Worksheet 10

Name: Prathmesh Sonawane

UID: U39215370

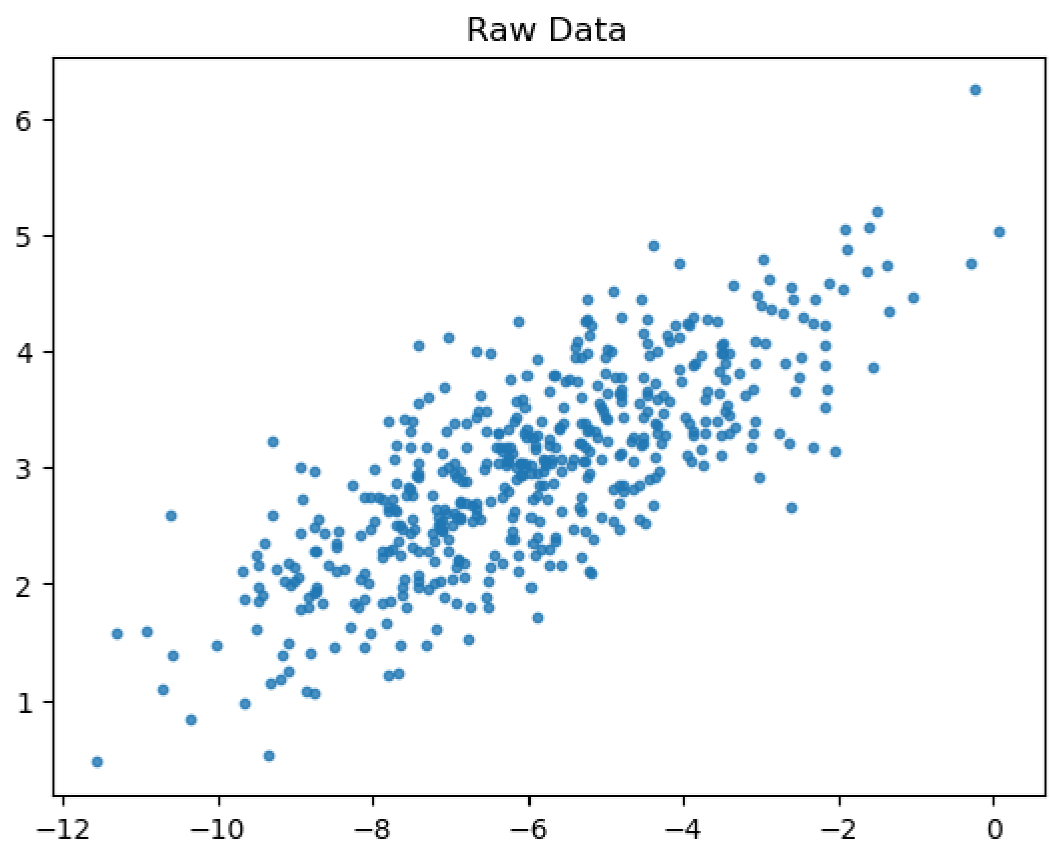
Topics

- Singular Value Decomposition

Feature Extraction

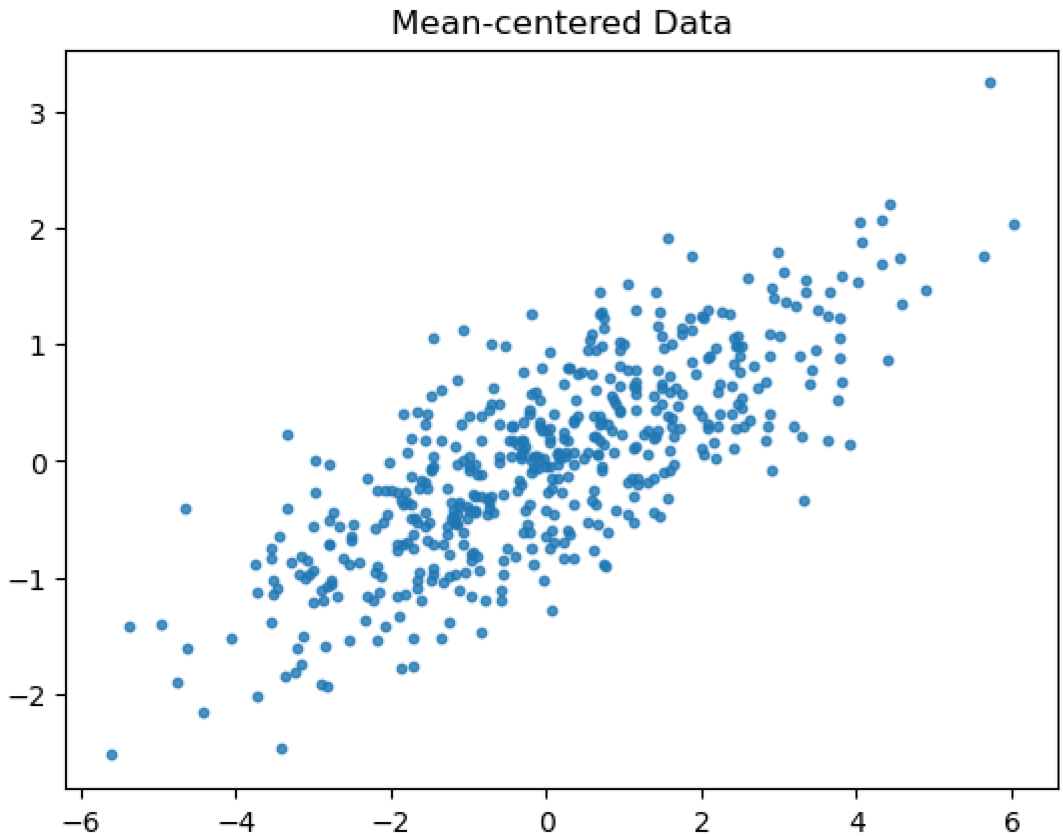
SVD finds features that are orthogonal. The Singular Values correspond to the importance of the feature or how much variance in the data it captures.

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()
```



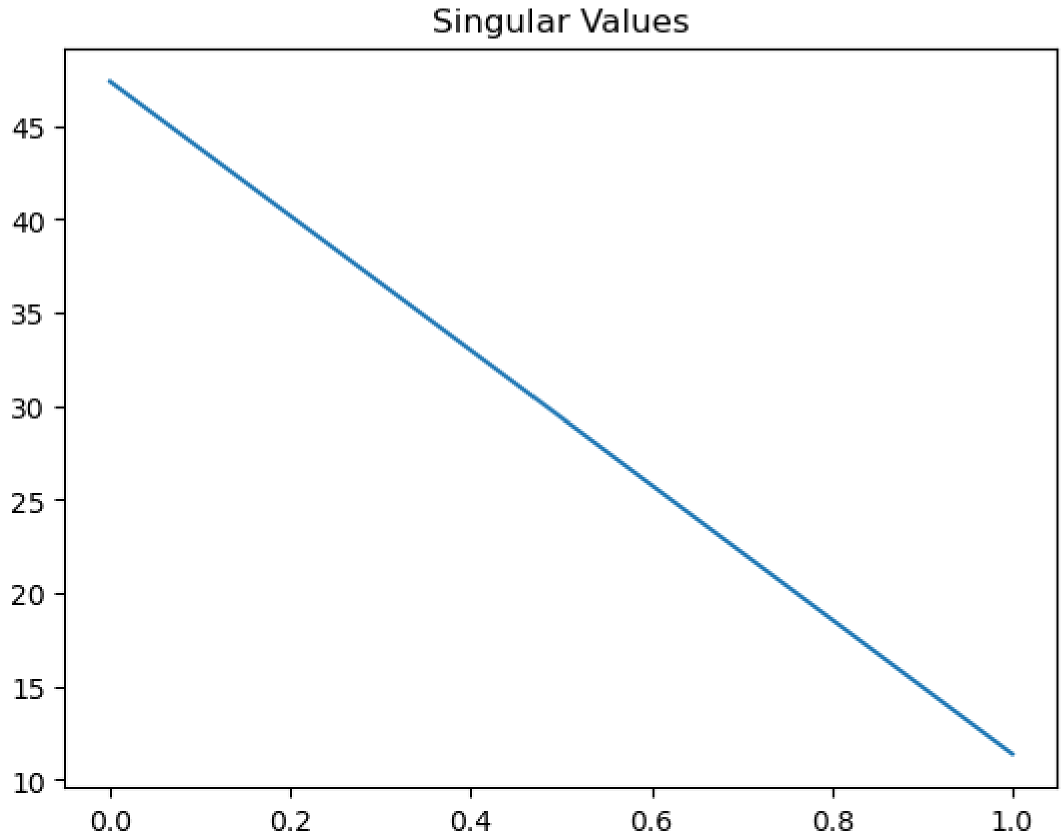
In [2]:

```
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()
```



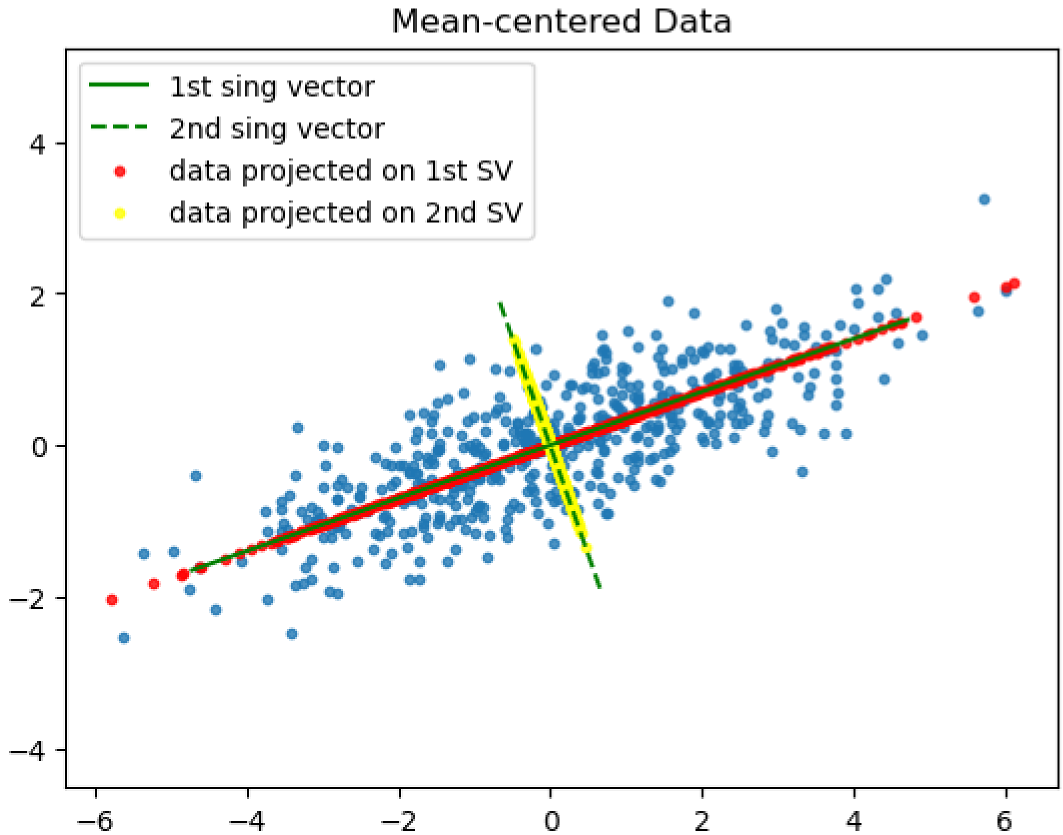
In [3]:

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

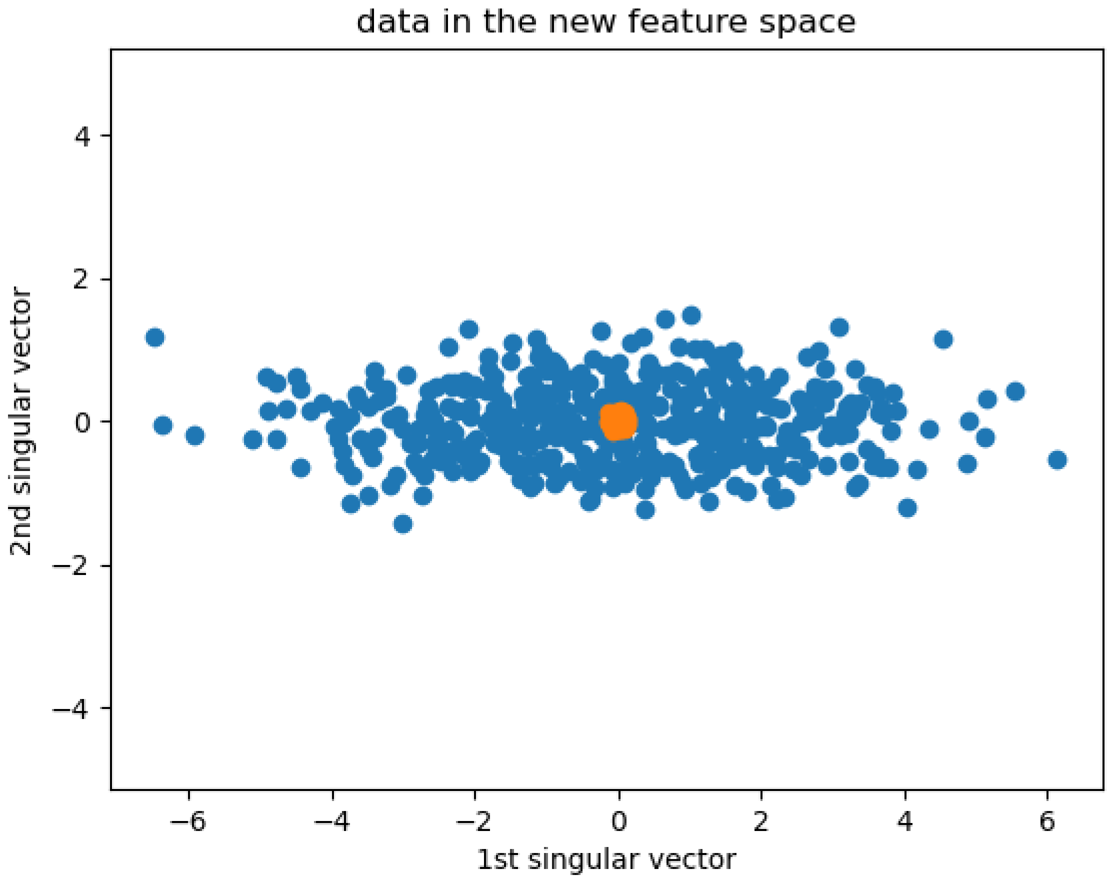


In [4]:

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

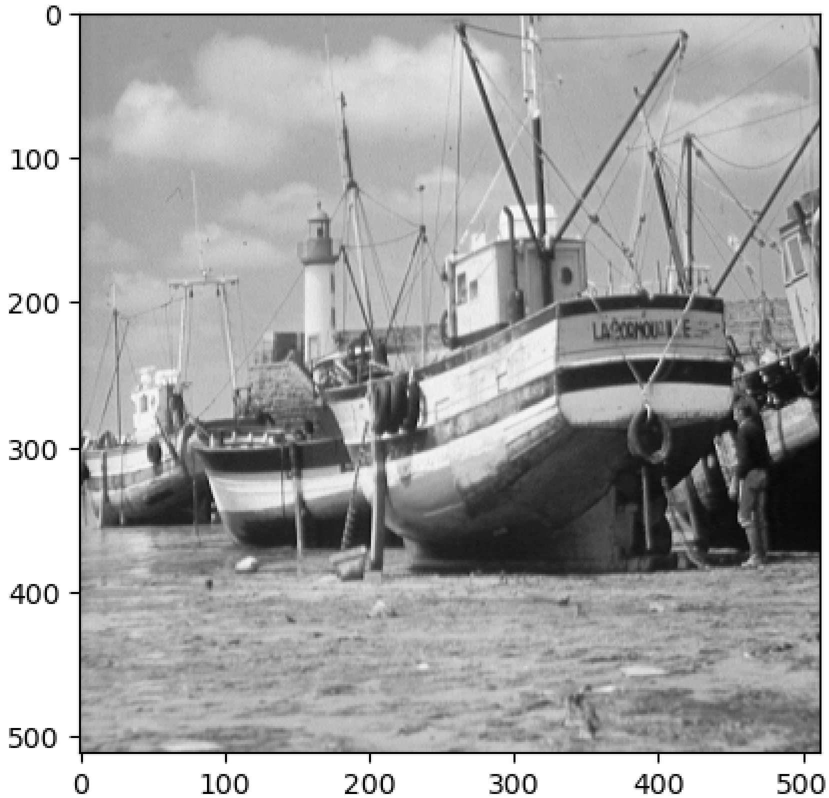


In [5]:
```
# show ouput from svd is the same
orthonormal_X = u
shifted_X = u.dot(np.diag(s))
plt.axis('equal')
plt.scatter(shifted_X[:,0], shifted_X[:,1])
plt.scatter(orthonormal_X[:,0], orthonormal_X[:,1])
plt.xlabel("1st singular vector")
plt.ylabel("2nd singular vector")
plt.title("data in the new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

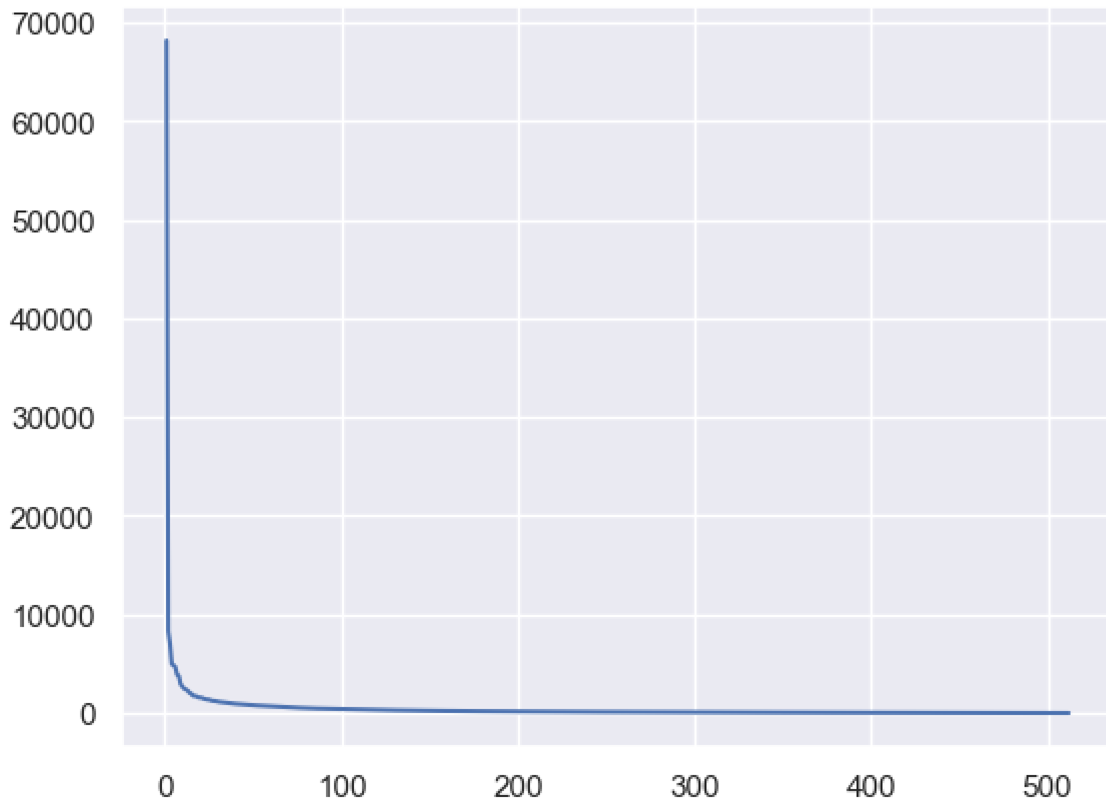
Out[6]:
```
<matplotlib.image.AxesImage at 0x10fd9b610>
```



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

In [15]:
```
u,s,vt=np.linalg.svd(boat,full_matrices=False)

plt.plot(range(1,len(s)+1),s)
```

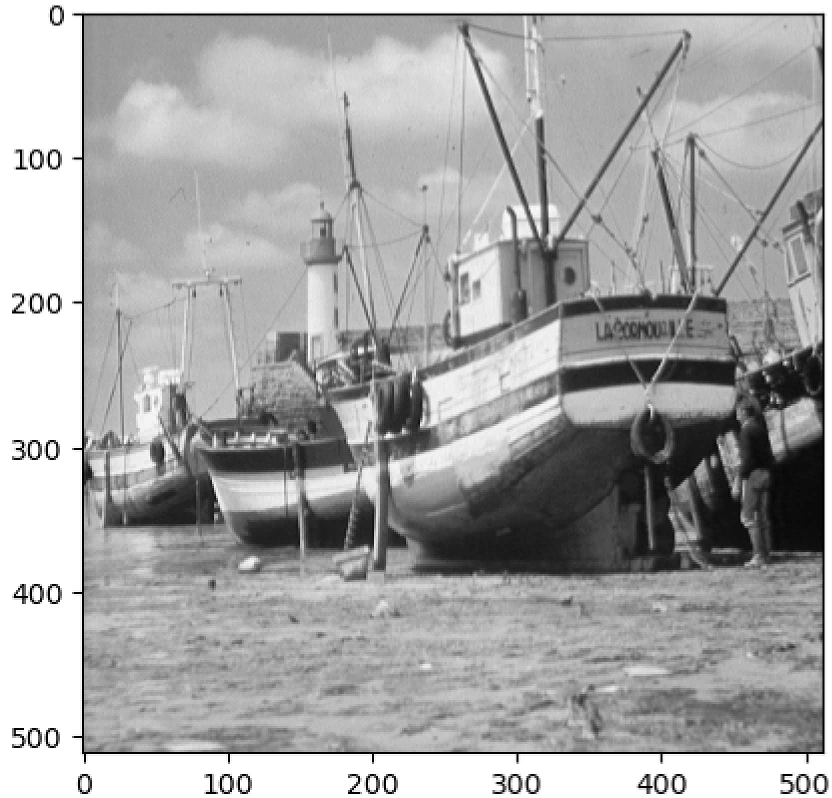
Out[15]:
```
[<matplotlib.lines.Line2D at 0x2894af750>]
```



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 0x10fc293d0>
```

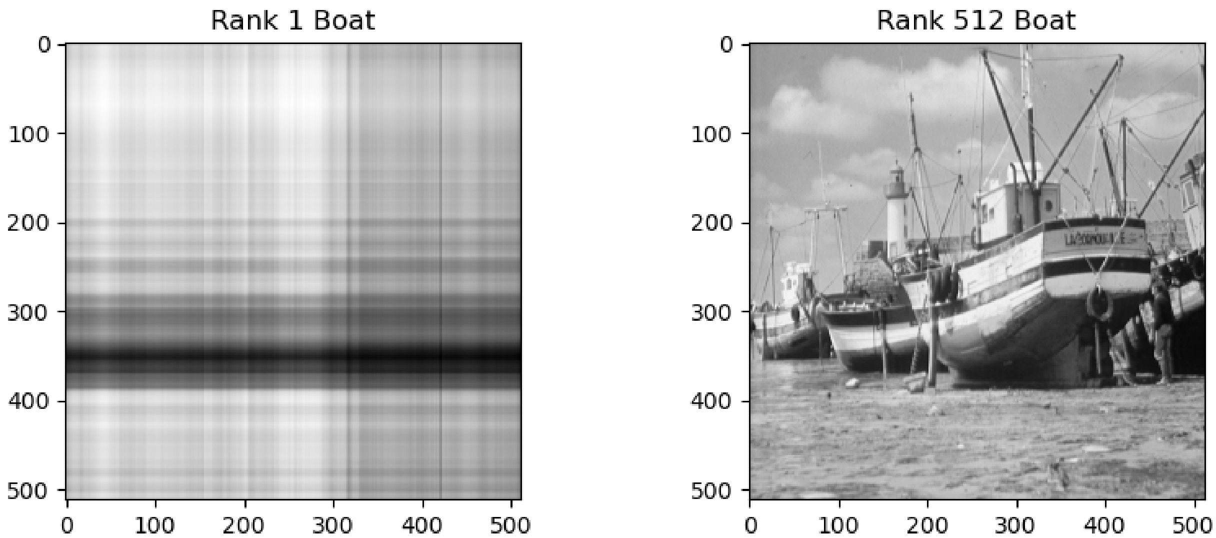


b) Create a new matrix `scopy` which is a copy of `s` with all but the first singular value set to 0.

In [9]:
```
scopy = s.copy()
scopy[1:] = 0.0
```

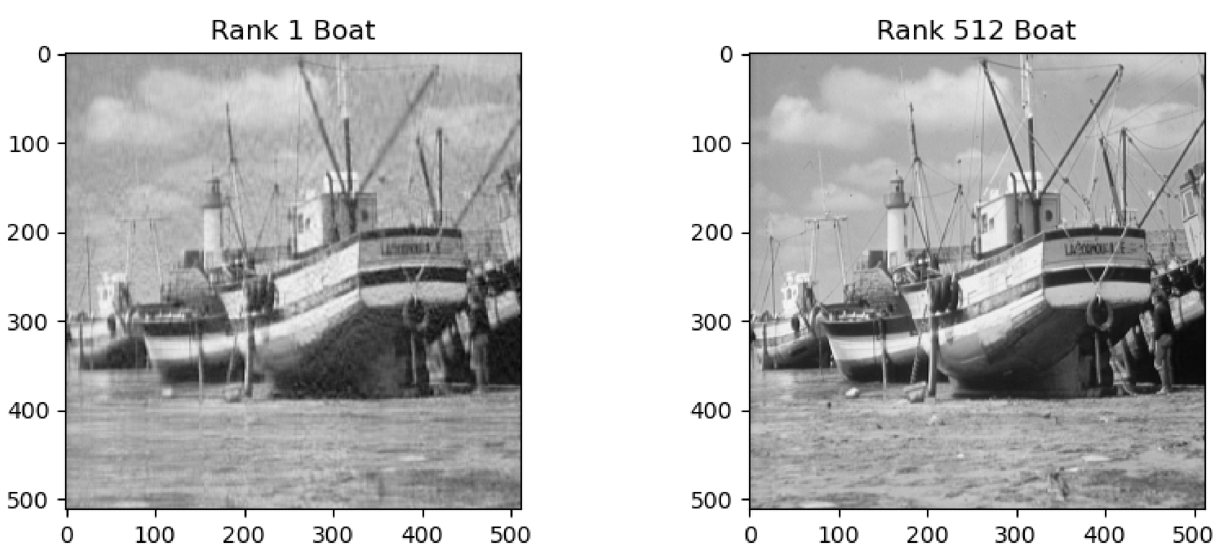
c) Create an approximation of the boat image by multiplying `u`, `scopy`, and `v` transpose. Plot them side by side.

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 [12]:
```
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 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()
```



Why you should care

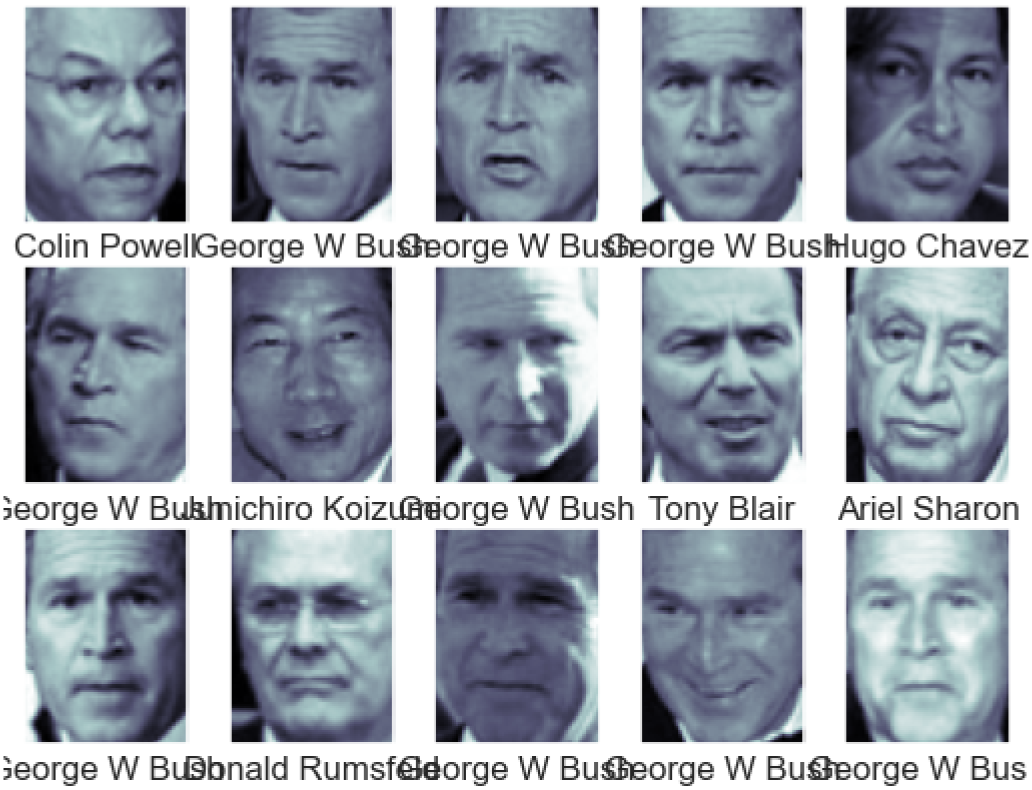
a) By using an approximation of the data, you can improve the performance of classification tasks since:

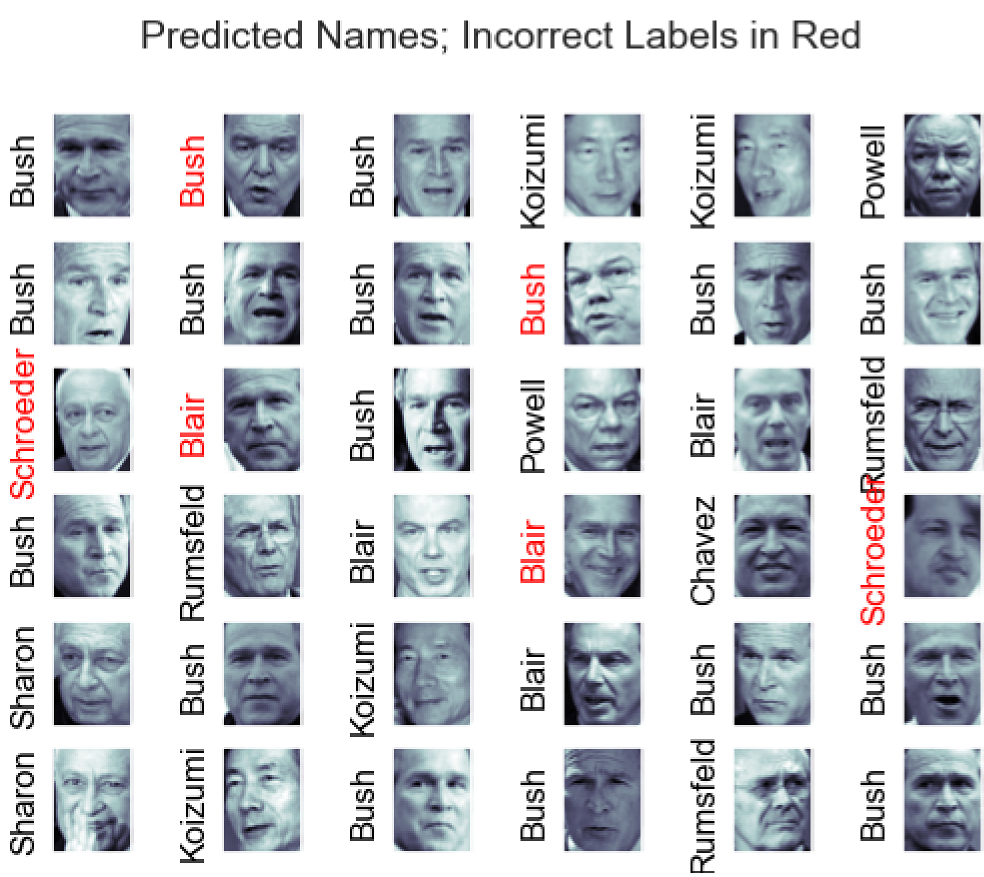
1. there is less noise interfering with classification
2. no relationship between features after SVD
3. the algorithm is sped up when reducing the dimension of the dataset

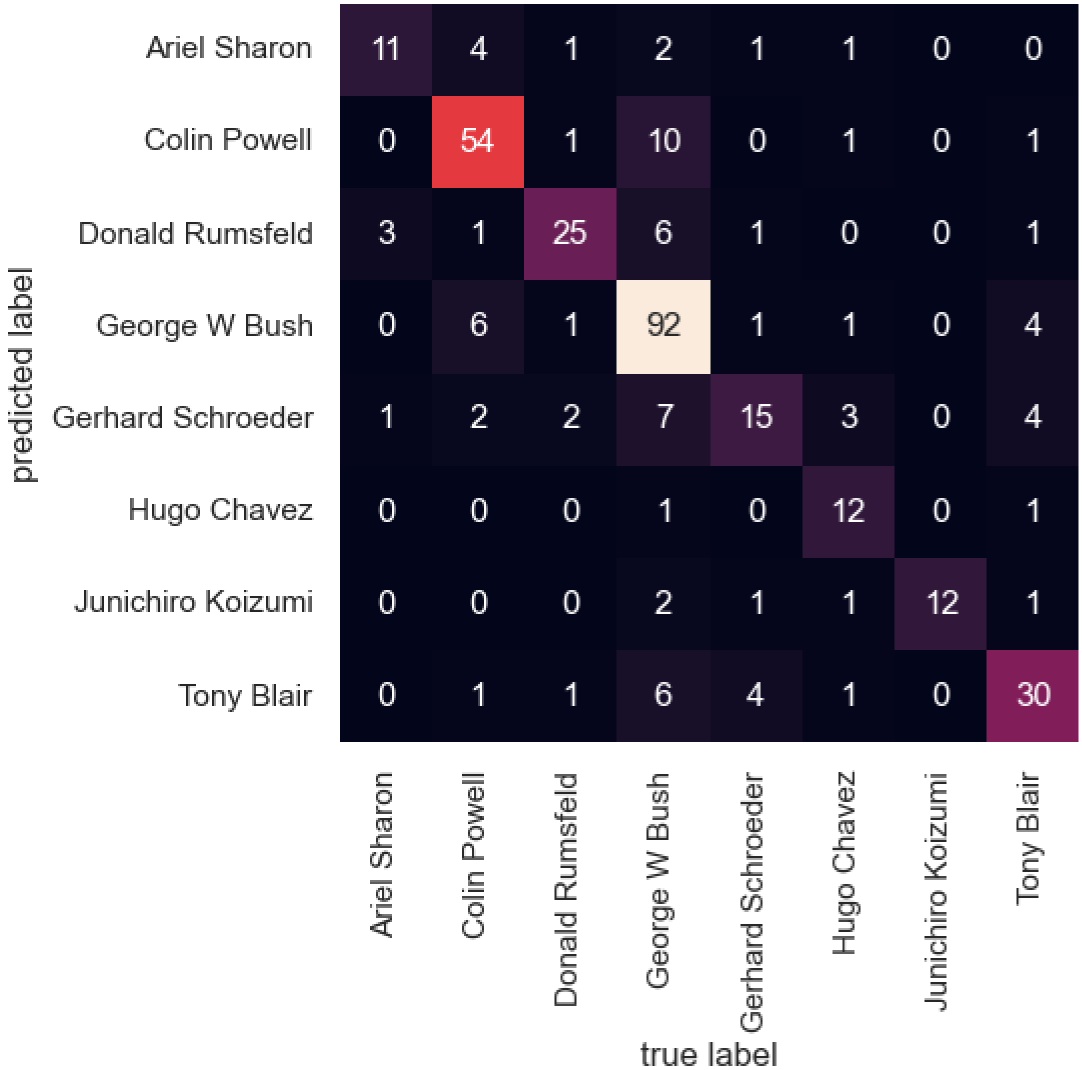
Below is some code to perform facial recognition on a dataset. Notice that, applied blindly, it does not perform well:

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, rand

# 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))
```







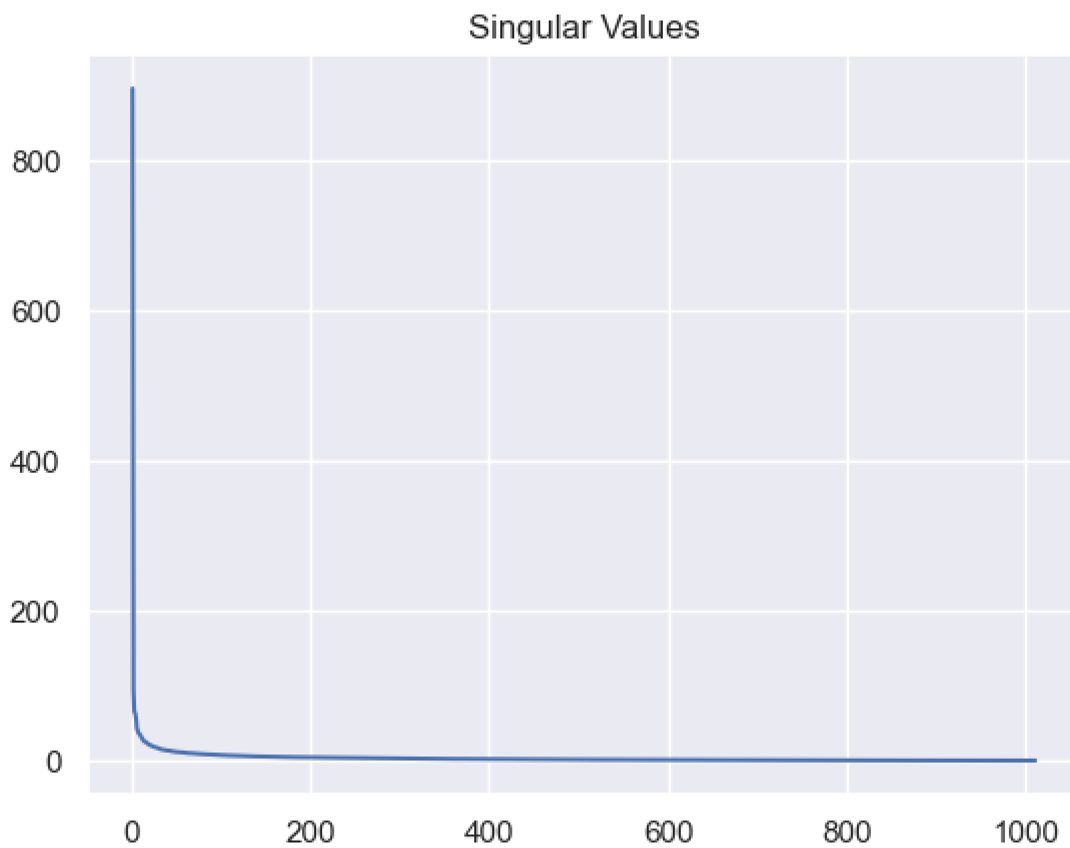

```
Accuracy =  0.744807121661721
```

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

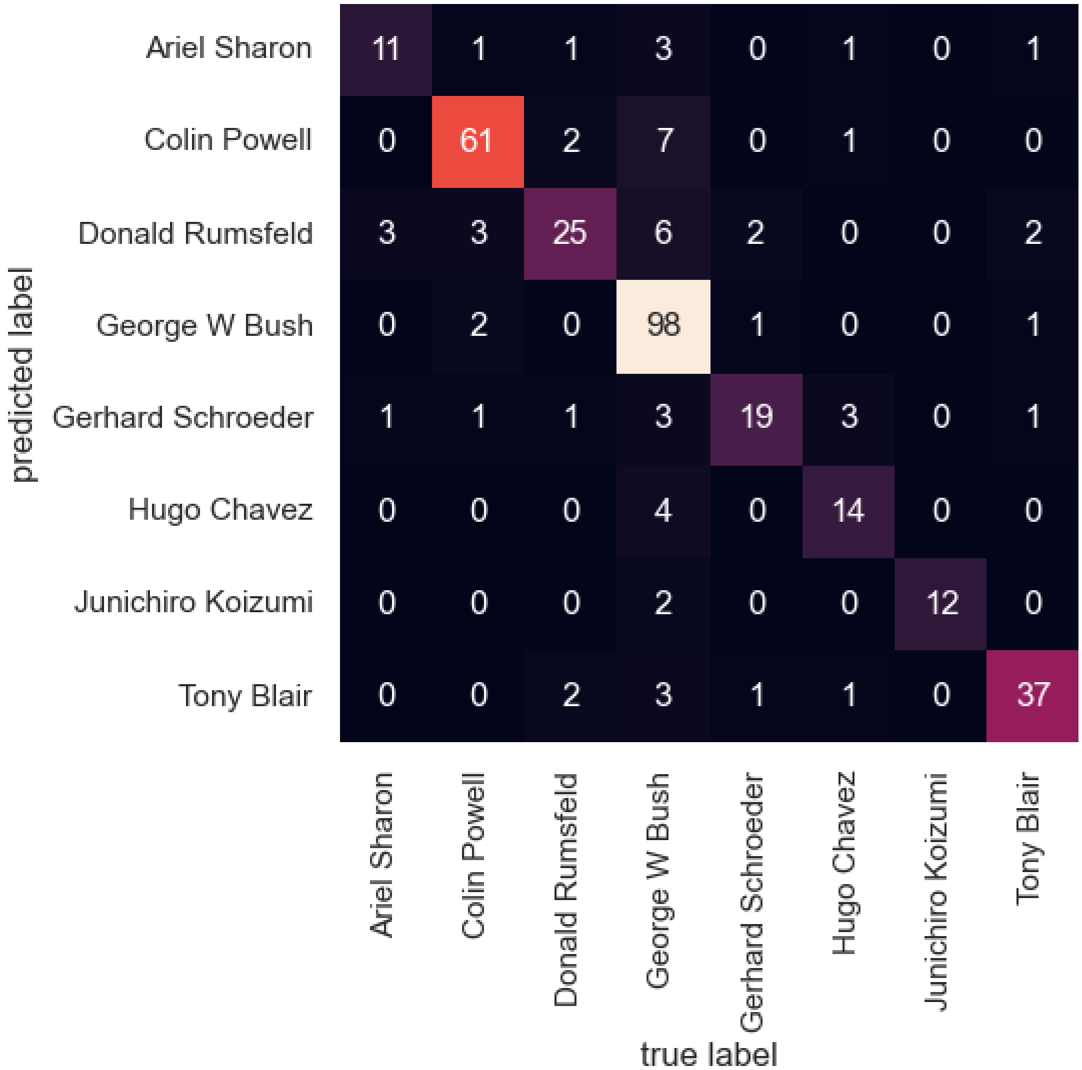
In [16]:

```
# 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=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
Bush
Schroeder
Bush
Koizumi
Koizumi
Powell
Bush
Bush
Bush
Powell
Bush
Bush
Schroeder
Bush
Rumsfeld
Powell
Blair
Rumsfeld
Bush
Rumsfeld
Blair
Bush
Chavez
Chavez
Sharon
Bush
Koizumi
Blair
Bush
Bush
Sharon
Koizumi
Bush
Rumsfeld
Rumsfeld
Bush




```
Accuracy =  0.8219584569732937
```

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).

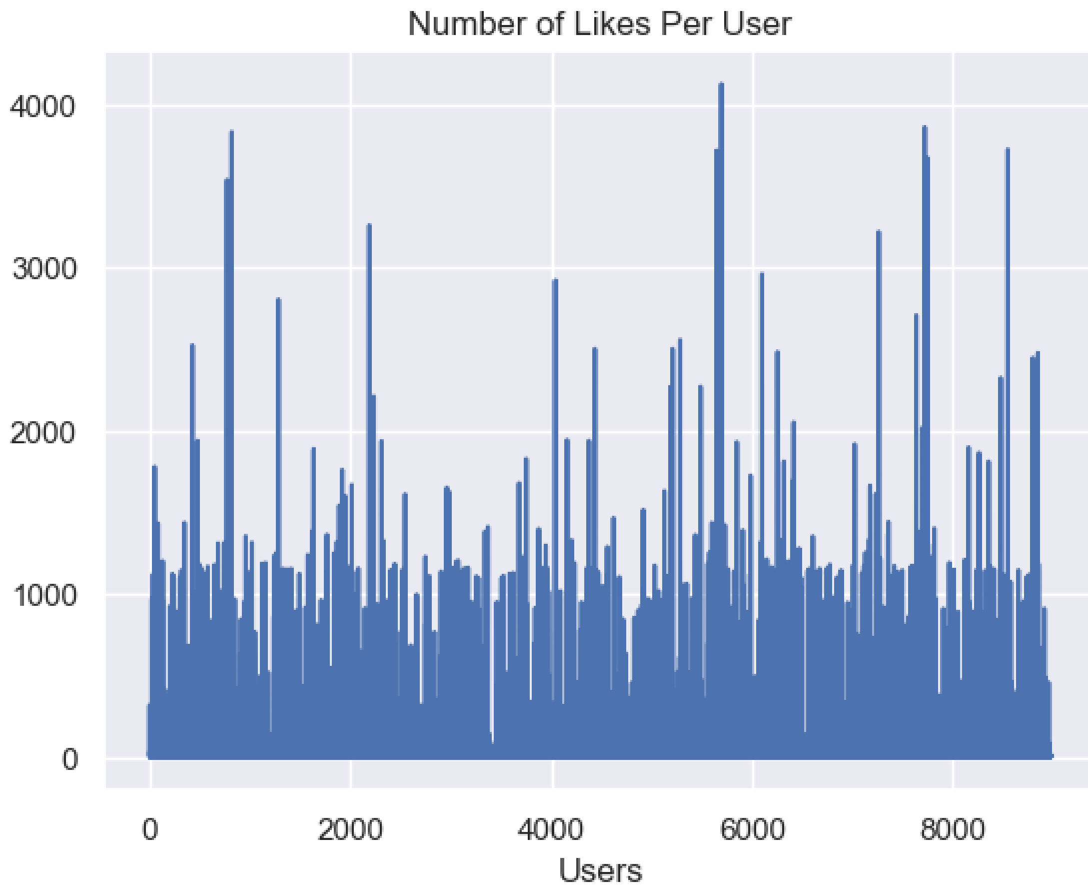
b) SVD can be used for anomaly detection.

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 [19]:

```
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()
```

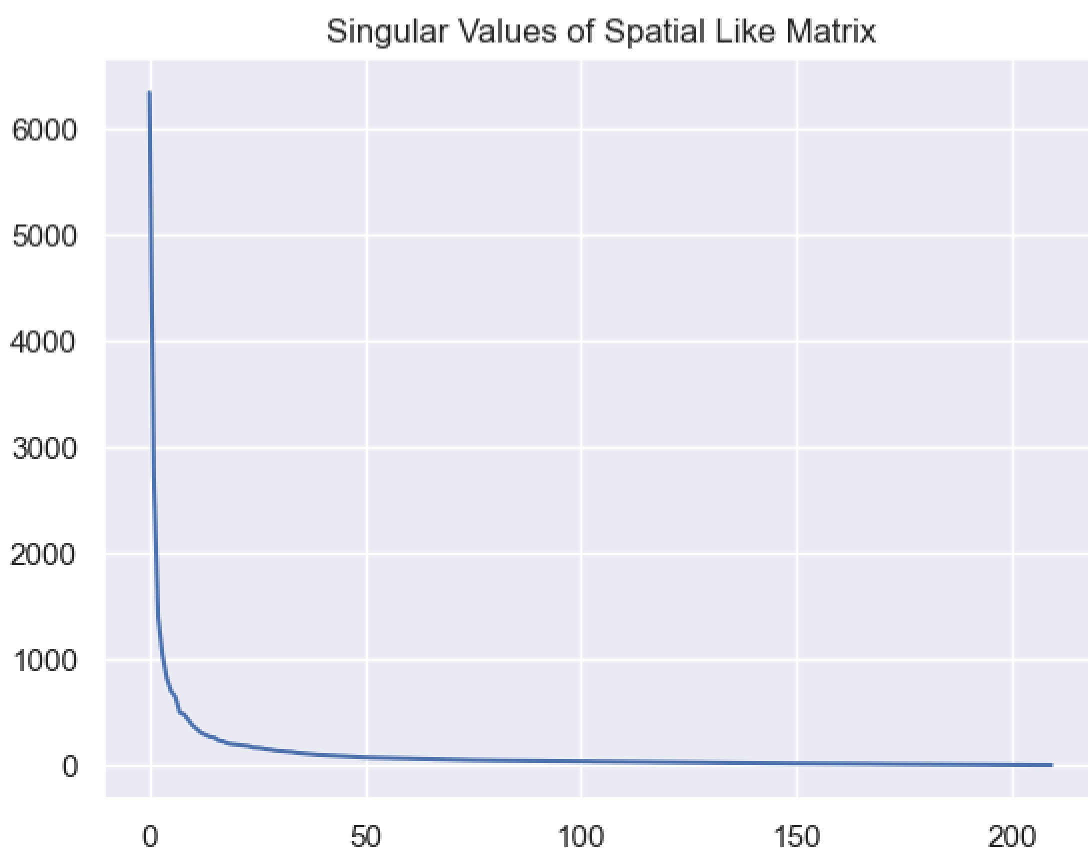
```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 2. 8.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

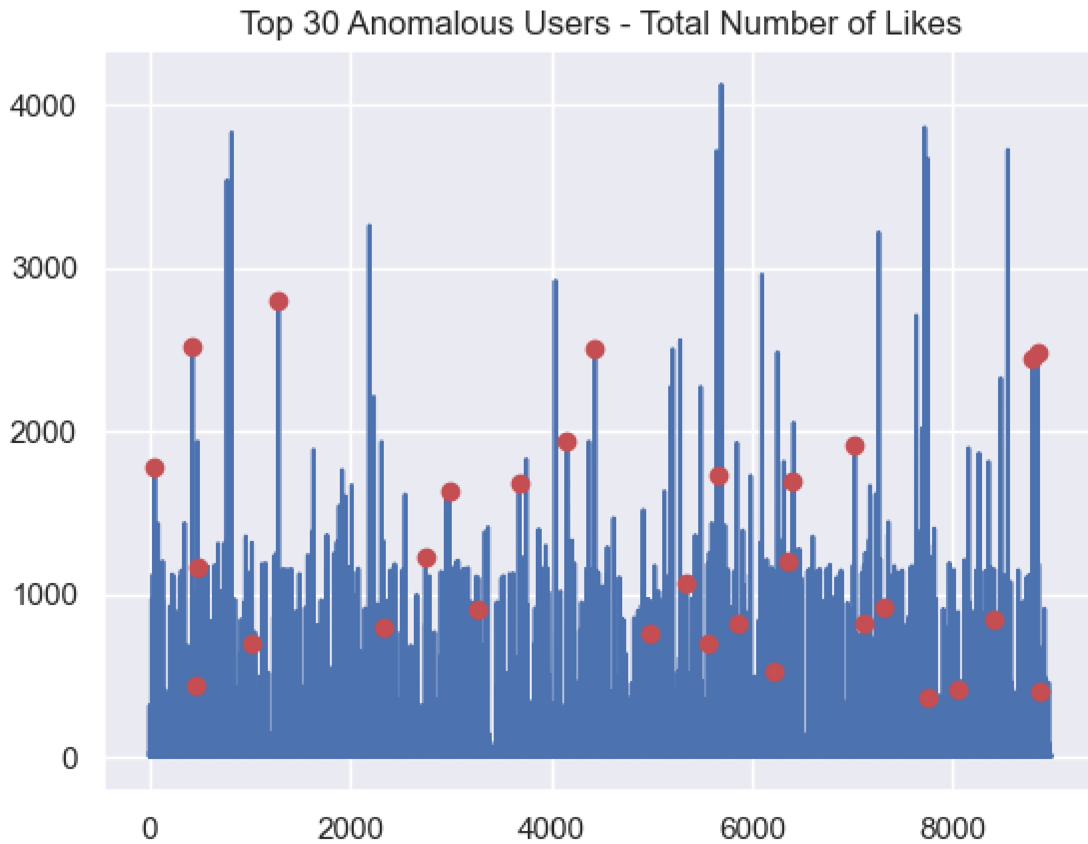


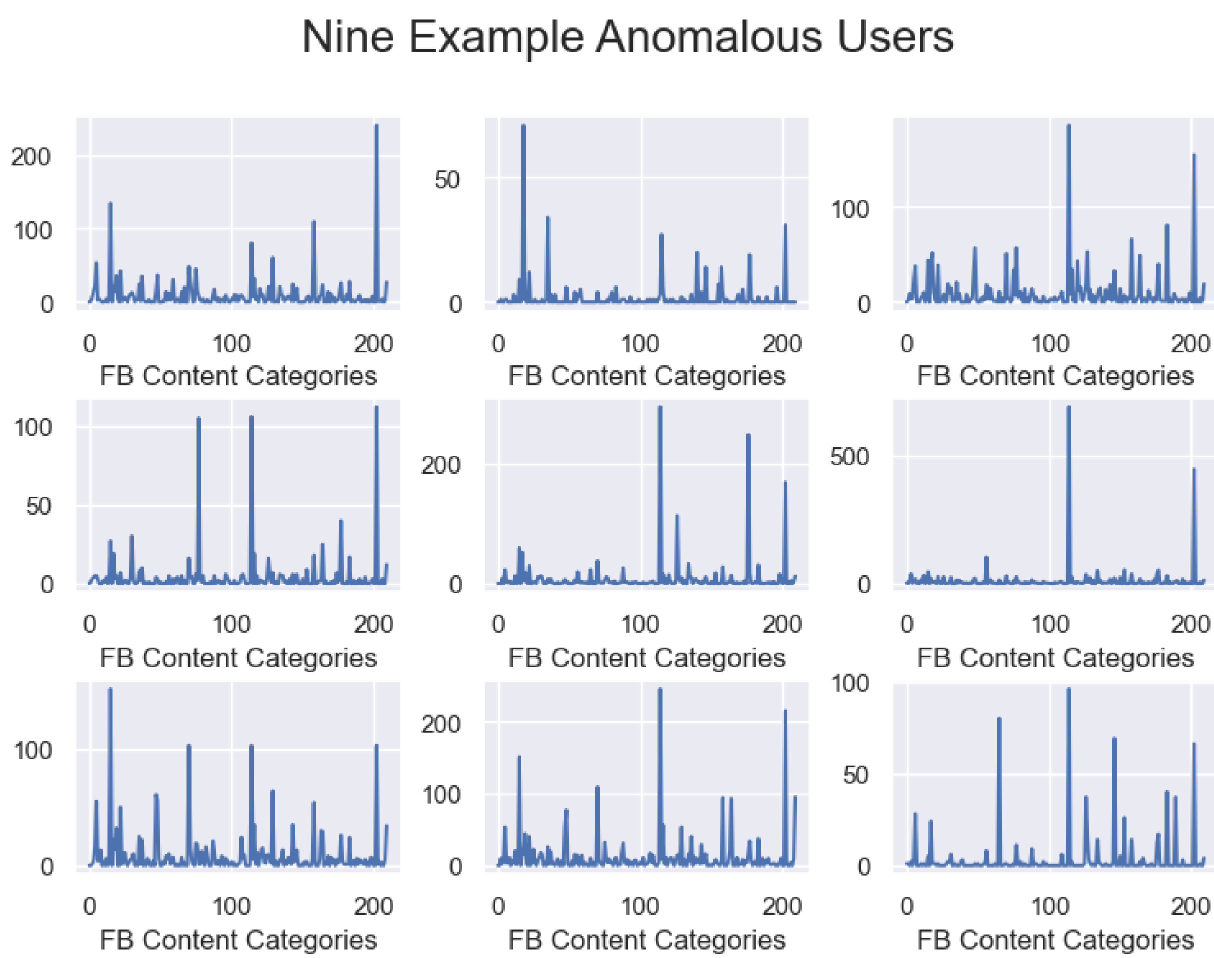
How users distribute likes across categories follows a general pattern that most users follow. This behavior can be captured using few singular vectors. And anomalous users can be easily identified.

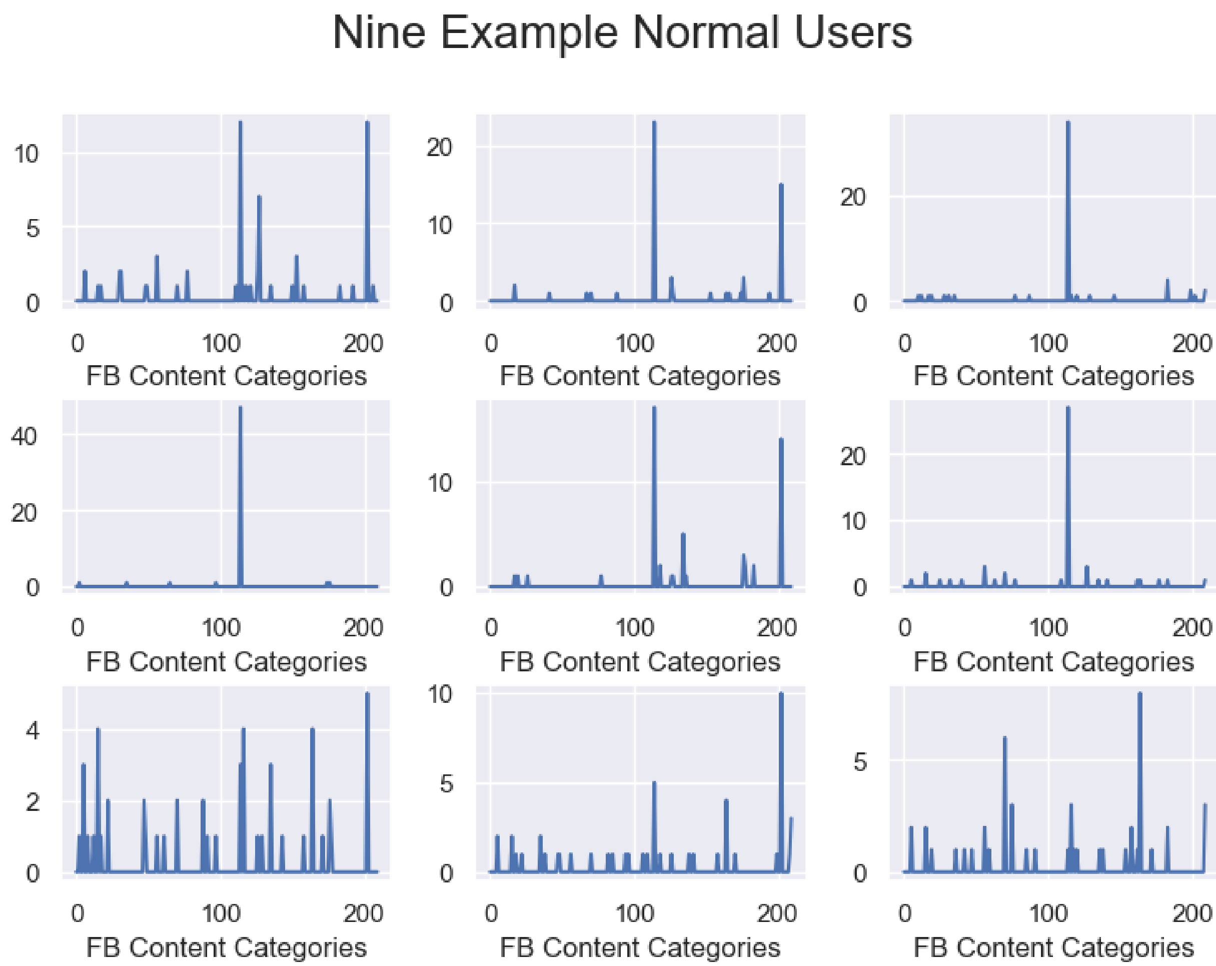
In [21]:
```
u,s,vt = np.linalg.svd(FBSpatial,full_matrices=False)
plt.plot(s)
_ = plt.title('Singular Values of Spatial Like Matrix')
plt.show()

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



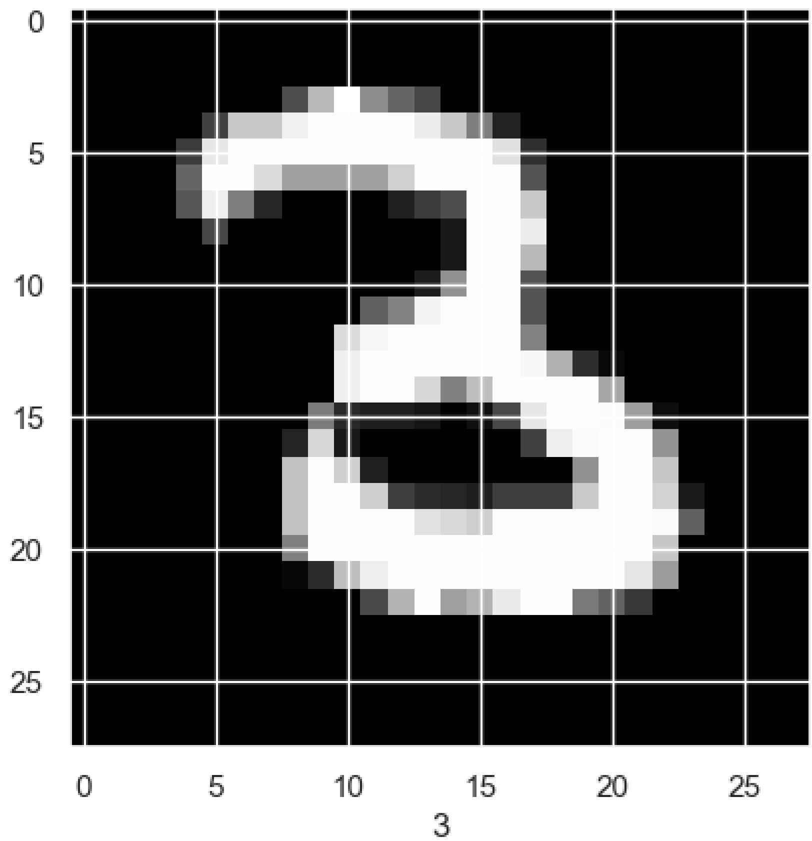
Challenge Problem

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

In [26]:

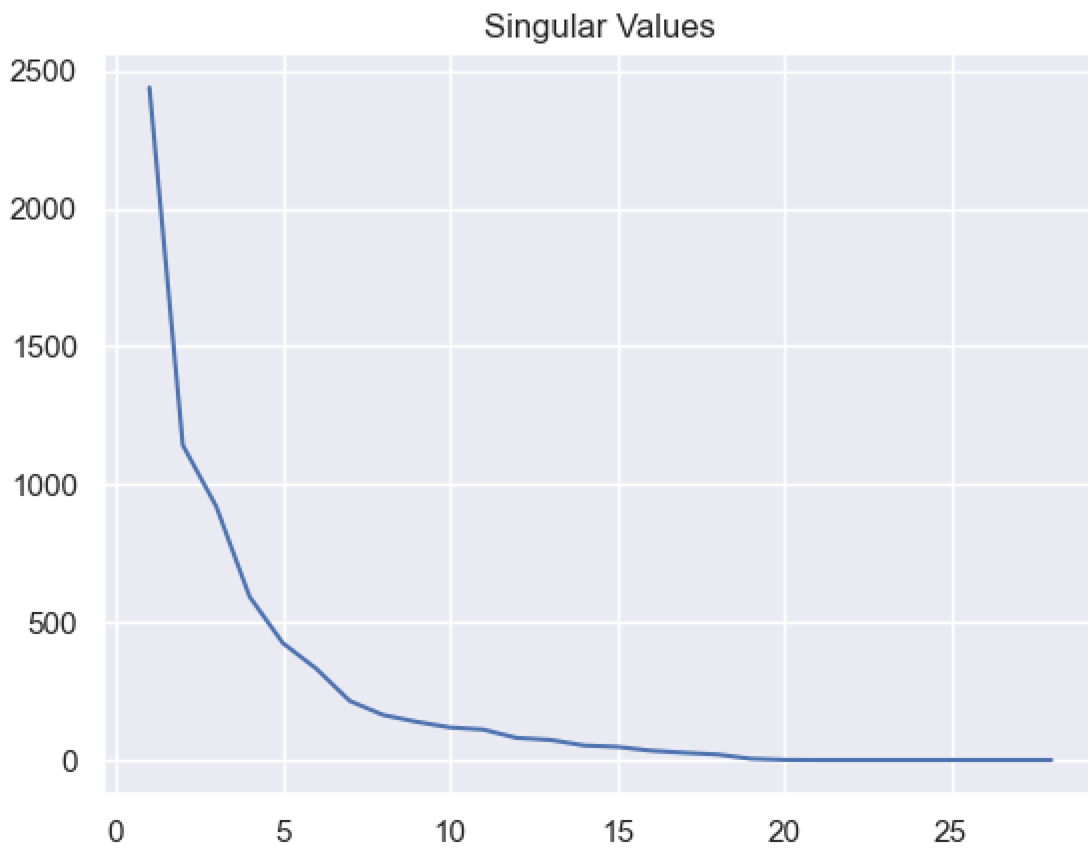
```
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml

X, y = fetch_openml(name="mnist_784", version=1, return_X_y=True, as_frame=Fals

# your code here

print(X.shape, y.shape)

idx = np.random.randint(0, len(X))

image = X[idx].reshape(28, 28)
plt.imshow(image, cmap="gray")
plt.xlabel(y[idx])
plt.show()
```

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



b) Plot its singular value plot.

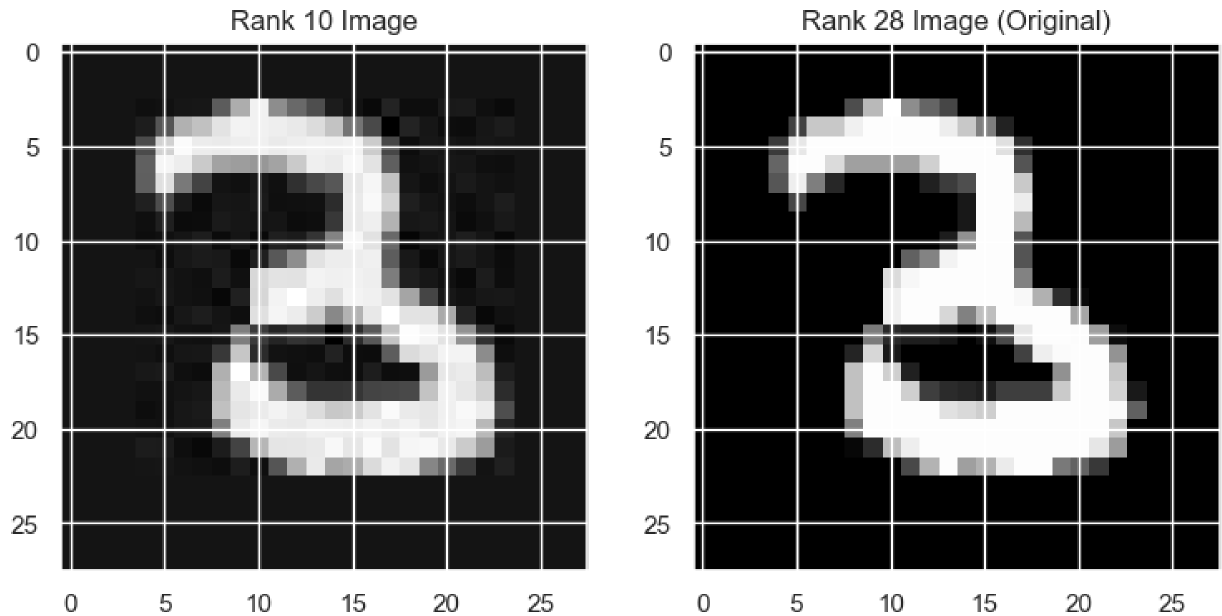
In [27]:
```
u,s,vt=np.linalg.svd(image,full_matrices=False)

plt.plot(range(1,len(s)+1),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

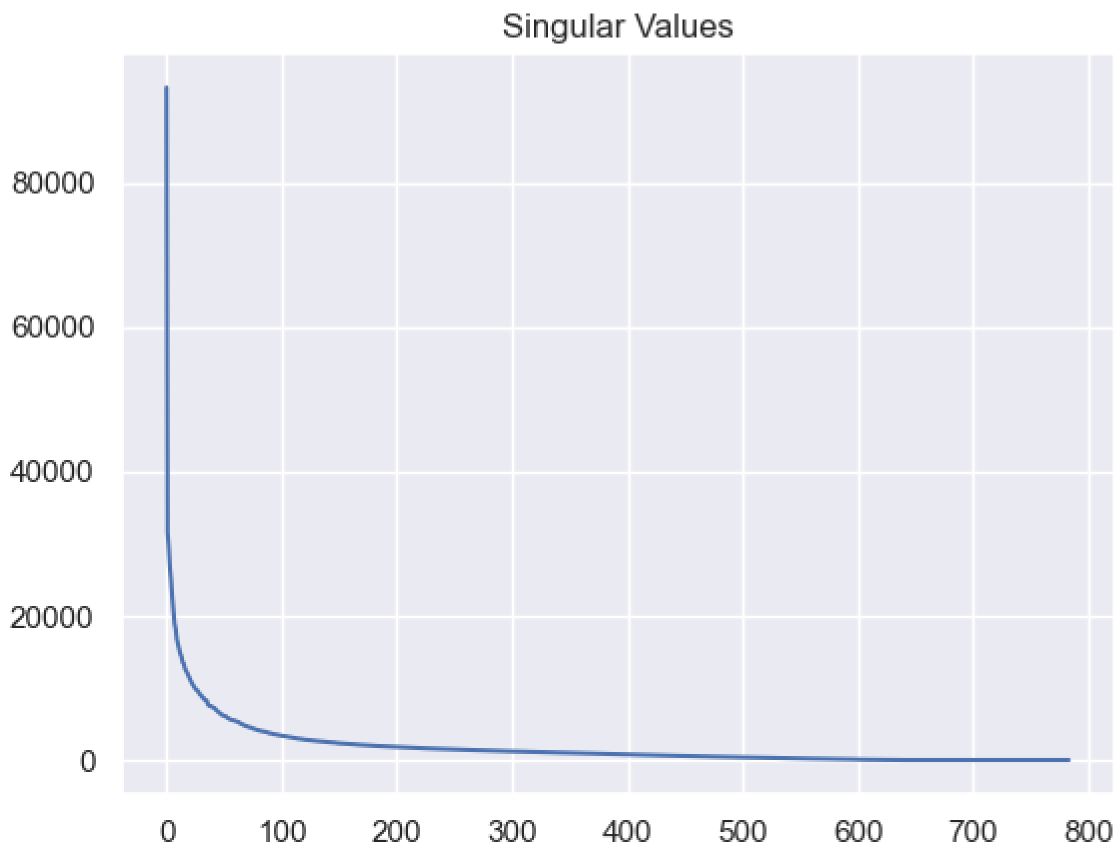
In [32]:
```
copy = s.copy()
#print(len(copy))
copy[10:] = 0.0
image_app = u.dot(np.diag(copy)).dot(vt)

plt.figure(figsize=(9,6))
plt.subplot(1,2,1)
plt.imshow(image_app, cmap='gray')
plt.title('Rank 10 Image')
plt.subplot(1,2,2)
plt.imshow(image, cmap='gray')
plt.title('Rank 28 Image (Original)')
plt.show()
```

28



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]:
```
from sklearn.model_selection import train_test_split

# take a small random sample, e.g., 5% of the data
X_sample, _, y_sample, _ = train_test_split(X, y, test_size=0.95, random_state=

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

plt.plot(s)
plt.title("Singular Values")
plt.show()
```

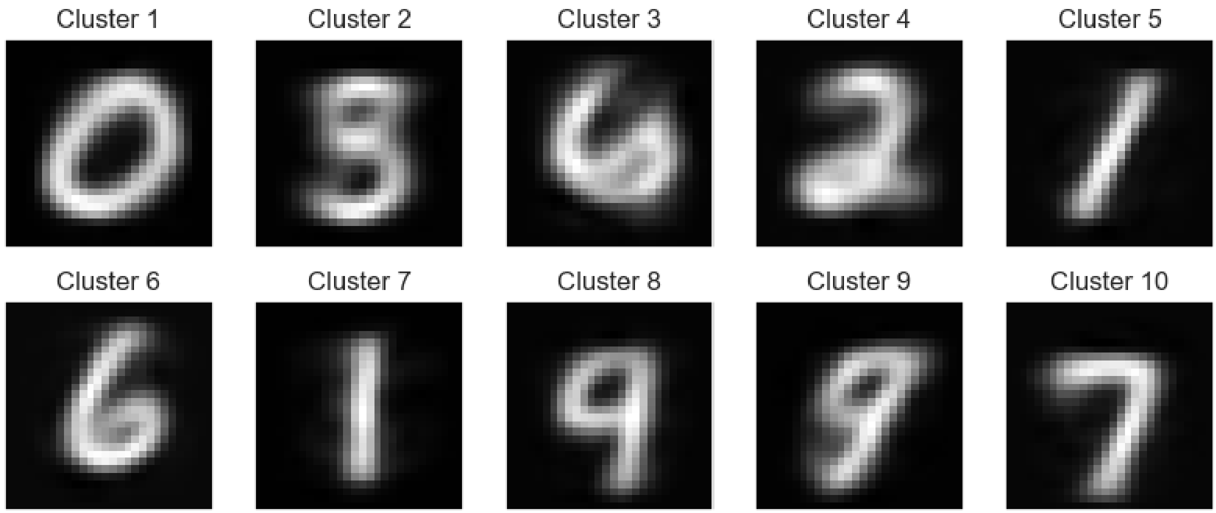


I would choose a rank close to 60 since that is where the singular values start to decline in vertical change. This means that after 60 singular values, incorporating the rest will not bring us much added information to be worthwhile to include it in our program due to runtime and space costs.

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 [38]:

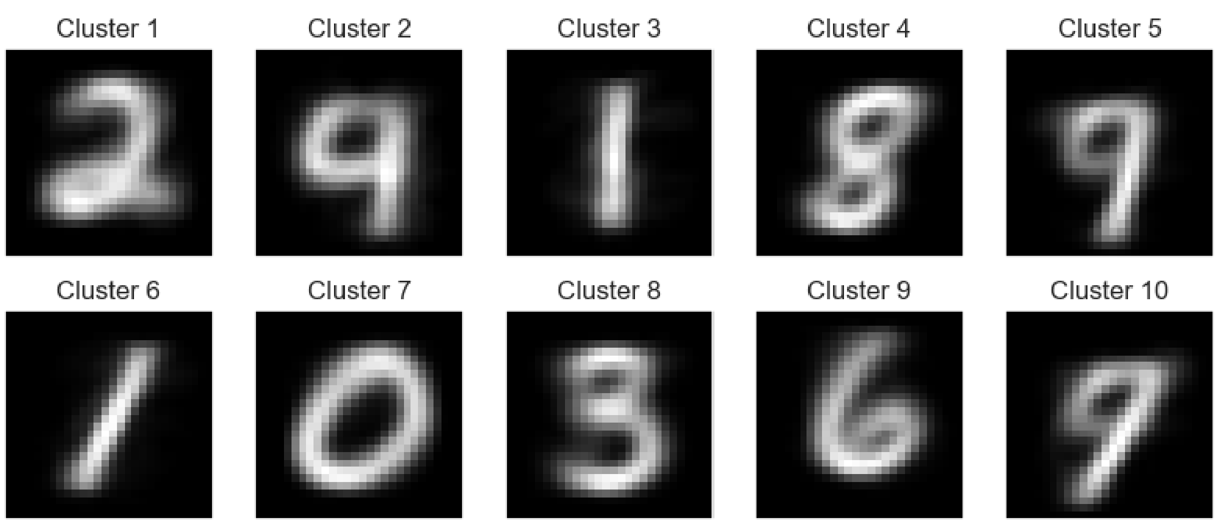
```
from sklearn.cluster import KMeans

scopy = np.copy(s)
scopy[50:] = 0.0

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

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

centers = kmeans.cluster_centers_.reshape(10, 28, 28)
fig, ax = plt.subplots(2, 5, figsize=(10, 4))
cnt = 0
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[], title=f'Cluster {cnt+1}')
    cnt += 1
    axi.imshow(center, cmap='gray')
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 [39]:

```
kmeans_origin = KMeans(n_clusters=10, init='k-means++', n_init='auto')
kmeans_origin.fit_predict(X_sample)

centers = kmeans_origin.cluster_centers_.reshape(10, 28, 28)
fig, ax = plt.subplots(2, 5, figsize=(10, 4))
cnt = 0
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[], title=f'Cluster {cnt+1}')
    cnt += 1
    axi.imshow(center, cmap='gray')
plt.show()
```



For the most part, there is not a great difference between the images created by the original dataset and the dataset undergoing SVD conversion. However, for some numbers like 9, the original dataset seems more crisp and readable compared to our SVD-version.

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`

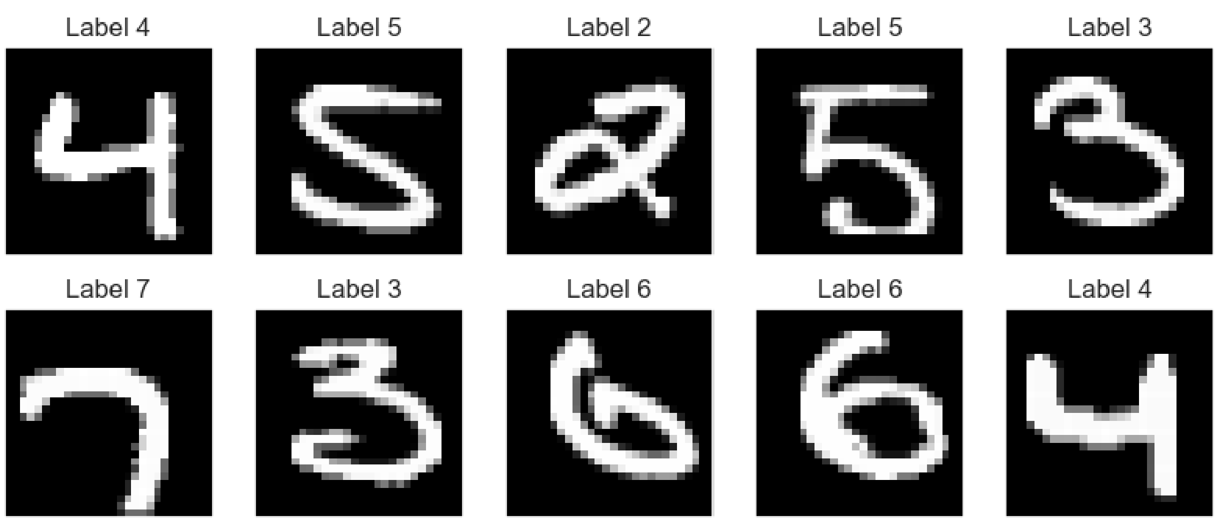
In [40]:
```
u, s, vt = np.linalg.svd(X_sample, full_matrices=False)
scopy = np.copy(s)
scopy[10:] = 0.0
B = u.dot(np.diag(scopy)).dot(vt)
A = X_sample
print(A.shape, B.shape)
O = A - B
print(O.shape)
```

```
(3500, 784) (3500, 784)
(3500, 784)
```

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 [41]:
```
diff = np.linalg.norm(O, axis=1)
print(diff.shape)
indices = np.argsort(diff)[-10:]    # ascending
images = A[indices]

fig, axes = plt.subplots(2, 5, figsize=(10, 4))
for i, ax in enumerate(axes.flatten()):
    ax.set(xticks=[], yticks=[], title=f'Label {y_sample[indices[i]]}')
    ax.imshow(images[i].reshape(28, 28), cmap='gray')
plt.show()
```

```
(3500,)
```



Rows that have the largest distances represent points that are strange, or ones that can not be accurately approximated by SVD. These points are the ones that differ significantly from the overall pattern of the data.