Q1

In [73]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import warnings
import pickle
from io import BytesIO
pd.options.display.float_format = '{:.5f}'.format
pd.set_option('display.max_columns', 500)
warnings.filterwarnings('ignore')
```

a

In [74]:
```
# Load the image using PIL
img = Image.open("Cliff_beach_BOEM_gov.jpg")
```

In [75]:
```
# Convert the image to a numpy array
img_arr = np.array(img)

# Get the dimensions of the image
width, height, colour  = img_arr.shape
print("Image width: {}, height: {}, that is number of pixels. And for eac
```

```
Image width: 1830, height: 2746, that is number of pixels. And for each p
ixel we have 3 colours (RGB).
```

Width and height value are the number of pixels. The value of the 3rd dimension is the colour from RGB method (0-255,0-255,0-255).

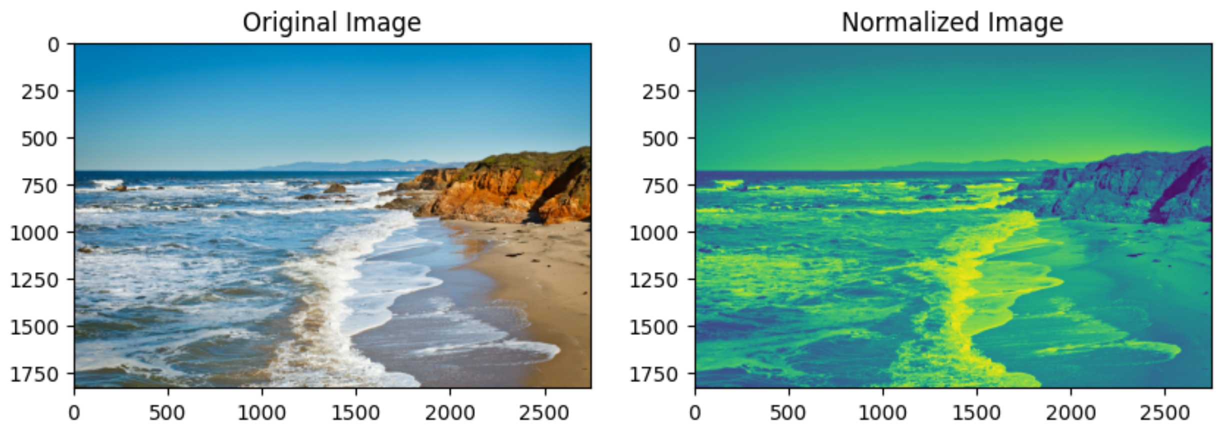
b

In [76]:
```
img_arr_sum = img_arr.sum(axis=2)
```

In [77]:
```
# Normalize the image by subtracting the mean and dividing by the standar
img_arr_normalized = (img_arr_sum - np.mean(img_arr_sum)) / np.std(img_ar
```

Plot the original and normalized images side by side:

In [78]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
ax1.imshow(img_arr)
ax1.set_title("Original Image")
ax2.imshow(img_arr_normalized)
ax2.set_title("Normalized Image")
plt.show()
```



We got the negative of the picture that we had before. The normalization process reduced the overall contrast of the image

In [79]:
```
# Save the Original image in memory using BytesIO
img_bytes = BytesIO()
img.save(img_bytes, format='JPEG')
# Check the size of the image in bytes
size_in_bytes = img_bytes.tell()
print("Size of the Original image in bytes:", size_in_bytes)
```

```
Size of the Original image in bytes: 675682
```

In [80]:
```
# Revert normalized image into uint8 type
img_arr_img_norm = Image.fromarray(img_arr_normalized.astype(np.uint8))
# Save the Normalized image in memory using BytesIO
img_bytes1 = BytesIO()
img_arr_img_norm.save(img_bytes, format='JPEG')
# Check the size of the image in bytes
size_in_bytes = img_bytes.tell()
print("Size of the Normalized image in bytes:", size_in_bytes)
```

```
Size of the Normalized image in bytes: 1400311
```

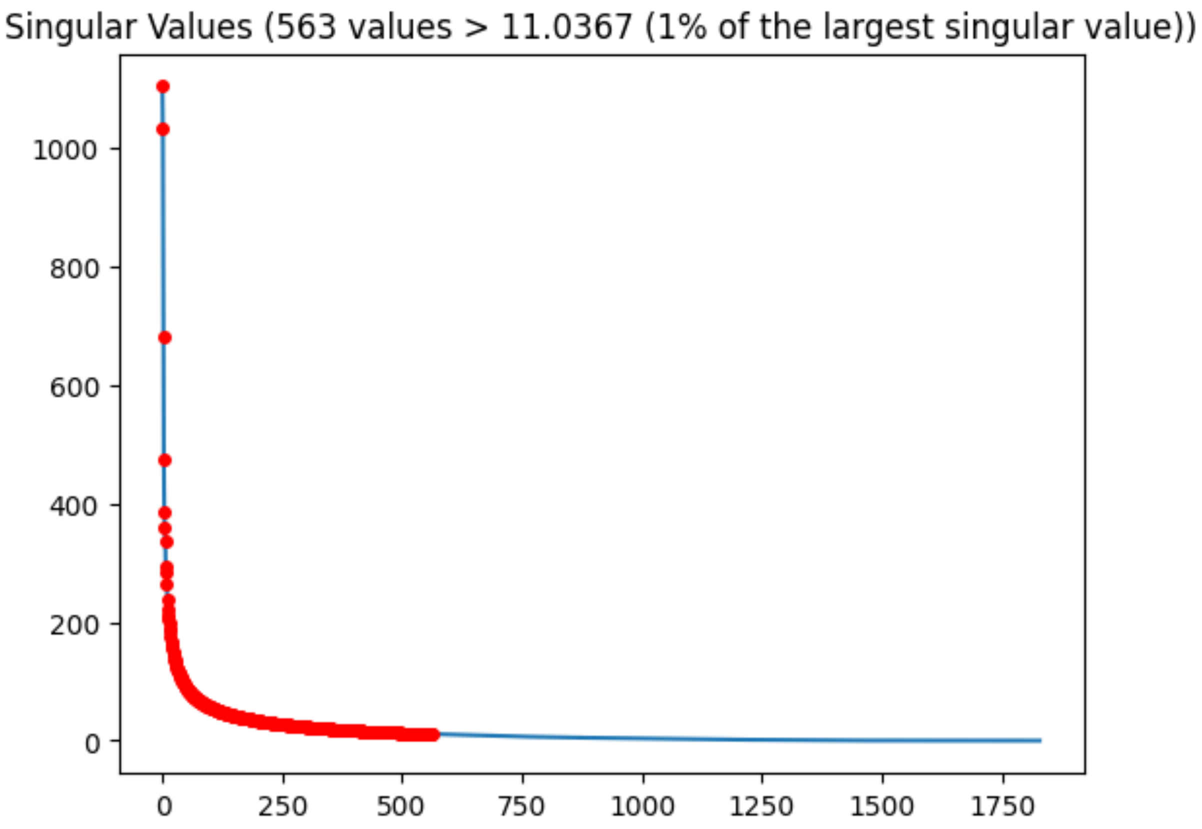
C

In [81]:
```
# Perform SVD decomposition and obtain the singular values
U, s, Vt = np.linalg.svd(img_arr_normalized)
print("Number of singular values is:", len(s))
```

```
Number of singular values is: 1830
```

We have exactly this number of singular values due the fact that 1830 is the width of our picture that is dim of the matrix

In [82]:
```
# Plot the singular values and the threshold
threshold = 0.01 * s[0] # 1% of the largest singular value
num_singular_values = np.sum((s > threshold))
plt.plot(s)
plt.plot(s[s > threshold], 'ro', markersize=4)
plt.title(f"Singular Values ({num_singular_values} values > {threshold:.4
plt.show()
```

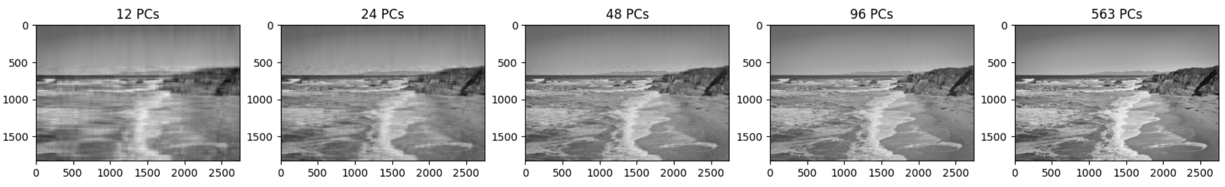


d

In [83]:
```
pcs = [12, 24, 48, 96, len(s[s > threshold])]

fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(20,4))
for i, num_pc in enumerate(pcs):
# Calculate the number of principal components required to obtain the des
    img_reconstructed = U[:, :num_pc] @ np.diag(s[:num_pc]) @ Vt[:num_pc,
    # Display the reconstructed image
    axs[i].imshow(img_reconstructed, cmap='gray')
    axs[i].set_title(f"{num_pc} PCs")
    # Calculate the size of the final reconstruction
    img_bytes = BytesIO()
    img_reconstructed_pic= Image.fromarray(img_reconstructed.astype(np.ui
    img_reconstructed_pic.save(img_bytes, format='JPEG')
    # Check the size of the image in bytes
    size = img_bytes.tell()
    print(f"Reconstructed Image Size with {num_pc} PCs: {size} bytes")
plt.show()
```

```
Reconstructed Image Size with 12 PCs: 248213 bytes
Reconstructed Image Size with 24 PCs: 337995 bytes
Reconstructed Image Size with 48 PCs: 439669 bytes
Reconstructed Image Size with 96 PCs: 553668 bytes
Reconstructed Image Size with 563 PCs: 728039 bytes
```



First of all, we have used gray pictures (cmap='gray') because much easier to recognize the difference between pictures with different PC's with b&w than negative one.
So what we can conclude from the pictures? When we are reducing the number of PC's from 563 to 96 there is no big difference between pictures that means that first 96 PC's explain the most of the variance in the given normalized picture that is great evidence why we need PCA algorithm. But if we take a look on 12 PC's picture we would see clear difference with the full picture that means that 12 PC's is not enough for the good picture, but still we can see approximately what is on the picture that is good but not enough for us if we want to downgrade the number of PC's with minimum influence on the quality, but may be good for other purposes.
Also important to add that on the graph of singular values we've seen that first of them have much higher values than others that means that they explain most of the variance of the picture colours.

There is a correlation between the decreasing value of the PCs and the decreasing image size, indicating that as the value of the PCs decreases, the image size tends to decrease as well.

Q2

A

Load the files:

In [84]:
```
with open("exodus_pickled", "rb") as f:
    exodus_pickled = pickle.load(f)

features = np.loadtxt("features.txt", delimiter="\t", dtype="str")[:-1] #
labels = np.loadtxt("labels.txt")

print(f"Features dimension: {features.shape[0]}")
print(f"Labels dimension: {labels.shape[0]}")
print(f"Matrix dimensions: {exodus_pickled.shape}")
```

```
Features dimension: 15373
Labels dimension: 1201
Matrix dimensions: (1201, 15373)
```

| The features is the array that lists the unique triplets of words associated with each column of exodus_pickled. The labels array contains the hypothetical association of each verse to one of two groups: 0 (non-priestly) or 1 (priestly).
The number of raws in **exodus_pickled** is exactly the number of the psukim (where each one of them has a lable), and the number of columns is the number of the triple words included in the Shemot, which are our *features*.

B

In their current format, it's difficult to make a plot of the verses and check whether the association with the groups shows a real difference between them. This is because the data is high-dimensional, making it challenging to visualize the relationship between the priestly and non-priestly schools directly. We need to perform dimensionality reduction first.

C

SVD decomposition on the exodus_pickled matrix:

In [85]:
```
U, s, Vt = np.linalg.svd(exodus_pickled)
```

In [86]:
```
print(f"U dimensions: {U.shape}")
print(f"s dimension: {s.shape[0]}")
print(f"Vt dimensions: {Vt.shape}")
```

```
U dimensions: (1201, 1201)
s dimension: 1201
Vt dimensions: (15373, 15373)
```

The dimensions of the three components are:

U: The columns are the left singular vectors, which form an orthonormal basis for the column space of the input matrix. Here this matrix related to the Psukim.

s: These diagonal elements are the singular values in descending order (it is a 1-dimensional array here).

Vt: The rows are the right singular vectors, which form an orthonormal basis for the row space of the input matrix.In our example indicates the number of unique triplets of words in Shemot

The matrix **Vt** expresses the importance vectors (of triples) for each PC because its dimensions match the dimensions of the features.

D

Finding the top 10 absolute loading values for the first PC:

In [87]:
```
top_loading_indices = np.argsort(-np.abs(pd.DataFrame(Vt.T).iloc[:,0])) #
top_triple_words = features[top_loading_indices]
top_triple_words[:10]
```

Out[87]:
```
array(["('ו', 'צפה', 'את')", "('עשׂה', 'את', 'ה')", "('ו', 'עשׂה', 'את')",
       "('ו', 'נתן', 'את')", "('אמה', 'ו', 'חצי')", "('ו', 'את', 'ה')",
       "('ו', 'עשׂה', 'ל')", "('צפה', 'את', 'זהב')", "('ה', 'אחד', 'ו')",
       "('תכלת', 'ו', 'ארגמן')"], dtype='<U29')
```

The commonality between these ten triplets of words and their importance for the first PC might suggest a specific theme or pattern that differentiates the priestly and non-priestly schools.

Most of the ten triplets of words include 'את' ,'עשׂה' ,'ו', we can assume that the words from top ten triplets can be good indicator for the Sefer Shemot . The importance of these ten triplets of words for the first PC suggests that the first PC, as we have expected, have to explain the most info (variance of the information) as possible of the Shemot (in our example) and therefore the words that we see are very popular in Tonach and Shemot in particular.

E

After we checked the given link and serched for our top 10 triples we can assume that the given words more related to the Shemot as paterns. The words as we can see are more related to some giving procedd where Jews receive the Ten Commandments and establish the covenant with God.

F

Projecting the data on the first two PCs and display them graphically:
To project pasuk on first 2 PCs we have to take the dot product between the pasuk vector and Vt_1 , the dot product between the pasuk vector and Vt_2.

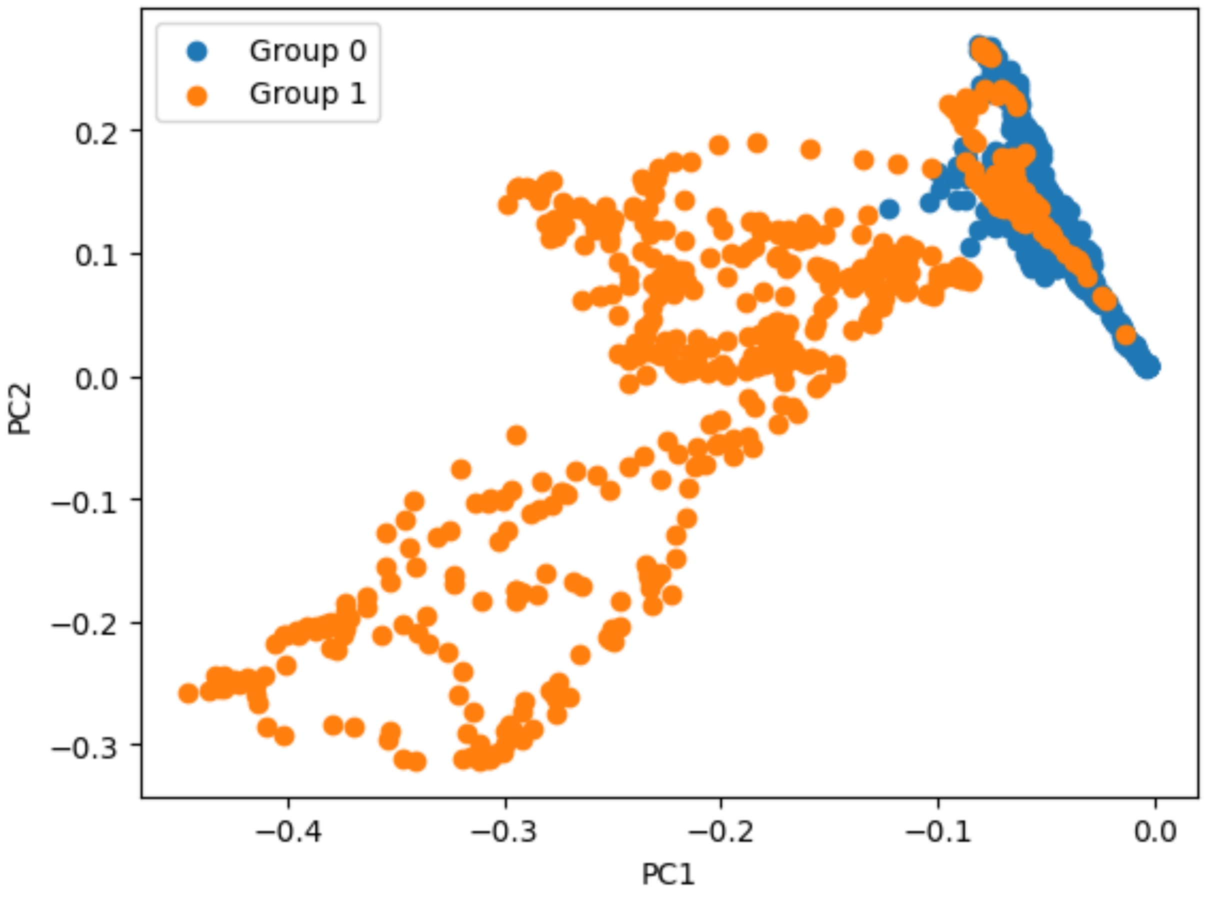
In [88]:
```
# Projecting the data
projected_data = exodus_pickled @ Vt[:2, :].T



group0_x, group0_y = [], []
group1_x, group1_y = [], []

for i in range(len(labels)):
    if labels[i] == 0:
        group0_x.append(projected_data[i, 0])
        group0_y.append(projected_data[i, 1])
    elif labels[i] == 1:
        group1_x.append(projected_data[i, 0])
        group1_y.append(projected_data[i, 1])

plt.scatter(group0_x, group0_y, label="Group 0")
plt.scatter(group1_x, group1_y, label="Group 1")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.show()
```



We can assume that *Goup 0* has the negative relation and *Group 1* has positiv correlation. We think that there is difference between groups (Group 0 is concentrated in up right corner and Group 1 is not) so the hypotetical devision has sence. So overall, our assumption from just checking triples in psukim that there is some difference in the groups was right.

G

The research question is about exploring the potential correlation between the hypothetical separation into group 0 or 1 and the two axes of variation in the context of PCA. The data being used includes features related to the occurrences of triples in verses, and the labels dataset indicating the grouping of data points into group 0 or 1.

The contributions of SVD in this research question involve its application in the PCA technique to identify the axes of variation that explain the most variance in the data. SVD helps with decomposition, includes insights into how the hypothetical separation into groups 0 and 1 may be related to these axes of variation.

Additional tools that could help us in investigating this matter could include **cluster analysis** techniques to identify patterns and structures within the data, **statistical hypothesis testing** to assess the significance of observed correlations, and LSTM which is the neural net architecture, that looks onto dependencies in sequences.