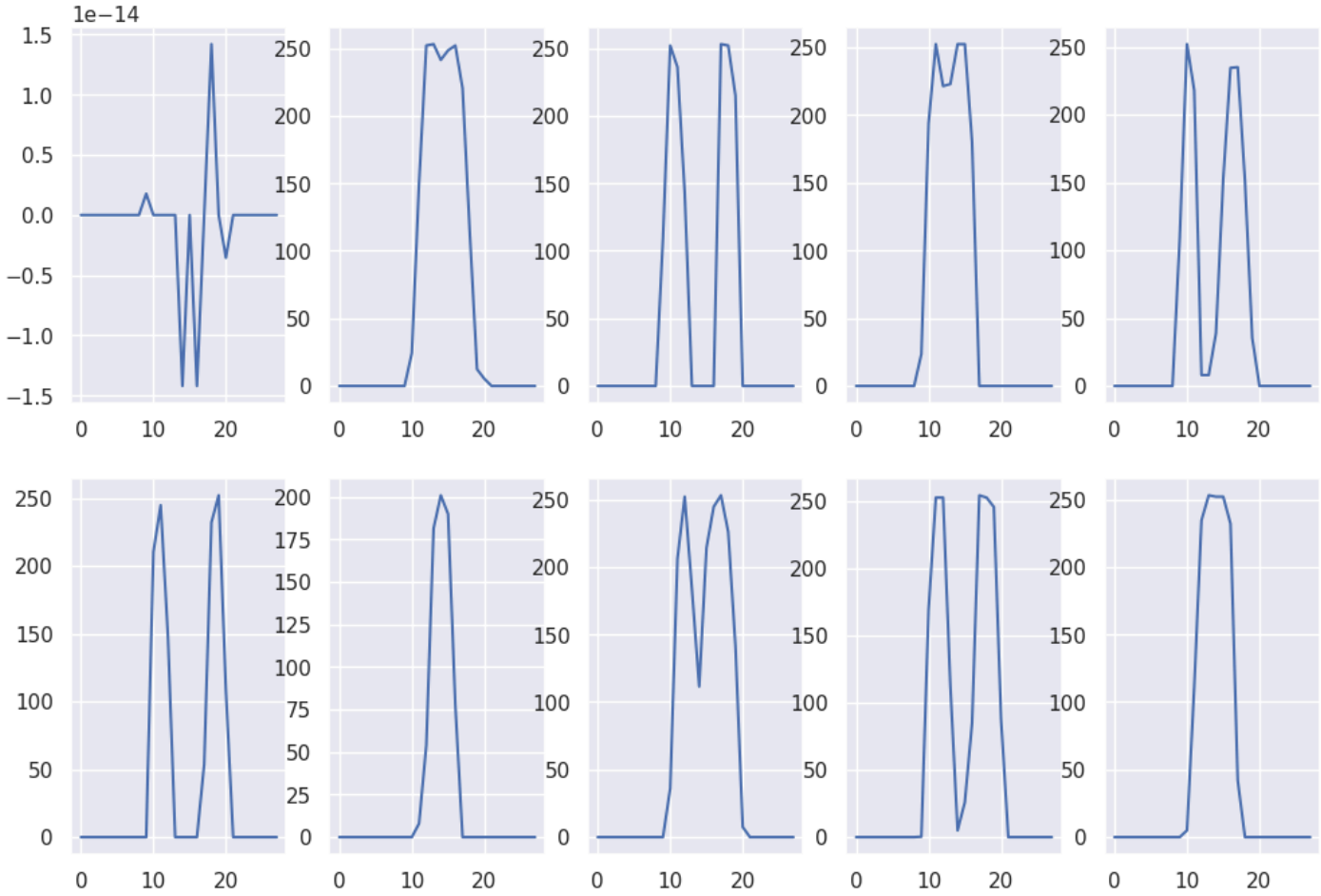
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
e default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_ini
t` explicitly to suppress the warning
  warnings.warn(
```



f) Repeat e) on the original dataset (if you used a subset of the dataset, keep using that same subset). Comment on any differences (or lack thereof) you observe between the centroids created here vs the ones you created in e).

In [49]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=10, init='k-means++')
kmeans.fit_predict(randomDigit)


centroids = kmeans.cluster_centers_
fig, axes = plt.subplots(2, 5, figsize=(12, 8))
for i, ax in enumerate(axes.flatten()):
    ax.plot(centroids[i])
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



The clusters for 1 and 3 are flipped and changed from the original image and the new image created from conducting SVD.

g) Create a matrix (let's call it `O`) that is the difference between the original dataset and the rank-10 approximation of the dataset. i.e. if the original dataset is `A` and the rank-10 approximation is `B`, then `O = A - B`

In [51]:
```
FBSpatial = randomDigit[:,:]
FBSnorm = np.linalg.norm(FBSpatial,axis=1,ord=1)


u,s,vt = np.linalg.svd(randomDigit,full_matrices=False)


RANK = 10
scopy = s.copy()
scopy[RANK:] = 0.
N = u @ np.diag(scopy) @ vt
O = FBSpatial - N
print(O)
```

```
[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -5.01095558e-14 -3.63670422e-14  2.68782766e-14  4.16231319e-15
  -1.66959771e-14  6.77603003e-02 -1.12601779e+00  9.08343880e-01
  -2.08234363e+00 -8.91844221e-01  3.83458014e+00 -3.16787069e+00
   2.20838053e+00  2.07019437e+00 -5.44573926e+00  6.09887865e+00
  -9.38996854e-01  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   2.24545323e-15 -1.02318244e-13  5.00162796e-14  6.52718341e-15
  -9.12298376e-16  4.92207077e+00 -3.67146211e+00  2.31417371e+00
  -3.12967381e+00 -7.08353963e-01  5.42055462e+00 -4.82263834e+00
   2.96842615e+00  2.76089576e+00 -8.17468037e+00  8.46549706e+00
   2.22855885e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -4.88796580e-15 -7.22049874e-14  5.99621946e-14  1.62138298e-14
   2.56689031e-15  8.67593811e-01  3.55035891e-01 -4.08850434e-01
   1.34864562e+00  6.97126870e-01 -2.54828926e+00  2.04007302e+00
  -1.49658526e+00 -1.40705815e+00  3.52887162e+00 -4.08251336e+00
   1.29642071e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   8.68973933e-15 -4.10502926e-14  1.07316018e-13  1.94846085e-14
  -2.14610848e-15  1.39905762e+00  1.55402992e-01 -3.19359706e-01
   1.38387075e+00  7.82051586e-01 -2.65101709e+00  2.08686963e+00
  -1.57269026e+00 -1.48080818e+00  3.62210097e+00 -4.26312370e+00
   1.71448707e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -5.03711708e-15 -9.75138903e-15  8.15937584e-14  1.71780499e-14
  -1.31136390e-14  2.46193829e+00 -2.08666845e+00  1.36147041e+00
  -2.03994850e+00 -5.59570736e-01  3.58621746e+00 -3.13392112e+00
   1.98914056e+00  1.85384606e+00 -5.32988101e+00  5.62639724e+00
   8.89043537e-01  0.00000000e+00  0.00000000e+00  0.00000000e+00
```

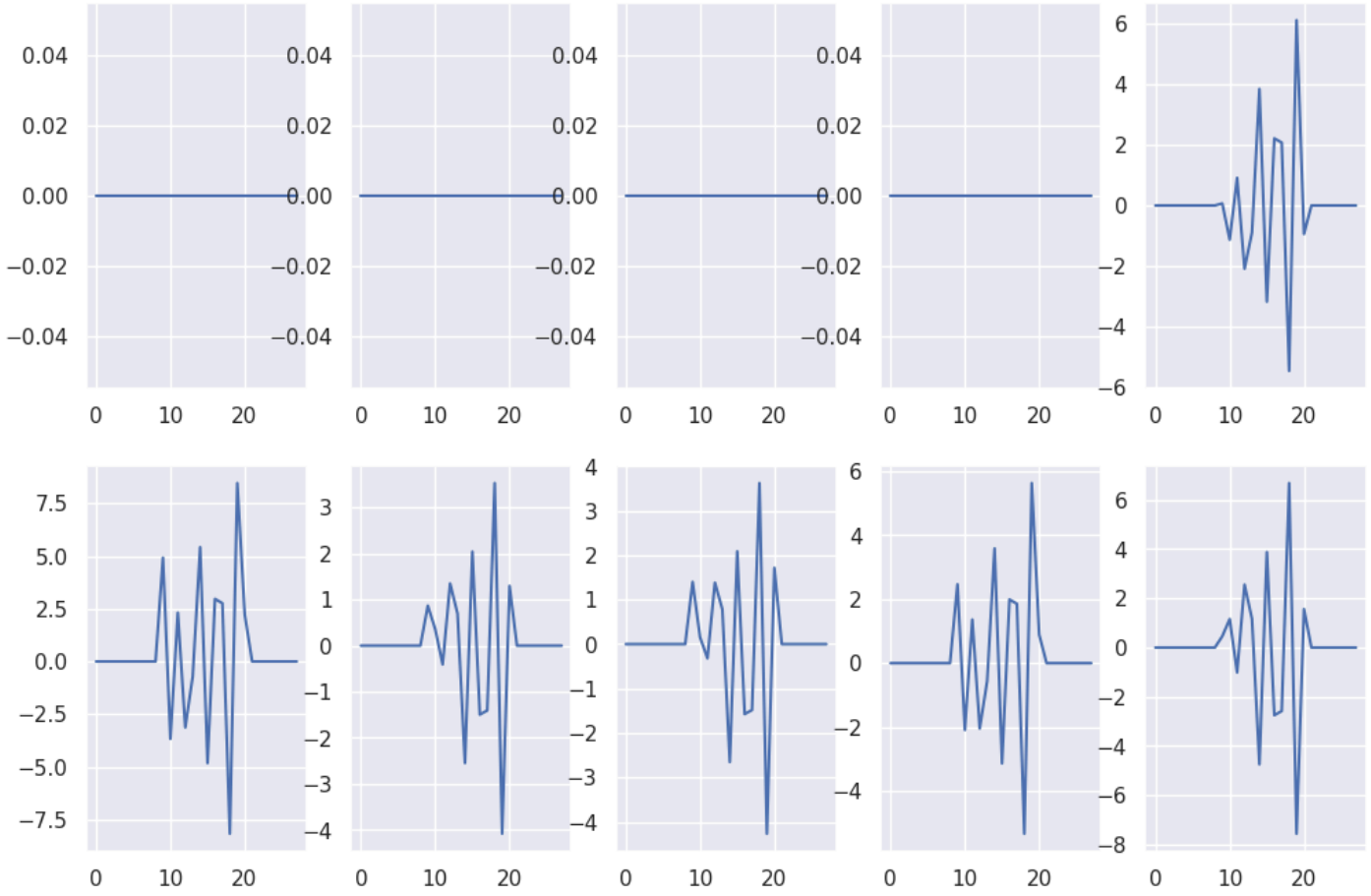
```
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.19171839e-14 -2.11530652e-14  6.17809784e-14  3.37339763e-14
 -6.50808514e-15  4.58561943e-01  1.15796821e+00 -1.00687977e+00
  2.55268226e+00  1.16430550e+00 -4.73919884e+00  3.87649076e+00
 -2.74659093e+00 -2.57717498e+00  6.67689498e+00 -7.55518361e+00
  1.56007915e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -1.52635827e-14 -2.90686409e-14  5.82717607e-14  5.83222642e-14
  7.45367594e-15  5.31012223e+00 -2.86867236e+00  1.60646814e+00
 -1.30535692e+00  1.31647425e-01  2.02931837e+00 -2.05300896e+00
  1.00113996e+00  9.14687465e-01 -3.40280057e+00  3.05728695e+00
  3.38905344e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.94611482e-14 -8.68806940e-14  5.41616990e-14  4.66677912e-15
  3.89207211e-14 -6.11920839e+00  1.30461856e+00 -2.20356244e-01
 -2.29021631e+00 -1.79302347e+00  4.65766648e+00 -3.40513018e+00
  2.87943425e+00  2.72725106e+00 -6.00226027e+00  7.60827139e+00
 -5.70971824e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -8.59651460e-15 -6.40815665e-14  5.55792221e-14 -7.54177064e-15
  3.36681044e-14 -7.96457020e+00  6.34721857e+00 -4.07576858e+00
  5.83463828e+00  1.47945528e+00 -1.01916491e+01  8.97537557e+00
 -5.62217177e+00 -5.23523408e+00  1.52425458e+01 -1.59583617e+01
 -3.23977513e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.01091200e-14 -9.23048853e-14  3.65489145e-14  1.36236611e-15
  1.17230073e-14  1.07990155e+01 -6.02449007e+00  3.42232840e+00
 -3.01600599e+00  1.11424745e-01  4.79320060e+00 -4.72470049e+00
  2.42005669e+00  2.22026156e+00 -7.86517535e+00  7.27753946e+00
  6.72038149e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -1.25434052e-14 -1.03159339e-13  7.36174139e-14  2.37431685e-14
 -2.10355697e-14 -7.78885878e+00  5.49812494e+00 -3.40798070e+00
  4.36142679e+00  8.65249566e-01 -7.48785252e+00  6.73254874e+00
 -4.06908755e+00 -3.77990931e+00  1.13900732e+01 -1.16621196e+01
 -3.80761135e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  6.84406409e-16 -7.22471645e-14  4.57709545e-14  2.33038004e-14
 -2.85826974e-14 -3.53344584e+00  2.97609499e+00 -1.93873925e+00
  2.89223606e+00  7.87731611e-01 -5.08148339e+00  4.44381606e+00
 -2.81707866e+00 -2.62525972e+00  7.55660832e+00 -7.97085839e+00
 -1.29288912e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.87746798e-14 -1.66157464e-14  3.45643485e-14  4.06958548e-15
 -6.70289884e-15  6.84713248e+00 -5.46154150e+00  3.50788202e+00
 -5.02521815e+00 -1.27585952e+00  8.77868661e+00 -7.73009141e+00
  4.84313864e+00  4.50987914e+00 -1.31280238e+01  1.37463339e+01
  2.78086330e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  6.41673902e-15 -2.03389470e-14  2.16153794e-14 -3.34996791e-15
  1.45417079e-14  2.19728240e+00  1.05025078e+00 -1.15858875e+00
  3.70207420e+00  1.88947008e+00 -6.98203271e+00  5.60241382e+00
 -4.09476587e+00 -3.84901721e+00  9.68647880e+00 -1.11798306e+01
```

```
  3.41955523e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -6.01413575e-15 -1.56471542e-14  1.37867692e-14 -1.42912262e-14
  1.30527543e-14  2.24022148e+00 -5.31892301e-01  1.24905194e-01
  7.35525350e-01  6.11903333e-01 -1.51540248e+00  1.09008455e+00
 -9.44763323e-01 -8.95879219e-01  1.92825695e+00 -2.48345230e+00
  2.04137194e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -2.88743443e-14 -2.63784605e-15  2.44793287e-14 -5.97803072e-15
  1.51791957e-15 -8.96283467e-03 -1.16728800e+00  9.52546163e-01
 -2.22032908e+00 -9.61590868e-01  4.09444987e+00 -3.37675248e+00
  2.36062608e+00  2.21328043e+00 -5.80676748e+00  6.51482612e+00
 -1.06254440e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.22264866e-14 -2.01334691e-14  6.80807208e-14  1.68689899e-14
  8.83170757e-15 -3.99961201e+00  3.72899895e-01  2.47013035e-01
 -2.40673094e+00 -1.56549164e+00  4.72181866e+00 -3.60936295e+00
  2.84907452e+00  2.68922846e+00 -6.30256471e+00  7.64188914e+00
 -4.16458104e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -8.04615318e-15 -6.70770201e-14  5.22371070e-14  2.51458824e-14
 -2.09536477e-14 -1.08933572e+01  5.25540916e+00 -2.78255111e+00
  1.48426695e+00 -7.86357435e-01 -1.96229441e+00  2.39630344e+00
 -7.85117314e-01 -6.86974459e-01  3.85907555e+00 -2.77033638e+00
 -7.51996913e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -8.46329271e-15 -7.86901674e-14  5.39731515e-14  2.02577630e-14
 -2.31012125e-15  6.85595872e+00 -4.75736134e+00  2.93277642e+00
 -3.68311662e+00 -6.94168746e-01  6.30351055e+00 -5.68902093e+00
  3.41598513e+00  3.17179039e+00 -9.61805434e+00  9.80788241e+00
  3.42570341e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -4.23349678e-14 -8.13221836e-14  6.23571410e-14 -1.36534569e-15
 -1.66643491e-14  4.52528583e+00 -2.30242104e+00  1.25309246e+00
 -8.42676418e-01  2.28871204e-01  1.23202604e+00 -1.33931721e+00
  5.66457224e-01  5.10683460e-01 -2.19440220e+00  1.81408924e+00
  3.01641783e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
```

```
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

h) The largest (using euclidean distance from the origin) rows of the matrix `O` could be considered anomalous data points. Briefly explain why. Plot the 10 images (by finding them in the original dataset) responsible for the 10 largest rows of that matrix `O`.

In [60]:
```
fig, axes = plt.subplots(2, 5, figsize=(12, 8))
for i, ax in enumerate(axes.flatten()):
  ax.plot(O[i])
```



Anomalies are outliers from the data points. The largest rows of matrix O could be considered as anomalous data points because they contain the strongest pattern in the image and therefore data points that are badly represented are farthest away from the important patterns, making them the strongest outliers.

In [ ]: