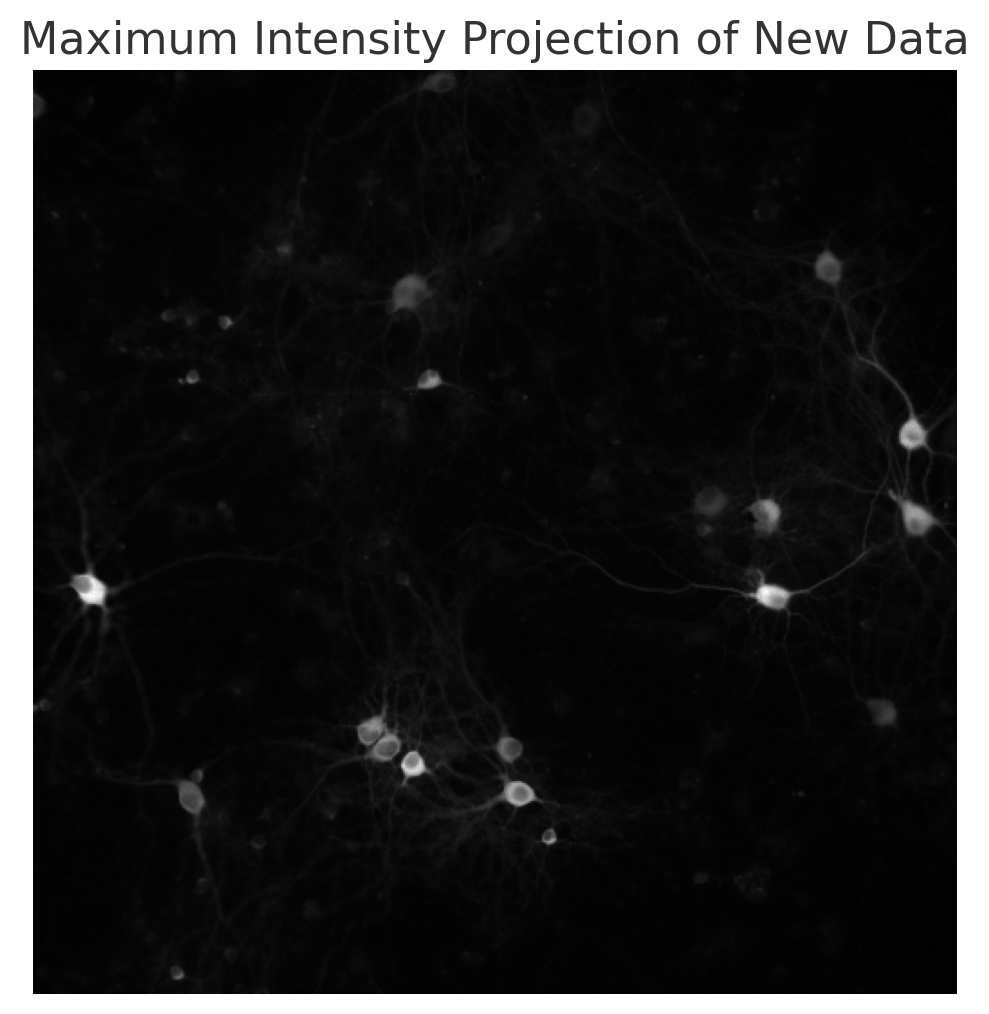
**PCA Analysis and Clustering of Ca Activity Data**

I calculated maximum intensity projection image from one of my recordings, and select ROI’s by hand as a ground truth selection, then I created a mask using ImageJ.

Max intensity projection:



Masked image:



I want to write a python code that makes this steps automatically.

#loaded tiff video file

#normalized the images so their pixel values mostly fall between 0 and 1

#converted the images as float32 <- the data type that is fast for GPU computation

import numpy as np

import tifffile

# Define the normalization function

def normalize99(img):

X = img.copy()

x01 = np.percentile(X, 1)

x99 = np.percentile(X, 99)

X = (X - x01) / (x99 - x01)

return X.astype(np.float32)

# Load the TIF file

file\_path = 'path\_to\_your\_tif\_file.tif'

tif = tifffile.TiffFile(file\_path)

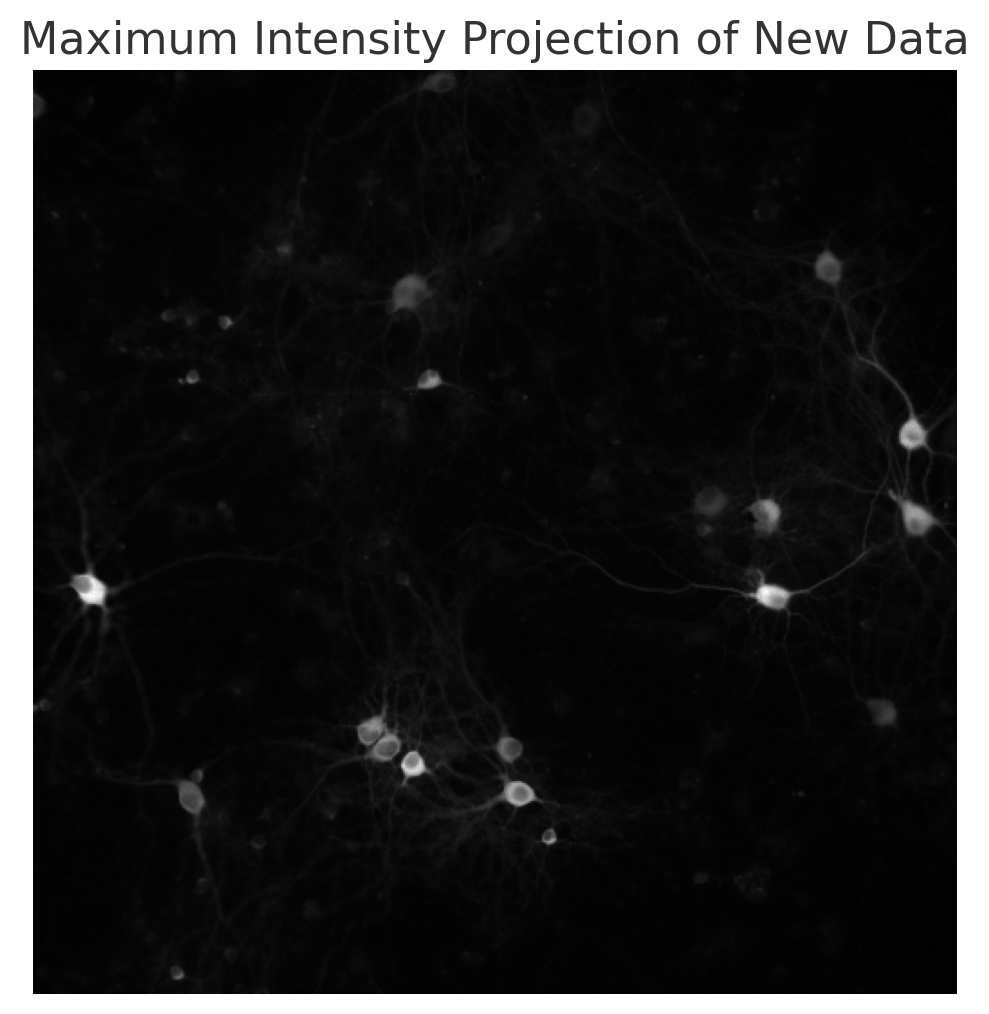
# Read the images into a list and apply normalization

normalized\_frames = [normalize99(frame.asarray()) for frame in tif.series[0].pages]

# Compute the Maximum Intensity Projection (MIP)

mip = np.max(normalized\_frames, axis=0)

mip image:



# Scikit-image provides a function named try\_all\_threshold that applies various thresholding methods to an image and displays the results side by side for easy comparison. I thought that this part can be asked to user.

from skimage.filters import try\_all\_threshold

import matplotlib.pyplot as plt

# normalize intensity across image

max\_img\_filtered = max\_img.copy() / gaussian\_filter(mip, 100)

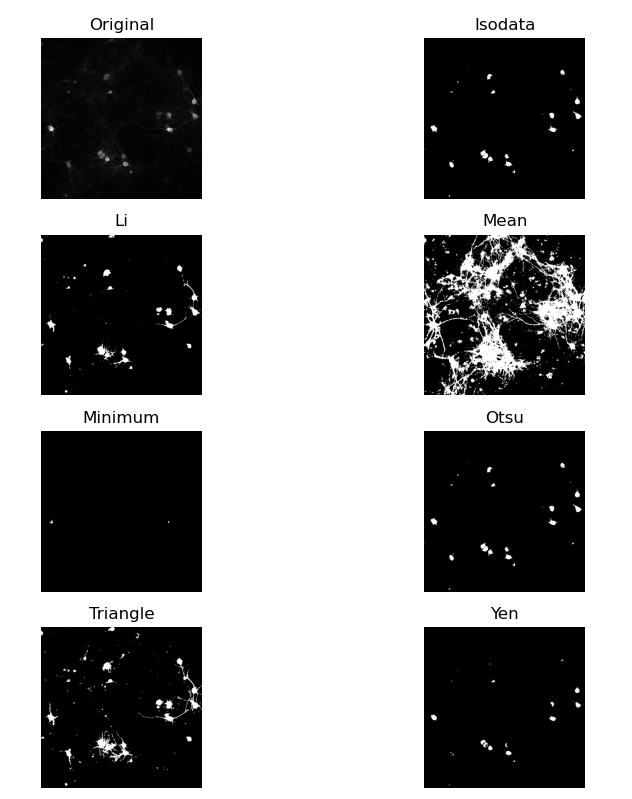
# high pass filter

max\_img\_filtered = max\_img\_filtered - gaussian\_filter(mip, 10)

max\_img\_filtered = normalize99(max\_img\_filtered)

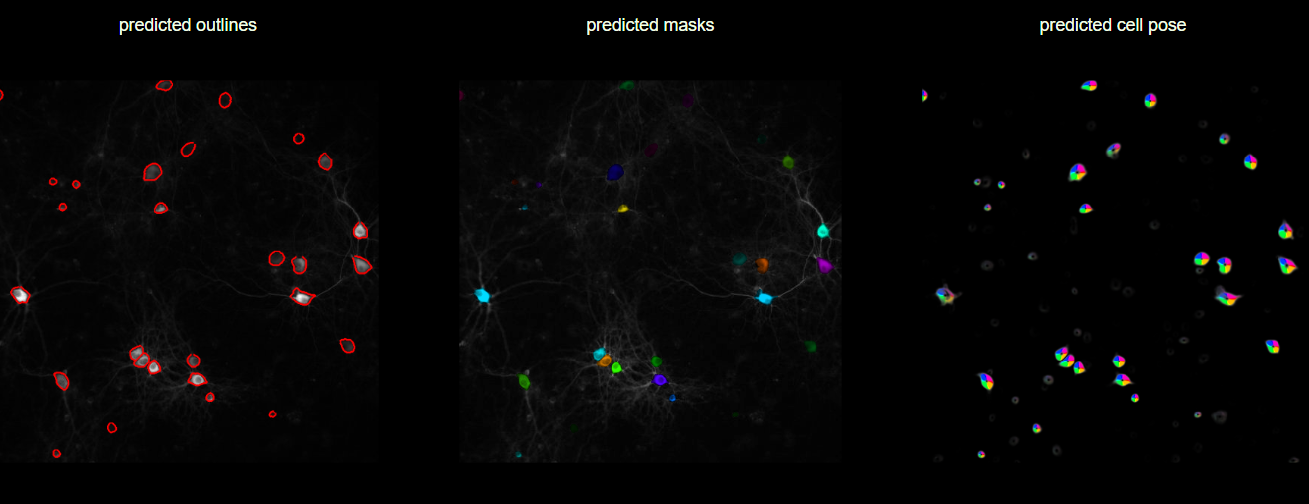
fig, ax = try\_all\_threshold(mip, figsize=(10, 8), verbose=False)

plt.show()



In this project I don’t need to get all the cells, but in long term I need to use deep learning algorithms for proper segmentation.

This is the results that I get using Cellpose, and it’s super impressive https://www.cellpose.org/:



# Calculate the Otsu threshold

threshold\_value = filters.threshold\_otsu(mip)

# Generate the binary mask

binary\_mask = mip > threshold\_value

# Display results

fig, ax = plt.subplots(1, 2, figsize=(12, 5))

# Histogram and Otsu threshold

ax[0].hist(mip.ravel(), bins=256)

ax[0].axvline(threshold\_value, color='r', linestyle='dashed', linewidth=2)

ax[0].set\_title('Histogram of Normalized MIP')

ax[0].set\_xlabel('Intensity Value')

ax[0].set\_ylabel('Frequency')

# Binary mask

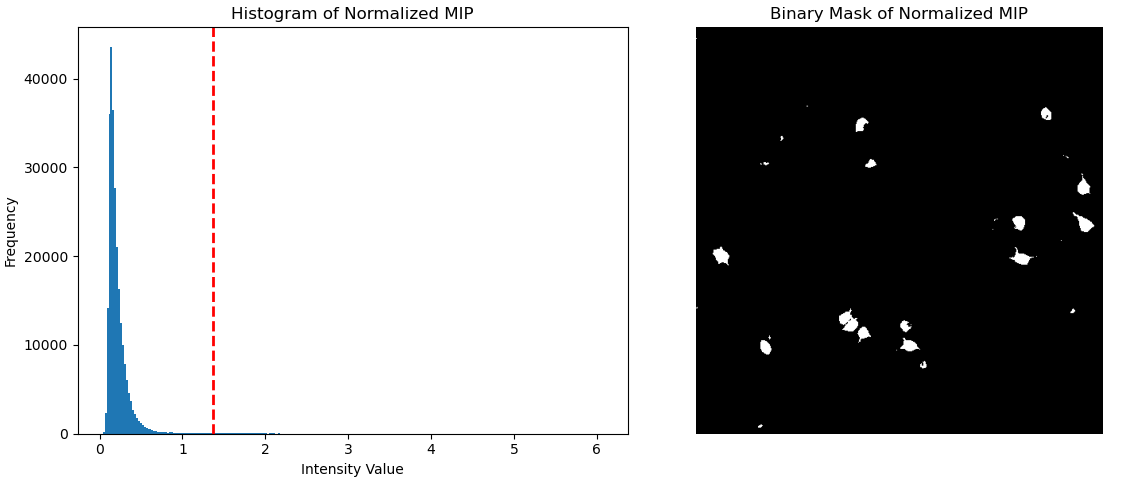
ax[1].imshow(binary\_mask, cmap='gray')

ax[1].set\_title('Binary Mask of Normalized MIP')

ax[1].axis('off')

plt.tight\_layout()

plt.show()



#Assigning a different colour for each cell

import numpy as np

import matplotlib.pyplot as plt

from skimage.measure import label

from skimage.color import label2rgb

import random

# Function to generate a unique color for each label

def generate\_unique\_colors(num\_labels):

random.seed(42) # Seed for reproducibility

colors = [(0, 0, 0)] # Background color

for \_ in range(1, num\_labels + 1):

# Generate a unique color for each label, avoiding too dark colors

colors.append((random.random()\*0.6 + 0.4, random.random()\*0.6 + 0.4, random.random()\*0.6 + 0.4))

return colors

# Label the connected regions in the binary mask

labeled\_mask = label(binary\_mask, connectivity=2)

# Generate unique colors

num\_labels = labeled\_mask.max()

unique\_colors = generate\_unique\_colors(num\_labels)

# Create an RGB image with the unique colors

colored\_labeled\_image = np.zeros((\*labeled\_mask.shape, 3), dtype=float)

for label, color in enumerate(unique\_colors):

colored\_labeled\_image[labeled\_mask == label] = color

# Display the colored labeled image

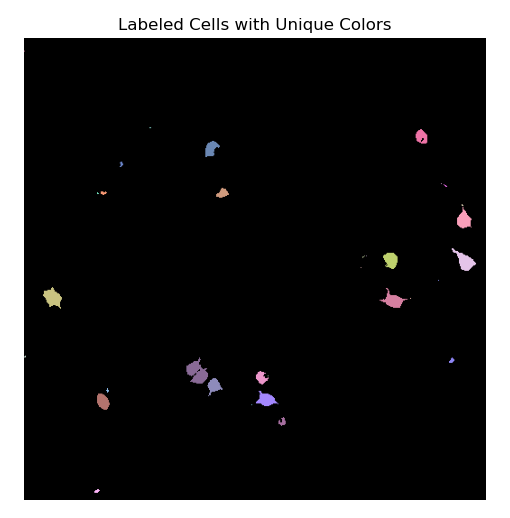
plt.figure(figsize=(6, 6))

plt.imshow(colored\_labeled\_image)

plt.title('Labeled Cells with Unique Colors')

plt.axis('off')

plt.show()



Here I have two problems: 1. Cells with very small radius, 2. Different cells identified as one single cell. I will correct this issue in the future.

cell\_intensities = {label: [] for label in range(1, num\_labels + 1)}

# Iterate over each frame to calculate average intensities

for frame in normalized\_frames:

for label in cell\_intensities.keys():

cell\_mask = labeled\_mask == label

cell\_intensity = frame[cell\_mask].mean()

cell\_intensities[label].append(cell\_intensity)

# Plotting

fig, axs = plt.subplots(2, 1, figsize=(12, 10))

# Display the labeled mask

colored\_labeled\_image = np.zeros((\*labeled\_mask.shape, 3), dtype=float)

for label, color in enumerate(unique\_colors):

colored\_labeled\_image[labeled\_mask == label] = color

axs[0].imshow(colored\_labeled\_image)

axs[0].set\_title('Labeled Cells with Unique Colors')

axs[0].axis('off')

# Display the activity data

for label, intensities in cell\_intensities.items():

axs[1].plot(intensities, color=unique\_colors[label], label=f'Cell {label}')

axs[1].set\_title('Average Intensity Over Time for Each Cell')

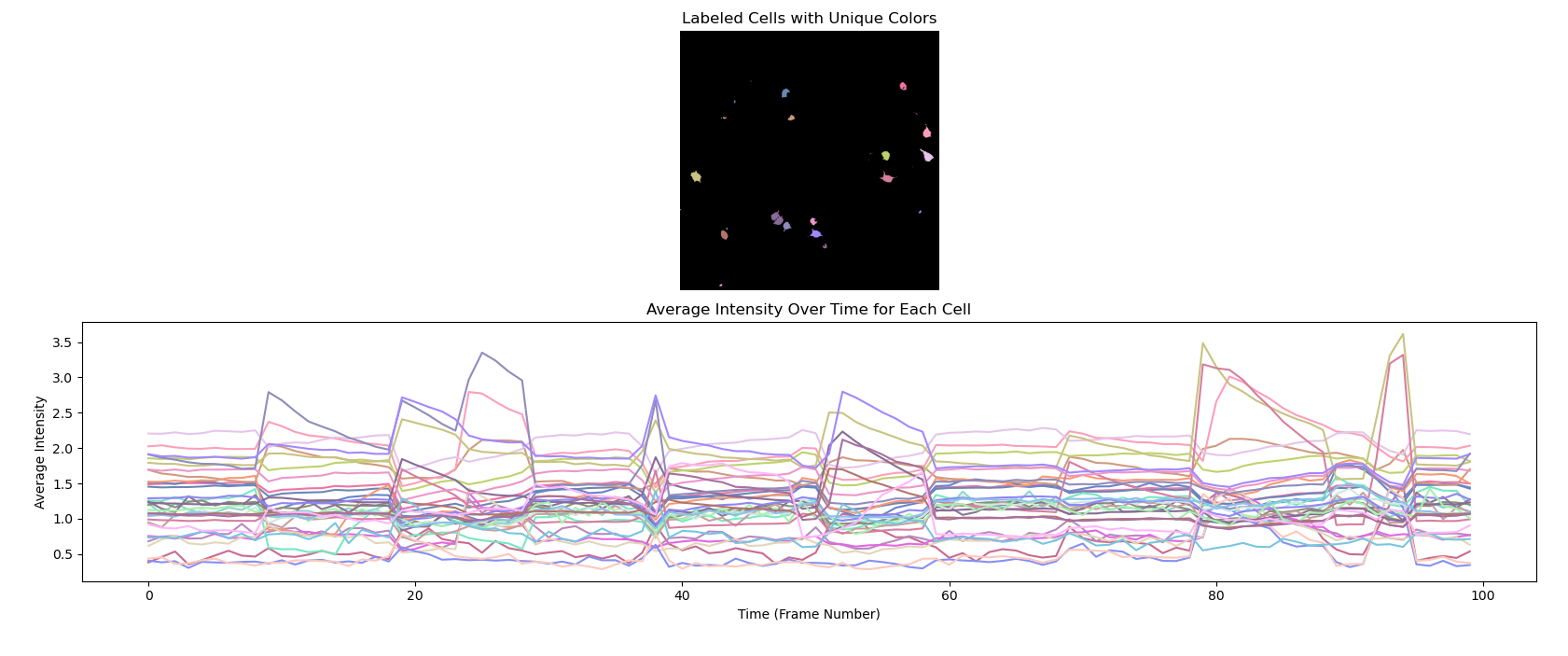
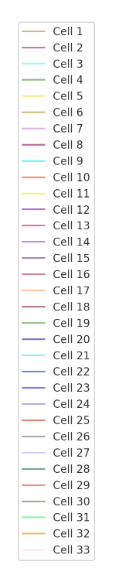
axs[1].set\_xlabel('Time (Frame Number)')

axs[1].set\_ylabel('Average Intensity')

axs[1].legend(bbox\_to\_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

plt.tight\_layout()

plt.show()



To find the correlation between cells, first I plotted the correlation matrix between cell activities:

# Convert the cell intensities dictionary to a list of lists for easier manipulation

intensity\_values = list(cell\_intensities.values())

# Convert the list of lists into a NumPy array for easier mathematical operations

intensity\_array = np.array(intensity\_values)

# Calculate the correlation matrix between the activities of the cells

correlation\_matrix = np.corrcoef(intensity\_array)

# Display the correlation matrix

plt.figure(figsize=(10, 8))

plt.imshow(correlation\_matrix, cmap='viridis', interpolation='nearest')

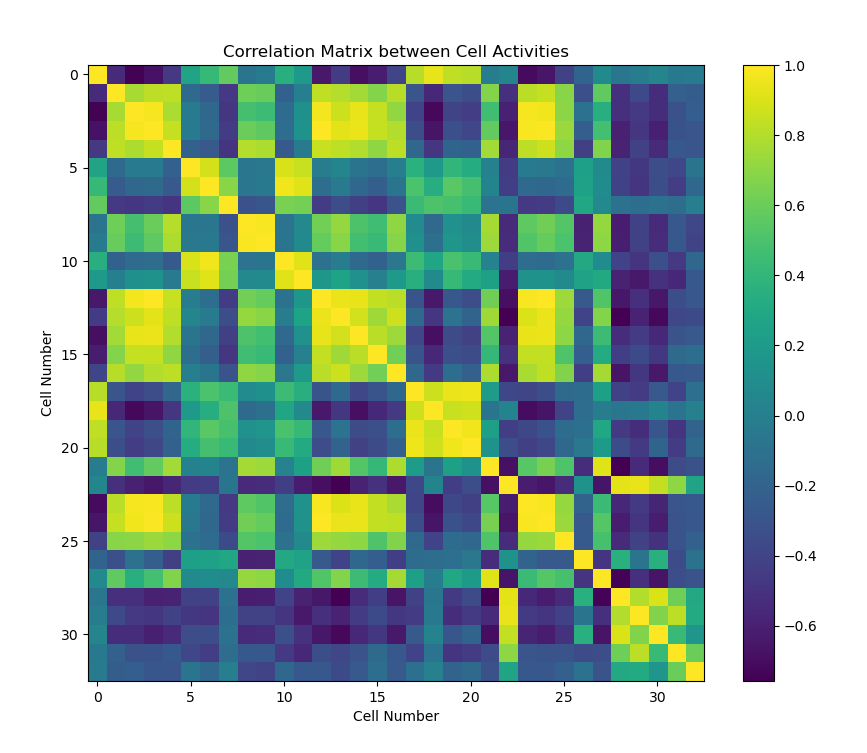
plt.colorbar()

plt.title('Correlation Matrix between Cell Activities')

plt.xlabel('Cell Number')

plt.ylabel('Cell Number')

plt.show()



A value close to 1 indicates a strong positive correlation, close to -1 indicates a strong negative correlation. Values near 0 indicate little to no linear relationship between the activities of the cells. Then, I calculated the PCA and clustered them with Kmeans clustering method.

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

pca = PCA(n\_components=2)

pca.fit(correlation\_matrix)

components = pca.components\_

explained\_variance = pca.explained\_variance\_

# Plot the first two principal components

plt.figure(figsize=(12, 6))

for i, (component, variance) in enumerate(zip(components, explained\_variance), start=1):

plt.subplot(1, 2, i)

plt.bar(range(len(component)), component)

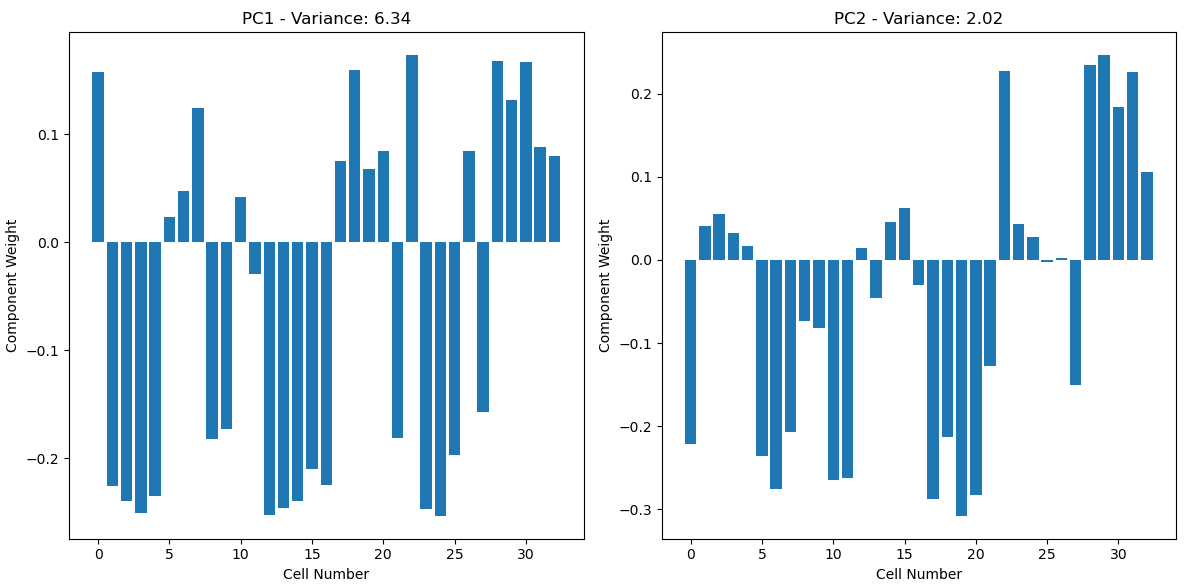
plt.title(f'PC{i} - Variance: {variance:.2f}')

plt.xlabel('Cell Number')

plt.ylabel('Component Weight')

plt.tight\_layout() # Correct placement

plt.show()



# Project the original correlation matrix data onto the first two principal components

pca\_projection = pca.transform(correlation\_matrix)

inertia = []

for n\_clusters in range(1, 11):

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)

kmeans.fit(pca\_projection)

inertia.append(kmeans.inertia\_)

optimal\_clusters = 3 # Adjust this value based on the Elbow method result

kmeans = KMeans(n\_clusters=optimal\_clusters, random\_state=42)

cluster\_labels = kmeans.fit\_predict(pca\_projection)

# Plot the clusters with cell numbers

plt.figure(figsize=(10, 8))

colors = plt.cm.viridis(np.linspace(0, 1, optimal\_clusters))

for i, color in zip(range(optimal\_clusters), colors):

plt.scatter(pca\_projection[cluster\_labels == i, 0], pca\_projection[cluster\_labels == i, 1],

color=color, marker='o', edgecolor='k', s=50, label=f'Cluster {i+1}')

# Label each point with its cell number

for i, (x, y) in enumerate(pca\_projection):

plt.text(x, y, str(i + 1), color='black', fontsize=9)

plt.title('Clusters of Cell Activities (PCA Projection)')

plt.xlabel('PCA 1')

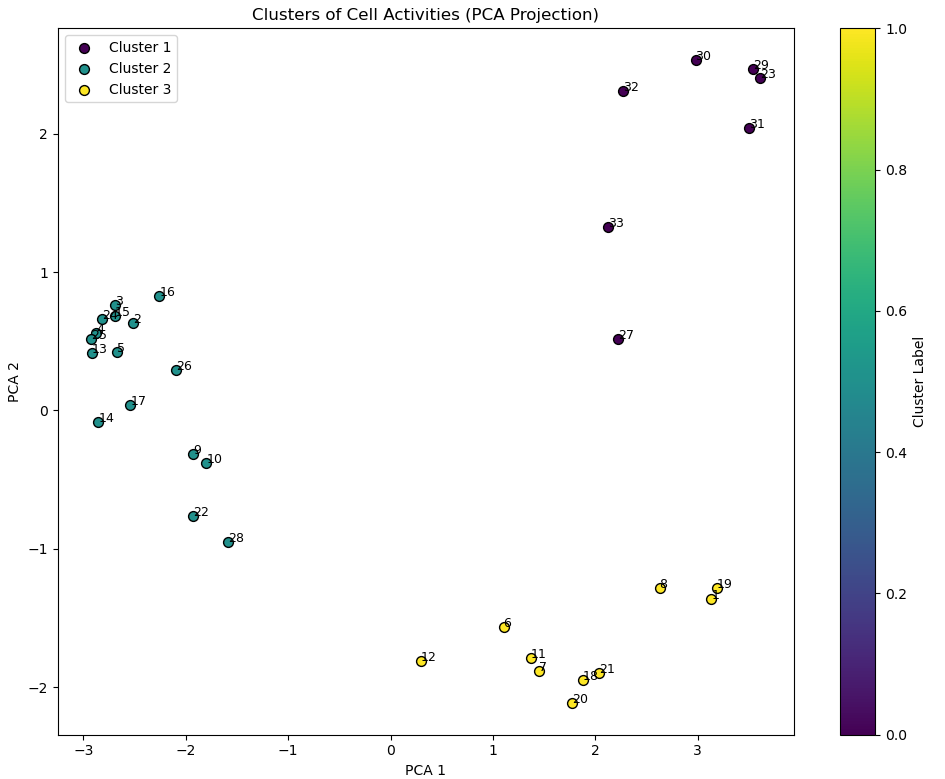
plt.ylabel('PCA 2')

plt.colorbar(label='Cluster Label')

plt.legend()

plt.tight\_layout()

plt.show()



I will apply the same methods to my data after adding a specific drug and see the activity response of the cells. Thank you for reading and many thanks to chatgpt 😊