Worksheet 10

Name: Kian Boon Tan

UID: U93983891

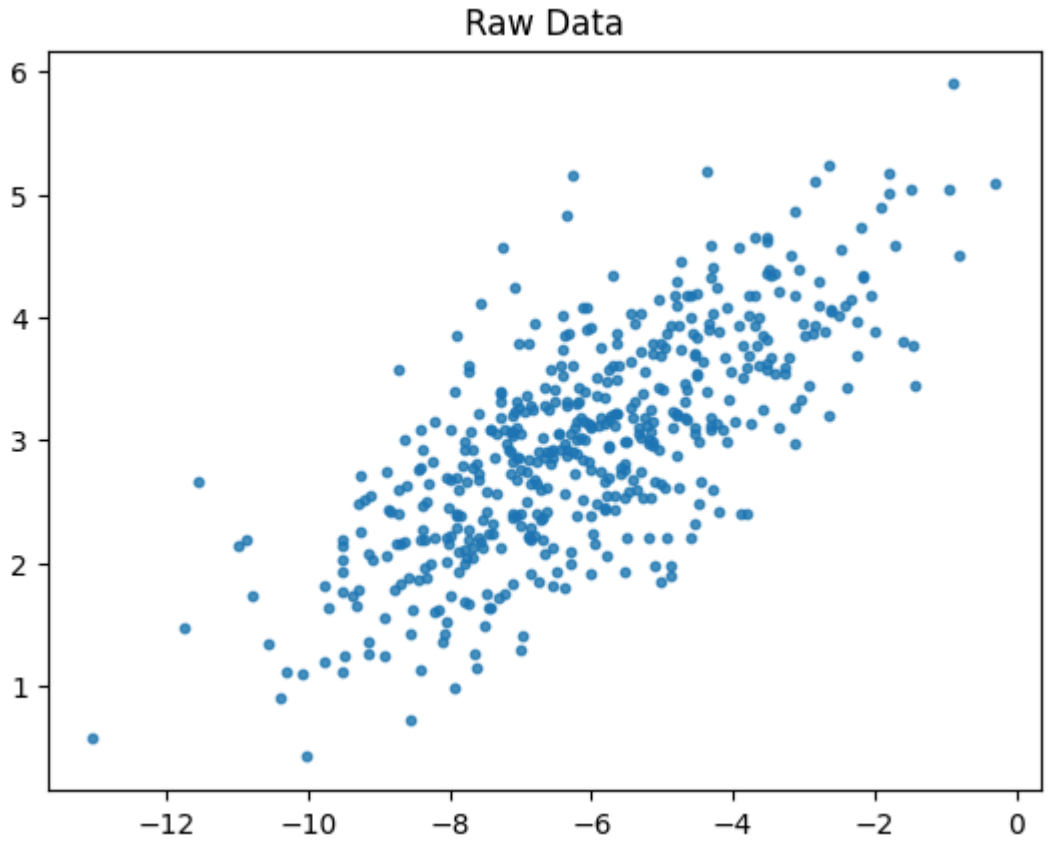
Topics

- Singular Value Decomposition

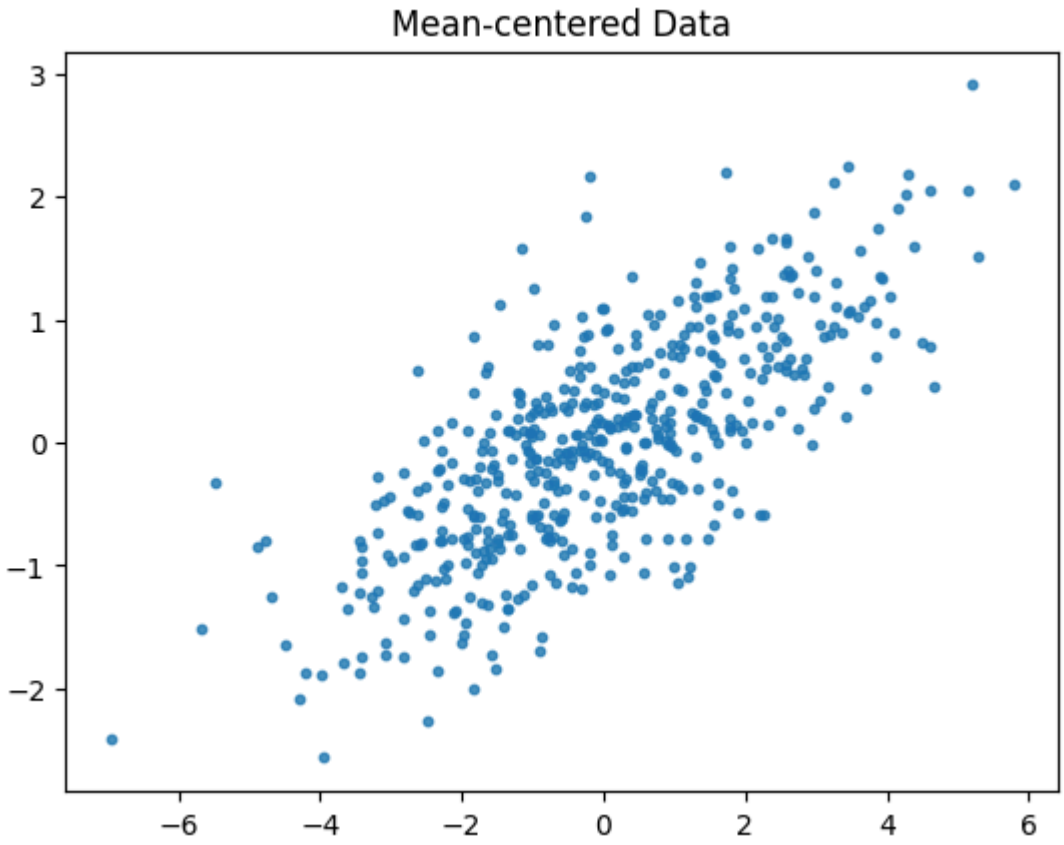
Feature Extraction

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.

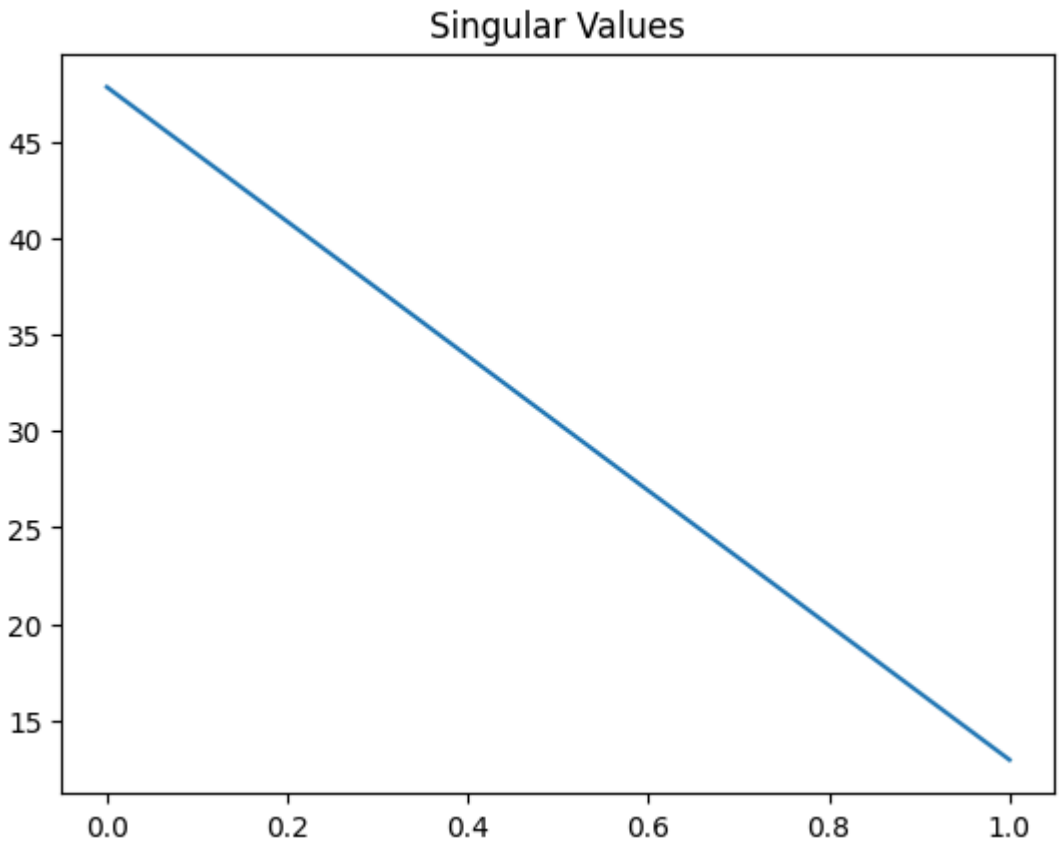
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

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 [2]:
```
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()
```

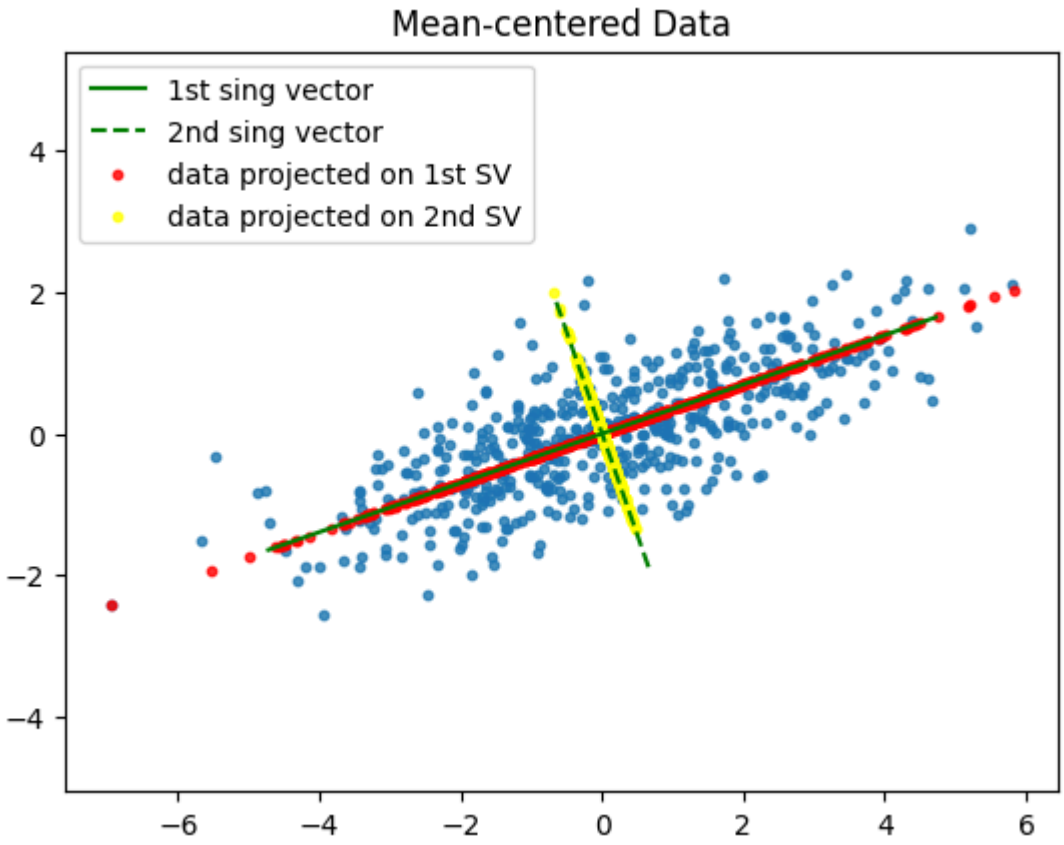


In [3]:
```
u,s,vt=np.linalg.svd(X, full_matrices=False)
plt.plot(s) # only 2 singular values
plt.title("Singular Values")
plt.show()
```

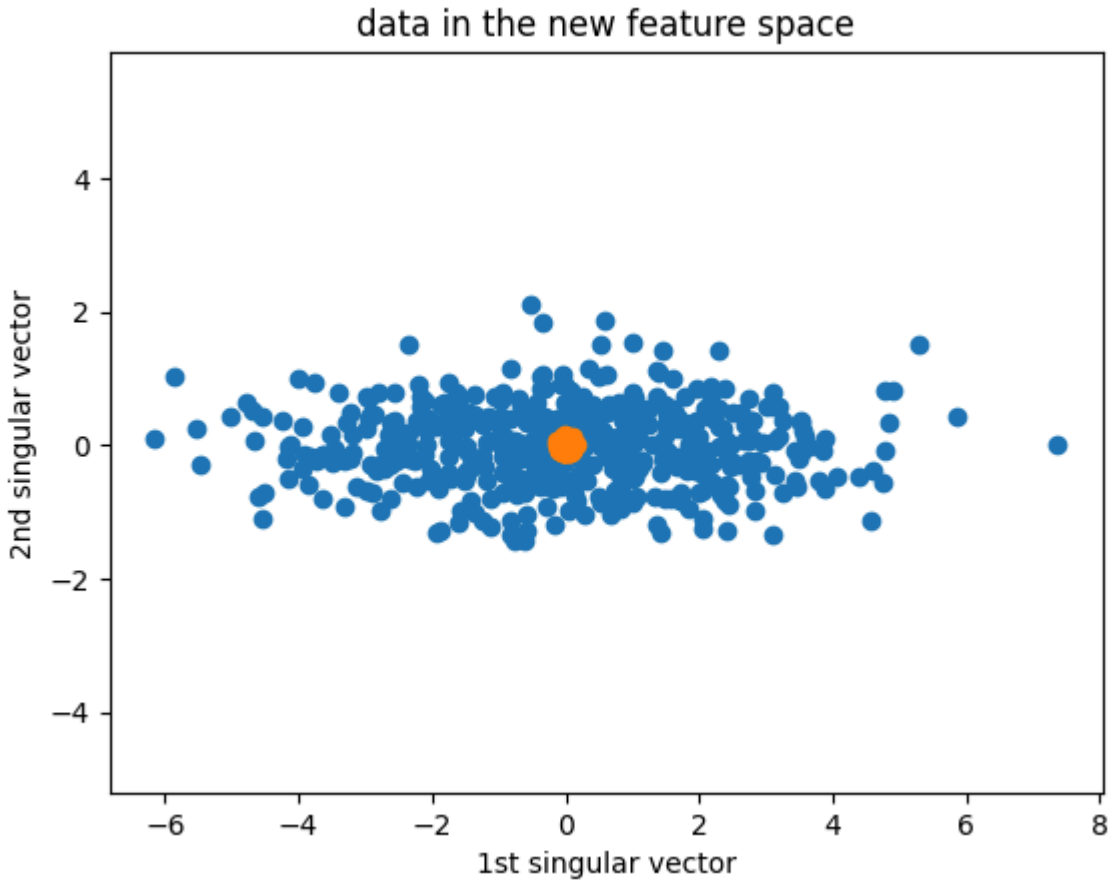


In [4]:

```
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 p
plt.scatter(approx1[:, 0] , approx1[:, 1], s=10, alpha=0.8, color="yellow", label="dat
plt.axis('equal')
plt.legend()
plt.title("Mean-centered Data")
plt.show()
```

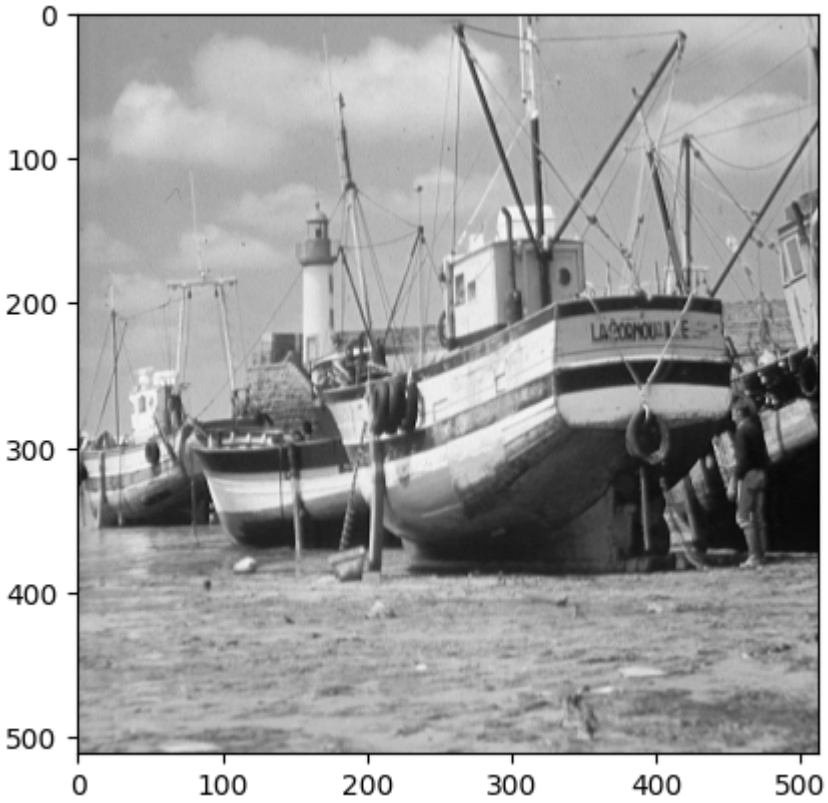


In [5]:
```
# show ouput from svd is the same
orthonormal_X = u
shifted_X = u.dot(np.diag(s))
plt.axis('equal')
plt.scatter(shifted_X[:,0], shifted_X[:,1])
plt.scatter(orthonormal_X[:,0], orthonormal_X[:,1])
plt.xlabel("1st singular vector")
plt.ylabel("2nd singular vector")
plt.title("data in the new feature space")
plt.show()
```



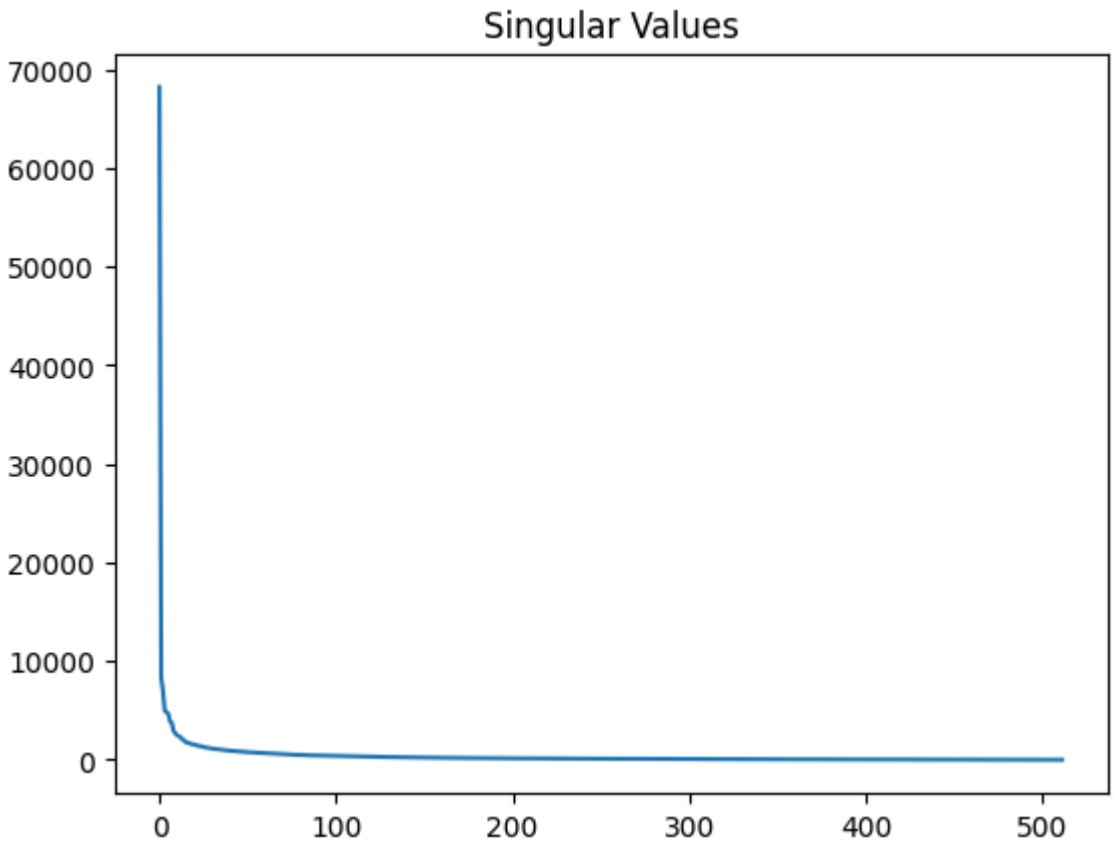
In [6]:
```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

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

Out[6]: <matplotlib.image.AxesImage at 0x1aff92c7370>



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

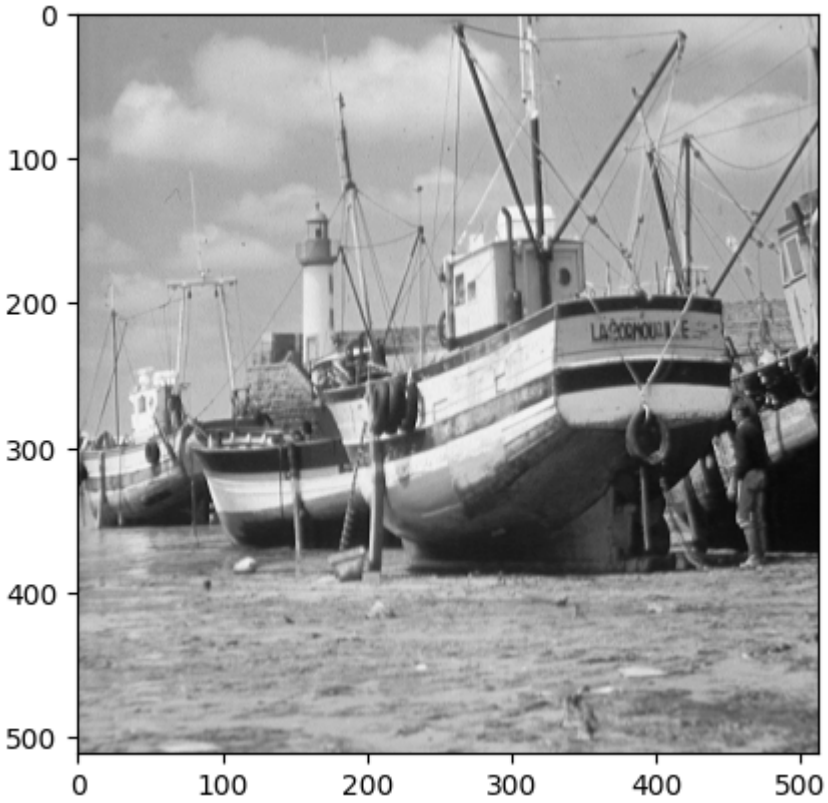
In [7]:
```
u,s,vt=np.linalg.svd(boat,full_matrices=False)

# Plot singular values.
plt.plot(s)
plt.title("Singular Values")
plt.show()
```



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

In [8]:
```
boat_copy = u.dot(np.diag(s)).dot(vt)
plt.figure()
plt.imshow(boat_copy, cmap = cm.Greys_r)
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x1aff999c0d0>
```

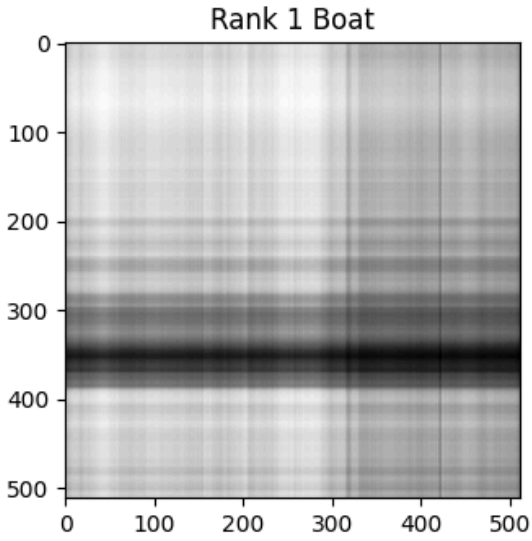


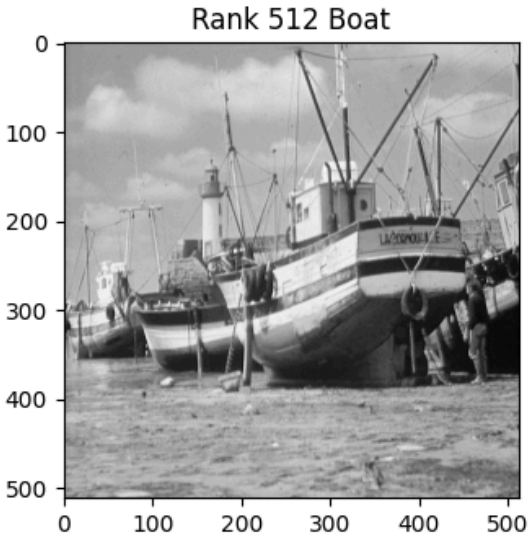
b) Create a new matrix `scopy` which is a copy of `s` with all but the first singular value set to 0.

In [9]:
```
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 [10]:
```
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()
```

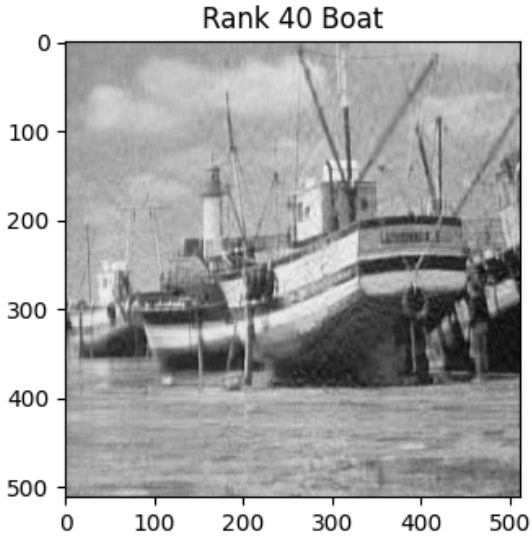


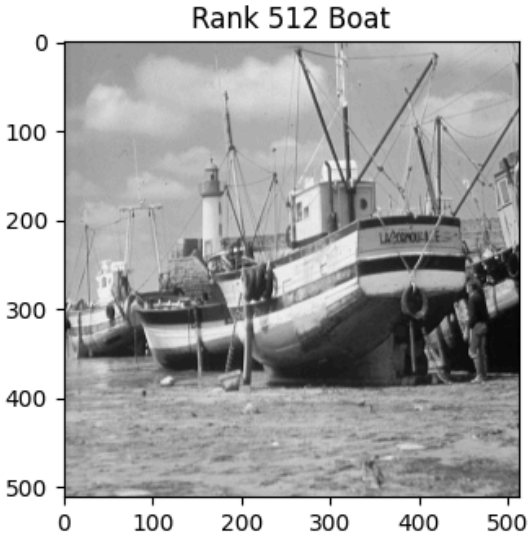


d) Repeat c) with 40 singular values instead of just 1.

In [11]:
```
# Create new scopy with 40 singular values.
scopy = s.copy()
scopy[40:] = 0.0
```

In [12]:
```
# Replot images.
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

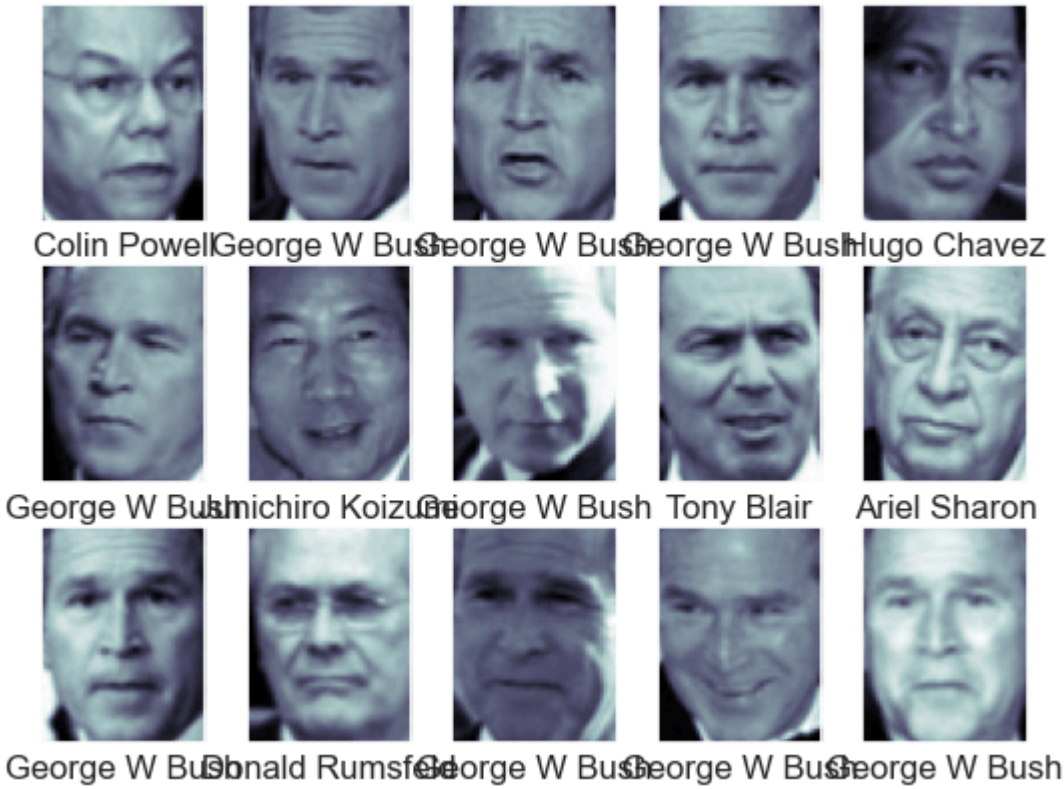
a) By using an approximation of the data, you can improve the performance of classification tasks since:

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

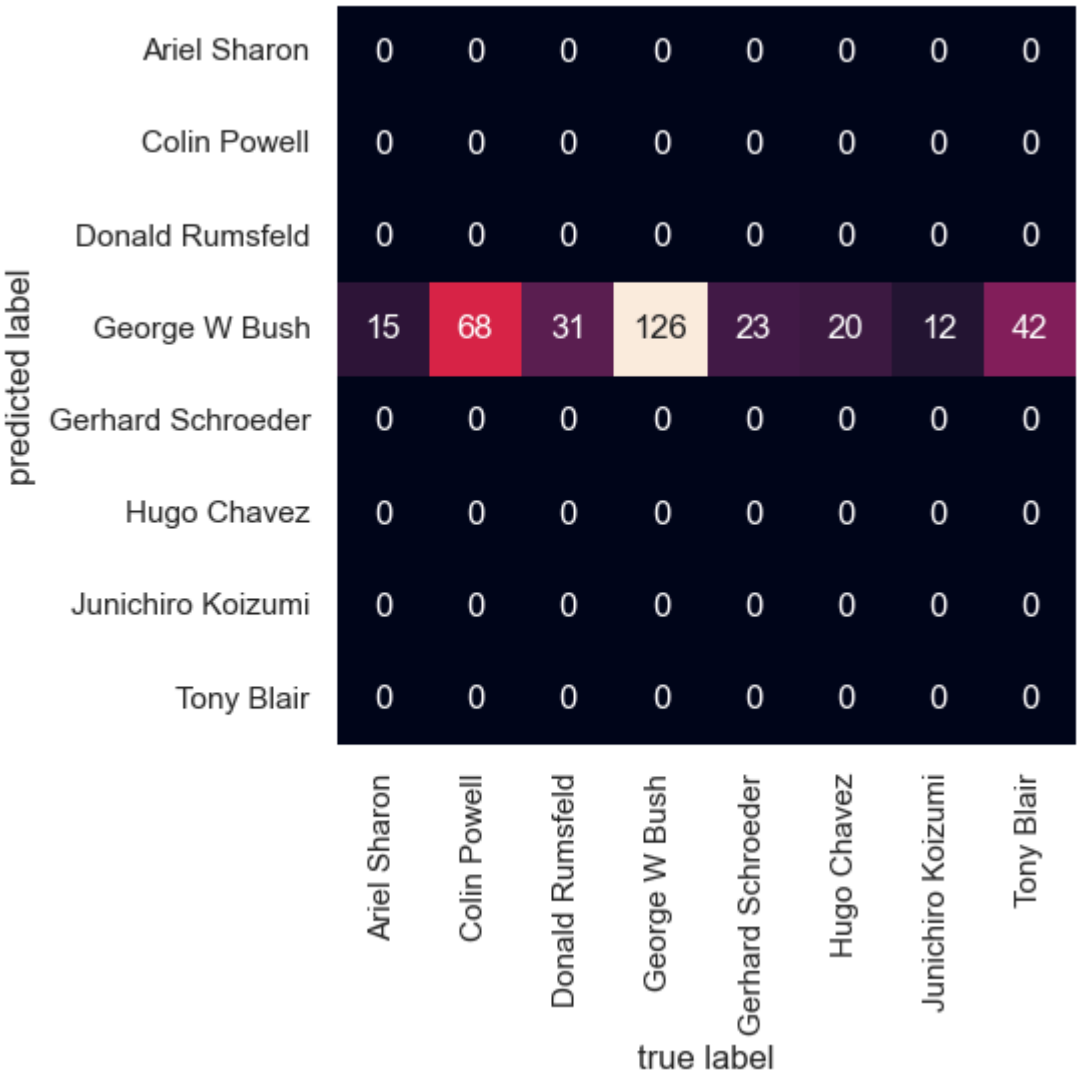
Below is some code to perform facial recognition on a dataset. Notice that, applied blindly, it does not perform well:

In [13]:
```
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

# 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))
```



Predicted Names; Incorrect Labels in Red

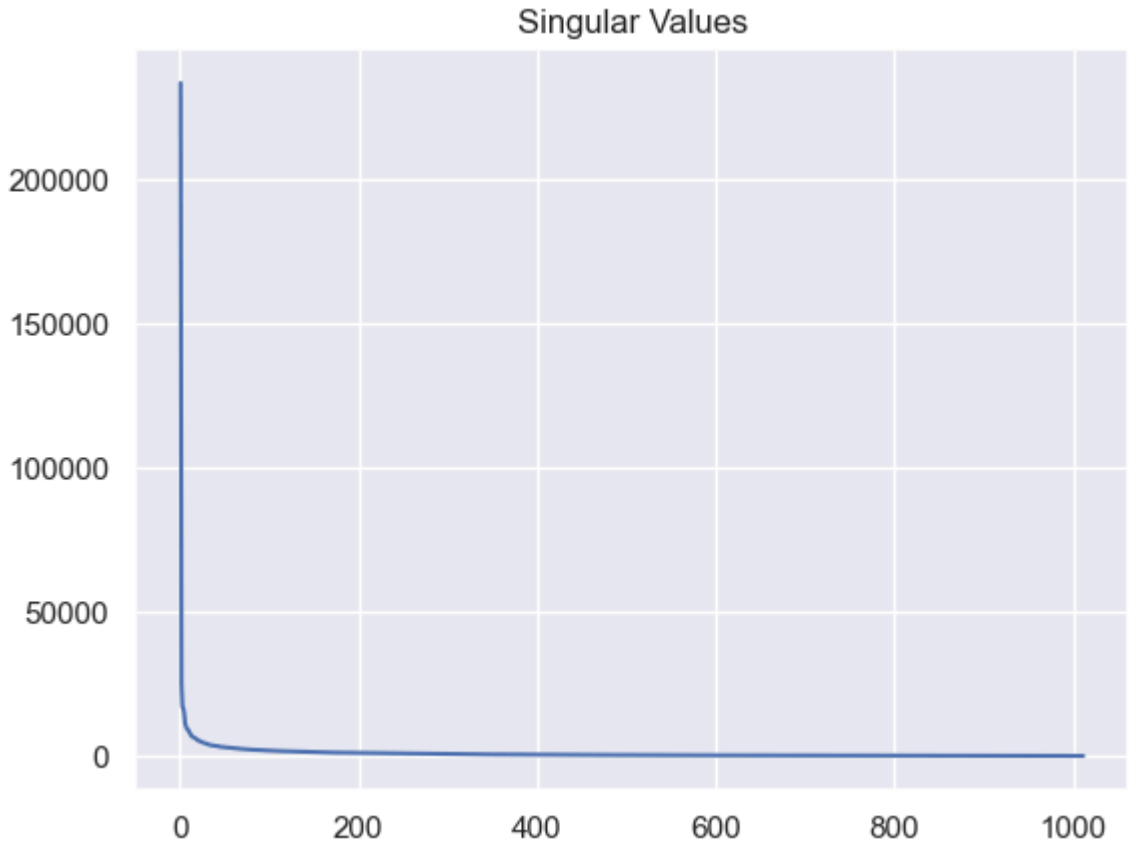



```
Accuracy =  0.37388724035608306
```

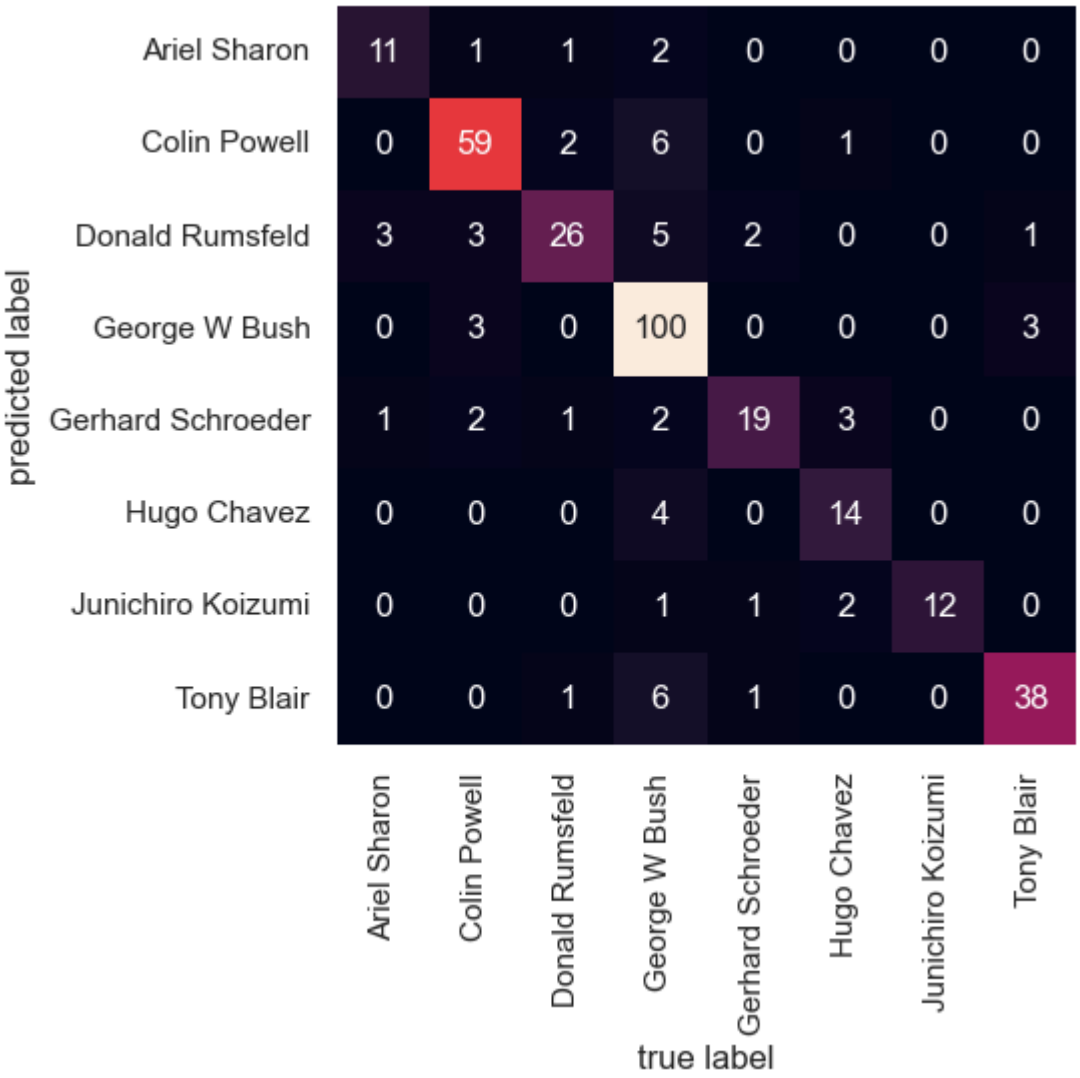
By performing SVD before applying the classification tool, we can reduce the dimension of the dataset.

In [14]:
```
# 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()

# ELIMINATES NOISE
# extract principal components (chosen components = 100 / 40 etc.)
pca = PCA(n_components=100, 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))
```



Predicted Names; Incorrect Labels in Red



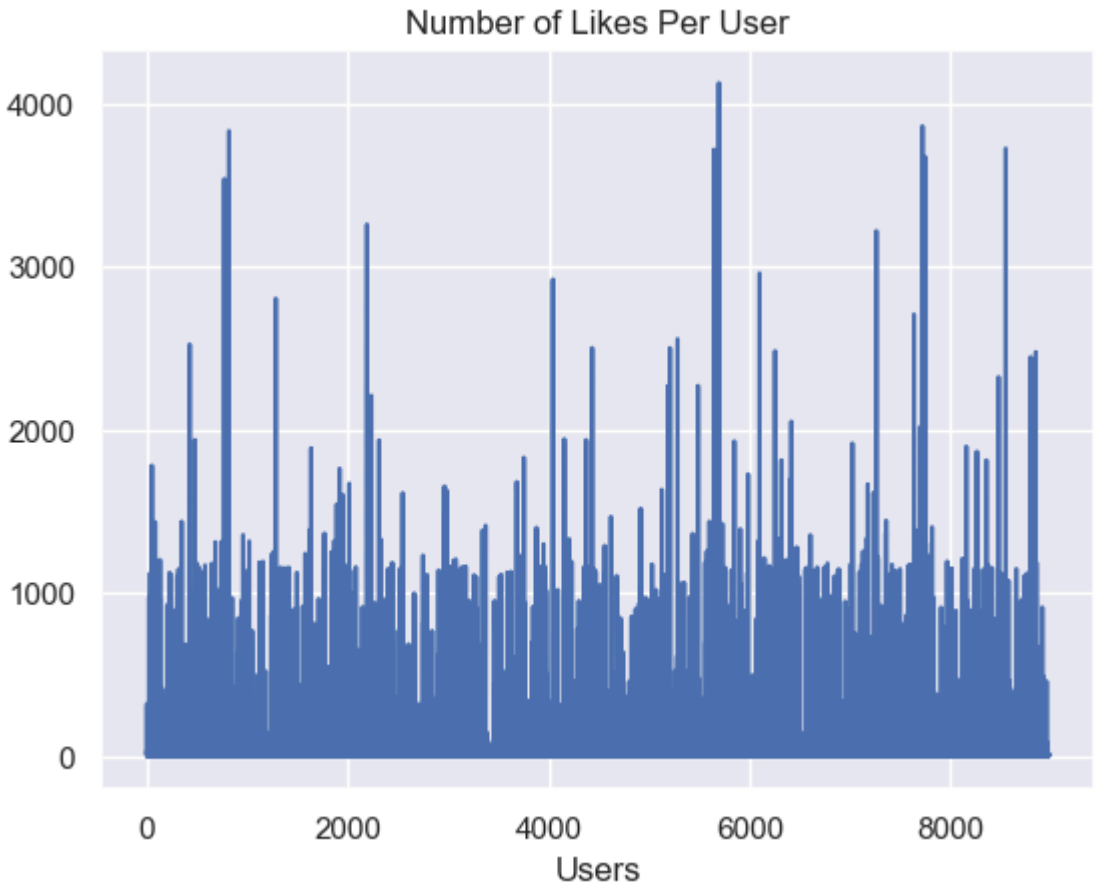

```
Accuracy =  0.827893175074184
```

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.

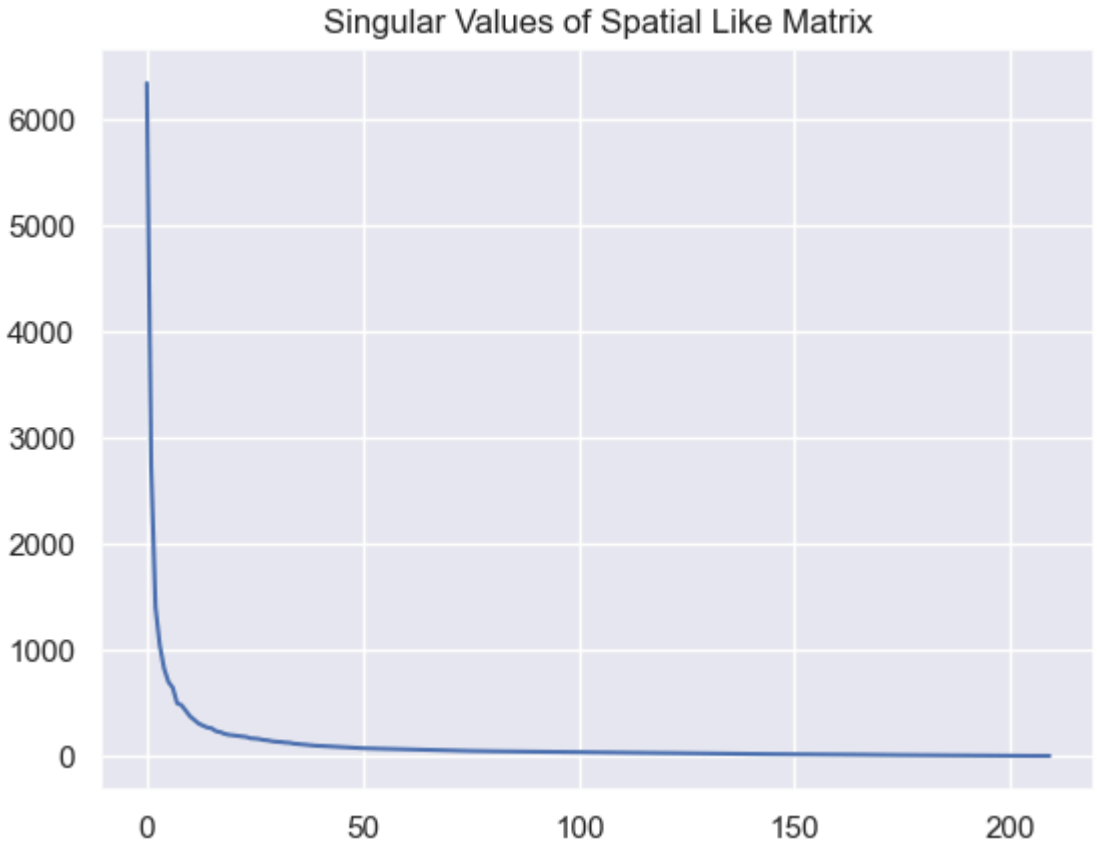
In [16]:
```
data = np.loadtxt('spatial_data.txt')

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()
```

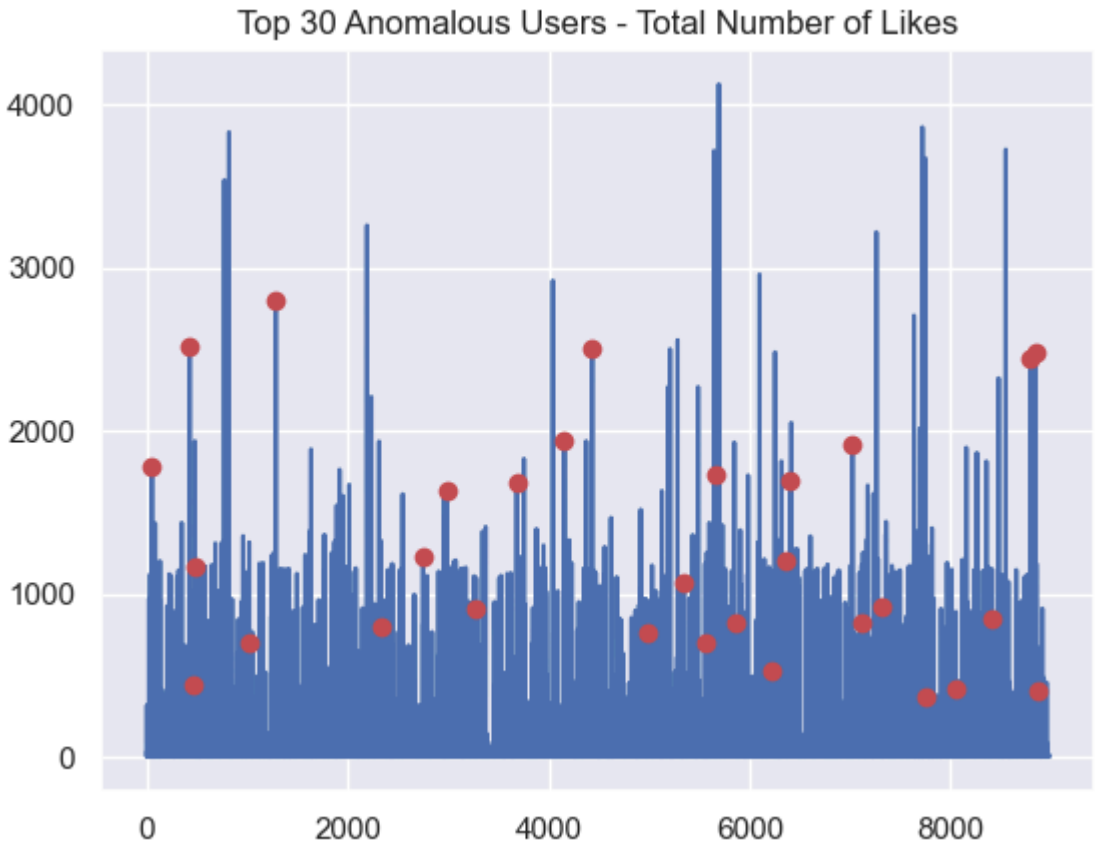


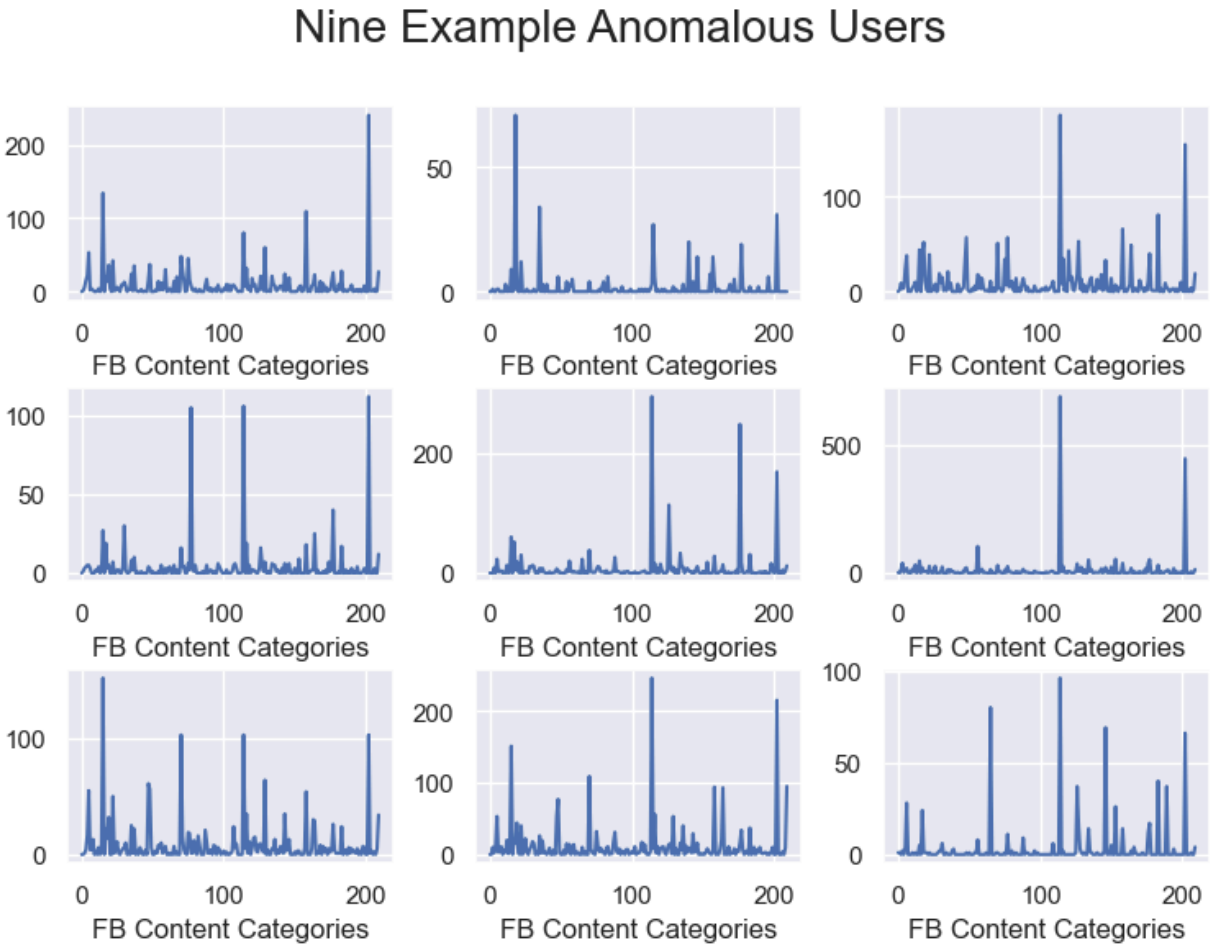
How users distribute likes across categories follows a general pattern that most users follow. This behavior can be captured using few singular vectors. And anomalous users can be easily identified.

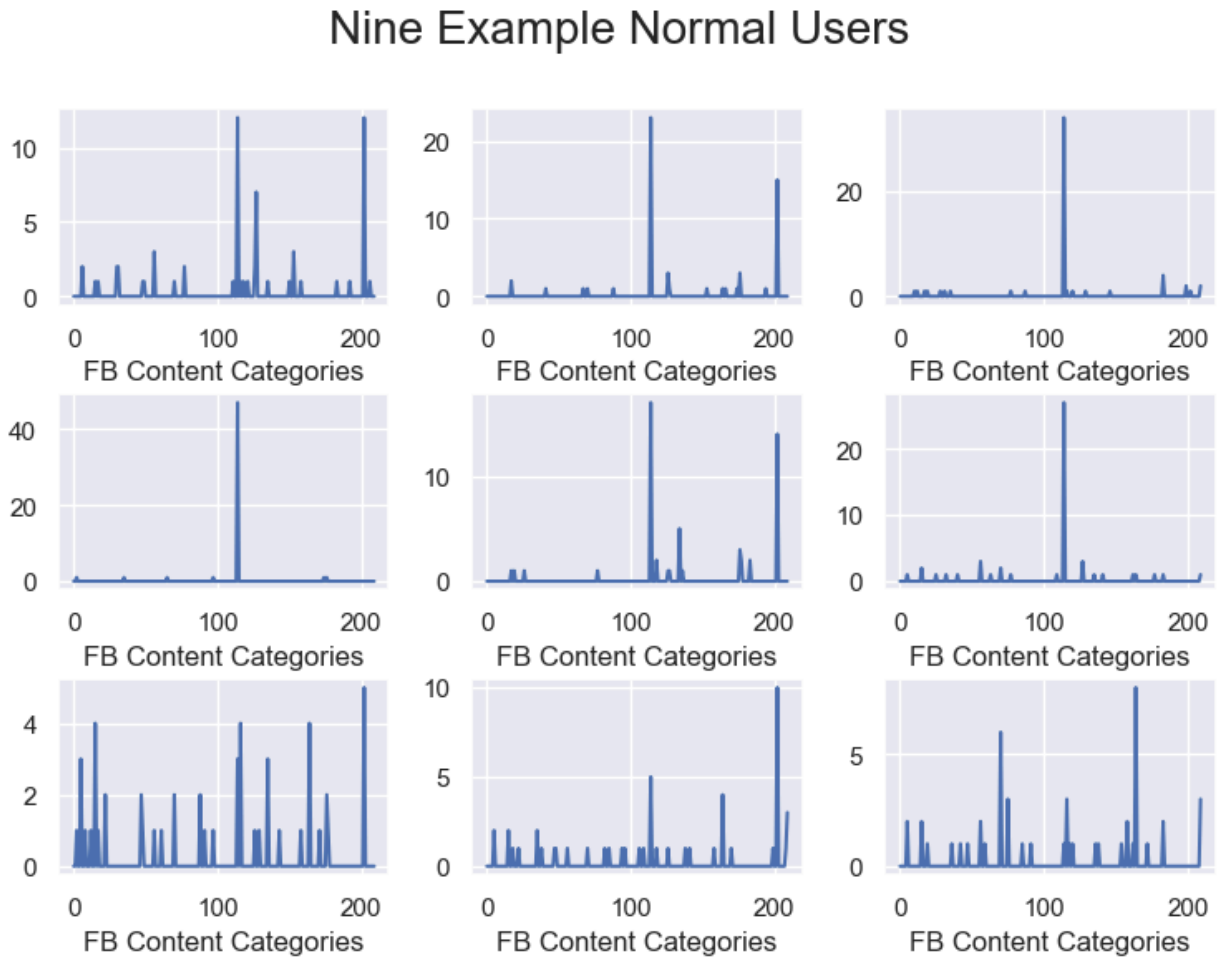
In [18]:
```
u,s,vt = np.linalg.svd(FBSpatial,full_matrices=False)
plt.plot(s)
_ = plt.title('Singular Values of Spatial Like Matrix')
plt.show()

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()
```




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



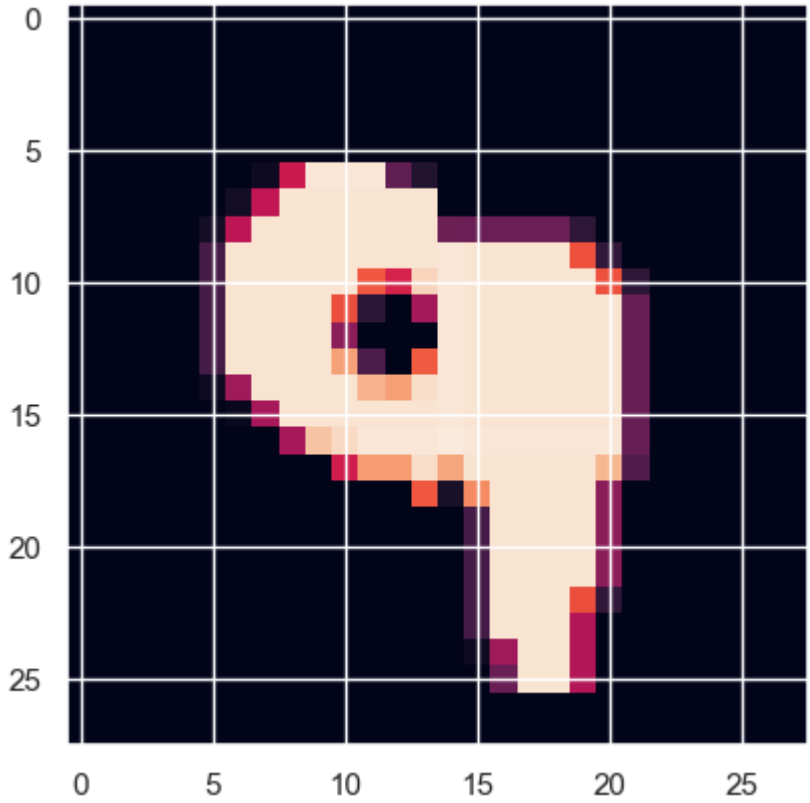
Challenge Problem

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

In [22]:
```
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)
```

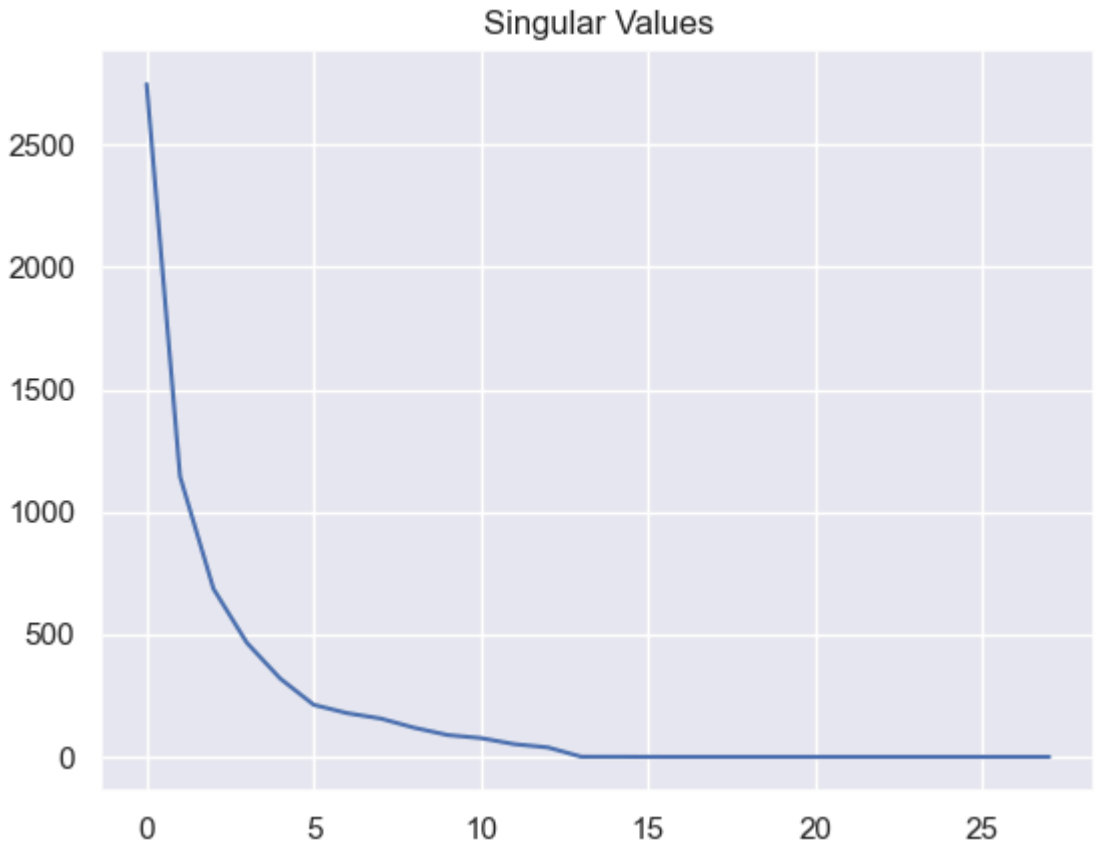
In [39]:
```
# your code here
# Pick a random image and convert it to view in an image.
image_index = np.random.randint(0, len(X))
image = X[image_index]
image_display = np.reshape(image, (28, 28))

# Display random digit.
plt.figure()
plt.imshow(image_display)
```

Out[39]:
```
<matplotlib.image.AxesImage at 0x1af989d5fd0>
```



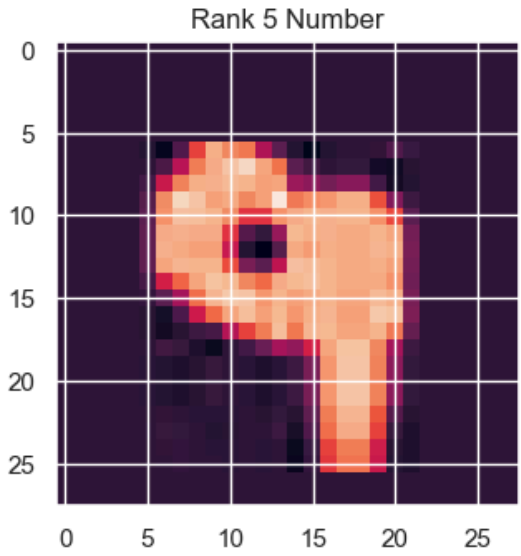
b) Plot its singular value plot.

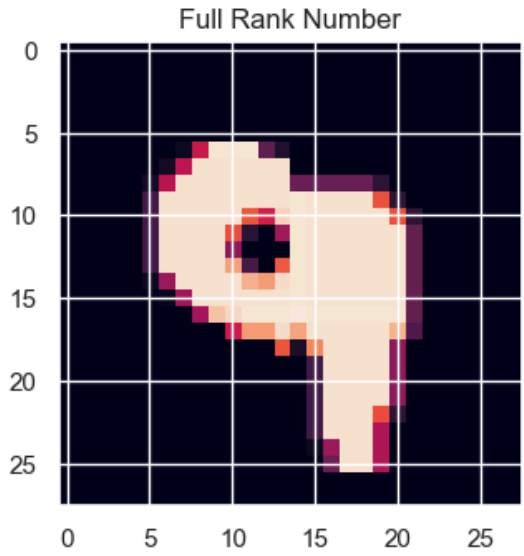
In [40]:
```
u,s,vt=np.linalg.svd(image_display,full_matrices=False)

# Plot singular values.
plt.plot(s)
plt.title("Singular Values")
plt.show()
```



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

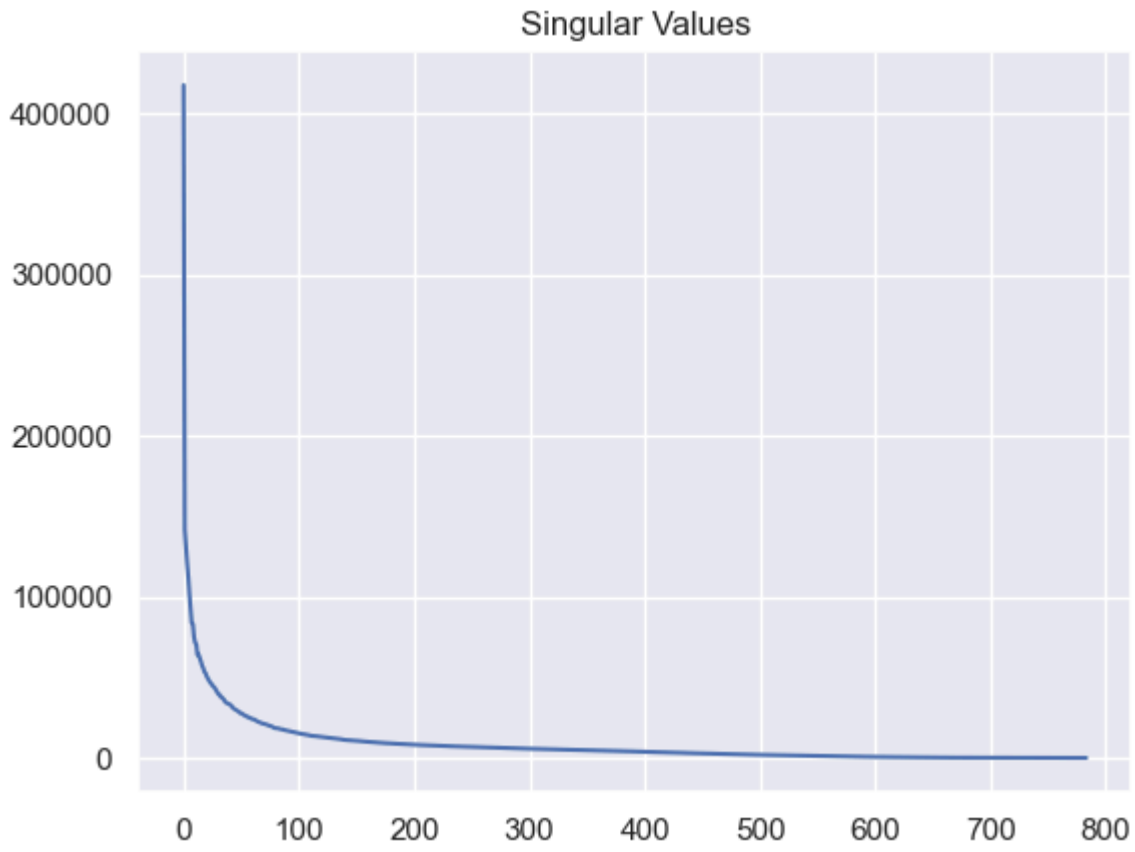
In [42]:
```
# Set some singular values to 0.
RANK = 5
scopy = s.copy()
scopy[RANK:] = 0.0

# Replot images.
reduced_image_display = u.dot(np.diag(scopy)).dot(vt)

plt.figure(figsize=(9,6))
plt.subplot(1,2,1)
plt.imshow(reduced_image_display)
plt.title(f'Rank {RANK} Number')
plt.subplot(1,2,2)
plt.imshow(image_display)
plt.title('Full Rank Number')
_ = 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 [45]:
```
# Perform SVD on the entire dataset.
u,s,vt=np.linalg.svd(X,full_matrices=False)

# Plot its singular values.
plt.plot(s)
plt.title("Singular Values")
plt.show()
```



By considering the graph of its singular values, we can observe that there are hugely diminishing returns after 100 singular values. Hence, we can just consider the first 100.

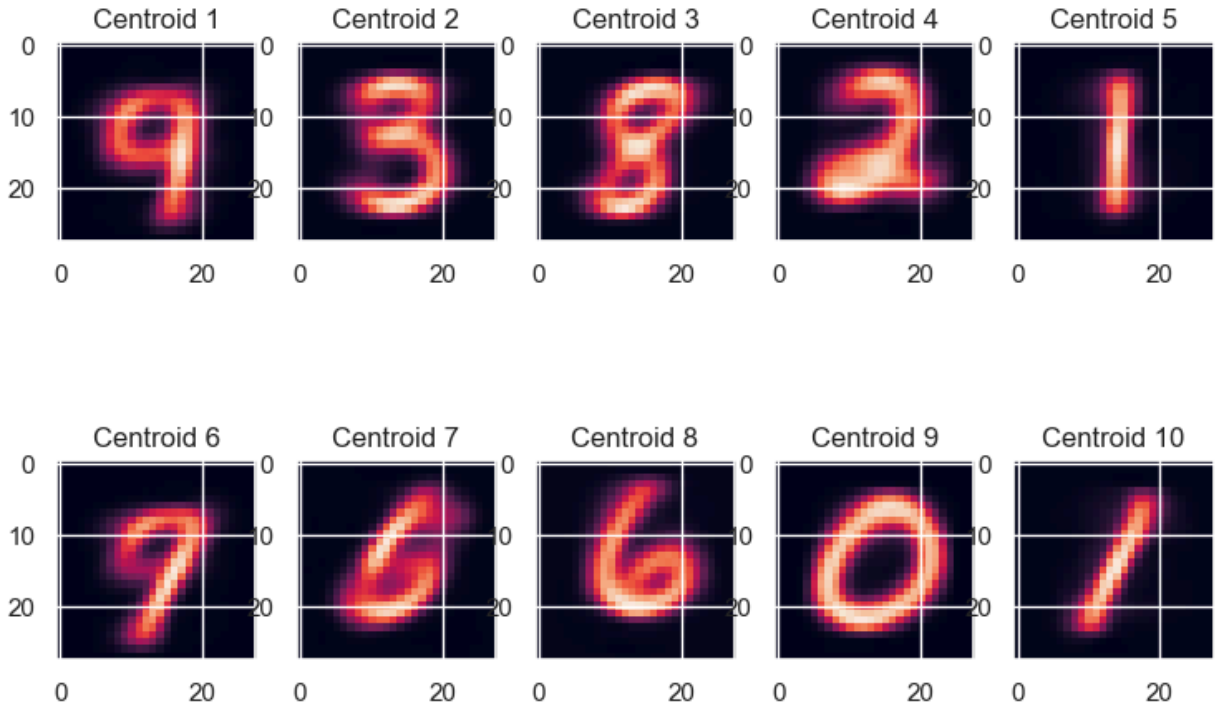
In [63]:
```
# Get an approximation on the data by considering the first RANK singular values.
RANK = 100
scopy = s.copy()
scopy[RANK:] = 0.0

approximated_X = u.dot(np.diag(scopy)).dot(vt)
```

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 [47]:
```
from sklearn.cluster import KMeans

# Perform KMeans on approximated dataset.
kmeans = KMeans(n_clusters=10, init='k-means++')
kmeans.fit_predict(X=approximated_X)
```

Out[47]:
```
array([2, 8, 0, ..., 5, 6, 7])
```

In [57]:
```
# Plot the centroid of each cluster.
centers = kmeans.cluster_centers_

plt.figure(figsize=(9,6))

for i in range(len(centers)):
    # Calculate its subplot position.
    plt.subplot(2, 5, i + 1)

    # Translate it into image dimensions and plot.
    center = centers[i]
    center_image = np.reshape(center, (28, 28))
    plt.imshow(center_image)
    plt.title(f'Centroid {i + 1}')

plt.show()
```

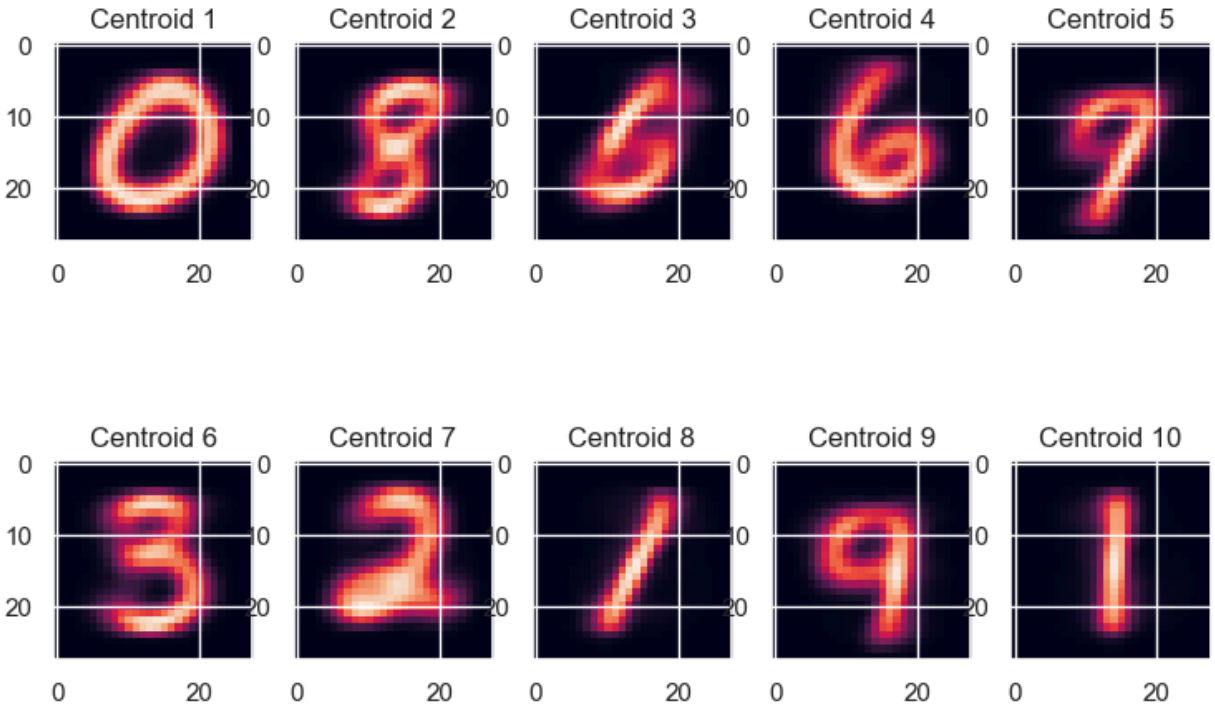


f) Repeat e) on the original dataset (if you used a subset of the dataset, keep using that same subset). Comment on any differences (or lack thereof) you observe between the centroids created here vs the ones you created in e).

In [65]:
```
# Repeat KMeans on original dataset.
kmeans = KMeans(n_clusters=10, init='k-means++')
kmeans.fit_predict(X=X)
```

Out[65]:
```
array([1, 0, 8, ..., 4, 2, 3])
```

In [66]:
```
# Plot the centroid of each cluster.
centers = kmeans.cluster_centers_

plt.figure(figsize=(9,6))

for i in range(len(centers)):
    # Calculate its subplot position.
    plt.subplot(2, 5, i + 1)

    # Translate it into image dimensions and plot.
    center = centers[i]
    center_image = np.reshape(center, (28, 28))
    plt.imshow(center_image)
    plt.title(f'Centroid {i + 1}')

plt.show()
```



The centroids calculated using both datasets are exactly the same, in ascending order, `0 1 1 2 3 6 6 8 9 9`. This suggests that the rank-100 approximation that we made for the approximated dataset is a good estimate of the original dataset (i.e. does not deviate by a lot), given that the differences between the datasets did not result in a difference between the centroids that were calculated by KMeans.

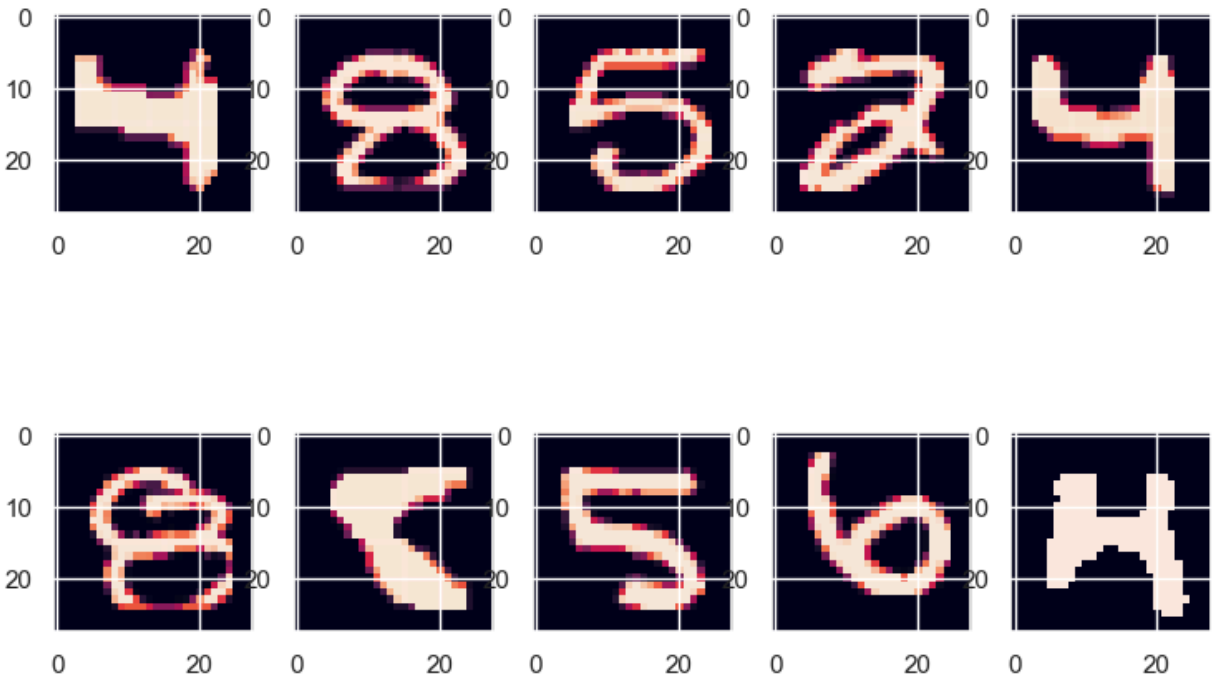
g) Create a matrix (let's call it `O`) that is the difference between the original dataset and the rank-10 approximation of the dataset. i.e. if the original dataset is `A` and the rank-10 approximation is `B`, then `O = A - B`

In [60]:
```
# Get the rank-10 approximation of the dataset.
scopy = s.copy()
scopy[10:] = 0.0
rank_10_X = u.dot(np.diag(scopy)).dot(vt)

O = X - rank_10_X
```

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 [67]:
```
# Find the indexes of the 10 largest rows in O.
Onorm = np.linalg.norm(O, axis=1)
anom_indices = np.argsort(Onorm)[-10:]
anomalous_data = X[anom_indices]

plt.figure(figsize=(9,6))

for i in range(len(centers)):
    # Calculate its subplot position.
    plt.subplot(2, 5, i + 1)
```

```
    # Translate it into image dimensions and plot.
    anom = anomalous_data[i]
    anom_image = np.reshape(anom, (28, 28))
    plt.imshow(anom_image)

plt.show()
```



The largest rows represent the data in X that deviate the most from its rank-10 approximation (i.e. it deviates a lot from its most significant singular vectors / patterns observed).