1)

a) $X = \sum_{i=1}^{k} \sigma_i u_i v_i^T$

$X = U \Sigma V^T$

$X^T = (U \Sigma V^T)^T \Rightarrow X^T = V \Sigma^T U^T$

$XX^T = (U \Sigma V^T)(U \Sigma V^T)^T = U \Sigma V^T V \Sigma^T U^T \Rightarrow XX^T = U \Sigma \Sigma^T U^T$

$X^T X = (U \Sigma V^T)^T (U \Sigma V^T) = V \Sigma^T U^T U \Sigma V^T \Rightarrow X^T X = V \Sigma^T \Sigma V^T$

b)

i) $Xw = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} 3w_1 \\ w_2 \\ 0 \\ 0 \end{pmatrix} \approx \begin{pmatrix} 6 \\ 2 \\ 1 \\ 2 \end{pmatrix}$ $\quad 3w_1 = 6 \Rightarrow w_1 = 2, w_2 = 2$

$\|y - Xw\|$ is minimized when $w = \begin{pmatrix} 2 \\ 2 \\ a \\ b \end{pmatrix}$ where $a, b \in \mathbb{R}$

$\|w\|$ is minimized when $a = b = 0 \Rightarrow w_{min} = \begin{pmatrix} 2 \\ 2 \\ 0 \\ 0 \end{pmatrix}$

ii) $Xw = U \Sigma V^T w = 0$

We claim that columns $k+1$ to d of V provide a basis for w that satisfy $Xw = 0$. Intuitively, only the first k columns of V remain when multiplied with $U\Sigma$, so taking any of the $k+1$ to d cols and multiplying it with $U\Sigma V^T$ gives 0 because V is orthogonal.

More formally, we claim that $w = \sum_{i=k+1}^{d} w_i v_i$.

$Xw = X \sum_{i=k+1}^{d} w_i v_i = \sum_{i=k+1}^{d} w_i (X v_i) = \sum_{i=k+1}^{d} w_i U \Sigma V^T v_i = \sum_{i=k+1}^{d} w_i U \Sigma e_i$

However, $\sigma_i = 0$ for $i > k$, so $\Sigma e_i = 0$ above. Thus, $\sum_{i=k+1}^{d} w_i U (\Sigma e_i) = \sum_{i=k+1}^{d} w_i U (0) = 0$.

Another way to consider this is $X = U \Sigma V^T$, so $XV = U \Sigma V^T V = U \Sigma$.

$\Rightarrow [Xv_1 \dots Xv_d] = [U\sigma_1 \dots U\sigma_d]$

Since $\sigma_i = 0$ for $i > k$, only the first k columns of $U\Sigma$ are non-zero. Therefore, $U\sigma_{k+1} = \dots = U\sigma_d = 0$, so $Xv_{k+1} = \dots = Xv_d = 0$. This means $v_{k+1}, \dots, v_d$ form a basis for w.

iii) $\tilde{y} = U^T y$, $\tilde{w} = V^T w$

$U^T U = U U^T = I \Rightarrow U^T = U^{-1}$

$\tilde{y} = U^T y \Rightarrow U\tilde{y} = UU^T y \Rightarrow y = U\tilde{y}$

$$\begin{aligned}
\|y - Xw\|_2^2 &= \|U\tilde{y} - U\Sigma V^T w\|_2^2 \\
&= \|U\tilde{y} - U\Sigma\tilde{w}\|_2^2 \\
&= \|U(\tilde{y} - \Sigma\tilde{w})\|_2^2 \\
&= (U(\tilde{y} - \Sigma\tilde{w}))^T (U(\tilde{y} - \Sigma\tilde{w})) \\
&= (\tilde{y} - \Sigma\tilde{w})^T U^T U (\tilde{y} - \Sigma\tilde{w}) \\
&= (\tilde{y} - \Sigma\tilde{w})^T (\tilde{y} - \Sigma\tilde{w}) \\
&= \|\tilde{y} - \Sigma\tilde{w}\|_2^2
\end{aligned}$$

$$\Rightarrow \boxed{\|y - Xw\|_2^2 = \|\tilde{y} - \Sigma\tilde{w}\|_2^2}$$

$\|y - Xw\|_2^2$ is the minimum when $\Sigma\tilde{w} = \tilde{y}$. We know that because Σ is a diagonal matrix, $\tilde{y}_i = \sigma_i \tilde{w}_i$, but $\sigma_i = 0$ for $i > k$, so $\boxed{\tilde{w}_i = \frac{\tilde{y}_i}{\sigma_i} \text{ where } 1 \le i \le k}$ minimizes the error.

Thus, $\tilde{w}$ has the smallest norm when $\boxed{\tilde{w}_i = 0 \text{ for } i > k \text{ and } \tilde{w}_i = \frac{\tilde{y}_i}{\sigma_i} \text{ otherwise}}$, which is the same as letting $\boxed{\tilde{w} = \Sigma^+ \tilde{y}}$

iv) Since $\tilde{w} = \Sigma^+ \tilde{y}$ and $\tilde{y} = U^T y$, $\tilde{w} = \Sigma^+ U^T y$.

Then, $\tilde{w} = V^T w$, so $V^T w = \Sigma^+ U^T y \Rightarrow VV^T w = V\Sigma^+ U^T y \Rightarrow \boxed{w = V\Sigma^+ U^T y}$

c) code attached

2)

a) $M=\begin{pmatrix}0&0&0&1\\1&0&0&0\\0&1&0&0\\0&1&1&0\end{pmatrix}$, $A=\begin{pmatrix}0&0&0&1\\1&0&0&0\\0&\frac{1}{2}&0&0\\0&\frac{1}{2}&1&0\end{pmatrix}$

b) code attached

c) code attached

d) $M=\begin{pmatrix}1&1&\dots&1\\1&1&\dots&1\\0&0&\dots&0\\\vdots&\vdots&&\vdots\\0&0&\dots&0\end{pmatrix}$ where $M\in\mathbb{R}^{n\times n}$, $M_{ij}=1$ if $i=1$ or $i=2$ and 0 otherwise

$A=\begin{pmatrix}\frac{1}{2}&\frac{1}{2}&\dots&\frac{1}{2}\\\frac{1}{2}&\frac{1}{2}&\dots&\frac{1}{2}\\0&0&\dots&0\\\vdots&\vdots&&\vdots\\0&0&\dots&0\end{pmatrix}\Rightarrow G=\alpha A+(1-\alpha)\frac{1}{n}\mathbb{1}\mathbb{1}^T=\alpha A+\frac{1-\alpha}{n}1_{n\times n}\Rightarrow G=\begin{pmatrix}\frac{\alpha}{2}+\frac{1-\alpha}{n}&\frac{\alpha}{2}+\frac{1-\alpha}{n}&\dots&\frac{\alpha}{2}+\frac{1-\alpha}{n}\\\frac{\alpha}{2}+\frac{1-\alpha}{n}&\frac{\alpha}{2}+\frac{1-\alpha}{n}&\dots&\frac{\alpha}{2}+\frac{1-\alpha}{n}\\\frac{1-\alpha}{n}&\frac{1-\alpha}{n}&\dots&\frac{1-\alpha}{n}\\\vdots&\vdots&&\vdots\\\frac{1-\alpha}{n}&\frac{1-\alpha}{n}&\dots&\frac{1-\alpha}{n}\end{pmatrix}$

$G_{ij}=\frac{\alpha}{2}+\frac{1-\alpha}{n}$ if $i=1$ or $i=2$ and $\frac{1-\alpha}{n}$ otherwise

$\pi=\begin{pmatrix}\pi_1\\\vdots\\\pi_n\end{pmatrix}$ where $\sum_{i=1}^{n}\pi_i=1$

$\pi=G\pi=\begin{pmatrix}\left(\frac{\alpha}{2}+\frac{1-\alpha}{n}\right)(\pi_1+\dots+\pi_n)\\\left(\frac{\alpha}{2}+\frac{1-\alpha}{n}\right)(\pi_1+\dots+\pi_n)\\\frac{1-\alpha}{n}(\pi_1+\dots+\pi_n)\\\vdots\\\frac{1-\alpha}{n}(\pi_1+\dots+\pi_n)\end{pmatrix}=\begin{pmatrix}\frac{\alpha}{2}+\frac{1-\alpha}{n}\\\frac{\alpha}{2}+\frac{1-\alpha}{n}\\\frac{1-\alpha}{n}\\\vdots\\\frac{1-\alpha}{n}\end{pmatrix}$

$\pi^T=\left(\frac{\alpha}{2}+\frac{1-\alpha}{n}\quad\frac{\alpha}{2}+\frac{1-\alpha}{n}\quad\frac{1-\alpha}{n}\dots\frac{1-\alpha}{n}\right)^T=(x\ x\ y\dots y)^T$

$x=\frac{\alpha}{2}+\frac{1-\alpha}{n}$, $y=\frac{1-\alpha}{n}$ where x is the PageRank for Youtube and Wikipedia and y is the PageRank for the other $n-2$ pages.

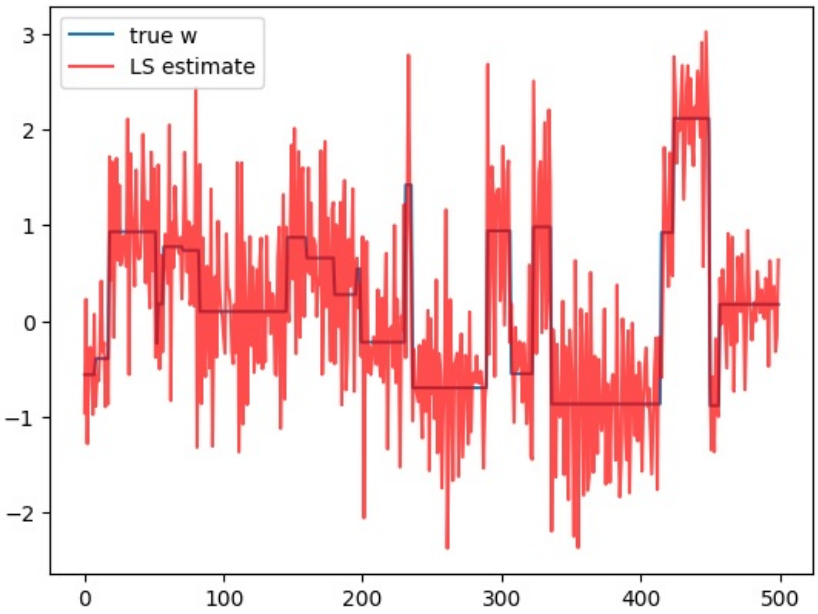
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
import sys
import numpy.linalg as la
from tabulate import tabulate
import random
%matplotlib inline
from PIL import Image
```

1.c

In [2]:
```
# Load data
data = np.load("blurring.npz")
X = data['X']
y = data['y']

U, S, Vt = np.linalg.svd(X, full_matrices=False)

# Estimate w using X and y with regular least squares
# Your code here
w_LS = Vt.T @ np.diag(1 / S) @ U.T @ y

# Estimate w using X and y with truncated SVD
# Your code here
w_15 = (Vt[:15, :]).T @ np.diag(1 / S[:15]) @ (U[:, :15]).T @ y
w_75 = (Vt[:75, :]).T @ np.diag(1 / S[:75]) @ (U[:, :75]).T @ y
w_200 = (Vt[:200, :]).T @ np.diag(1 / S[:200]) @ (U[:, :200]).T @ y
```

1.c.i

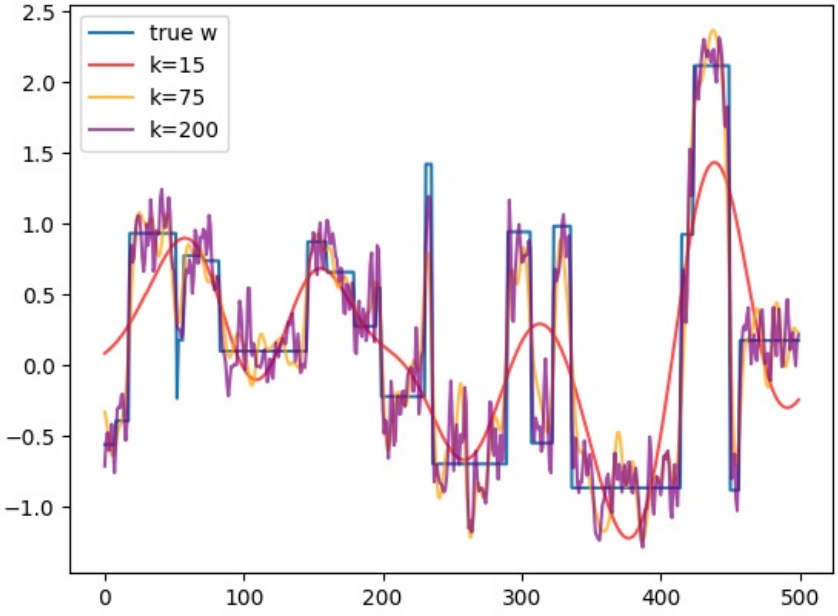
In [3]:
```
# Compare estimate to true value for LS
plt.plot(data['w'],label='true w')
plt.plot(w_LS, alpha=0.7,c='red',label='LS estimate')
plt.legend()
plt.show()
```



The standard least squares does not account for noise in a dataset, so we see the LS estimate deviating from the true weight in order to incorporate the noisy data.

1.c.ii

In [4]:
```
plt.plot(data['w'],label='true w')
plt.plot(w_15, alpha=0.7,c='red',label='k=15')
plt.plot(w_75, alpha=0.7,c='orange',label='k=75')
plt.plot(w_200, alpha=0.7,c='purple',label='k=200')
plt.legend()
plt.show()
```



We use the truncated SVD in order to produce a weight vector based on the k most important singular values. By ignoring potentially very small singular values that may arise from added noise, the truncated SVD performs better than the regular LS since it can ignore noise more effectively. Since a lower k indicates a loss of information from the dataset when $k < rank(X)$, we see smoother curves for higher k values. However, it is important to consider balancing a smoother approximation and losing too much information with lower k values.

2b

In [5]:
```
n = 4
trials = 1000
```

In [6]:
```
A = np.array([[0, 0, 0, 1], [1, 0, 0, 0], [0, 0.5, 0, 0], [0, 0.5, 1, 0]])
pi = np.array(np.random.rand(n)).reshape((n, 1))
normalized_pi = pi / np.sum(pi)

for i in range(trials):
    pi = A @ normalized_pi
    normalized_pi = pi / np.sum(pi)
normalized_pi
```

Out[6]:
```
array([[0.28571429],
       [0.28571429],
       [0.14285714],
       [0.28571429]])
```

Thus, $\pi_1 = \pi_2 = \pi_4 = 0.2857$ and $\pi_3 = 0.1429$.

2c

In [7]:
```
alpha = 0.8

G = alpha * A + (1 - alpha) / n * np.ones(n).T @ np.ones(n)
G = G / G.sum(axis=0)
pi = np.array(np.random.rand(n)).reshape((n, 1))
normalized_pi = pi / np.sum(pi)

for i in range(trials):
    pi = G @ normalized_pi
    normalized_pi = pi / np.sum(pi)
normalized_pi
```

Out[7]:
```
array([[0.26724138],
       [0.25862069],
       [0.18965517],
       [0.28448276]])
```

Thus, $\pi_1 = 0.2672$, $\pi_2 = 0.2586$, $\pi_3 = 0.1897$, and $\pi_4 = 0.2845$.

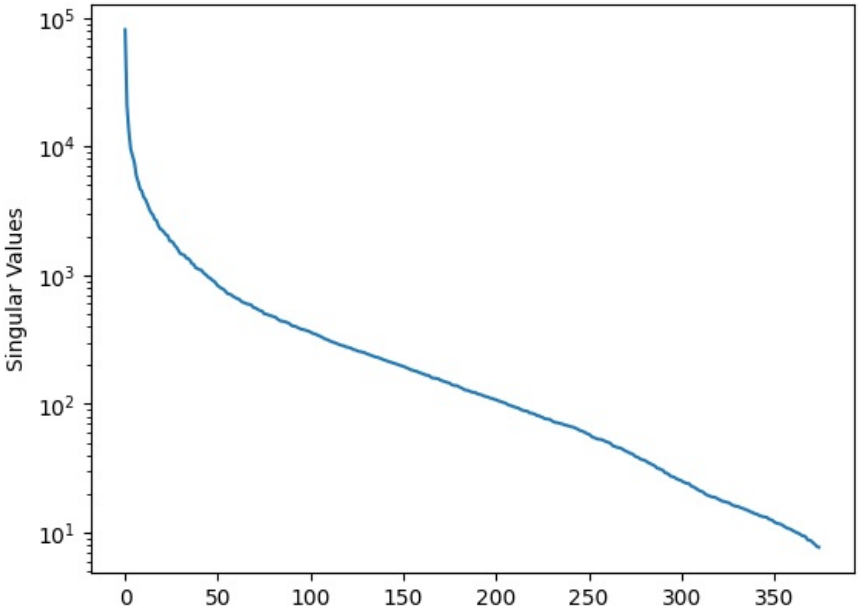
3a

In [8]:
```
# feel free to use your favorite picture
# or work with the provided one
img = Image.open("disaster-girl.jpg", mode="r")
img = np.array(img).astype(np.int32)
```

```
plt.imshow(img)
plt.title('Original Picture')
plt.show()

# YOUR CODE BELOW
# note that images usually have 3 channels
# so, you can reshape it
# img_stack = face.reshape((img.shape[0], -1))
# and find its SVD.
# another option is to find the SVD of every color channel

img_stack = img.reshape((img.shape[0], -1))
U, S, Vt = np.linalg.svd(img_stack, full_matrices=False)
```

In [9]:

```
plt.plot(S)
plt.ylabel('Singular Values')
plt.yscale('log')
plt.show()
```



The singular values decrease as Σ gives the singular matrices in non-increasing order.

3b

In [10]:

```
num_channels = 3
```

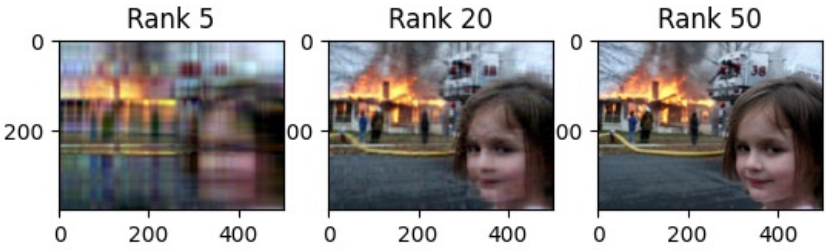
In [11]:

```
def compress(image, k):
    """
    Perform svd decomposition and truncate it (using k singular values/vectors)
```

```
    Parameters:
        image (np.array): input image (probably, colourful)
        k (int): approximation rank (number of singular values)

    Returns:
      reconst_matrix (np.array): reconstructed matrix (tensor in colourful case)

      s (np.array): array of singular values
    """
    # YOUR CODE IS HERE
    image = np.array(image).astype(np.int32)
    truncated = []
    s = []
    for channel in range(num_channels):
        current_img = image[:, :, channel]
        U, S, Vt = np.linalg.svd(current_img, full_matrices=False)
        new_channel = U[:, :k] @ np.diag(S[:k]) @ Vt[:k, :]
        truncated.append(new_channel)
        s.append(S[:k])
    reconst_matrix = np.stack(truncated, axis=-1)
    reconst_matrix = np.array(reconst_matrix).astype(np.int32)

    return reconst_matrix, np.array(s)
```

In [17]:

```
rank5_img, rank5_s = compress(img, 5)
rank20_img, rank20_s = compress(img, 20)
rank50_img, rank50_s = compress(img, 50)

fig, ax = plt.subplots(1, 3)

ax[0].imshow(rank5_img)
ax[0].set_title('Rank 5')

ax[1].imshow(rank20_img)
ax[1].set_title('Rank 20')

ax[2].imshow(rank50_img)
ax[2].set_title('Rank 50')

plt.show()
```



Is it clear that higher rank images more closely resemble the original image since we are losing less information with the truncation.

3c

In [13]:

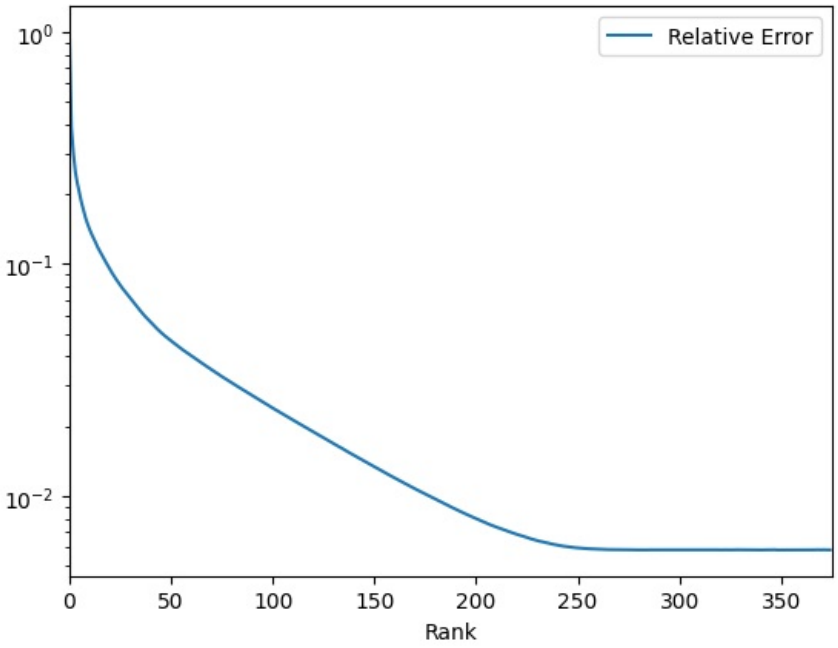
```
def calculate_error(actual, expected):
    return np.linalg.norm(expected - actual, 'fro') / np.linalg.norm(expected, 'fro')
```

In [21]:

```
max_k = min(img.shape[0], img.shape[1])
original_pred = compress(img, max_k)[0]
original_size = img.shape[0] * img.shape[1]

all_errors = []
all_sizes = []
for current_k in range(max_k):
    current_pred = compress(img, current_k)[0]
    total_error = 0
    total_size = img.shape[0] * current_k + img.shape[1] * current_k + current_k

    total_error = calculate_error(current_pred[:, :, 0], original_pred[:, :, 0])
    all_errors.append(total_error)
    all_sizes.append(total_size / original_size)
```

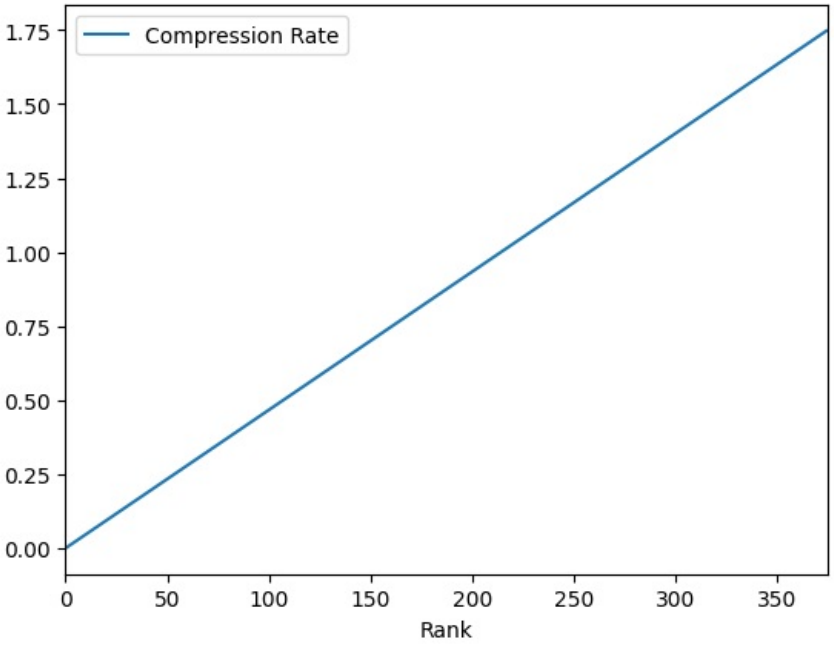
In [22]:

```
plt.plot(all_errors, label='Relative Error')
plt.xlabel('Rank')
plt.xlim((0, max_k))
plt.yscale('log')
plt.legend()
plt.show()
```



As rank increases, the relative error decreases. This makes sense because a higher rank means we are preserving more of the original image, so the discrepancy between the truncated and original decreases.

In [23]:
```
plt.plot(all_sizes, label='Compression Rate')
plt.xlabel('Rank')
plt.xlim((0, max_k))
plt.legend()
plt.show()
```



As rank increases, the compression rate increases. This makes sense because a higher rank means it takes more storage to preserve a greater amount of information. There are a variety of cutoffs we can use, but around $k = 75$ we see a slight decrease in slope of the relative error, so it may be a good cutoff. Also, the compression rate is above 0.25 when $k = 75$, which may be sufficient. We can further verify this by a holistic approach since doing the $k = 50$ approximation before yielded a relatively clear picture, so something around $k = 75$ allows us to now have to store much information by using a small rank but also lets us have a relatively accurate image.