In [1]:
```
!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install plotly
!pip install scikit-learn
!pip install scipy
```

```
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: pandas in /home/jupyter-doggo/.local/lib/pyt
hon3.9/site-packages (1.5.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /opt/tljh/user/li
b/python3.9/site-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.20.3 in /home/jupyter-doggo/.local/
lib/python3.9/site-packages (from pandas) (1.24.1)
Requirement already satisfied: pytz>=2020.1 in /opt/tljh/user/lib/python3.
9/site-packages (from pandas) (2022.7)
Requirement already satisfied: six>=1.5 in /opt/tljh/user/lib/python3.9/sit
e-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: numpy in /home/jupyter-doggo/.local/lib/pyth
on3.9/site-packages (1.24.1)
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: matplotlib in /opt/tljh/user/lib/python3.9/s
ite-packages (3.6.3)
Requirement already satisfied: numpy>=1.19 in /home/jupyter-doggo/.local/li
b/python3.9/site-packages (from matplotlib) (1.24.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/tljh/user/lib/pyth
on3.9/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: contourpy>=1.0.1 in /opt/tljh/user/lib/pytho
n3.9/site-packages (from matplotlib) (1.0.7)
Requirement already satisfied: python-dateutil>=2.7 in /opt/tljh/user/lib/p
ython3.9/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: packaging>=20.0 in /opt/tljh/user/lib/python
3.9/site-packages (from matplotlib) (22.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/tljh/user/lib/pyth
on3.9/site-packages (from matplotlib) (4.38.0)
Requirement already satisfied: cycler>=0.10 in /opt/tljh/user/lib/python3.
9/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/tljh/user/lib/pytho
n3.9/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: pillow>=6.2.0 in /opt/tljh/user/lib/python3.
9/site-packages (from matplotlib) (9.4.0)
Requirement already satisfied: six>=1.5 in /opt/tljh/user/lib/python3.9/sit
e-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: plotly in /home/jupyter-doggo/.local/lib/pyt
hon3.9/site-packages (5.13.0)
Requirement already satisfied: tenacity>=6.2.0 in /home/jupyter-doggo/.loca
l/lib/python3.9/site-packages (from plotly) (8.2.1)
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: scikit-learn in /opt/tljh/user/lib/python3.
9/site-packages (1.2.1)
Requirement already satisfied: scipy>=1.3.2 in /home/jupyter-doggo/.local/l
ib/python3.9/site-packages (from scikit-learn) (1.10.0)
Requirement already satisfied: numpy>=1.17.3 in /home/jupyter-doggo/.local/
lib/python3.9/site-packages (from scikit-learn) (1.24.1)
Requirement already satisfied: joblib>=1.1.1 in /opt/tljh/user/lib/python3.
9/site-packages (from scikit-learn) (1.2.0)
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/tljh/user/lib/p
ython3.9/site-packages (from scikit-learn) (3.1.0)
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: scipy in /home/jupyter-doggo/.local/lib/pyth
on3.9/site-packages (1.10.0)
Requirement already satisfied: numpy<1.27.0,>=1.19.5 in /home/jupyter-dogg
o/.local/lib/python3.9/site-packages (from scipy) (1.24.1)
```

In [2]:
```python
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import plotly.express as px
from sklearn.mixture import GaussianMixture
from scipy import stats
from scipy.optimize import fsolve
from scipy.stats import norm
import plotly
```

In [3]:
```python
plotly.offline.init_notebook_mode()
```

In [4]:
```python
# TO-DO:
# Change the directory accordingly to the folder on your system.

sample_data_path = '02-05-2023/sample_take_2/'
zone_0_folder_path = '02-05-2023/zone_0_sample_take_2/'
```

In [5]:
```python
# Get a list of all files in the directory
file_list = os.listdir(sample_data_path)

# Sort the list of files (Necessary on Linux)
file_list.sort()

# Print the list of files
file_list
```

Out[5]:
```
['goes15.2016.12.08.1200.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1300.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1500.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1530.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1545.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1600.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1615.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1630.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1645.v01.nc-var1-t0.csv',
 'goes15.2016.12.08.1700.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1200.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1400.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1415.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1430.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1445.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1500.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1530.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1545.v01.nc-var1-t0.csv',
 'goes15.2016.12.09.1600.v01.nc-var1-t0.csv',
 'goes15.2016.12.10.1300.v01.nc-var1-t0.csv',
 'goes15.2016.12.11.1400.v01.nc-var1-t0.csv',
 'goes15.2016.12.11.1415.v01.nc-var1-t0.csv',
 'goes15.2016.12.11.1430.v01.nc-var1-t0.csv',
 'goes15.2016.12.11.1445.v01.nc-var1-t0.csv',
 'goes15.2016.12.11.1500.v01.nc-var1-t0.csv',
 'goes15.2016.12.11.1530.v01.nc-var1-t0.csv',
 'goes15.2016.12.11.1545.v01.nc-var1-t0.csv',
 'goes15.2016.12.11.1600.v01.nc-var1-t0.csv',
 'goes15.2016.12.12.1700.v01.nc-var1-t0.csv',
 'goes15.2016.12.12.1715.v01.nc-var1-t0.csv',
 'goes15.2016.12.12.1730.v01.nc-var1-t0.csv',
 'goes15.2016.12.12.1745.v01.nc-var1-t0.csv',
 'goes15.2016.12.12.1800.v01.nc-var1-t0.csv',
 'goes15.2016.12.12.1830.v01.nc-var1-t0.csv',
 'goes15.2016.12.12.1845.v01.nc-var1-t0.csv',
 'goes15.2016.12.12.1900.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1400.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1415.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1430.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1445.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1500.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1530.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1545.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1600.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1615.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1630.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1645.v01.nc-var1-t0.csv',
 'goes15.2016.12.13.1700.v01.nc-var1-t0.csv',
 'goes15.2016.12.15.1700.v01.nc-var1-t0.csv',
 'goes15.2016.12.15.1800.v01.nc-var1-t0.csv',
 'goes15.2016.12.15.1900.v01.nc-var1-t0.csv',
 'goes15.2016.12.19.1400.v01.nc-var1-t0.csv',
 'goes15.2016.12.19.1500.v01.nc-var1-t0.csv',
 'goes15.2016.12.19.1600.v01.nc-var1-t0.csv',
 'goes15.2016.12.19.1700.v01.nc-var1-t0.csv',
 'goes15.2016.12.20.1600.v01.nc-var1-t0.csv',
```

```
 'goes15.2016.12.20.1700.v01.nc-var1-t0.csv',
 'goes15.2016.12.20.1800.v01.nc-var1-t0.csv',
 'goes15.2016.12.22.1700.v01.nc-var1-t0.csv',
 'goes15.2016.12.22.1800.v01.nc-var1-t0.csv',
 'goes15.2016.12.22.1900.v01.nc-var1-t0.csv',
 'goes15.2016.12.26.1700.v01.nc-var1-t0.csv',
 'goes15.2016.12.26.1800.v01.nc-var1-t0.csv',
 'goes15.2016.12.26.1900.v01.nc-var1-t0.csv',
 'goes15.2016.12.27.1800.v01.nc-var1-t0.csv',
 'goes15.2016.12.27.1900.v01.nc-var1-t0.csv',
 'goes15.2016.12.27.2000.v01.nc-var1-t0.csv',
 'goes15.2017.01.05.1700.v01.nc-var1-t0.csv',
 'goes15.2017.01.05.1800.v01.nc-var1-t0.csv',
 'goes15.2017.01.05.1900.v01.nc-var1-t0.csv',
 'goes15.2017.11.10.1400.v01.nc-var1-t0.csv',
 'goes15.2017.11.10.1500.v01.nc-var1-t0.csv',
 'goes15.2017.11.10.1600.v01.nc-var1-t0.csv']
```

In [6]:

```
prefix_dict = {}
first_index_list = []

for i, filename in enumerate(file_list):
    prefix = filename[:18]
    if prefix in prefix_dict:
        continue
    else:
        prefix_dict[prefix] = i
        first_index_list.append(i)

num_groups = len(first_index_list)

print('Number of days in the dataset: ', num_groups)

print(first_index_list)
```

```
Number of days in the dataset:  14
[0, 10, 19, 20, 28, 36, 48, 51, 55, 58, 61, 64, 67, 70]
```

In [7]:

```
# Get the boundaries
LP = pd.read_csv('/srv/scratch/NOAA/Lake_Partition.csv')
LP
```

Out[7]:

	latitude	first_bound	second_bound	left_zone	mid_zone	right_zone
0	40.06	-87.5465	-84.8170	2	3	4
1	40.10	-87.5465	-84.8170	2	3	4
2	40.14	-87.5465	-84.8170	2	3	4
3	40.18	-87.5465	-84.8170	2	3	4
4	40.22	-87.5465	-84.8170	2	3	4
...	...	...	...	...	...	...
193	47.78	-86.1781	-84.9473	6	6	6
194	47.82	-86.1781	-84.9473	6	6	6
195	47.86	-86.1781	-84.9473	6	6	6
196	47.90	-86.1781	-84.9473	6	6	6
197	47.94	-86.1781	-84.9473	6	6	6

198 rows × 6 columns

---

0. Clean & Extract Lake Michigan Data

In [8]:
```
# # Create the directory if it doesn't exist
# if not os.path.exists(zone_0_folder_path):
#     os.makedirs(zone_0_folder_path)
#     print('New folder created!')
# else:
#     # Delete all files in the directory if it already exists
#     print("Folder exists, now deleting existing files.")
#     for file in os.listdir(zone_0_folder_path):
#         file_path = os.path.join(zone_0_folder_path, file)
#         try:
#             if os.path.isfile(zone_0_folder_path):
#                 os.remove(zone_0_folder_path)
#         except Exception as e:
#             print(e)
```

In [9]:
```
# for fn in file_list:
#     file = pd.read_csv(sample_data_path+fn)
#     flp = pd.merge(file, LP, on = 'latitude')
#     par_list = []
#     for i in range(len(flp)):
#         a = flp['longitude'][i]
#         b = flp['first_bound'][i]
#         c = flp['second_bound'][i]
#         if a < b:
#             par_list.append(flp['left_zone'][i])
#         elif a > c:
#             par_list.append(flp['right_zone'][i])
#         else:
```

```
#             par_list.append(flp['mid_zone'][i])
#     flp['partition'] = par_list
#     flp = flp.loc[:, ['value', 'datetime', 'latitude', 'longitude', 'parti
#     s_flp = flp[flp['partition'] == 0].reset_index().rename(columns={'inde

#     ## TO-DO: Change the directory, clearly lable as "zone_0"
#     s_flp.to_csv(zone_0_folder_path+str(fn),index = False)
#     print(fn)
```

Since there is no difference in file names, we can retrieve the `zone 0` files just by changing the parent directory path.

Let's look into the cases

There are `13` days in the tester pack, and we will perform analysis on them.

---

1. Case 1

`Duration` : 2016.12.08 1200-1600 UTC

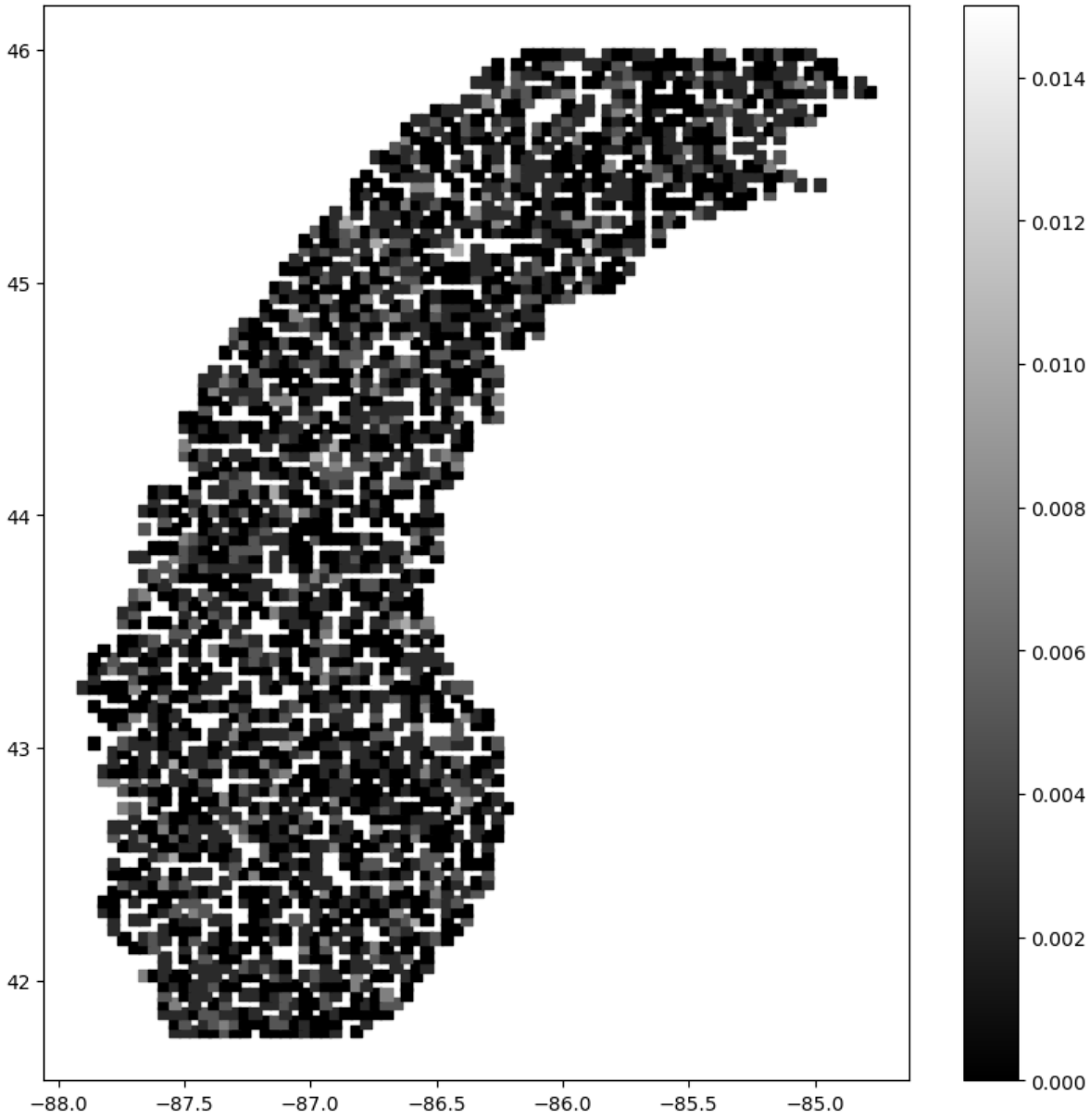
1.1 2016.12.08.1200

But before this, let's take a look at what happened at `1200` .

As we can see, there are blue noises on the image in the Lake Michigan area.

In [10]:
```
df_0 = pd.read_csv(zone_0_folder_path + file_list[0])

x_0, y_0, i_0 = df_0.longitude, df_0.latitude, df_0.value
plt.figure(figsize=(10, 10))
plt.scatter(x_0.values, y_0.values, c=i_0.values, cmap=cm.gray, marker='s')
plt.colorbar(orientation='vertical')
len(x_0), len(y_0), len(i_0)
```

Out[10]:
```
(2346, 2346, 2346)
```



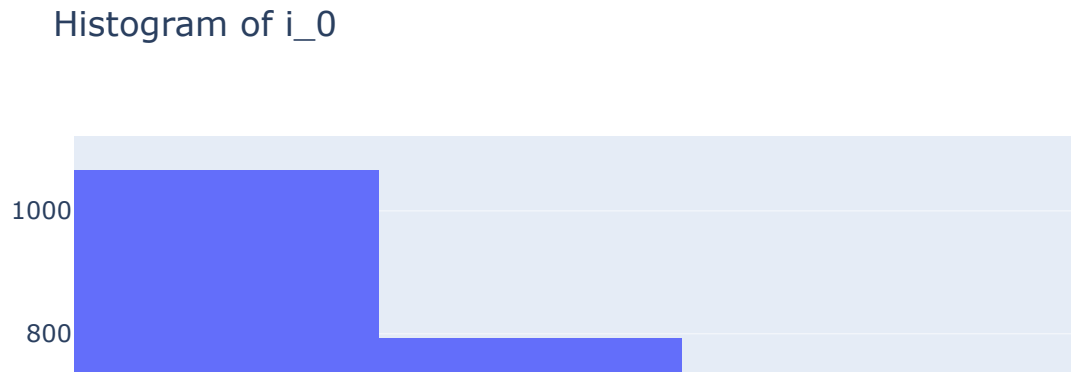
In [11]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_0 = np.nanmean(i_0.values)
std_i_0 = np.nanstd(i_0.values)

# Check number of nan values and 0s in i, just in case
num_nan_0 = i_0.isna().sum().sum()
num_zeros_0 = (i_0 == 0).sum().sum()

# Check min and max in i

i_0_max = i_0.max().max()
i_0_min = i_0.min().min()
i_0_mode = i_0.mode()[0]

print('Number of nan values in i_0 = ', num_nan_0)
print('Number of 0 values in i_0 = ', num_zeros_0)
print('                values in i_0 = ', len(i_0))
print('-------------------------------------------------------')
```

```
print('Mean of values in i_0 = ', mean_i_0)
print('STD of 1-D values in i_0 = ', std_i_0)
print('Mean value >= STD =====> ', mean_i_0 >= std_i_0)
print('-------------------------------------------------------')
print("Max value in i_0 = ", i_0_max)
print("Min value in i_0 = ", i_0_min)
print('Mode value in i_0 = ', i_0_mode)
```

```
Number of nan values in i_0 =  0
Number of 0 values in i_0 =  1066
Number of values in i_0 =  2346
-------------------------------------------------------
Mean of values in i_0 =  0.0020428388742540497
STD of 1-D values in i_0 =  0.0022900731580302335
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_0 =  0.015
Min value in i_0 =  0.0
Mode value in i_0 =  0.0
```

In [12]:

```
# # Plot histogram
# plt.hist(i_0.values.flatten(), bins=50)

# # Set plot labels and title
# plt.xlabel('Value')
# plt.ylabel('Count')
# plt.title('Distribution of Values in DataFrame i_0')

# # Display plot
# plt.show()

fig = px.histogram(i_0.values.flatten(), title="Histogram of i_0")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



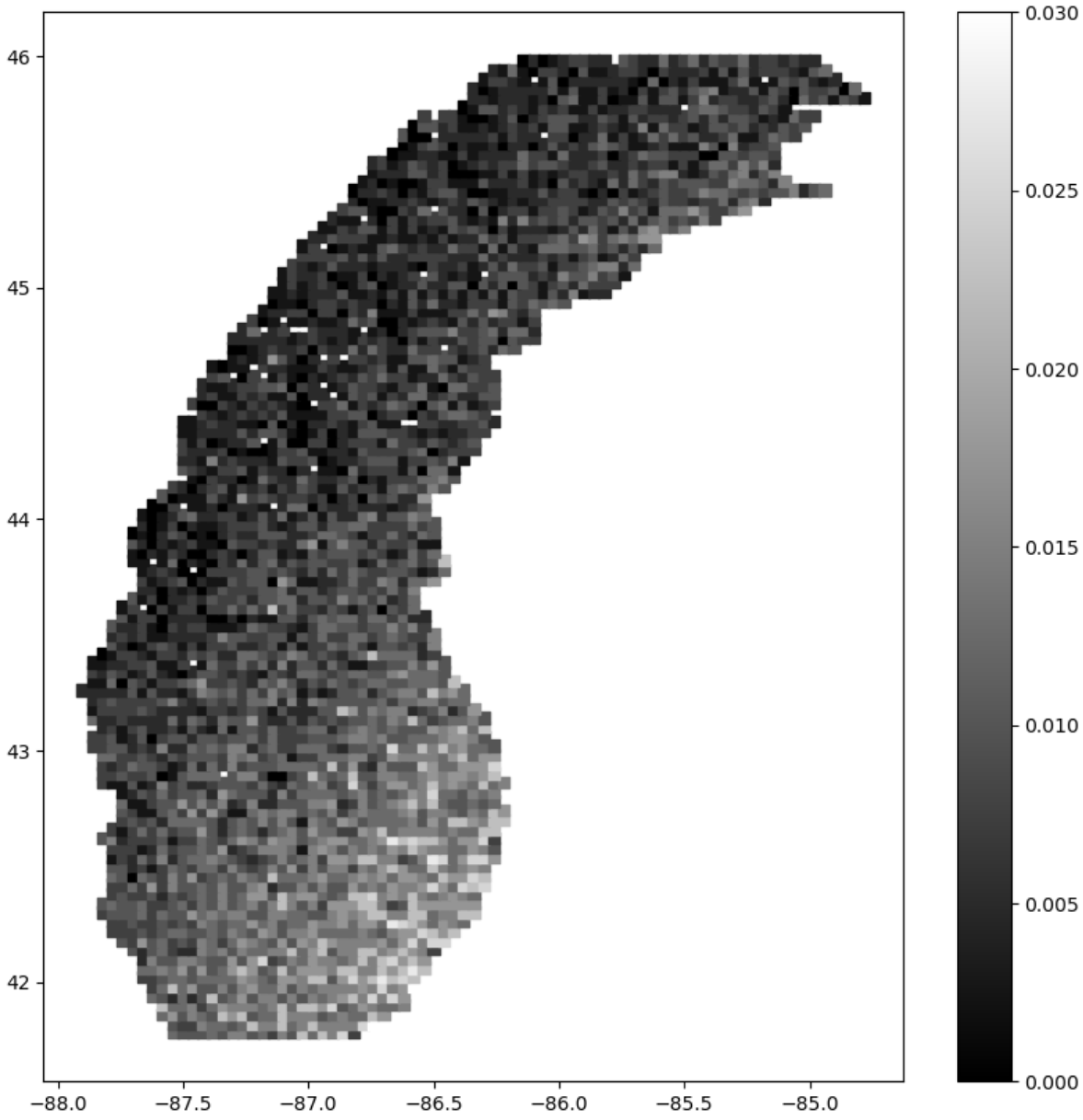
1.2 2016.12.08.1300

At `1300`, we assume that the sun is going up since **13:00 UTC** is **8:00 CDT**.

Most of the Lake Michigan area is still covered in darkness.

In [13]:
```
df_1 = pd.read_csv(zone_0_folder_path + file_list[1])

x_1, y_1, i_1 = df_1.longitude, df_1.latitude, df_1.value
plt.figure(figsize=(10, 10))
plt.scatter(x_1.values, y_1.values, c=i_1.values, cmap=cm.gray, marker='s')
plt.colorbar(orientation='vertical')
len(x_1), len(y_1), len(i_1)
```

Out[13]:
```
(3558, 3558, 3558)
```



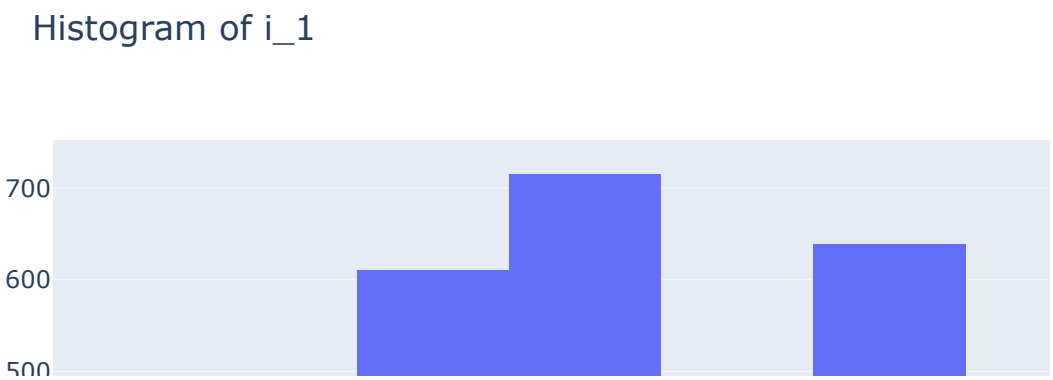
In [14]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_1 = np.nanmean(i_1.values)
std_i_1 = np.nanstd(i_1.values)

# Check number of nan values and 0s in i, just in case
num_nan_1 = i_1.isna().sum().sum()
num_zeros_1 = (i_1 == 0).sum().sum()

# Check min and max in i

i_1_max = i_1.max().max()
i_1_min = i_1.min().min()
i_1_mode = i_1.mode()[0]

print('Number of nan values in i_1 = ', num_nan_1)
print('Number of 0 values in i_1 = ', num_zeros_1)
print('                values in i_1 = ', len(i_1))
print('-------------------------------------------------------')
```

```
print('Mean of values in i_1 = ', mean_i_1)
print('STD of 1-D values in i_1 = ', std_i_1)
print('Mean value >= STD =====> ', mean_i_1 >= std_i_1)
print('-------------------------------------------------------')
print("Max value in i_1 = ", i_1_max)
print("Min value in i_1 = ", i_1_min)
print('Mode value in i_1 = ', i_1_mode)
```

```
Number of nan values in i_1 =  0
Number of 0 values in i_1 =  155
Number of values in i_1 =  3558
-------------------------------------------------------
Mean of values in i_1 =  0.008943929010118043
STD of 1-D values in i_1 =  0.005206204872047027
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_1 =  0.03
Min value in i_1 =  0.0
Mode value in i_1 =  0.0075
```

In [15]:

```
# # Plot histogram
# plt.hist(i_1.values.flatten(), bins=50)

# # Set plot labels and title
# plt.xlabel('Value')
# plt.ylabel('Count')
# plt.title('Distribution of Values in DataFrame i_1')

# # Display plot
# plt.show()

fig = px.histogram(i_1.values.flatten(), title="Histogram of i_1")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



1.3 2016.12.08.1500

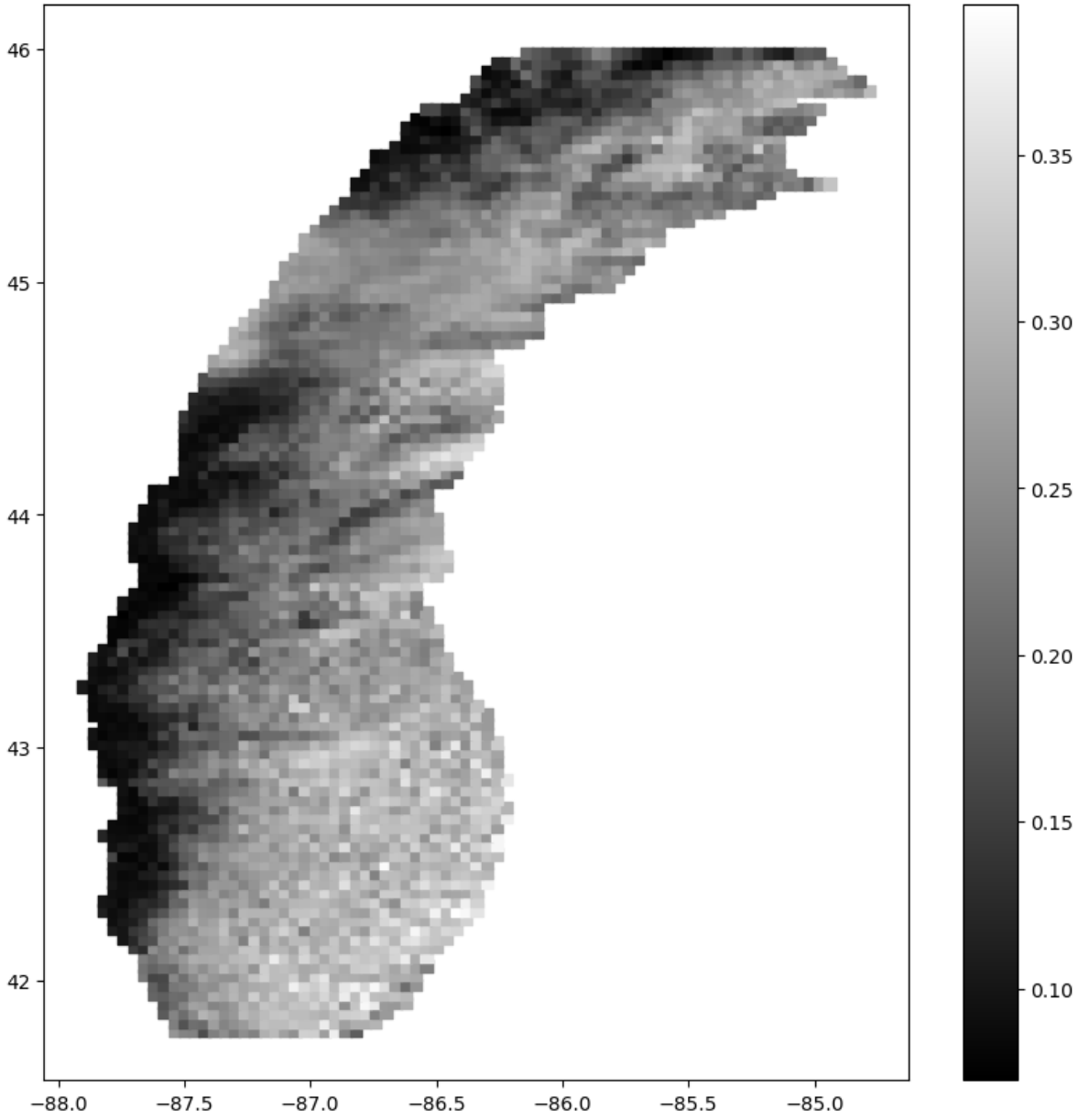
At `1500`, we assume that the sun has already gone up since **15:00 UTC** is **10:00 CDT**.

As shown in the picture above, the cloud is visible in this case. Now, let's dive into the same process just to see what is going on.

One interesting fact I would like to point out is that how the number of points in the sample is `3,599`, which should be the ideal case. The previous cases all have less number of points in the datasets.

In [16]:
```
df_2 = pd.read_csv(zone_0_folder_path + file_list[2])

x_2, y_2, i_2 = df_2.longitude, df_2.latitude, df_2.value
plt.figure(figsize=(10, 10))
plt.scatter(x_2.values, y_2.values, c=i_2.values, cmap=cm.gray, marker='s')
plt.colorbar(orientation='vertical')
len(x_2), len(y_2), len(i_2)
```

Out[16]: (3599, 3599, 3599)



In [17]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_2 = np.nanmean(i_2.values)
std_i_2 = np.nanstd(i_2.values)

# Check number of nan values and 0s in i, just in case
num_nan_2 = i_2.isna().sum().sum()
num_zeros_2 = (i_2 == 0).sum().sum()

# Check min and max in i

i_2_max = i_2.max().max()
i_2_min = i_2.min().min()
i_2_mode = i_2.mode()[0]

print('Number of nan values in i_2 = ', num_nan_2)
print('Number of 0 values in i_2 = ', num_zeros_2)
```

```
print('Number of values in i_2 = ', len(i_2))
print('-------------------------------------------------------')
print('Mean of values in i_2 = ', mean_i_2)
print('STD of 1-D values in i_2 = ', std_i_2)
print('Mean value >= STD =====> ', mean_i_2 >= std_i_2)
print('-------------------------------------------------------')
print("Max value in i_2 = ", i_2_max)
print("Min value in i_2 = ", i_2_min)
print('Mode value in i_2 = ', i_2_mode)
```

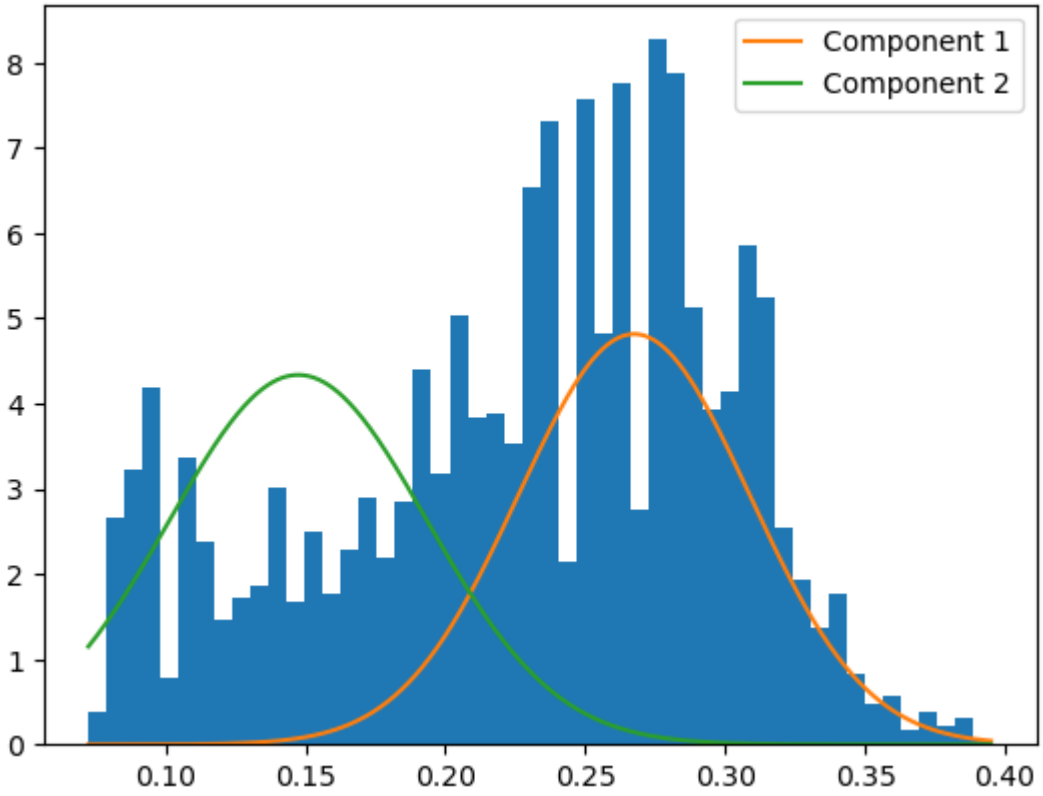
```
Number of nan values in i_2 =  0
Number of 0 values in i_2 =  0
Number of values in i_2 =  3599
-------------------------------------------------------
Mean of values in i_2 =  0.22888440950041677
STD of 1-D values in i_2 =  0.07059967323918688
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_2 =  0.39499998
Min value in i_2 =  0.0725
Mode value in i_2 =  0.2525
```

In [18]:

```
# # Plot histogram
# plt.hist(i_2.values.flatten(), bins=50)

# # Set plot labels and title
# plt.xlabel('Value')
# plt.ylabel('Count')
# plt.title('Distribution of Values in DataFrame i_2')

# # Display plot
# plt.show()

fig = px.histogram(i_2.values.flatten(), title="Histogram of i_2")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```

In [19]:
```
print(i_2)
x_2 = np.array(i_2).reshape(-1, 1)
print(x_2)
# Fit a mixture of two Gaussians to the data
gmm_2 = GaussianMixture(n_components=2).fit(x_2)

# Get the means and standard deviations of the two Gaussian components
mu1_2, mu2_2 = gmm_2.means_.flatten()
sigma1_2, sigma2_2 = np.sqrt(gmm_2.covariances_.flatten())

# Visualize the results
plt.hist(i_2, density=True, bins = 50)
# x_axis = i_2.unique()
x_axis_2 = np.linspace(min(i_2), max(i_2), 3000)
plt.plot(x_axis_2, 0.5 * np.exp(-(x_axis_2 - mu1_2)**2 / (2 * sigma1_2**2))
plt.plot(x_axis_2, 0.5 * np.exp(-(x_axis_2 - mu2_2)**2 / (2 * sigma2_2**2))
plt.legend()
plt.show()
```

```
0       0.2600
1       0.2400
2       0.2375
3       0.1700
4       0.2325
         ...
3594    0.1150
3595    0.1050
3596    0.1825
3597    0.1875
3598    0.1800
Name: value, Length: 3599, dtype: float64
[[0.26      ]
 [0.24      ]
 [0.2375    ]
 ...
 [0.18249999]
 [0.1875    ]
 [0.17999999]]
```



In [20]:
```
print('The mean of the 1st normal distribution = ', mu1_2)
print('The mean of the 2nd normal distribution = ', mu2_2)
```

```
The mean of the 1st normal distribution =  0.26746443243580437
The mean of the 2nd normal distribution =  0.14746206711022558
```

In [21]:
```
print('The std of the 1st normal distribution = ', sigma1_2)
print('The std of the 2nd normal distribution = ', sigma2_2)
```

```
The std of the 1st normal distribution =  0.04141630919302785
The std of the 2nd normal distribution =  0.04599689616140695
```

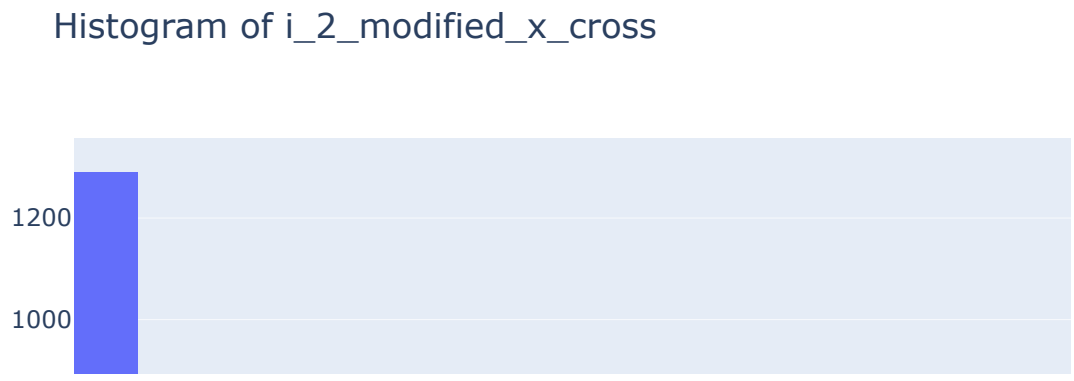
In [22]:
```
# Define the two normal distributions
mean1, std1 = mu1_2, sigma1_2
mean2, std2 = mu2_2, sigma2_2

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_2 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_2[0])
```

```
Cross point of two normal distributions: x = 0.20894456536927047
```

In [23]:
```
# What if we reduce by x_cross

# If we were to reduce the values by x_cross value as mentioned above
i_2_modified_x_cross = i_2.apply(lambda x: max(0, x - x_cross_2[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_2_modified_x_cross = np.nanmean(i_2_modified_x_cross.values)
std_i_2_modified_x_cross = np.nanstd(i_2_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_2_modified_x_cross = i_2_modified_x_cross.isna().sum().sum()
num_zeros_2_modified_x_cross = (i_2_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_2_modified_x_cross_max = i_2_modified_x_cross.max().max()
i_2_modified_x_cross_min = i_2_modified_x_cross.min().min()
# i_2_modified_x_cross_min = 0
i_2_modified_x_cross_mode = i_2_modified_x_cross.mode()[0]

print('Number of nan values in i_2_modified_x_cross = ', num_nan_2_modified_
print('Number of 0 values in i_2_modified_x_cross = ', num_zeros_2_modified_
print('Number of values in i_2_modified_x_cross = ', len(i_2_modified_x_cros
print('-------------------------------------------------------')
print('Mean of values in i_2_modified_x_cross = ', mean_i_2_modified_x_cross
print('STD of 1-D values in i_2_modified_x_cross = ', std_i_2_modified_x_cro
print('Mean value >= STD =====> ', mean_i_2_modified_x_cross >= std_i_2_modi
print('-------------------------------------------------------')
print("Max value in i_2_modified_x_cross = ", i_2_modified_x_cross_max)
print("Min value in i_2_modified_x_cross = ", i_2_modified_x_cross_min)
print('Mode value in i_2_modified_x_cross = ', i_2_modified_x_cross_mode)
```

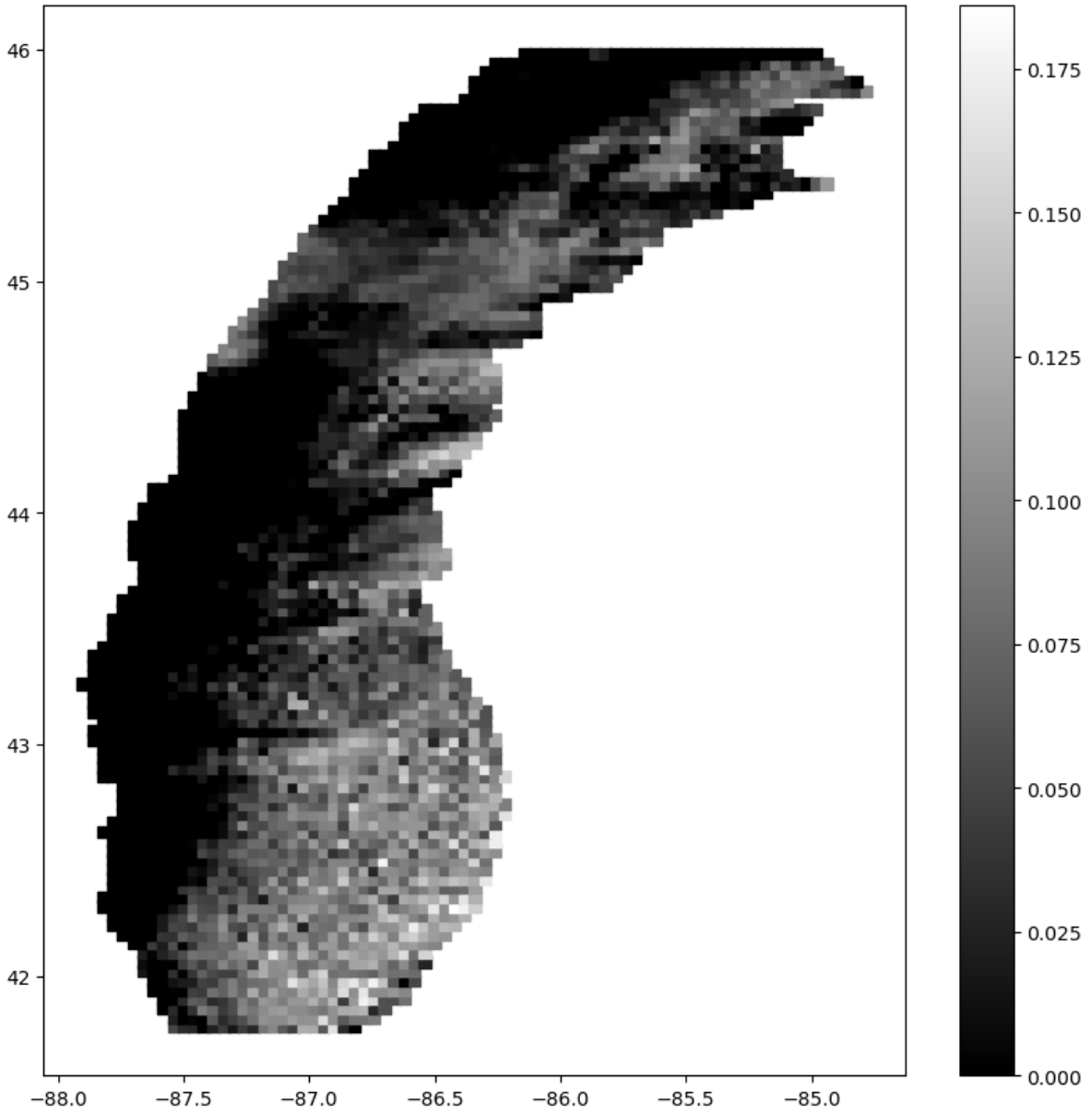
```
Number of nan values in i_2_modified_x_cross =  0
Number of 0 values in i_2_modified_x_cross =  1249
Number of values in i_2_modified_x_cross =  3599
--------------------------------------------------
Mean of values in i_2_modified_x_cross =  0.041269870545211
STD of 1-D values in i_2_modified_x_cross =  0.04220211110194128
Mean value >= STD =====>  False
--------------------------------------------------
Max value in i_2_modified_x_cross =  0.1860554146307295
Min value in i_2_modified_x_cross =  0.0
Mode value in i_2_modified_x_cross =  0.0
```

In [24]:
```
fig = px.histogram(i_2_modified_x_cross.values.flatten(), title="Histogram c
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [25]:
```
# x_2, y_2, i_2 = df_2.longitude, df_2.latitude, df_2.value
plt.figure(figsize=(10, 10))
plt.scatter(df_2.longitude, y_2.values, c=i_2_modified_x_cross.values, cmap=
plt.colorbar(orientation='vertical')
len(x_2), len(y_2), len(i_2_modified_x_cross)
```

Out[25]: (3599, 3599, 3599)



In [26]:

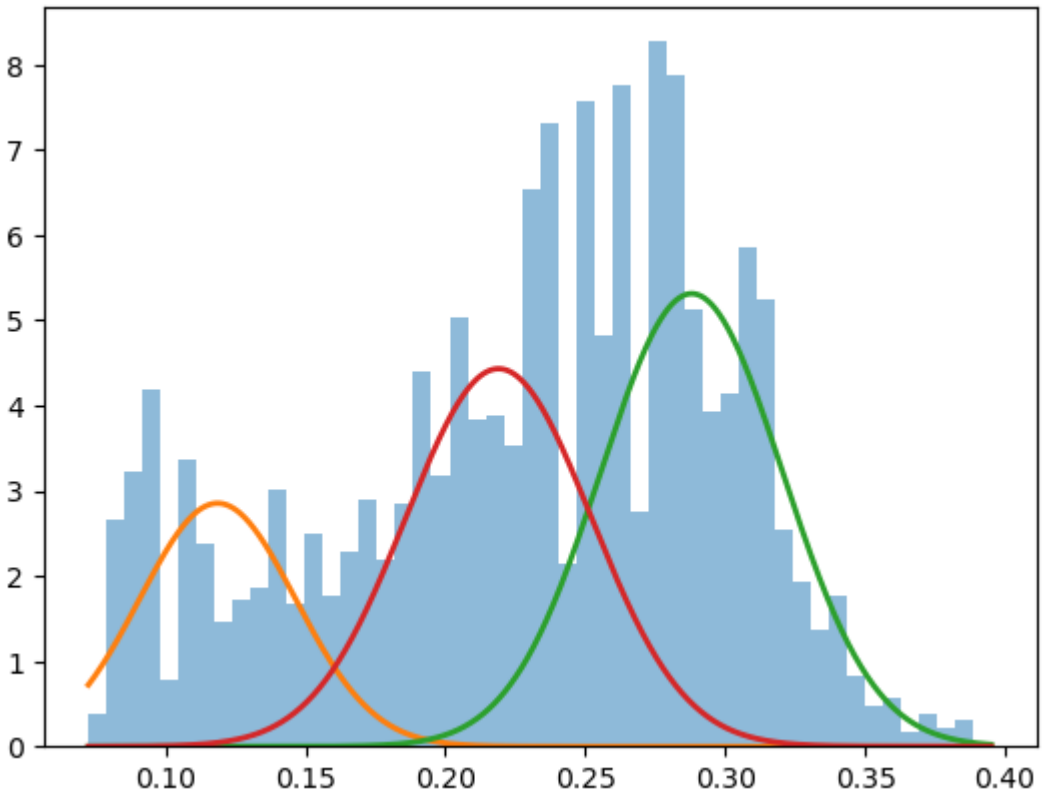
```
# Load the 1-D data into a numpy array 'data'
data = df_2.value.to_numpy()

# Define the range of number of components (i.e., Gaussian distributions) to
n_components_range = range(1, 11)

# Fit the GMM model with different number of components and calculate the AI
lowest_aic = float("inf")
best_n_components = 1

for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, covariance_type='full')
    gmm.fit(data.reshape(-1, 1))
    aic = gmm.aic(data.reshape(-1, 1))
    if aic < lowest_aic:
        lowest_aic = aic
        best_n_components = n_components
```

```
# Print the number of Gaussian distributions that give the lowest AIC score
print("Number of Gaussian distributions:", best_n_components)
```

```
Number of Gaussian distributions: 10
```

In [27]:

```
from scipy.stats import norm

# generate some 1D data with multiple Gaussian distributions
np.random.seed(42)
x = df_2.value.to_numpy()

# fit Gaussian mixture model to the data
gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(x.reshape(-1, 1))

# get parameters of each Gaussian distribution
mus, sigmas, weights = zip(*[(mu[0], sigma[0], weight) for mu, sigma, weight

# plot the data and the fitted Gaussian distributions
plt.hist(x, bins=50, density=True, alpha=0.5)
x_range = np.linspace(x.min(), x.max(), num=1000)
for mu, sigma, weight in zip(mus, sigmas, weights):
    plt.plot(x_range, norm.pdf(x_range, loc=mu, scale=sigma) * weight, linew
plt.show()

# get the number of Gaussian distributions
n_components = gmm.n_components

print(f"Number of Gaussian distributions in the data: {n_components}")
```




```
Number of Gaussian distributions in the data: 3
```

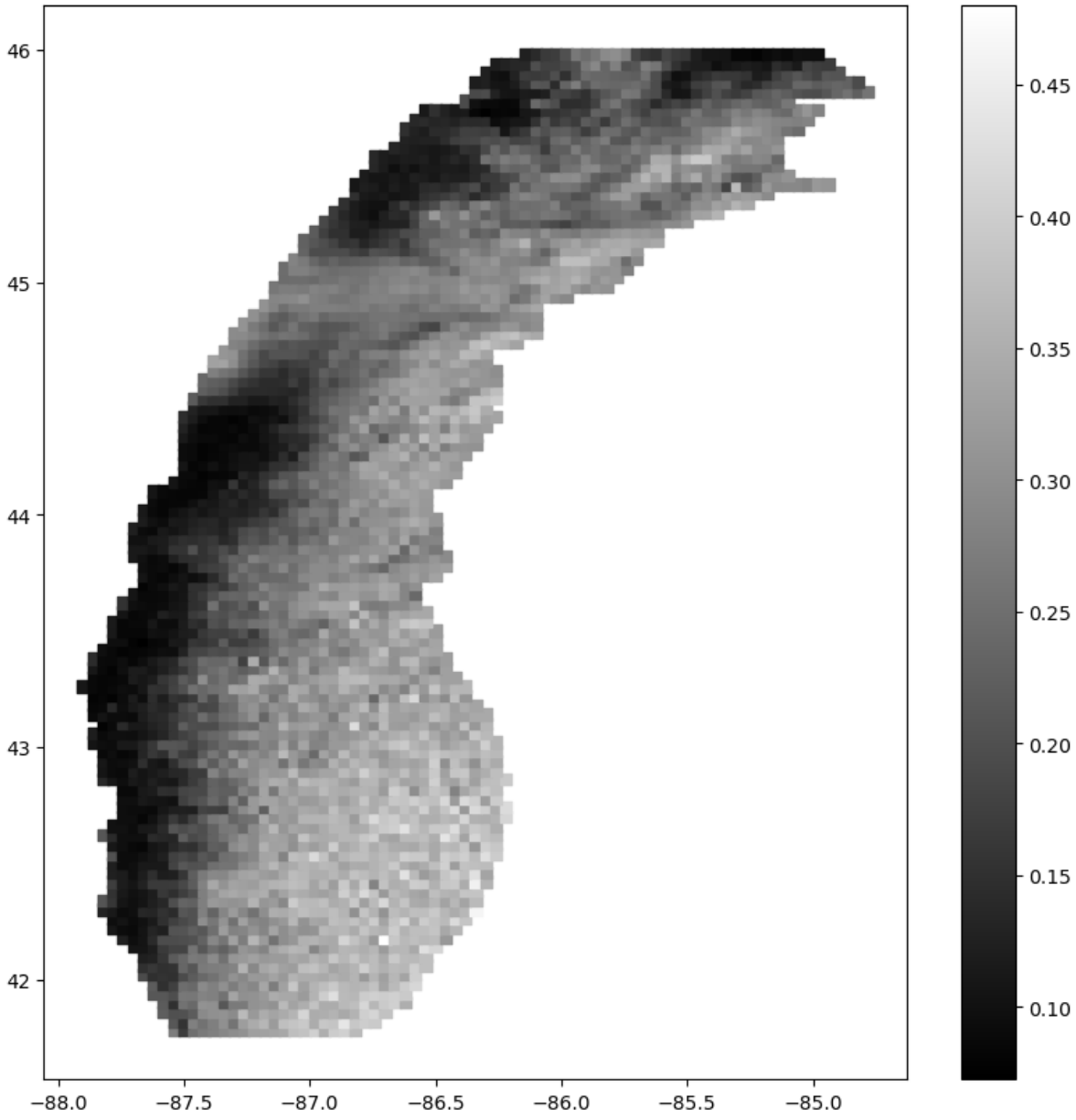
1.4 2016.12.08.1530

At `1530`, we assume that the sun has already gone up since **15:30 UTC** is **10:30 CDT**.

As shown in the picture above, the cloud is visible in this case. Now, let's dive into the same process just to see what is going on.

In [28]:
```
df_3 = pd.read_csv(zone_0_folder_path + file_list[3])

x_3, y_3, i_3 = df_3.longitude, df_3.latitude, df_3.value
plt.figure(figsize=(10, 10))
plt.scatter(x_3.values, y_3.values, c=i_3.values, cmap=cm.gray, marker='s')
plt.colorbar(orientation='vertical')
len(x_3), len(y_3), len(i_3)
```

Out[28]:
```
(3599, 3599, 3599)
```



In [29]:

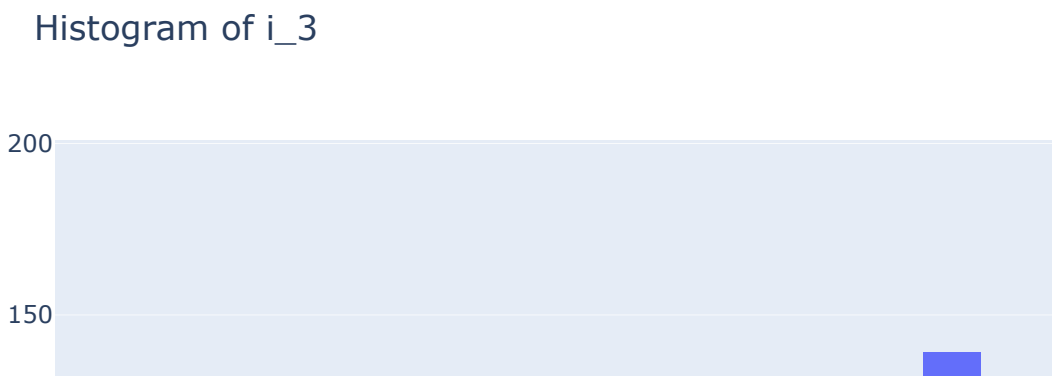
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_3 = np.nanmean(i_3.values)
std_i_3 = np.nanstd(i_3.values)

# Check number of nan values and 0s in i, just in case
num_nan_3 = i_3.isna().sum().sum()
num_zeros_3 = (i_3 == 0).sum().sum()

# Check min and max in i

i_3_max = i_3.max().max()
i_3_min = i_3.min().min()
i_3_mode = i_3.mode()[0]

print('Number of nan values in i_3 = ', num_nan_3)
print('Number of 0 values in i_3 = ', num_zeros_3)
print('Number of values in i_3 = ', len(i_3))
```

```
print('-----------------------------------------------------')
print('Mean of values in i_3 = ', mean_i_3)
print('STD of 1-D values in i_3 = ', std_i_3)
print('Mean value >= STD =====> ', mean_i_3 >= std_i_3)
print('-----------------------------------------------------')
print("Max value in i_3 = ", i_3_max)
print("Min value in i_3 = ", i_3_min)
print('Mode value in i_3 = ', i_3_mode)
```

```
Number of nan values in i_3 =  0
Number of 0 values in i_3 =  0
Number of values in i_3 =  3599
-----------------------------------------------------
Mean of values in i_3 =  0.2517206131269797
STD of 1-D values in i_3 =  0.08945432948660141
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_3 =  0.48
Min value in i_3 =  0.0725
Mode value in i_3 =  0.3175
```

In [30]:

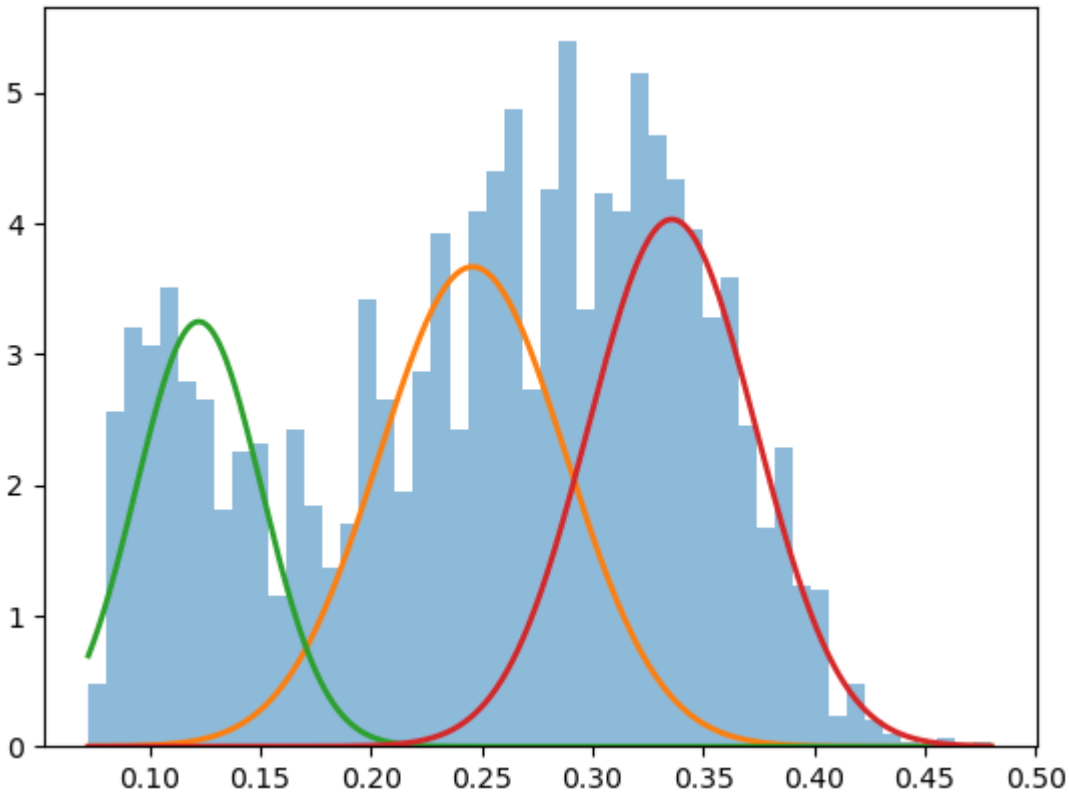
```
# # Plot histogram
# plt.hist(i_3.values.flatten(), bins=50)

# # Set plot labels and title
# plt.xlabel('Value')
# plt.ylabel('Count')
# plt.title('Distribution of Values in DataFrame i_3')

# # Display plot
# plt.show()

fig = px.histogram(i_3.values.flatten(), title="Histogram of i_3")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



In [31]:

```
from scipy.stats import norm

# generate some 1D data with multiple Gaussian distributions
np.random.seed(42)
x = df_3.value.to_numpy()

# fit Gaussian mixture model to the data
gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(x.reshape(-1, 1))

# get parameters of each Gaussian distribution
mus, sigmas, weights = zip(*[(mu[0], sigma[0], weight) for mu, sigma, weight

# plot the data and the fitted Gaussian distributions
plt.hist(x, bins=50, density=True, alpha=0.5)
x_range = np.linspace(x.min(), x.max(), num=1000)
for mu, sigma, weight in zip(mus, sigmas, weights):
    plt.plot(x_range, norm.pdf(x_range, loc=mu, scale=sigma) * weight, linew
plt.show()

# get the number of Gaussian distributions
n_components = gmm.n_components
```

```
print(f"Number of Gaussian distributions in the data: {n_components}")
```




```
Number of Gaussian distributions in the data: 3
```

1.5 2016.12.08.1545

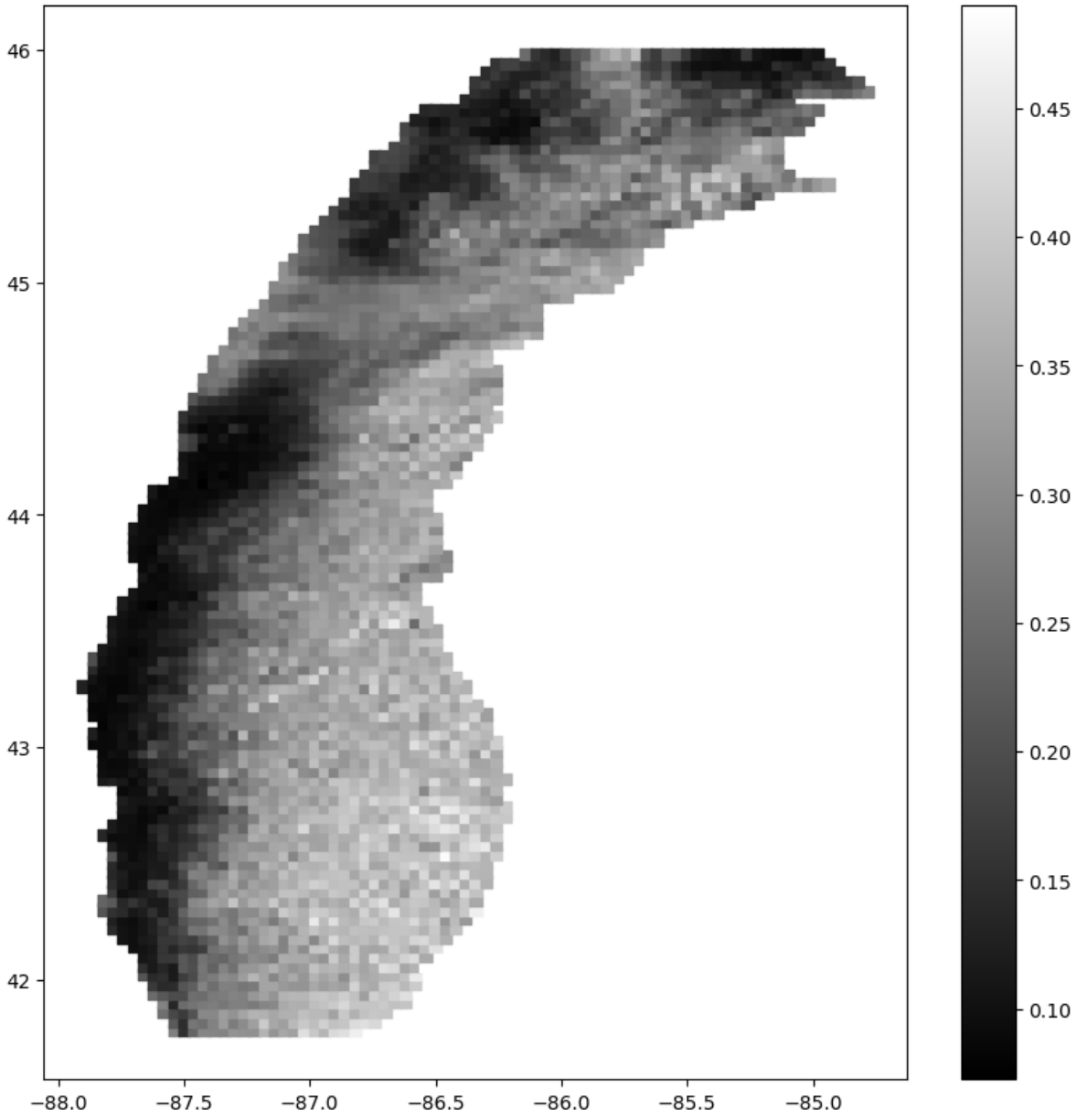
At `1545`, we assume that the sun has already gone up since **15:45 UTC** is **10:45 CDT**.

image

As shown in the picture above, the cloud is visible in this case. Now, let's dive into the same process just to see what is going on.

In [32]:
```
df_4 = pd.read_csv(zone_0_folder_path + file_list[4])

x_4, y_4, i_4 = df_4.longitude, df_4.latitude, df_4.value
plt.figure(figsize=(10, 10))
plt.scatter(x_4.values, y_4.values, c=i_4.values, cmap=cm.gray, marker='s')
plt.colorbar(orientation='vertical')
len(x_4), len(y_4), len(i_4)
```

Out[32]:
```
(3599, 3599, 3599)
```



In [33]:

```
# Calculate the mean and standard deviation of the 1-D array

mean_i_4 = np.nanmean(i_4.values)
std_i_4 = np.nanstd(i_4.values)

# Check number of nan values and 0s in i, just in case
num_nan_4 = i_4.isna().sum().sum()
num_zeros_4 = (i_4 == 0).sum().sum()

# Check min and max in i

i_4_max = i_4.max().max()
i_4_min = i_4.min().min()
i_4_mode = i_4.mode()[0]

print('Number of nan values in i_4 = ', num_nan_4)
print('Number of 0 values in i_4 = ', num_zeros_4)
print('Number of values in i_4 = ', len(i_4))
```

```
print('-----------------------------------------------------')
print('Mean of values in i_4 = ', mean_i_4)
print('STD of 1-D values in i_4 = ', std_i_4)
print('Mean value >= STD =====> ', mean_i_4 >= std_i_4)
print('-----------------------------------------------------')
print("Max value in i_4 = ", i_4_max)
print("Min value in i_4 = ", i_4_min)
print('Mode value in i_4 = ', i_4_mode)
```

```
Number of nan values in i_4 =  0
Number of 0 values in i_4 =  0
Number of values in i_4 =  3599
-----------------------------------------------------
Mean of values in i_4 =  0.26068838185968324
STD of 1-D values in i_4 =  0.0964002509087906
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_4 =  0.48999998
Min value in i_4 =  0.0725
Mode value in i_4 =  0.3375
```

In [34]:
```
# # Plot histogram
# plt.hist(i_4.values.flatten(), bins=50)

# # Set plot labels and title
# plt.xlabel('Value')
# plt.ylabel('Count')
# plt.title('Distribution of Values in DataFrame i_4')

# # Display plot
# plt.show()
```

In [35]:
```
fig = px.histogram(i_4.values.flatten(), title="Histogram of i_4")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```

1.6 Combine 2016.12.08.1500 till 2016.12.08.1700

At `1700`, we assume that the sun has already gone up since **17:00 UTC** is **12:00 CDT**.

We now attempt to add the available set of data together and perform a simple analysis.

In [36]:
```
df_5 = pd.read_csv(zone_0_folder_path + file_list[5])
df_6 = pd.read_csv(zone_0_folder_path + file_list[6])
df_7 = pd.read_csv(zone_0_folder_path + file_list[7])
df_8 = pd.read_csv(zone_0_folder_path + file_list[8])
df_9 = pd.read_csv(zone_0_folder_path + file_list[9])

x_5, y_5, i_5 = df_4.longitude, df_5.latitude, df_5.value
x_6, y_6, i_6 = df_6.longitude, df_6.latitude, df_6.value
x_7, y_7, i_7 = df_7.longitude, df_7.latitude, df_7.value
x_8, y_8, i_8 = df_8.longitude, df_8.latitude, df_8.value
x_9, y_9, i_9 = df_9.longitude, df_9.latitude, df_9.value
```

```
# ...of dataframes to be appended
df_case_0 = pd.concat([df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9], ignc
```

```
df_case_0
```

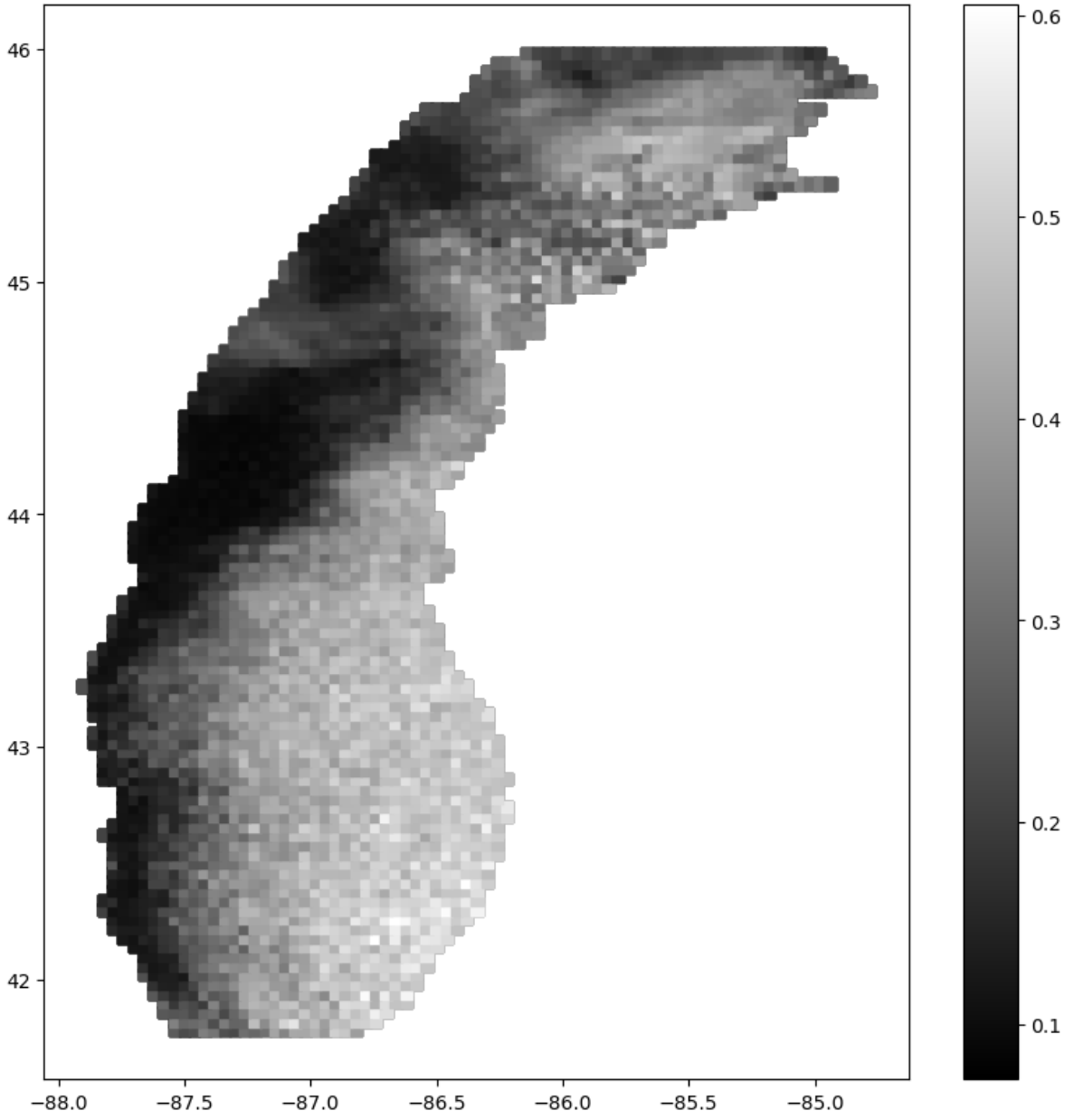
Out[37]:

	corresponding row	value	datetime	latitude	longitude	partition
0	7942	0.2600	2016-12-08 15:00:00	41.78	-87.54	0
1	7943	0.2400	2016-12-08 15:00:00	41.78	-87.50	0
2	7944	0.2375	2016-12-08 15:00:00	41.78	-87.46	0
3	7945	0.1700	2016-12-08 15:00:00	41.78	-87.42	0
4	7946	0.2325	2016-12-08 15:00:00	41.78	-87.38	0
...	...	...	...	...	...	...
28787	23438	0.2775	2016-12-08 17:00:00	45.98	-85.14	0
28788	23439	0.2200	2016-12-08 17:00:00	45.98	-85.10	0
28789	23440	0.2000	2016-12-08 17:00:00	45.98	-85.06	0
28790	23441	0.1650	2016-12-08 17:00:00	45.98	-85.02	0
28791	23442	0.1825	2016-12-08 17:00:00	45.98	-84.98	0

28792 rows × 6 columns

In [38]:
```
x_case_0, y_case_0, i_case_0 = df_case_0.longitude, df_case_0.latitude, df_c
plt.figure(figsize=(10, 10))
plt.scatter(x_case_0.values, y_case_0.values, c=i_case_0.values, cmap=cm.gra
plt.colorbar(orientation='vertical')
len(x_case_0), len(y_case_0), len(i_case_0)
```

Out[38]:
```
(28792, 28792, 28792)
```



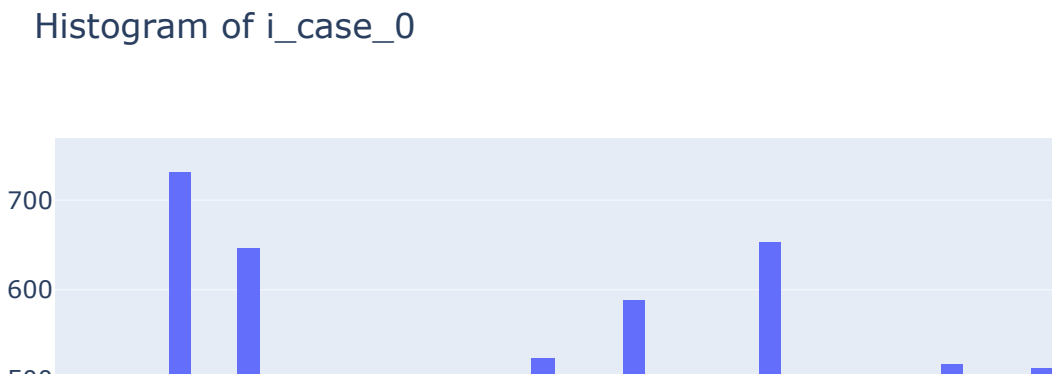
In [39]:

```
# Calculate the mean and standard deviation of the 1-D array

mean_i_case_0 = np.nanmean(i_case_0.values)
std_i_case_0 = np.nanstd(i_case_0.values)

# Check number of nan values and 0s in i, just in case
num_nan_case_0 = i_case_0.isna().sum().sum()
num_zeros_case_0 = (i_case_0 == 0).sum().sum()

# Check min and max in i

i_case_0_max = i_case_0.max().max()
i_case_0_min = i_case_0.min().min()
i_case_0_mode = i_case_0.mode()[0]

print('Number of nan values in i_case_0 = ', num_nan_case_0)
print('Number of 0 values in i_case_0 = ', num_zeros_case_0)
print('Number of values in i_case_0 = ', len(i_case_0))
```

```
print('-------------------------------------------------------')
print('Mean of values in i_case_0 = ', mean_i_case_0)
print('STD of 1-D values in i_case_0 = ', std_i_case_0)
print('Mean value >= STD =====> ', mean_i_case_0 >= std_i_case_0)
print('-------------------------------------------------------')
print("Max value in i_case_0 = ", i_case_0_max)
print("Min value in i_case_0 = ", i_case_0_min)
print('Mode value in i_case_0 = ', i_case_0_mode)
```

```
Number of nan values in i_case_0 =  0
Number of 0 values in i_case_0 =  0
Number of values in i_case_0 =  28792
-------------------------------------------------------
Mean of values in i_case_0 =  0.27718706176153096
STD of 1-D values in i_case_0 =  0.11081211274999676
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_case_0 =  0.60499996
Min value in i_case_0 =  0.0725
Mode value in i_case_0 =  0.29
```

In [40]:
```
fig = px.histogram(i_case_0.values.flatten(), title="Histogram of i_case_0")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```

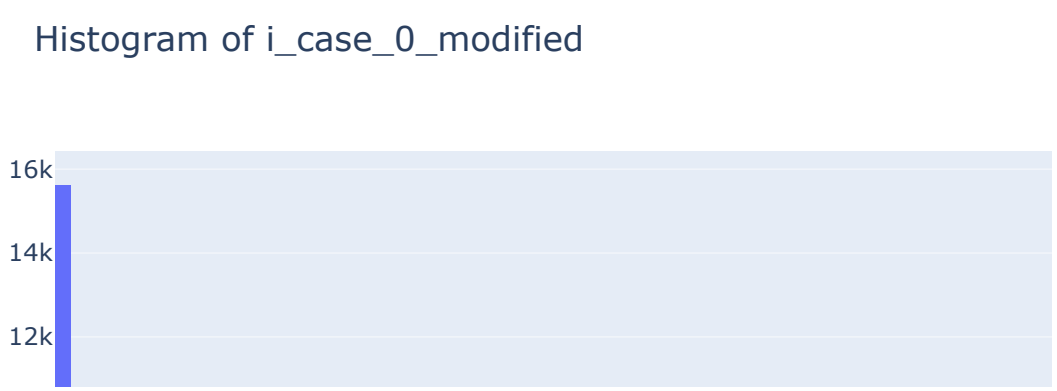


In [41]:
```
# If we were to reduce the values by the mode
i_case_0_modified = i_case_0.apply(lambda x: max(0, x - i_case_0_mode))
```

In [42]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_case_0_modified = np.nanmean(i_case_0_modified.values)
std_i_case_0_modified = np.nanstd(i_case_0_modified.values)

# Check number of nan values and 0s in i, just in case
num_nan_case_0_modified = i_case_0_modified.isna().sum().sum()
num_zeros_case_0_modified = (i_case_0_modified == 0).sum().sum()

# Check min and max in i

i_case_0_modified_max = i_case_0_modified.max().max()
i_case_0_modified_min = i_case_0_modified.min().min()
i_case_0_modified_mode = i_case_0_modified.mode()[0]

print('Number of nan values in i_case_0_modified = ', num_nan_case_0_modifie
print('Number of 0 values in i_case_0_modified = ', num_zeros_case_0_modifie
print('Number of values in i_case_0_modified = ', len(i_case_0_modified))
print('-------------------------------------------------------')
print('Mean of values in i_case_0_modified = ', mean_i_case_0_modified)
print('STD of 1-D values in i_case_0_modified = ', std_i_case_0_modified)
print('Mean value >= STD =====> ', mean_i_case_0_modified >= std_i_case_0_mo
print('-------------------------------------------------------')
print("Max value in i_case_0_modified = ", i_case_0_modified_max)
print("Min value in i_case_0_modified = ", i_case_0_modified_min)
print('Mode value in i_case_0_modified = ', i_case_0_modified_mode)
```

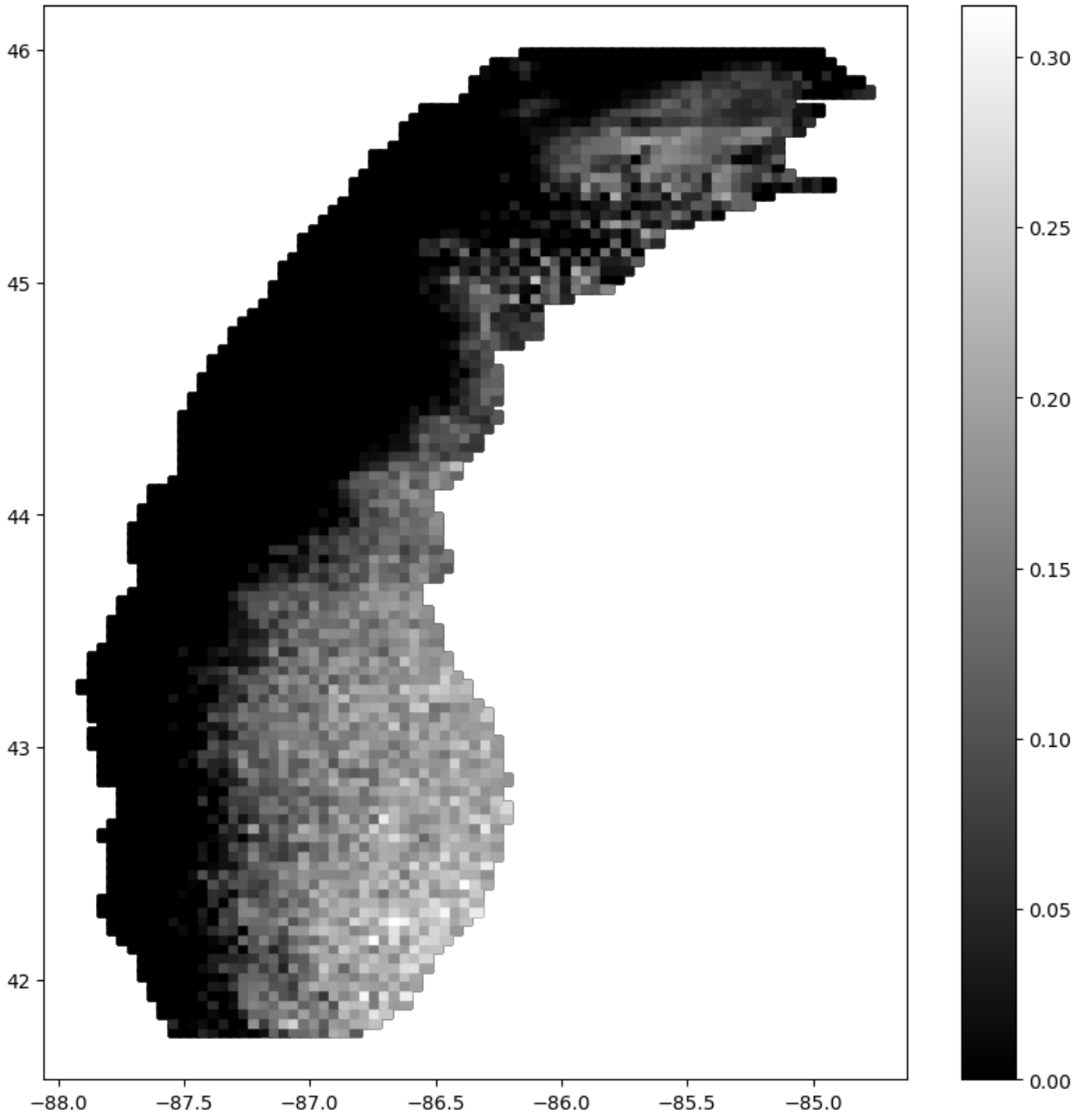
```
Number of nan values in i_case_0_modified =  0
Number of 0 values in i_case_0_modified =  15613
Number of values in i_case_0_modified =  28792
-------------------------------------------------------
Mean of values in i_case_0_modified =  0.04018398925257016
STD of 1-D values in i_case_0_modified =  0.05911838400063641
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_case_0_modified =  0.31499996
Min value in i_case_0_modified =  0.0
Mode value in i_case_0_modified =  0.0
```

In [43]:
```
fig = px.histogram(i_case_0_modified.values.flatten(), title="Histogram of i
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [44]:
```
# x_case_0, y_case_0, i_case_0 = df_case_0.longitude, df_case_0.latitude, df
plt.figure(figsize=(10, 10))
plt.scatter(x_case_0.values, y_case_0.values, c=i_case_0_modified.values, cm
plt.colorbar(orientation='vertical')
len(x_case_0), len(y_case_0), len(i_case_0_modified)
```

Out[44]: (28792, 28792, 28792)



In [45]:
```
# What if we reduce by mean - std of the sum

mean_minus_std_case_0 = mean_i_case_0 - std_i_case_0

print(mean_minus_std_case_0)
```

```
0.1663749490115342
```

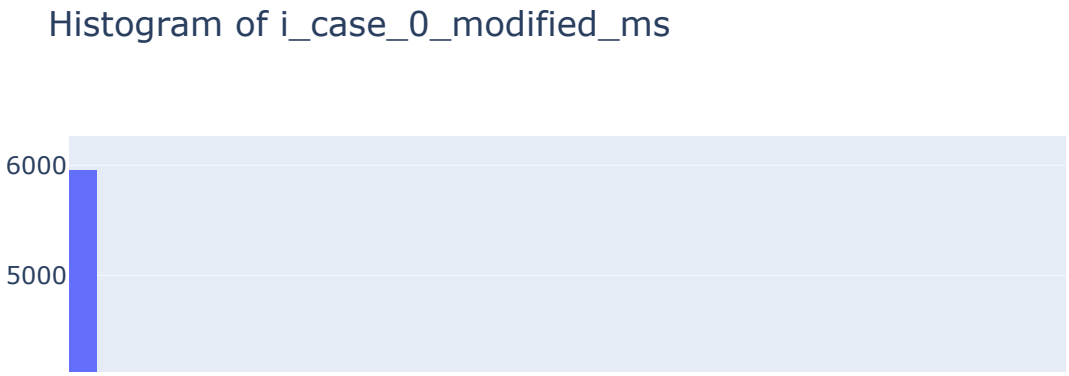
In [46]:
```
# If we were to reduce the values by the mode
i_case_0_modified_ms = i_case_0.apply(lambda x: max(0, x - mean_minus_std_ca
```

In [47]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_case_0_modified_ms = np.nanmean(i_case_0_modified_ms.values)
std_i_case_0_modified_ms = np.nanstd(i_case_0_modified_ms.values)

# Check number of nan values and 0s in i, just in case
odified_ms = i_case_0_modified_ms.isna().sum().sum()
```

```
num_zeros_case_0_modified_ms = (i_case_0_modified_ms == 0).sum().sum()

# Check min and max in i

i_case_0_modified_ms_max = i_case_0_modified_ms.max().max()
i_case_0_modified_ms_min = i_case_0_modified_ms.min().min()
i_case_0_modified_ms_mode = i_case_0_modified_ms.mode()[0]

print('Number of nan values in i_case_0_modified_ms = ', num_nan_case_0_modi
print('Number of 0 values in i_case_0_modified_ms = ', num_zeros_case_0_modi
print('Number of values in i_case_0_modified_ms = ', len(i_case_0_modified_m
print('-------------------------------------------------------')
print('Mean of values in i_case_0_modified_ms = ', mean_i_case_0_modified_ms
print('STD of 1-D values in i_case_0_modified_ms = ', std_i_case_0_modified_
print('Mean value >= STD =====> ', mean_i_case_0_modified_ms >= std_i_case_0
print('-------------------------------------------------------')
print("Max value in i_case_0_modified_ms = ", i_case_0_modified_ms_max)
print("Min value in i_case_0_modified_ms = ", i_case_0_modified_ms_min)
print('Mode value in i_case_0_modified_ms = ', i_case_0_modified_ms_mode)
```

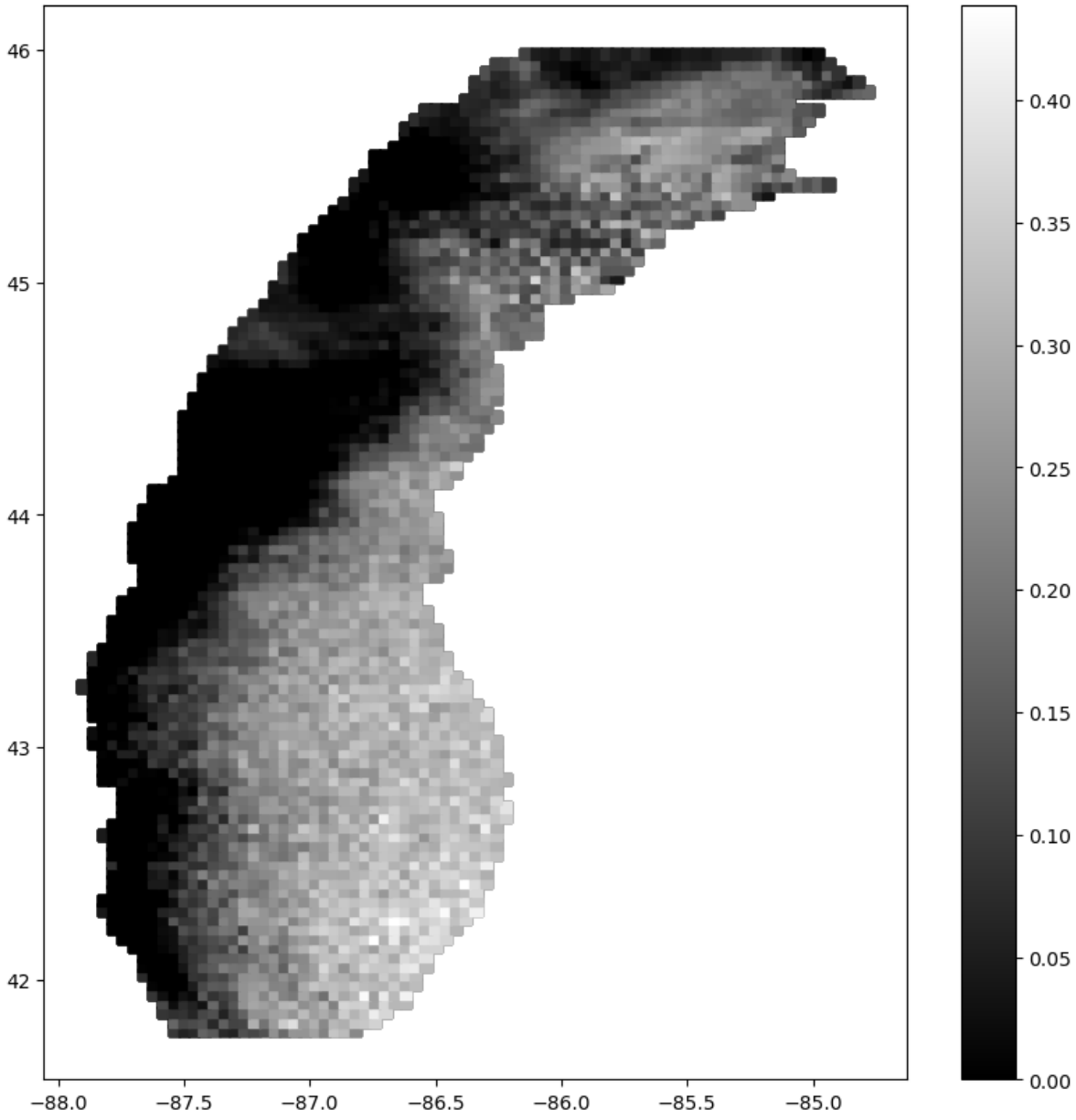
```
Number of nan values in i_case_0_modified_ms =  0
Number of 0 values in i_case_0_modified_ms =  5776
Number of values in i_case_0_modified_ms =  28792
-------------------------------------------------------
Mean of values in i_case_0_modified_ms =  0.11982118526571715
STD of 1-D values in i_case_0_modified_ms =  0.09840863260882153
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_case_0_modified_ms =  0.4386250109884658
Min value in i_case_0_modified_ms =  0.0
Mode value in i_case_0_modified_ms =  0.0
```

In [48]:
```
fig = px.histogram(i_case_0_modified_ms.values.flatten(), title="Histogram c
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [49]:
```
# x_case_0, y_case_0, i_case_0 = df_case_0.longitude, df_case_0.latitude, df
plt.figure(figsize=(10, 10))
plt.scatter(x_case_0.values, y_case_0.values, c=i_case_0_modified_ms.values,
plt.colorbar(orientation='vertical')
len(x_case_0), len(y_case_0), len(i_case_0_modified_ms)
```

Out[49]: (28792, 28792, 28792)



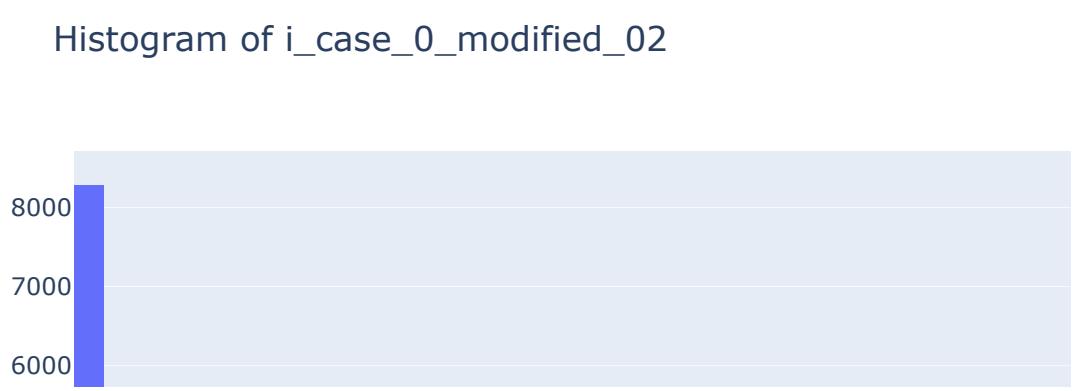
In [50]:
```
# What if we reduce by 0.2

# If we were to reduce the values by 0.2 (hard cap)
i_case_0_modified_02 = i_case_0.apply(lambda x: max(0, x - 0.2))
```

In [51]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_case_0_modified_02 = np.nanmean(i_case_0_modified_02.values)
std_i_case_0_modified_02 = np.nanstd(i_case_0_modified_02.values)

# Check number of nan values and 0s in i, just in case
num_nan_case_0_modified_02 = i_case_0_modified_02.isna().sum().sum()
num_zeros_case_0_modified_02 = (i_case_0_modified_02 == 0).sum().sum()

# Check min and max in i

i_case_0_modified_02_max = i_case_0_modified_02.max().max()
i_case_0_modified_02_min = i_case_0_modified_02.min().min()
```

```
i_case_0_modified_02_mode = i_case_0_modified_02.mode()[0]

print('Number of nan values in i_case_0_modified_02 = ', num_nan_case_0_modi
print('Number of 0 values in i_case_0_modified_02 = ', num_zeros_case_0_modi
print('Number of values in i_case_0_modified_02 = ', len(i_case_0_modified_0
print('-----------------------------------------------------')
print('Mean of values in i_case_0_modified_02 = ', mean_i_case_0_modified_02
print('STD of 1-D values in i_case_0_modified_02 = ', std_i_case_0_modified_
print('Mean value >= STD =====> ', mean_i_case_0_modified_02 >= std_i_case_0
print('-----------------------------------------------------')
print("Max value in i_case_0_modified_02 = ", i_case_0_modified_02_max)
print("Min value in i_case_0_modified_02 = ", i_case_0_modified_02_min)
print('Mode value in i_case_0_modified_02 = ', i_case_0_modified_02_mode)
```

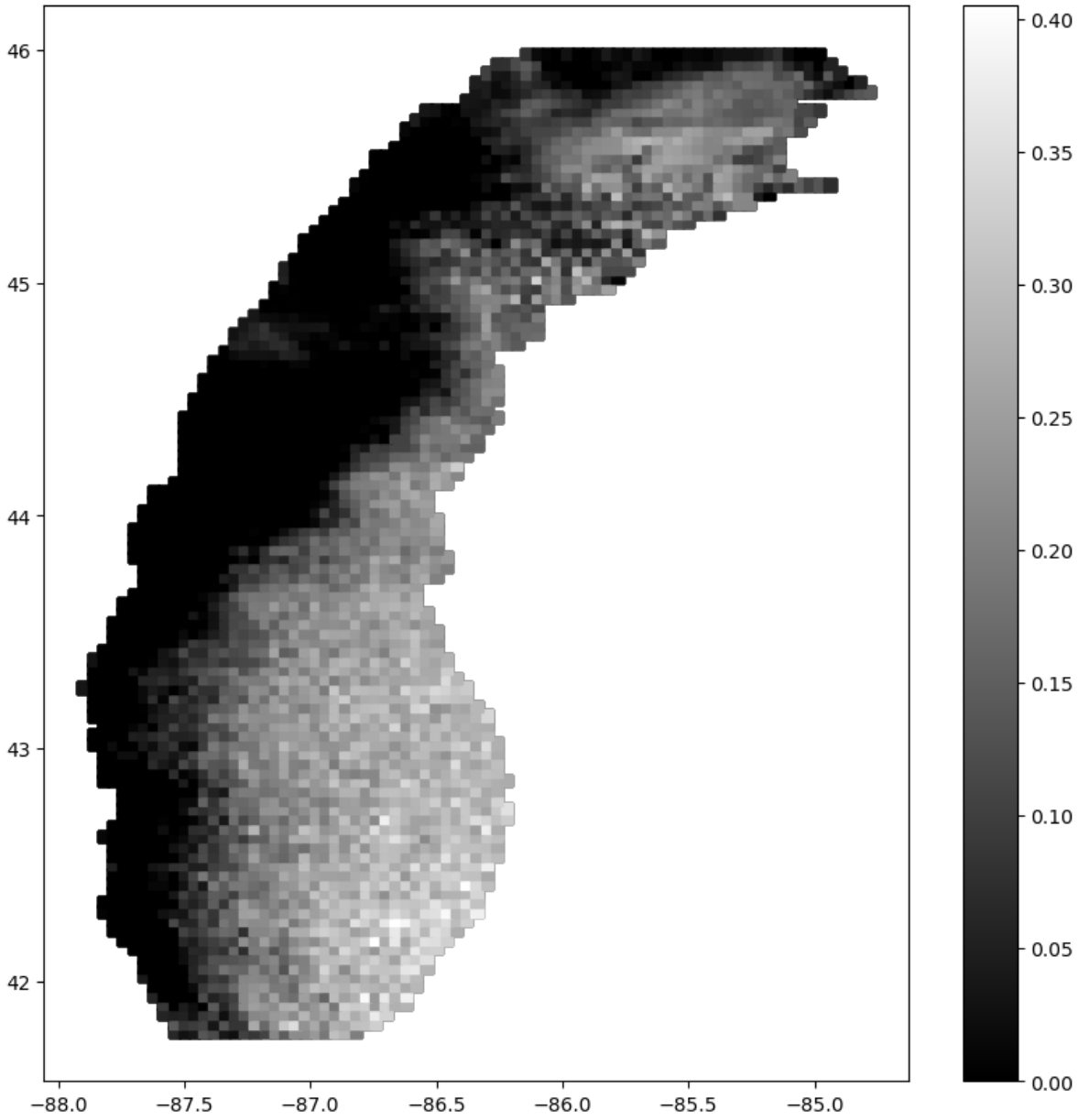
```
Number of nan values in i_case_0_modified_02 =  0
Number of 0 values in i_case_0_modified_02 =  8108
Number of values in i_case_0_modified_02 =  28792
-----------------------------------------------------
Mean of values in i_case_0_modified_02 =  0.09422790712420116
STD of 1-D values in i_case_0_modified_02 =  0.08932539277076187
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_case_0_modified_02 =  0.40499996
Min value in i_case_0_modified_02 =  0.0
Mode value in i_case_0_modified_02 =  0.0
```

In [52]:
```
fig = px.histogram(i_case_0_modified_02.values.flatten(), title="Histogram c
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [53]:
```
# x_case_0, y_case_0, i_case_0 = df_case_0.longitude, df_case_0.latitude, df
plt.figure(figsize=(10, 10))
plt.scatter(x_case_0.values, y_case_0.values, c=i_case_0_modified_02.values,
plt.colorbar(orientation='vertical')
len(x_case_0), len(y_case_0), len(i_case_0_modified_02)
```

Out[53]: (28792, 28792, 28792)



Observation:

It still seems to me that we shall deduct `mean - 1 x std` as the threshold, across the given day.

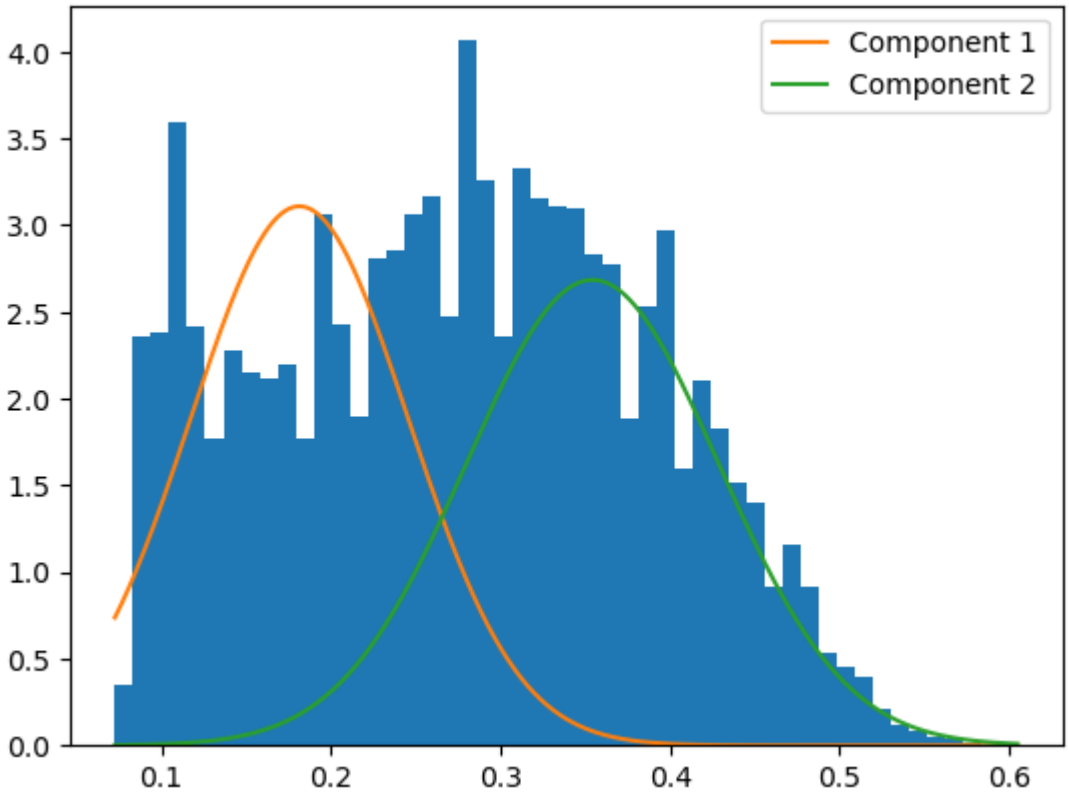
EXPERIMENT:

Call me stoopid if you will, let's try to see if it is going to split into 2 normal distributions?

In [54]:
```
xx_case_0 = np.array(i_case_0).reshape(-1, 1)
```

```
# Fit a mixture of two Gaussians to the data
gmm_case_0 = GaussianMixture(n_components=2).fit(xx_case_0)

# Get the means and standard deviations of the two Gaussian components
mu1_case_0, mu2_case_0 = gmm_case_0.means_.flatten()
sigma1_case_0, sigma2_case_0 = np.sqrt(gmm_case_0.covariances_.flatten())

# Visualize the results
plt.hist(i_case_0, density=True, bins = 50)
# x_axis = i_11.unique()
xx_axis_case_0 = np.linspace(min(i_case_0), max(i_case_0), 3000)
plt.plot(xx_axis_case_0, 0.5 * np.exp(-(xx_axis_case_0 - mu1_case_0)**2 / (2
plt.plot(xx_axis_case_0, 0.5 * np.exp(-(xx_axis_case_0 - mu2_case_0)**2 / (2
plt.legend()
plt.show()
```

```
OpenBLAS warning: precompiled NUM_THREADS exceeded, adding auxiliary array
for thread metadata.
OpenBLAS warning: precompiled NUM_THREADS exceeded, adding auxiliary array
for thread metadata.
```



In [55]:
```
print('The mean of the 1st normal distribution = ', mu1_case_0)
print('The mean of the 2nd normal distribution = ', mu2_case_0)
```

```
The mean of the 1st normal distribution =  0.18134201527889374
The mean of the 2nd normal distribution =  0.3542286740841984
```

In [56]:
```
print('The std of the 1st normal distribution = ', sigma1_case_0)
print('The std of the 2nd normal distribution = ', sigma2_case_0)
```

```
The std of the 1st normal distribution =  0.0641704881533064
nd normal distribution =  0.07430995337301305
```

In [57]:
```
# Define the two normal distributions
mean1, std1 = mu1_case_0, sigma1_case_0
mean2, std2 = mu2_case_0, sigma2_case_0

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_case_0 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_case_0[0])
```

```
Cross point of two normal distributions: x = 0.2654884369574946
```

My $0.02 here

`What-if` : We filter based on the cross point?

The cross point of two normal distributions is the point where the two distributions intersect. This point is significant because it represents the value at which the probability of observing a particular outcome is equal under both distributions.

When two normal distributions intersect, they have the same mean, but they may have different standard deviations. The point of intersection is also known as the "crossover point" or "break-even point."

The location of the cross point can be used to make statistical inferences about the populations represented by the two distributions. For example, if the cross point is close to the mean of one distribution but far from the mean of the other distribution, this may indicate that the two populations are distinct and have different characteristics.

The cross point can also be used to calculate the probability that an observation comes from one distribution versus the other. This can be useful in a variety of fields, including finance, economics, and psychology, where researchers may be interested in comparing the characteristics of different groups or populations.

Therefore, when it comes to the current situation, we assume that there are two groups in the picture, one group of pixels are non-cloud pixels, another group would inevitably be the cloud pixels. Then, the value of intensity at the cross point/intersect shall be considered as the threshold value.

In [58]:
```
# What if we reduce by x_cross

# If we were to reduce the values by x_cross value as mentioned above
i_case_0_modified_x_cross = i_case_0.apply(lambda x: max(0, x - x_cross_case
```

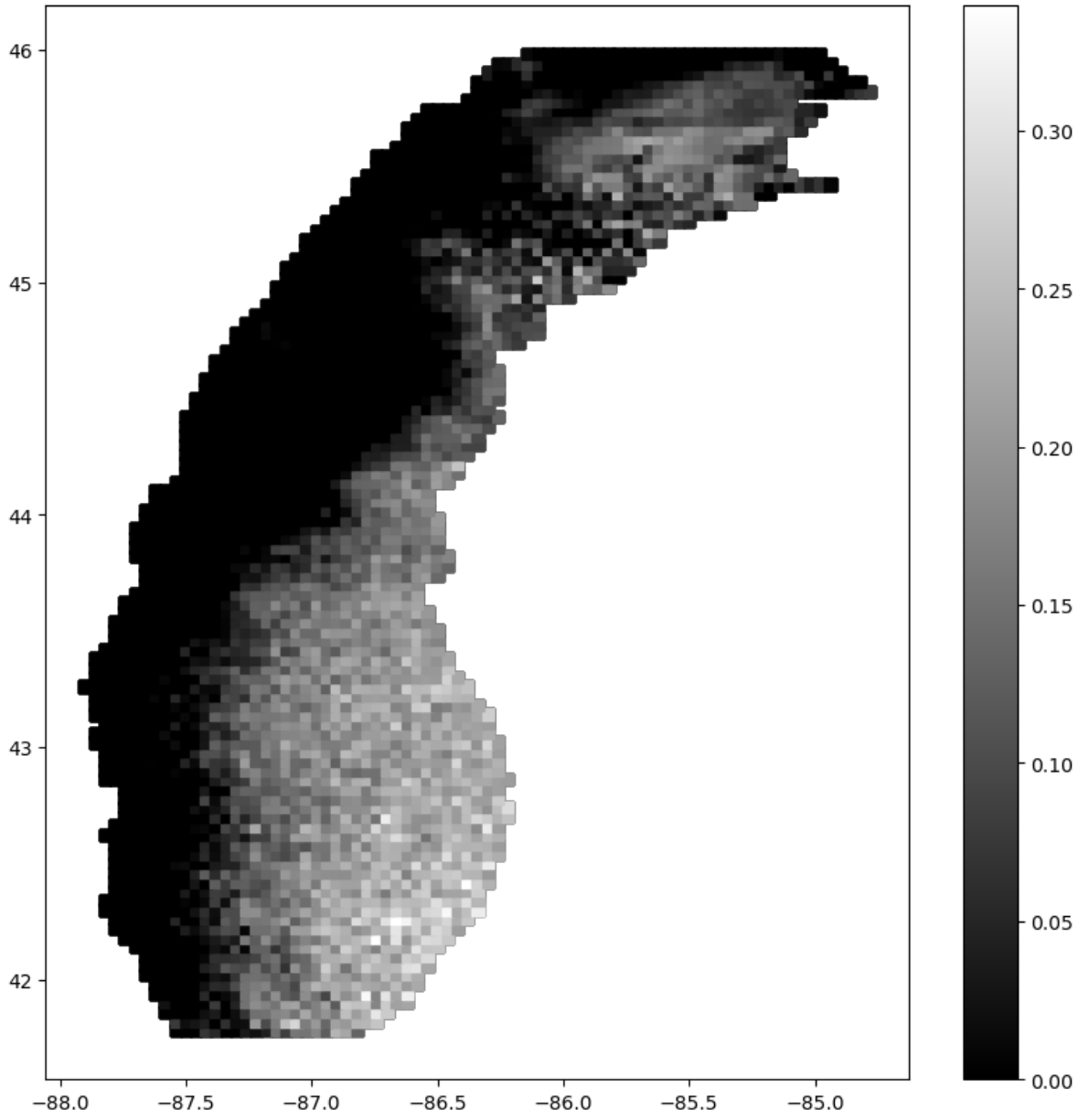
In [59]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_case_0_modified_x_cross = np.nanmean(i_case_0_modified_x_cross.values
std_i_case_0_modified_x_cross = np.nanstd(i_case_0_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_case_0_modified_x_cross = i_case_0_modified_x_cross.isna().sum().sum
num_zeros_case_0_modified_x_cross = (i_case_0_modified_x_cross == 0).sum().s

# Check min and max in i

i_case_0_modified_x_cross_max = i_case_0_modified_x_cross.max().max()
i_case_0_modified_x_cross_min = i_case_0_modified_x_cross.min().min()
# i_case_0_modified_x_cross_min = 0
i_case_0_modified_x_cross_mode = i_case_0_modified_x_cross.mode()[0]

print('Number of nan values in i_case_0_modified_x_cross = ', num_nan_case_0
print('Number of 0 values in i_case_0_modified_x_cross = ', num_zeros_case_0
print('Number of values in i_case_0_modified_x_cross = ', len(i_case_0_modif
print('-------------------------------------------------------')
print('Mean of values in i_case_0_modified_x_cross = ', mean_i_case_0_modifi
print('STD of 1-D values in i_case_0_modified_x_cross = ', std_i_case_0_modi
print('Mean value >= STD =====> ', mean_i_case_0_modified_x_cross >= std_i_c
print('-------------------------------------------------------')
print("Max value in i_case_0_modified_x_cross = ", i_case_0_modified_x_cross
print("Min value in i_case_0_modified_x_cross = ", i_case_0_modified_x_cross
print('Mode value in i_case_0_modified_x_cross = ', i_case_0_modified_x_cros
```

```
Number of nan values in i_case_0_modified_x_cross =  0
Number of 0 values in i_case_0_modified_x_cross =  13332
Number of values in i_case_0_modified_x_cross =  28792
-------------------------------------------------------
Mean of values in i_case_0_modified_x_cross =  0.05253676986236225
STD of 1-D values in i_case_0_modified_x_cross =  0.06793152690583791
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_case_0_modified_x_cross =  0.3395115230425054
Min value in i_case_0_modified_x_cross =  0.0
Mode value in i_case_0_modified_x_cross =  0.0
```

In [60]:
```
fig = px.histogram(i_case_0_modified_x_cross.values.flatten(), title="Histog
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```

In [61]:
```
# x_case_0, y_case_0, i_case_0 = df_case_0.longitude, df_case_0.latitude, df
plt.figure(figsize=(10, 10))
plt.scatter(x_case_0.values, y_case_0.values, c=i_case_0_modified_x_cross.va
plt.colorbar(orientation='vertical')
len(x_case_0), len(y_case_0), len(i_case_0_modified_x_cross)
```

Out[61]:
```
(28792, 28792, 28792)
```



In [ ]:

In [ ]:

In [ ]:

2. Case 2

`Duration` : 2016.12.09 1200-1600 UTC

image

> [Q]:

> In this case, we can see that the day starts at 1300 UTC, aka 8AM CDT. Around that time, cloud was barely visible. However, in each 15 mins, the intensity of the cloud goes up significantly, we can see clouds that sort follow SLAP pattern (short-lake axis parallel). Just want to check with you if you think at 1500 (the second image), if the cloud exists. The intensity of the cloud is low across the board.

> [A]:

> Yes, this is a good use case. I think we need to ensure that the threshold is not higher than the 1500 image, because that is a definite cloud

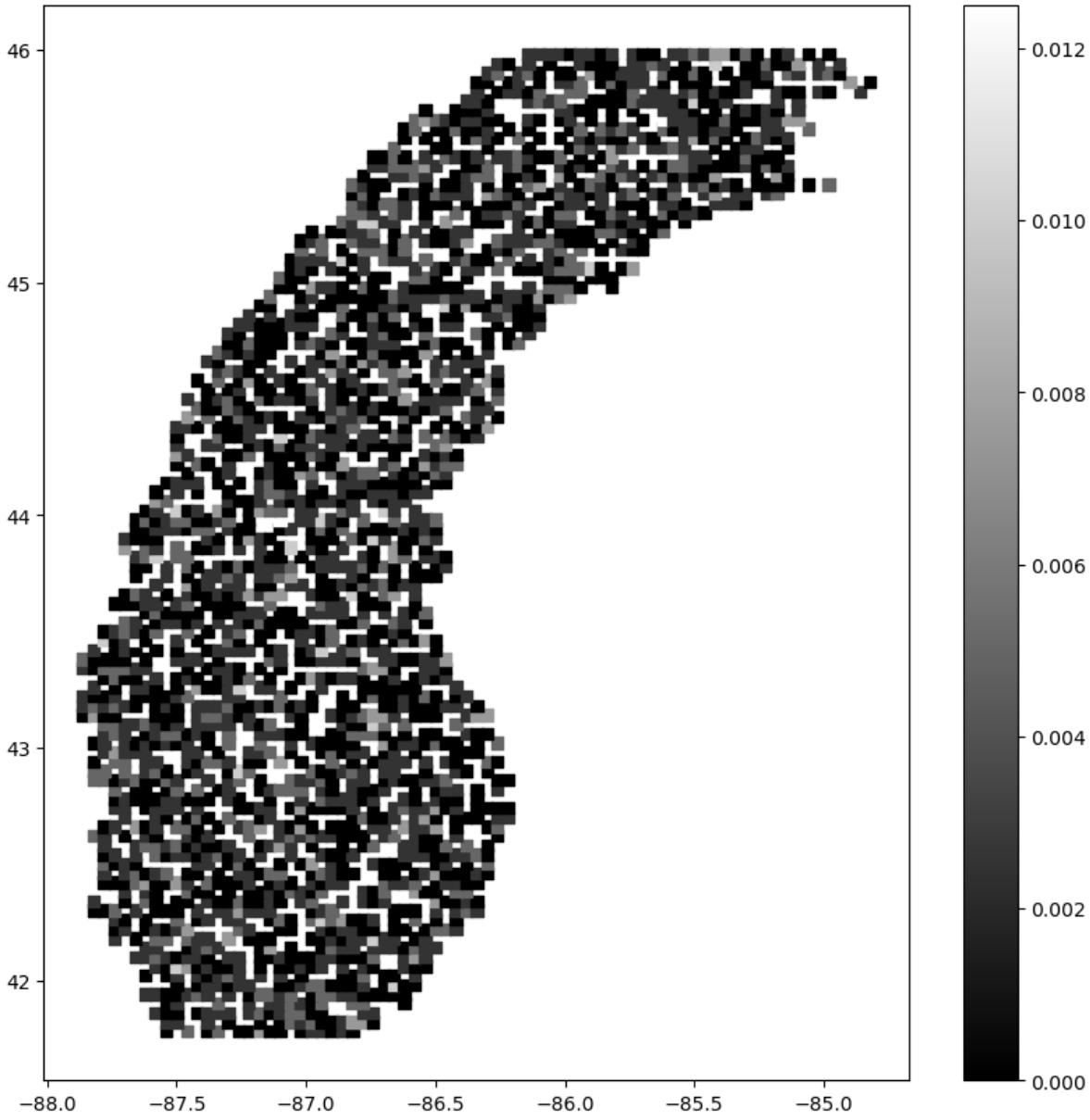
2.1 2016.12.09.1200

But before this, let's take a look at what happened at `1200`.

As we can see, there are blue noises on the image in the Lake Michigan area.

In [62]:
```
df_10 = pd.read_csv(zone_0_folder_path + file_list[10])

x_10, y_10, i_10 = df_10.longitude, df_10.latitude, df_10.value
plt.figure(figsize=(10, 10))
plt.scatter(x_10.values, y_10.values, c=i_10.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_10), len(y_10), len(i_10)
```

Out[62]:
```
(2418, 2418, 2418)
```



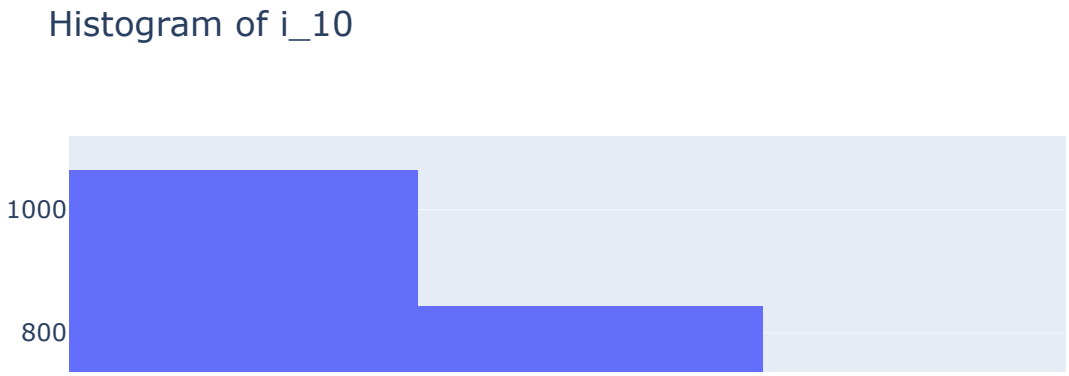
In [63]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_10 = np.nanmean(i_10.values)
std_i_10 = np.nanstd(i_10.values)

# Check number of nan values and 0s in i, just in case
num_nan_10 = i_10.isna().sum().sum()
num_zeros_10 = (i_10 == 0).sum().sum()

# Check min and max in i

i_10_max = i_10.max().max()
i_10_min = i_10.min().min()
i_10_mode = i_10.mode()[0]

print('Number of nan values in i_10 = ', num_nan_10)
print('Number of 0 values in i_10 = ', num_zeros_10)
print('                values in i_10 = ', len(i_10))
print('-------------------------------------------------------')
```

```
print('Mean of values in i_10 = ', mean_i_10)
print('STD of 1-D values in i_10 = ', std_i_10)
print('Mean value >= STD =====> ', mean_i_10 >= std_i_10)
print('-------------------------------------------------------')
print("Max value in i_10 = ", i_10_max)
print("Min value in i_10 = ", i_10_min)
print('Mode value in i_10 = ', i_10_mode)
```

```
Number of nan values in i_10 =  0
Number of 0 values in i_10 =  1064
Number of values in i_10 =  2418
-------------------------------------------------------
Mean of values in i_10 =  0.002081265507444169
STD of 1-D values in i_10 =  0.0022845347341692427
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_10 =  0.012499999
Min value in i_10 =  0.0
Mode value in i_10 =  0.0
```

In [64]:

```
fig = px.histogram(i_10.values.flatten(), title="Histogram of i_10")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



In [65]:
```
len(i_10) - len(i_0)
```

Out[65]: 72

In [66]:
```
# Since they have different length, we try to perform an union join, aka out
df_i_0_i_10 = pd.merge(df_0, df_10, on=['longitude', 'latitude'], how='outer
df_i_0_i_10
```

Out[66]:

	corresponding row_x	value_x	datetime_x	latitude	longitude	partition_x	corresponding row_y	v
0	5360.0	0.0025	2016-12-08 12:00:00	41.78	-87.54	0.0	5305.0	
1	5361.0	0.0025	2016-12-08 12:00:00	41.78	-87.50	0.0	NaN	
2	5362.0	0.0025	2016-12-08 12:00:00	41.78	-87.46	0.0	5306.0	
3	5363.0	0.0000	2016-12-08 12:00:00	41.78	-87.42	0.0	NaN	
4	5364.0	0.0000	2016-12-08 12:00:00	41.78	-87.38	0.0	5307.0	
...	...	...	...	...	...	...	...	
3166	NaN	NaN	NaN	45.98	-85.50	NaN	18118.0	
3167	NaN	NaN	NaN	45.98	-85.46	NaN	18119.0	
3168	NaN	NaN	NaN	45.98	-85.34	NaN	18122.0	
3169	NaN	NaN	NaN	45.98	-85.30	NaN	18123.0	
3170	NaN	NaN	NaN	45.98	-84.98	NaN	18128.0	

3171 rows × 10 columns

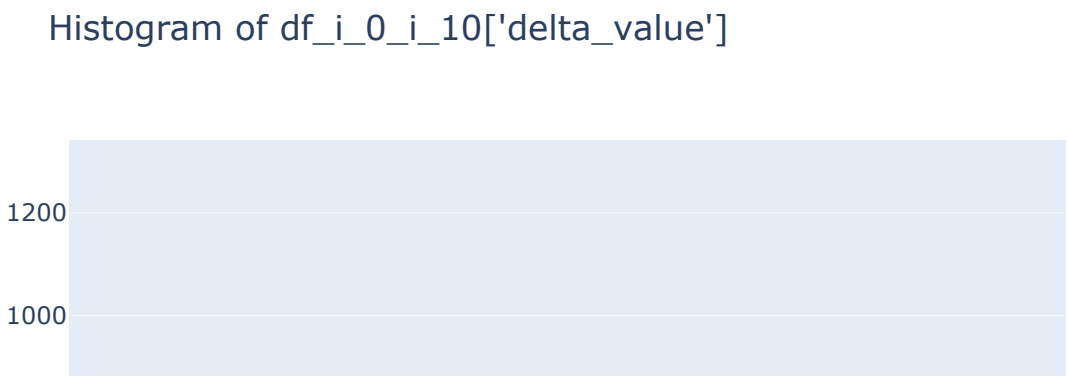
In [67]:
```
df_i_0_i_10 = df_i_0_i_10.fillna(0)

df_i_0_i_10['delta_value'] = df_i_0_i_10['value_x'] - df_i_0_i_10['value_y']
df_i_0_i_10
```

Out[67]:

	corresponding row_x	value_x	datetime_x	latitude	longitude	partition_x	corresponding row_y	v
0	5360.0	0.0025	2016-12-08 12:00:00	41.78	-87.54	0.0	5305.0	
1	5361.0	0.0025	2016-12-08 12:00:00	41.78	-87.50	0.0	0.0	
2	5362.0	0.0025	2016-12-08 12:00:00	41.78	-87.46	0.0	5306.0	
3	5363.0	0.0000	2016-12-08 12:00:00	41.78	-87.42	0.0	0.0	
4	5364.0	0.0000	2016-12-08 12:00:00	41.78	-87.38	0.0	5307.0	
...	...	...	...	...	...	...	...	
3166	0.0	0.0000	0	45.98	-85.50	0.0	18118.0	
3167	0.0	0.0000	0	45.98	-85.46	0.0	18119.0	
3168	0.0	0.0000	0	45.98	-85.34	0.0	18122.0	
3169	0.0	0.0000	0	45.98	-85.30	0.0	18123.0	
3170	0.0	0.0000	0	45.98	-84.98	0.0	18128.0	

3171 rows × 11 columns

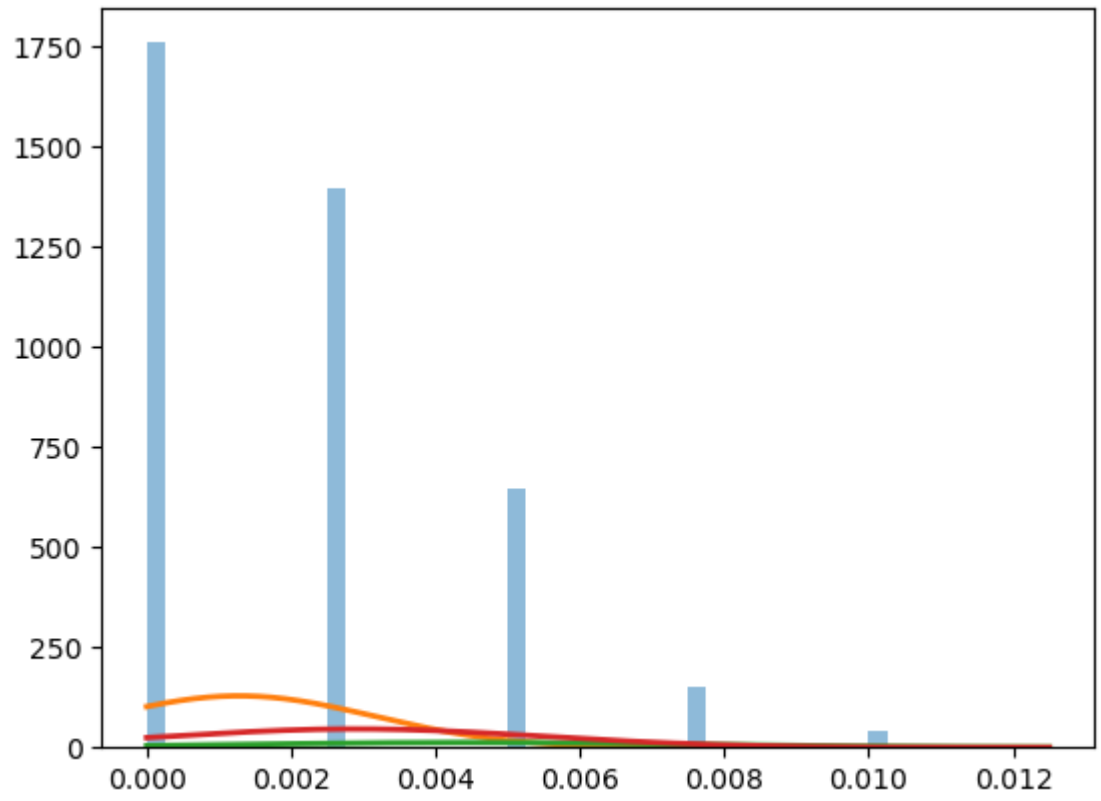
In [68]:
```
fig = px.histogram(df_i_0_i_10['delta_value'], title="Histogram of df_i_0_i_
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



It seems that there is little to no difference between days at `1200` before sun goes up. And the max values are all under 0.02. Let's dive into other hours and see the differences.

In [69]:
```
from scipy.stats import norm

# generate some 1D data with multiple Gaussian distributions
np.random.seed(42)
x = df_10.value.to_numpy()

# fit Gaussian mixture model to the data
gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(x.reshape(-1, 1))

# get parameters of each Gaussian distribution
mus, sigmas, weights = zip(*[(mu[0], sigma[0], weight) for mu, sigma, weight

# plot the data and the fitted Gaussian distributions
plt.hist(x, bins=50, density=True, alpha=0.5)
x_range = np.linspace(x.min(), x.max(), num=1000)
for mu, sigma, weight in zip(mus, sigmas, weights):
    ange, norm.pdf(x_range, loc=mu, scale=sigma) * weight, linew
plt.show()
```

```
# get the number of Gaussian distributions
n_components = gmm.n_components

print(f"Number of Gaussian distributions in the data: {n_components}")
```




```
Number of Gaussian distributions in the data: 3
```

2.2 2016.12.09.1400

At `1400`, we assume that the sun is going up since **14:00 UTC** is **9:00 CDT**.

image

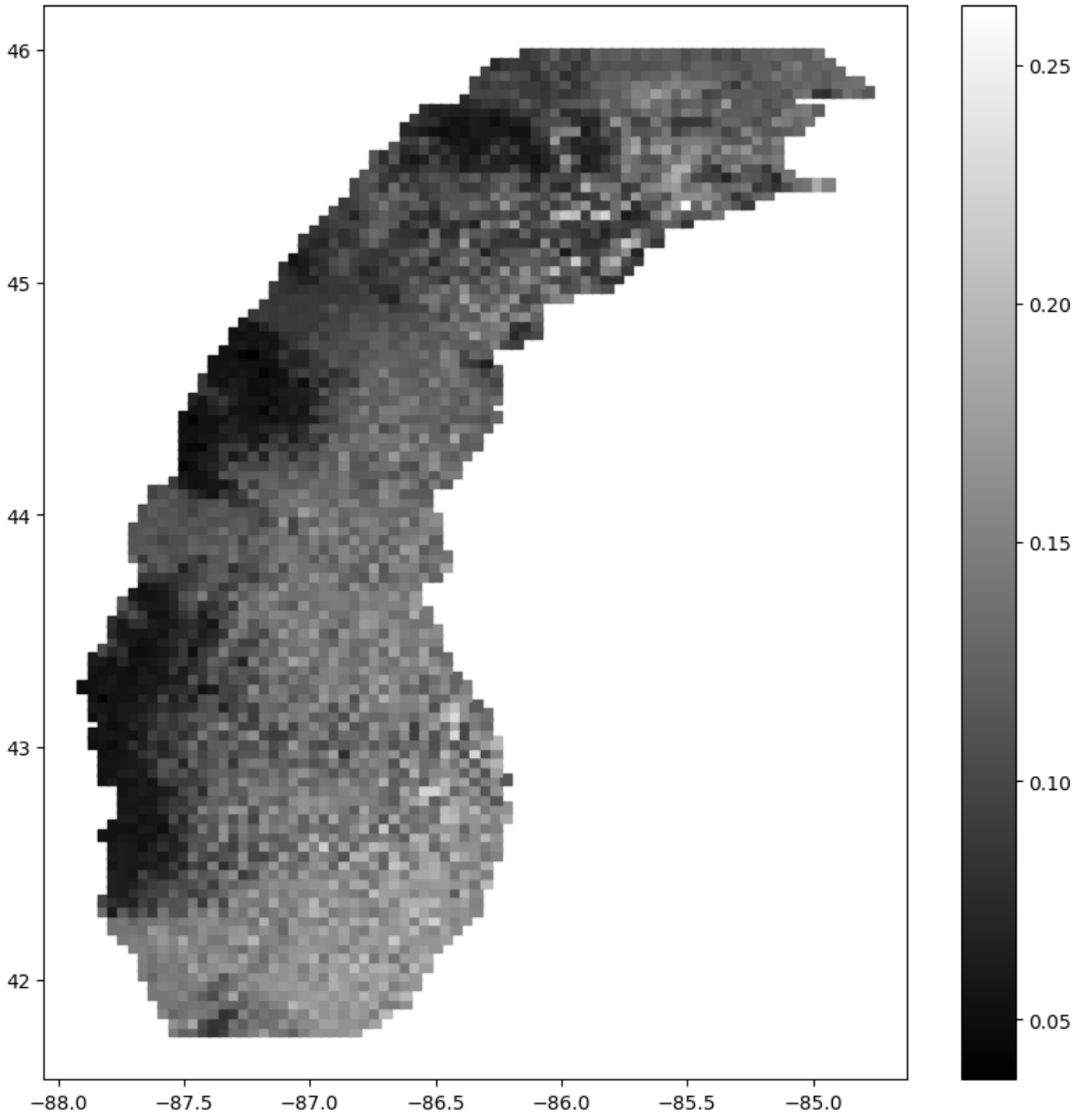
The sun rises, and we can see the cloud clearly.

In [70]:
```
df_11 = pd.read_csv(zone_0_folder_path + file_list[11])

print(file_list[11])
```

```
goes15.2016.12.09.1400.v01.nc-var1-t0.csv
```

In [71]:
```
x_11, y_11, i_11 = df_11.longitude, df_11.latitude, df_11.value
plt.figure(figsize=(10, 10))
plt.scatter(x_11.values, y_11.values, c=i_11.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_11), len(y_11), len(i_11)
```

Out[71]:
```
(3599, 3599, 3599)
```



In [72]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_11 = np.nanmean(i_11.values)
std_i_11 = np.nanstd(i_11.values)

# Check number of nan values and 0s in i, just in case
num_nan_11 = i_11.isna().sum().sum()
num_zeros_11 = (i_11 == 0).sum().sum()

# Check min and max in i

i_11_max = i_11.max().max()
i_11_min = i_11.min().min()
i_11_mode = i_11.mode()[0]

print('Number of nan values in i_11 = ', num_nan_11)
print('Number of 0 values in i_11 = ', num_zeros_11)
print('Number of values in i_11 = ', len(i_11))
```

```
print('-----------------------------------------------------')
print('Mean of values in i_11 = ', mean_i_11)
print('STD of 1-D values in i_11 = ', std_i_11)
print('Mean value >= STD =====> ', mean_i_11 >= std_i_11)
print('-----------------------------------------------------')
print("Max value in i_11 = ", i_11_max)
print("Min value in i_11 = ", i_11_min)
print('Mode value in i_11 = ', i_11_mode)
```

```
Number of nan values in i_11 =  0
Number of 0 values in i_11 =  0
Number of values in i_11 =  3599
-----------------------------------------------------
Mean of values in i_11 =  0.12021603039955543
STD of 1-D values in i_11 =  0.0361471038703567
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_11 =  0.2625
Min value in i_11 =  0.037499998
Mode value in i_11 =  0.1175
```

In [73]:

```
fig = px.histogram(i_11.values.flatten(), title="Histogram of i_11")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```

EXPERIMENT:

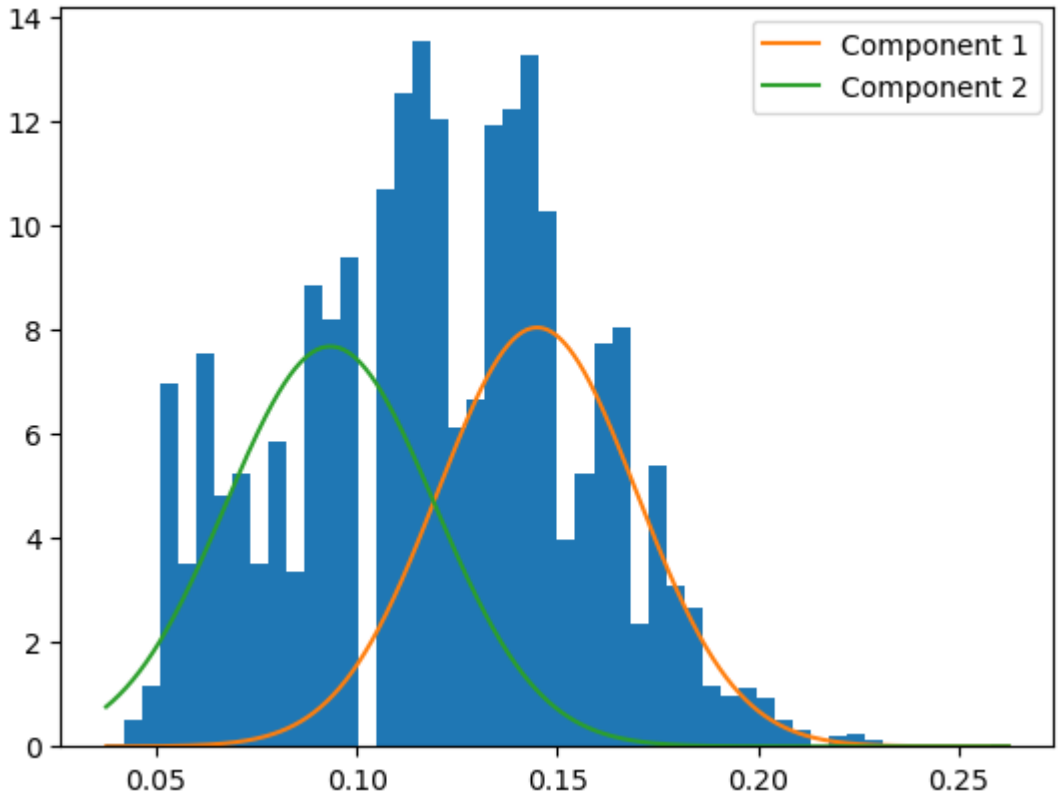
Call me stoopid if you will, let's try to see if it is going to split into 2 normal distributions?

In [74]:

```
print(i_11)
x_11 = np.array(i_11).reshape(-1, 1)
print(x_11)
# Fit a mixture of two Gaussians to the data
gmm_11 = GaussianMixture(n_components=2).fit(x_11)

# Get the means and standard deviations of the two Gaussian components
mu1_11, mu2_11 = gmm_11.means_.flatten()
sigma1_11, sigma2_11 = np.sqrt(gmm_11.covariances_.flatten())

# Visualize the results
plt.hist(i_11, density=True, bins = 50)
# x_axis = i_11.unique()
x_axis_11 = np.linspace(min(i_11), max(i_11), 3000)
11, 0.5 * np.exp(-(x_axis_11 - mu1_11)**2 / (2 * sigma1_11**
11, 0.5 * np.exp(-(x_axis_11 - mu2_11)**2 / (2 * sigma2_11**
```

```
plt.legend()
plt.show()
```

```
0       0.1450
1       0.1225
2       0.1250
3       0.0975
4       0.0800
         ...
3594    0.1275
3595    0.1250
3596    0.1200
3597    0.1200
3598    0.1350
Name: value, Length: 3599, dtype: float64
[[0.145     ]
 [0.1225    ]
 [0.125     ]
 ...
 [0.12      ]
 [0.12      ]
 [0.13499999]]
```



In [75]:
```
print('The mean of the 1st normal distribution = ', mu1_11)
print('The mean of the 2nd normal distribution = ', mu2_11)
```

```
The mean of the 1st normal distribution =  0.14491717727398112
The mean of the 2nd normal distribution =  0.09336999324565276
```

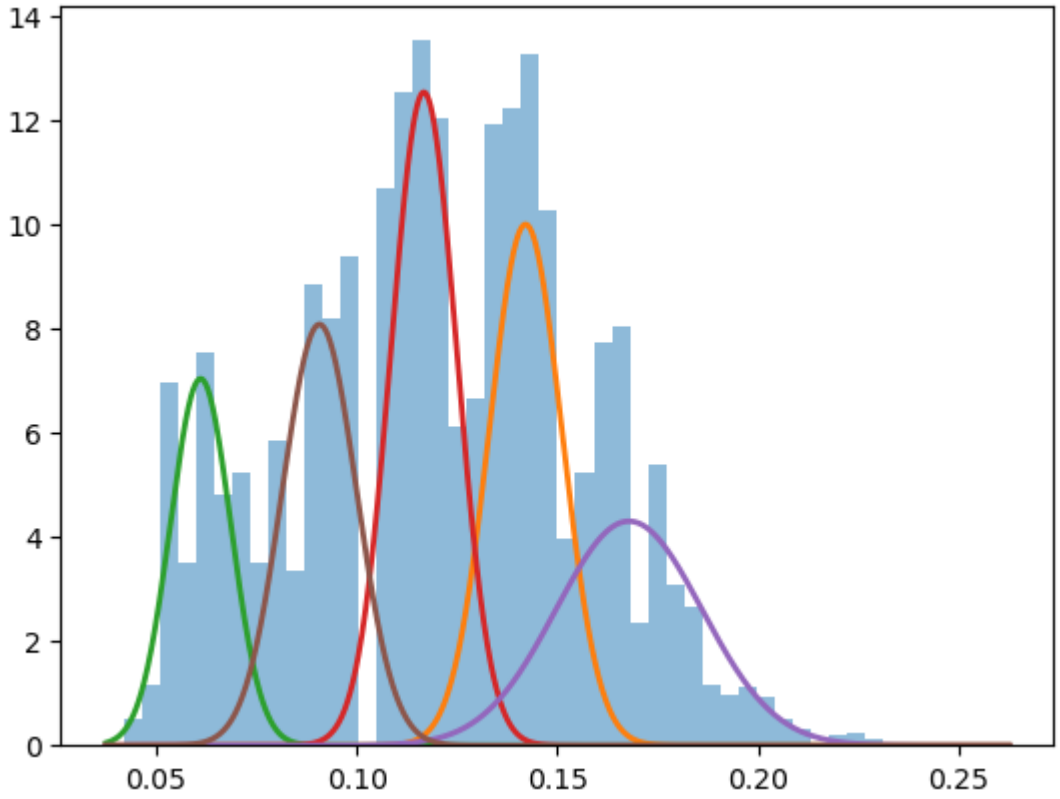
In [76]:
```
print('The std of the 1st normal distribution = ', sigma1_11)
print('The std of the 2nd normal distribution = ', sigma2_11)
```

```
The std of the 1st normal distribution =  0.024818829668032373
The std of the 2nd normal distribution =  0.02598984075198758
```

In [77]:
```python
# Define the two normal distributions
mean1, std1 = mu1_11, sigma1_11
mean2, std2 = mu2_11, sigma2_11

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_11 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_11[0])
```

```
Cross point of two normal distributions: x = 0.1191609860663127
```

In [78]:
```python
from scipy.stats import norm

# generate some 1D data with multiple Gaussian distributions
np.random.seed(42)
x = df_11.value.to_numpy()

# fit Gaussian mixture model to the data
gmm = GaussianMixture(n_components=5, random_state=42)
gmm.fit(x.reshape(-1, 1))

# get parameters of each Gaussian distribution
mus, sigmas, weights = zip(*[(mu[0], sigma[0], weight) for mu, sigma, weight

# plot the data and the fitted Gaussian distributions
plt.hist(x, bins=50, density=True, alpha=0.5)
x_range = np.linspace(x.min(), x.max(), num=1000)
for mu, sigma, weight in zip(mus, sigmas, weights):
    plt.plot(x_range, norm.pdf(x_range, loc=mu, scale=sigma) * weight, linew
plt.show()

# get the number of Gaussian distributions
n_components = gmm.n_components

print(f"Number of Gaussian distributions in the data: {n_components}")
```




```
Number of Gaussian distributions in the data: 5
```

In [79]:

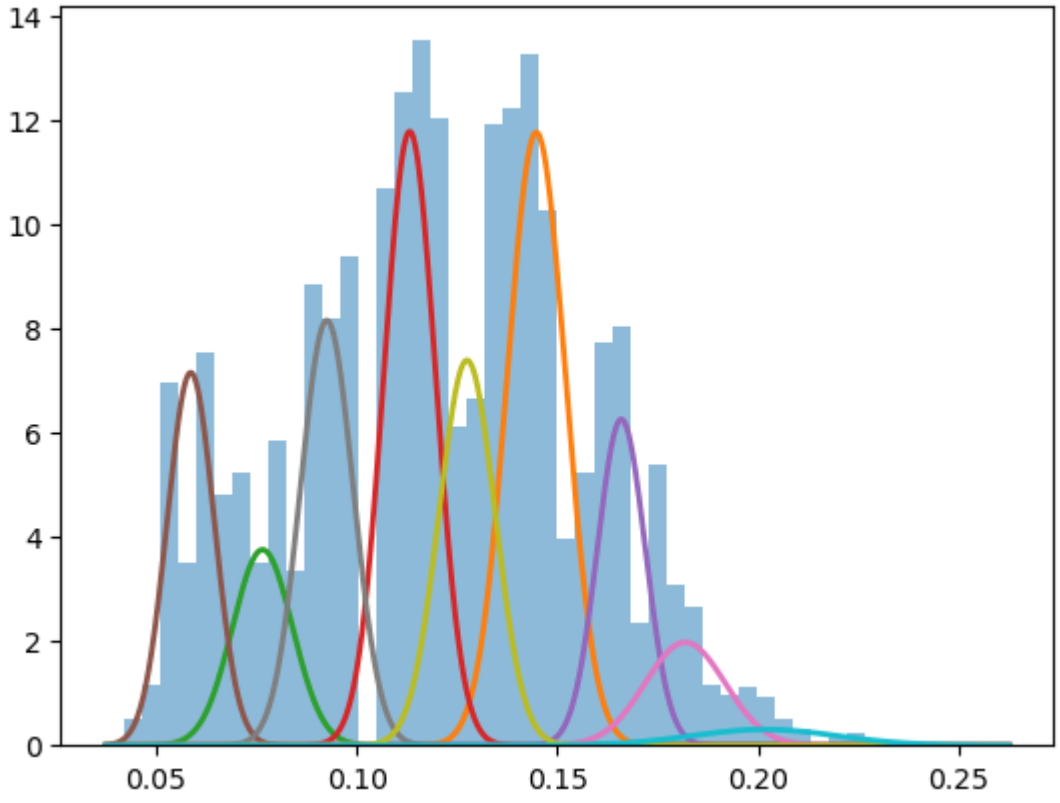
```
from sklearn.mixture import GaussianMixture

# Load the 1-D data into a numpy array 'data'
data = df_11.value.to_numpy()

# Define the range of number of components (i.e., Gaussian distributions) to
n_components_range = range(1, 11)

# Fit the GMM model with different number of components and calculate the AI
lowest_aic = float("inf")
best_n_components = 1

for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, covariance_type='full')
    gmm.fit(data.reshape(-1, 1))
    aic = gmm.aic(data.reshape(-1, 1))
    if aic < lowest_aic:
        lowest_aic = aic
        best_n_components = n_components

# Print the number of Gaussian distributions that give the lowest AIC score
print("Number of Gaussian distributions:", best_n_components)
```

```
Number of Gaussian distributions: 9
```

In [80]:

```
from scipy.stats import norm

# generate some 1D data with multiple Gaussian distributions
                                    2)
x = df_11.value.to_numpy()
```

```
# fit Gaussian mixture model to the data
gmm = GaussianMixture(n_components=9, random_state=42)
gmm.fit(x.reshape(-1, 1))

# get parameters of each Gaussian distribution
mus, sigmas, weights = zip(*[(mu[0], sigma[0], weight) for mu, sigma, weight

# plot the data and the fitted Gaussian distributions
plt.hist(x, bins=50, density=True, alpha=0.5)
x_range = np.linspace(x.min(), x.max(), num=1000)
for mu, sigma, weight in zip(mus, sigmas, weights):
    plt.plot(x_range, norm.pdf(x_range, loc=mu, scale=sigma) * weight, linew
plt.show()

# get the number of Gaussian distributions
n_components = gmm.n_components

print(f"Number of Gaussian distributions in the data: {n_components}")
```




```
Number of Gaussian distributions in the data: 9
```

In [ ]:

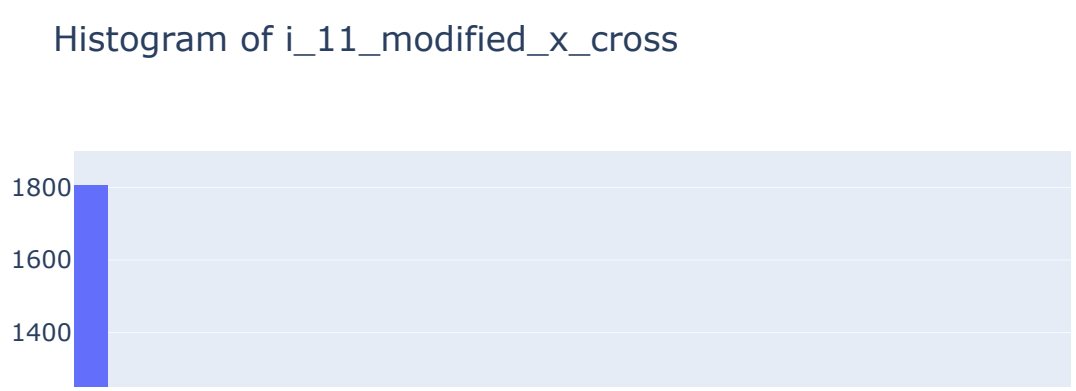
In [ ]:

In [81]:
```
# What if we reduce by x_cross

# If we were to reduce the values by x_cross value as mentioned above
i_11_modified_x_cross = i_11.apply(lambda x: max(0, x - x_cross_11[0]))
```

```
# Calculate the mean and standard deviation of the 1-D array

mean_i_11_modified_x_cross = np.nanmean(i_11_modified_x_cross.values)
std_i_11_modified_x_cross = np.nanstd(i_11_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_11_modified_x_cross = i_11_modified_x_cross.isna().sum().sum()
num_zeros_11_modified_x_cross = (i_11_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_11_modified_x_cross_max = i_11_modified_x_cross.max().max()
i_11_modified_x_cross_min = i_11_modified_x_cross.min().min()
# i_11_modified_x_cross_min = 0
i_11_modified_x_cross_mode = i_11_modified_x_cross.mode()[0]

print('Number of nan values in i_11_modified_x_cross = ', num_nan_11_modifie
print('Number of 0 values in i_11_modified_x_cross = ', num_zeros_11_modifie
print('Number of values in i_11_modified_x_cross = ', len(i_11_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_11_modified_x_cross = ', mean_i_11_modified_x_cro
print('STD of 1-D values in i_11_modified_x_cross = ', std_i_11_modified_x_c
print('Mean value >= STD =====> ', mean_i_11_modified_x_cross >= std_i_11_mo
print('-------------------------------------------------------')
print("Max value in i_11_modified_x_cross = ", i_11_modified_x_cross_max)
print("Min value in i_11_modified_x_cross = ", i_11_modified_x_cross_min)
print('Mode value in i_11_modified_x_cross = ', i_11_modified_x_cross_mode)
```

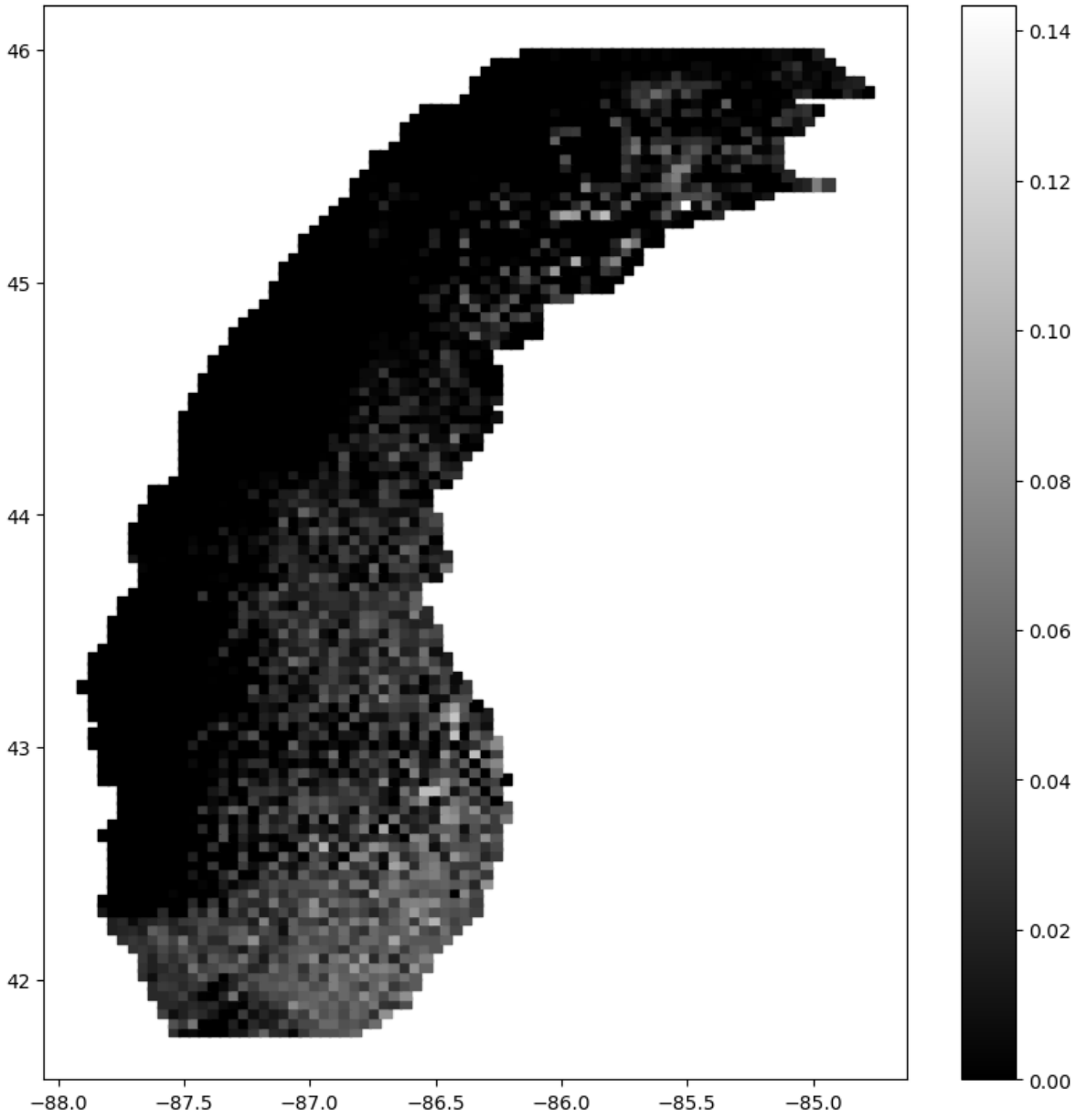
```
Number of nan values in i_11_modified_x_cross =  0
Number of 0 values in i_11_modified_x_cross =  1712
Number of values in i_11_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_11_modified_x_cross =  0.015309312336167802
STD of 1-D values in i_11_modified_x_cross =  0.020904537787925993
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_11_modified_x_cross =  0.1433390139336873
Min value in i_11_modified_x_cross =  0.0
Mode value in i_11_modified_x_cross =  0.0
```

In [82]:
```
fig = px.histogram(i_11_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [83]:
```
# x_11, y_11, i_11 = df_11.longitude, df_11.latitude, df_11.value
plt.figure(figsize=(10, 10))
plt.scatter(df_11.longitude, y_11.values, c=i_11_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_11), len(y_11), len(i_11_modified_x_cross)
```

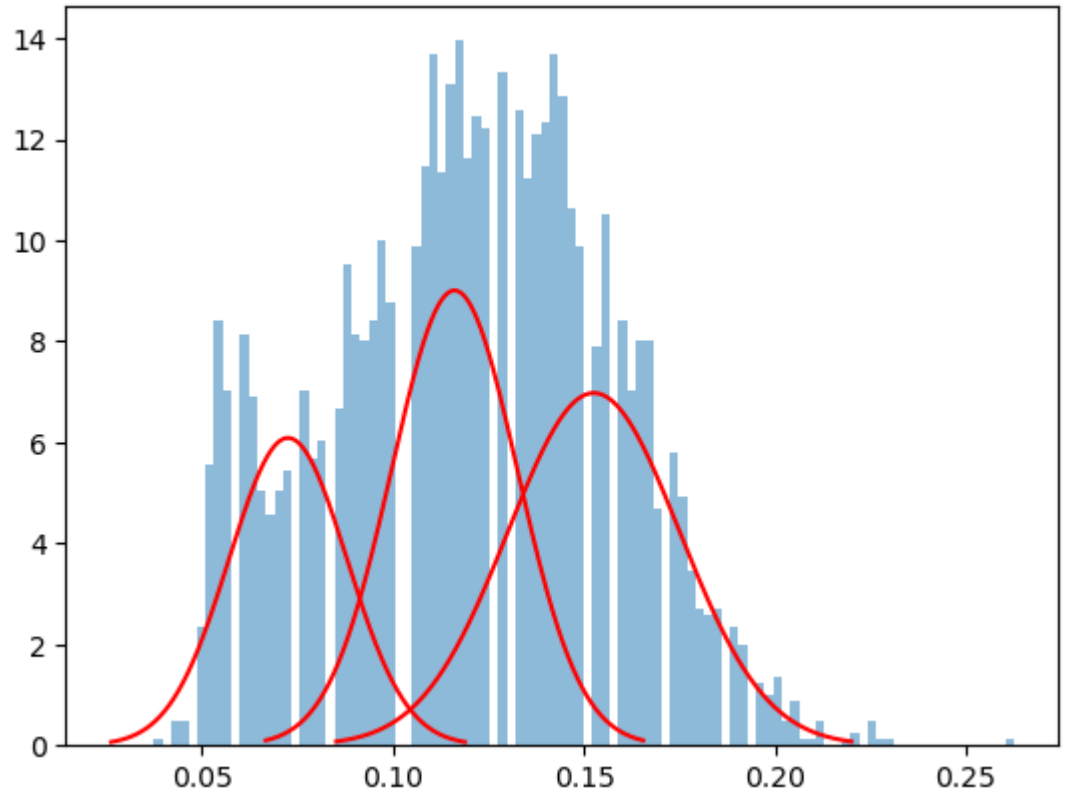
Out[83]: (3599, 3599, 3599)



In [84]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Create a sample 1-D array
i_3N_11 = np.array(i_11).reshape(-1, 1)

# Set the number of components to 3
n_components = 3

# Define the Gaussian mixture model
gmm = GaussianMixture(n_components=n_components)

# Fit the model to the data
gmm.fit(i_3N_11.reshape(-1, 1))

# Get the mean and standard deviation for each component
s_.flatten()
```

```
stds = np.sqrt(gmm.covariances_.flatten())

# Plot the histogram and the fitted Gaussian curves
plt.hist(i_3N_11, bins=100, density=True, alpha=0.5)
for i in range(n_components):
    x = np.linspace(means[i] - 3 * stds[i], means[i] + 3 * stds[i], 3000)
    plt.plot(x, gmm.weights_[i] * np.exp(-0.5 * ((x - means[i]) / stds[i]) *
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

2.3 2016.12.09.1600

At `1600`, it is noon time since **16:00 UTC** is **11:00 CDT**.

image

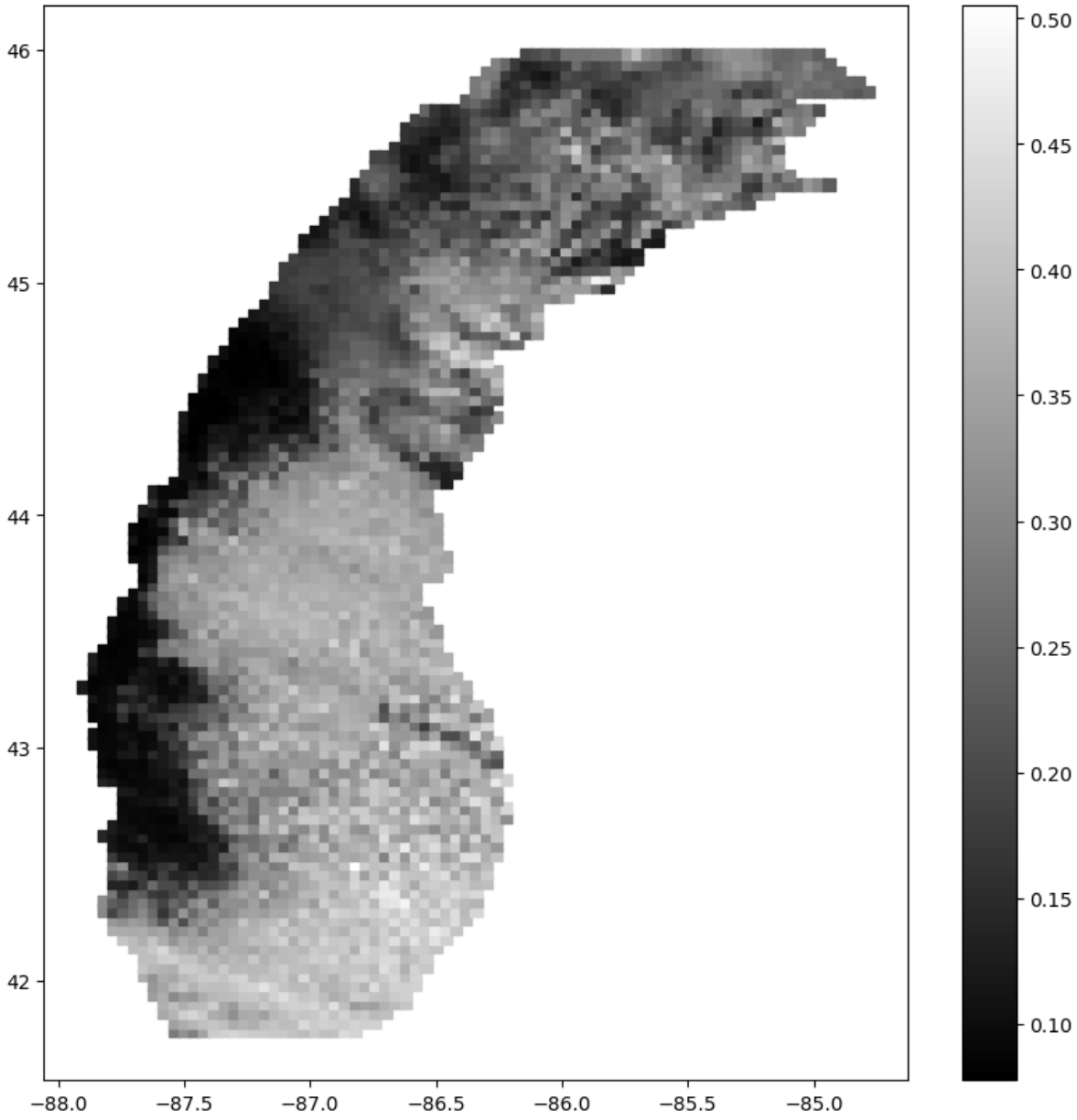
The cloud intensity reaches its peak values.

In [85]:
```
df_18 = pd.read_csv(zone_0_folder_path + file_list[18])
print(file_list[18])
```

```
goes15.2016.12.09.1600.v01.nc-var1-t0.csv
```

In [86]:
```
x_18, y_18, i_18 = df_18.longitude, df_18.latitude, df_18.value
plt.figure(figsize=(10, 10))
plt.scatter(x_18.values, y_18.values, c=i_18.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_18), len(y_18), len(i_18)
```

Out[86]: (3599, 3599, 3599)



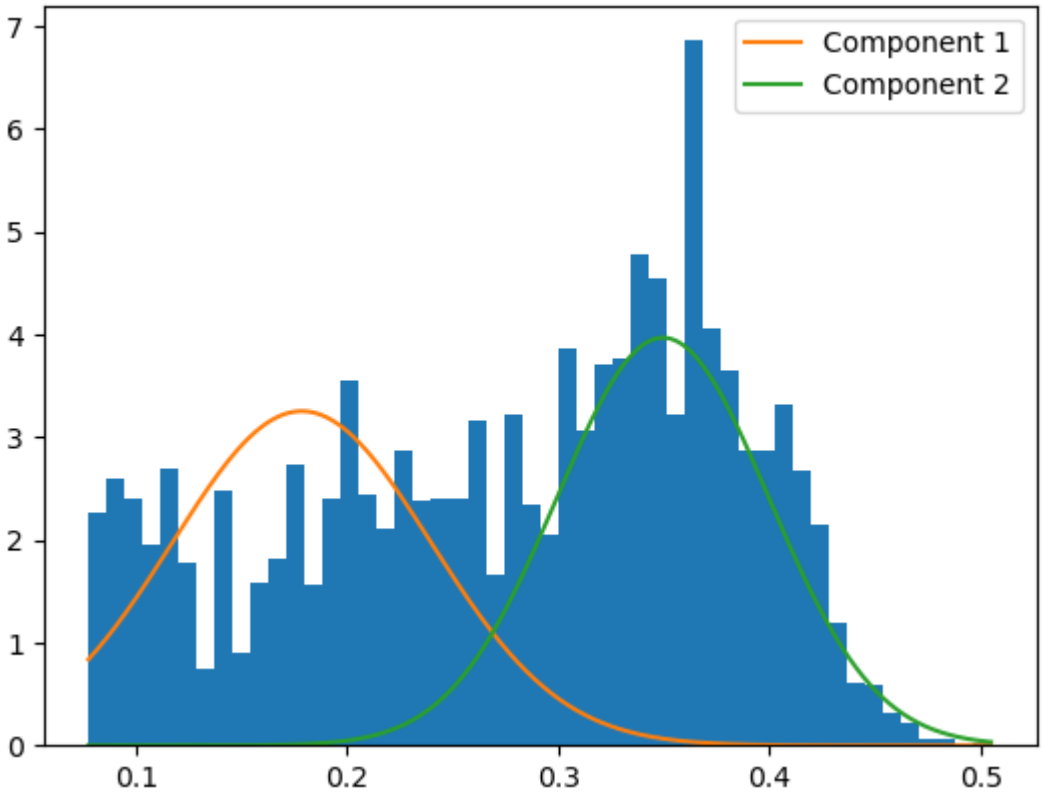
In [87]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_18 = np.nanmean(i_18.values)
std_i_18 = np.nanstd(i_18.values)

# Check number of nan values and 0s in i, just in case
num_nan_18 = i_18.isna().sum().sum()
num_zeros_18 = (i_18 == 0).sum().sum()

max in i
```

```
i_18_max = i_18.max().max()
i_18_min = i_18.min().min()
i_18_mode = i_18.mode()[0]

print('Number of nan values in i_18 = ', num_nan_18)
print('Number of 0 values in i_18 = ', num_zeros_18)
print('Number of values in i_18 = ', len(i_18))
print('-------------------------------------------------------')
print('Mean of values in i_18 = ', mean_i_18)
print('STD of 1-D values in i_18 = ', std_i_18)
print('Mean value >= STD =====> ', mean_i_18 >= std_i_18)
print('-------------------------------------------------------')
print("Max value in i_18 = ", i_18_max)
print("Min value in i_18 = ", i_18_min)
print('Mode value in i_18 = ', i_18_mode)
```

```
Number of nan values in i_18 =  0
Number of 0 values in i_18 =  0
Number of values in i_18 =  3599
-------------------------------------------------------
Mean of values in i_18 =  0.27877673916449014
STD of 1-D values in i_18 =  0.10067410604439496
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_18 =  0.505
Min value in i_18 =  0.0775
Mode value in i_18 =  0.36249998
```

In [88]:
```
fig = px.histogram(i_18.values.flatten(), title="Histogram of i_18")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```

In [89]:
```
x_18 = np.array(i_18).reshape(-1, 1)

# Fit a mixture of two Gaussians to the data
gmm_18 = GaussianMixture(n_components=2).fit(x_18)

# Get the means and standard deviations of the two Gaussian components
mu1_18, mu2_18 = gmm_18.means_.flatten()
sigma1_18, sigma2_18 = np.sqrt(gmm_18.covariances_.flatten())

# Visualize the results
plt.hist(i_18, density=True, bins = 50)
# x_axis = i_18.unique()
x_axis_18 = np.linspace(min(i_18), max(i_18), 3000)
plt.plot(x_axis_18, 0.5 * np.exp(-(x_axis_18 - mu1_18)**2 / (2 * sigma1_18**
plt.plot(x_axis_18, 0.5 * np.exp(-(x_axis_18 - mu2_18)**2 / (2 * sigma2_18**
plt.legend()
plt.show()
```



In [90]:
```
print('The mean of the 1st normal distribution = ', mu1_18)
print('The mean of the 2nd normal distribution = ', mu2_18)
```

```
The mean of the 1st normal distribution =  0.17853861849852065
The mean of the 2nd normal distribution =  0.3495810987762521
```

In [91]:
```
print('The std of the 1st normal distribution = ', sigma1_18)
print('The std of the 2nd normal distribution = ', sigma2_18)
```

```
The std of the 1st normal distribution =  0.061334753889830326
The std of the 2nd normal distribution =  0.05028220140051553
```

In [92]:
```
# Define the two normal distributions
mean1, std1 = mu1_18, sigma1_18
mean2, std2 = mu2_18, sigma2_18

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_18 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_18[0])
```

```
Cross point of two normal distributions: x = 0.268960587251013
```

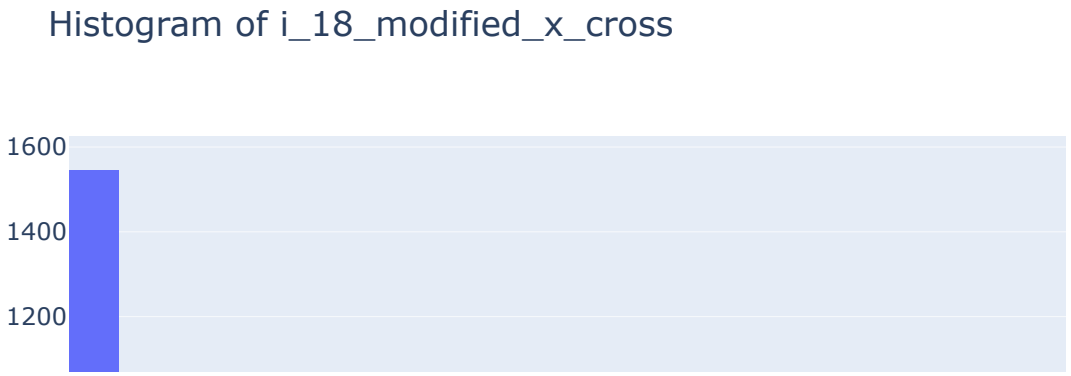
Then, if we were to use the same threshold to the imagery data at `1800 UTC`, let's see what is going to happen?

In [93]:
```
# What if we reduce by x_cross

# If we were to reduce the values by x_cross value as mentioned above
i_18_modified_x_cross = i_18.apply(lambda x: max(0, x - x_cross_18[0]))
```

In [94]:
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_18_modified_x_cross = np.nanmean(i_18_modified_x_cross.values)
std_i_18_modified_x_cross = np.nanstd(i_18_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_18_modified_x_cross = i_18_modified_x_cross.isna().sum().sum()
num_zeros_18_modified_x_cross = (i_18_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_18_modified_x_cross_max = i_18_modified_x_cross.max().max()
i_18_modified_x_cross_min = i_18_modified_x_cross.min().min()
# i_18_modified_x_cross_min = 0
i_18_modified_x_cross_mode = i_18_modified_x_cross.mode()[0]

print('Number of nan values in i_18_modified_x_cross = ', num_nan_18_modifie
print('Number of 0 values in i_18_modified_x_cross = ', num_zeros_18_modifie
print('Number of values in i_18_modified_x_cross = ', len(i_18_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_18_modified_x_cross = ', mean_i_18_modified_x_cro
print('STD of 1-D values in i_18_modified_x_cross = ', std_i_18_modified_x_c
print('Mean value >= STD =====> ', mean_i_18_modified_x_cross >= std_i_18_mo
print('-------------------------------------------------------')
print("Max value in i_18_modified_x_cross = ", i_18_modified_x_cross_max)
print("Min value in i_18_modified_x_cross = ", i_18_modified_x_cross_min)
print('Mode value in i_18_modified_x_cross = ', i_18_modified_x_cross_mode)
```

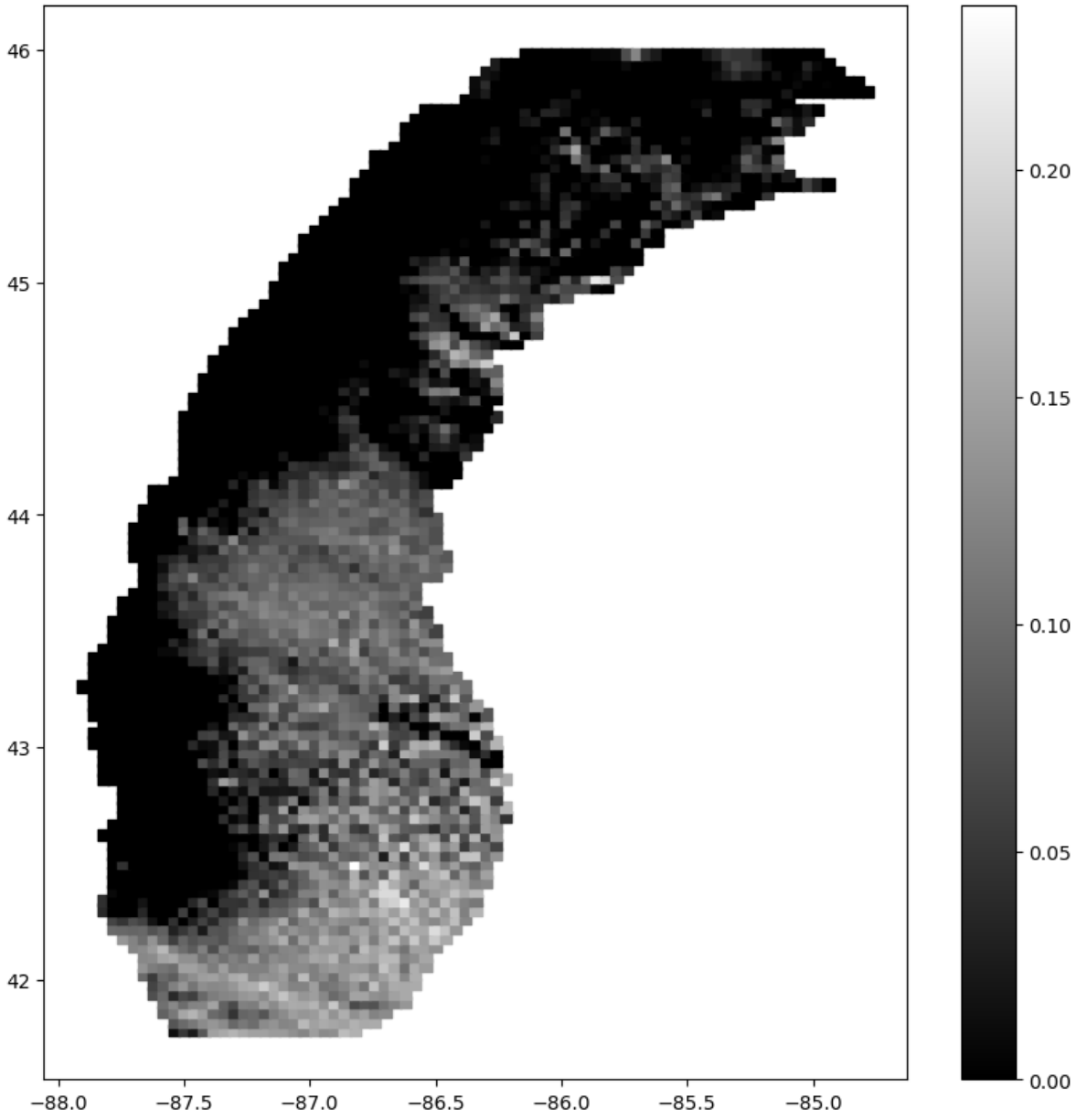
```
Number of nan values in i_18_modified_x_cross =  0
Number of 0 values in i_18_modified_x_cross =  1545
Number of values in i_18_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_18_modified_x_cross =  0.04871490476421763
STD of 1-D values in i_18_modified_x_cross =  0.05404667937151287
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_18_modified_x_cross =  0.236039412748987
Min value in i_18_modified_x_cross =  0.0
Mode value in i_18_modified_x_cross =  0.0
```

In [95]:
```
fig = px.histogram(i_18_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [96]:
```
x_18= df_18.longitude
plt.figure(figsize=(10, 10))
plt.scatter(x_18.values, y_18.values, c=i_18_modified_x_cross.values, cmap=c
plt.colorbar(orientation='vertical')
len(x_18), len(y_18), len(i_18_modified_x_cross)
```

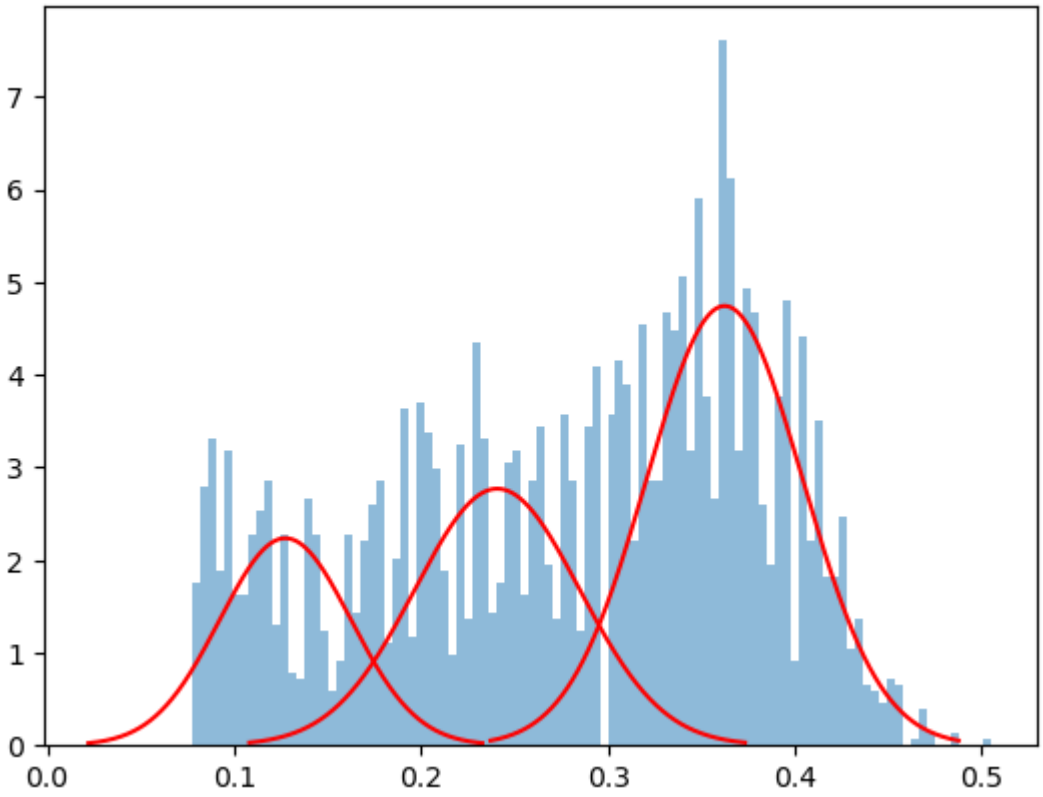
Out[96]: (3599, 3599, 3599)



In [97]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Create a sample 1-D array
i_3N_18 = np.array(i_18).reshape(-1, 1)

# Set the number of components to 3
n_components = 3

# Define the Gaussian mixture model
gmm = GaussianMixture(n_components=n_components)

# Fit the model to the data
gmm.fit(i_3N_18.reshape(-1, 1))

# Get the mean and standard deviation for each component
s_.flatten()
```

```
stds = np.sqrt(gmm.covariances_.flatten())

# Plot the histogram and the fitted Gaussian curves
plt.hist(i_3N_18, bins=100, density=True, alpha=0.5)
for i in range(n_components):
    x = np.linspace(means[i] - 3 * stds[i], means[i] + 3 * stds[i], 3000)
    plt.plot(x, gmm.weights_[i] * np.exp(-0.5 * ((x - means[i]) / stds[i]) *
plt.show()
```



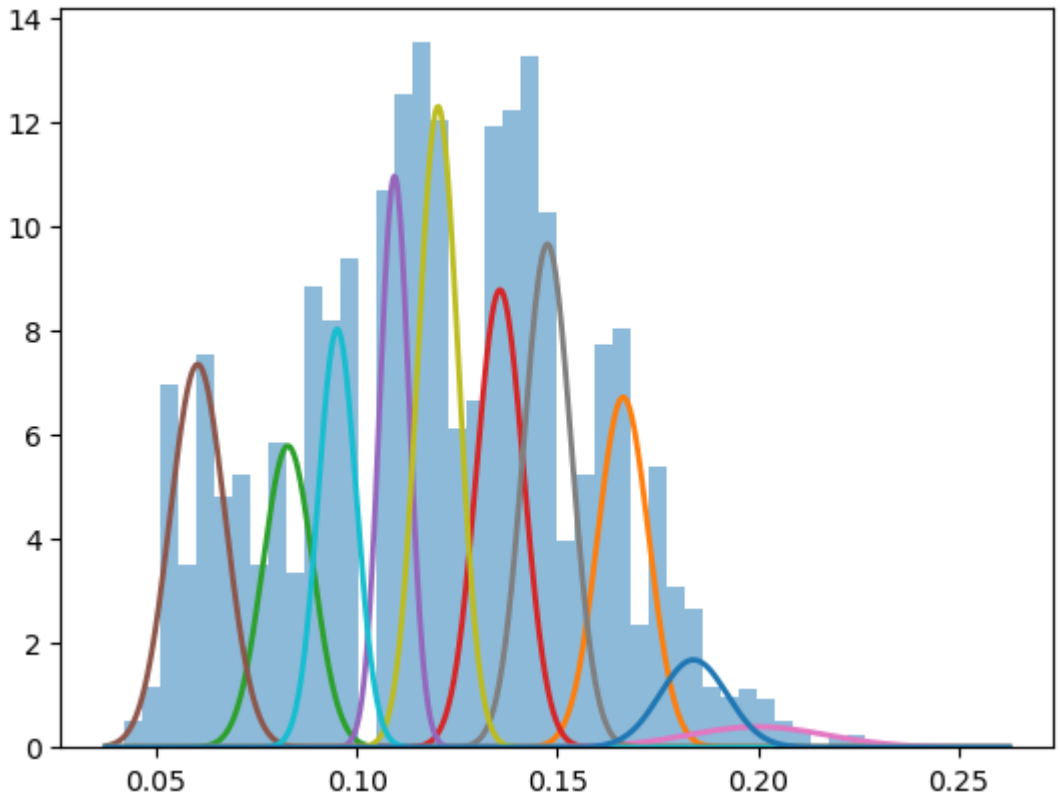
In [98]:

```
from sklearn.mixture import GaussianMixture

# Load the 1-D data into a numpy array 'data'
data = df_18.value.to_numpy()

# Define the range of number of components (i.e., Gaussian distributions) to
n_components_range = range(1, 11)

# Fit the GMM model with different number of components and calculate the AI
lowest_aic = float("inf")
best_n_components = 1

for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, covariance_type='full')
    gmm.fit(data.reshape(-1, 1))
    aic = gmm.aic(data.reshape(-1, 1))
    if aic < lowest_aic:
        lowest_aic = aic
        best_n_components = n_components

# Print the number of Gaussian distributions that give the lowest AIC score
print("Number of Gaussian distributions:", best_n_components)
```

```
Number of Gaussian distributions: 10
```

In [99]:

```python
from scipy.stats import norm

# generate some 1D data with multiple Gaussian distributions
np.random.seed(100)
x = df_11.value.to_numpy()

# fit Gaussian mixture model to the data
gmm = GaussianMixture(n_components=10, random_state=100)
gmm.fit(x.reshape(-1, 1))

# get parameters of each Gaussian distribution
mus, sigmas, weights = zip(*[(mu[0], sigma[0], weight) for mu, sigma, weight

# plot the data and the fitted Gaussian distributions
plt.hist(x, bins=50, density=True, alpha=0.5)
x_range = np.linspace(x.min(), x.max(), num=1000)
for mu, sigma, weight in zip(mus, sigmas, weights):
    plt.plot(x_range, norm.pdf(x_range, loc=mu, scale=sigma) * weight, linew
plt.show()

# get the number of Gaussian distributions
n_components = gmm.n_components

print(f"Number of Gaussian distributions in the data: {n_components}")
```




```
Number of Gaussian distributions in the data: 10
```

In [ ]:

Let's see what happens when we applying the same threshold to `1400 UTC`.

In [100…
```
x_11, y_11, i_11 = df_11.longitude, df_11.latitude, df_11.value
```

In [101…
```
# What if we reduce by x_cross

# If we were to reduce the values by x_cross value as mentioned above
i_11_modified_x_cross = i_11.apply(lambda x: max(0, x - x_cross_18[0]))
print(i_11_modified_x_cross)
```

```
0       0
1       0
2       0
3       0
4       0
       ..
3594    0
3595    0
3596    0
3597    0
3598    0
Name: value, Length: 3599, dtype: int64
```

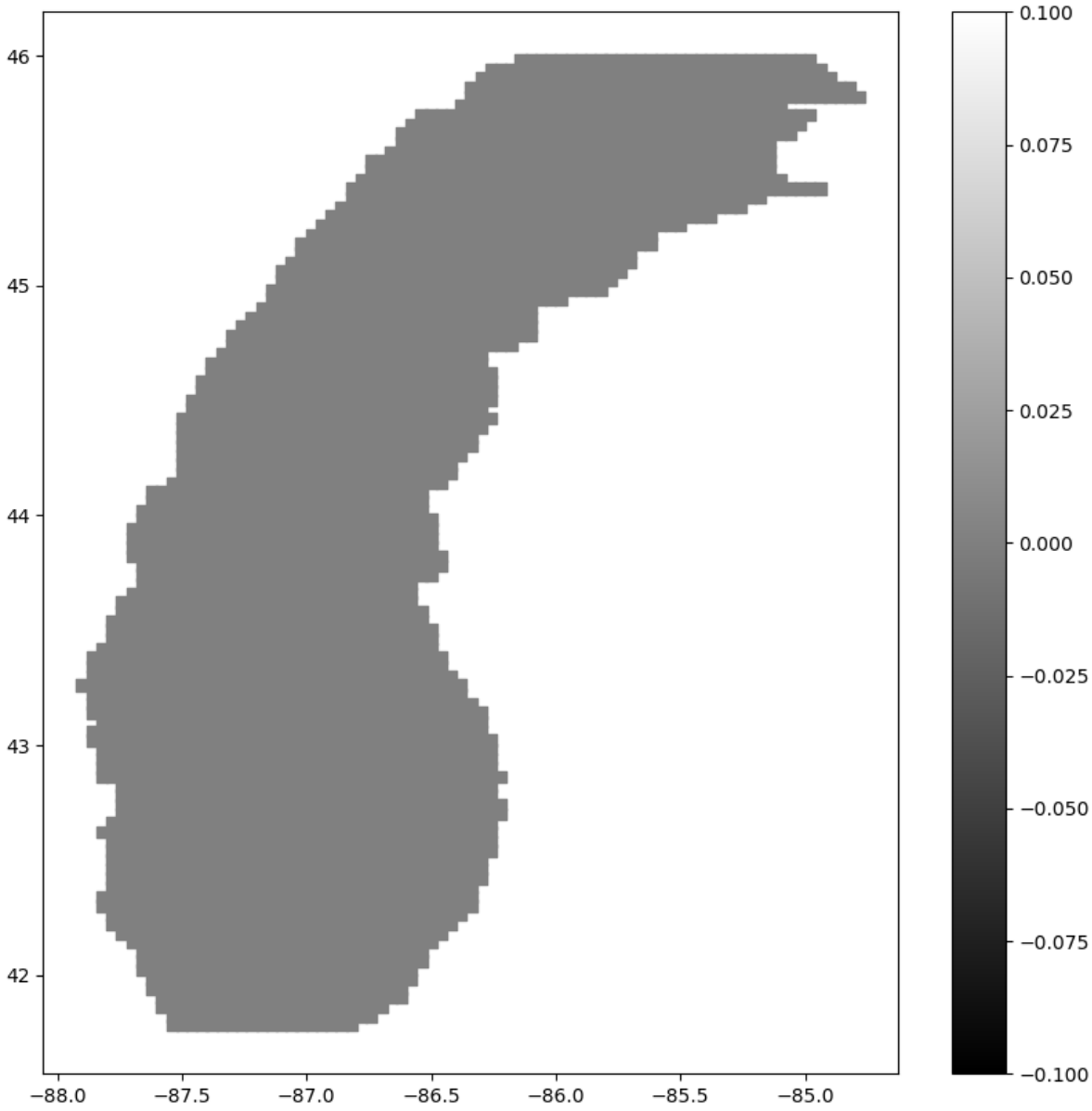
In [102…
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_11_modified_x_cross = np.nanmean(i_11_modified_x_cross.values)
std_i_11_modified_x_cross = np.nanstd(i_11_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_11_modified_x_cross = i_11_modified_x_cross.isna().sum().sum()
num_zeros_11_modified_x_cross = (i_11_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_11_modified_x_cross_max = i_11_modified_x_cross.max().max()
i_11_modified_x_cross_min = i_11_modified_x_cross.min().min()
# i_11_modified_x_cross_min = 0
i_11_modified_x_cross_mode = i_11_modified_x_cross.mode()[0]

print('Number of nan values in i_11_modified_x_cross = ', num_nan_11_modifie
print('Number of 0 values in i_11_modified_x_cross = ', num_zeros_11_modifie
print('Number of values in i_11_modified_x_cross = ', len(i_11_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_11_modified_x_cross = ', mean_i_11_modified_x_cro
print('STD of 1-D values in i_11_modified_x_cross = ', std_i_11_modified_x_c
print('Mean value >= STD =====> ', mean_i_11_modified_x_cross >= std_i_11_mo
print('-------------------------------------------------------')
print("Max value in i_11_modified_x_cross = ", i_11_modified_x_cross_max)
print("Min value in i_11_modified_x_cross = ", i_11_modified_x_cross_min)
print('Mode value in i_11_modified_x_cross = ', i_11_modified_x_cross_mode)
```

```
Number of nan values in i_11_modified_x_cross =  0
Number of 0 values in i_11_modified_x_cross =  3599
Number of values in i_11_modified_x_cross =  3599
--------------------------------------------------
Mean of values in i_11_modified_x_cross =  0.0
STD of 1-D values in i_11_modified_x_cross =  0.0
Mean value >= STD =====>  True
--------------------------------------------------
Max value in i_11_modified_x_cross =  0
Min value in i_11_modified_x_cross =  0
Mode value in i_11_modified_x_cross =  0
```

In [103…
```
fig = px.histogram(i_11_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```

In [104…
```
# x_11, y_11, i_11 = df_11.longitude, df_11.latitude, df_11.value
plt.figure(figsize=(10, 10))
plt.scatter(x_11.values, y_11.values, c=i_11_modified_x_cross.values, cmap=c
plt.colorbar(orientation='vertical')
len(x_11), len(y_11), len(i_11_modified_x_cross)
```

Out[104]: (3599, 3599, 3599)



Now, let's apply the intersect point value at i_11 as the threshold.

In [105…

```
# Define the two normal distributions
mean1, std1 = mu1_11, sigma1_11
mean2, std2 = mu2_11, sigma2_11

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_11 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_11[0])
```

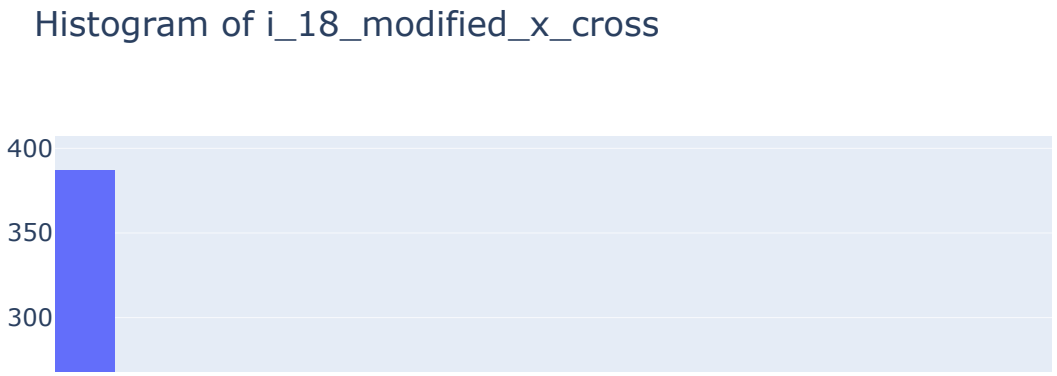
```
Cross point of two normal distributions: x = 0.1191609860663127
```

In [106…
```
# What if we reduce by x_cross

# If we were to reduce the values by x_cross value as mentioned above
i_18_modified_x_cross = i_18.apply(lambda x: max(0, x - x_cross_11[0]))
```

In [107…
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_18_modified_x_cross = np.nanmean(i_18_modified_x_cross.values)
std_i_18_modified_x_cross = np.nanstd(i_18_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_18_modified_x_cross = i_18_modified_x_cross.isna().sum().sum()
num_zeros_18_modified_x_cross = (i_18_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_18_modified_x_cross_max = i_18_modified_x_cross.max().max()
i_18_modified_x_cross_min = i_18_modified_x_cross.min().min()
# i_18_modified_x_cross_min = 0
i_18_modified_x_cross_mode = i_18_modified_x_cross.mode()[0]

print('Number of nan values in i_18_modified_x_cross = ', num_nan_18_modifie
print('Number of 0 values in i_18_modified_x_cross = ', num_zeros_18_modifie
print('Number of values in i_18_modified_x_cross = ', len(i_18_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_18_modified_x_cross = ', mean_i_18_modified_x_cro
print('STD of 1-D values in i_18_modified_x_cross = ', std_i_18_modified_x_c
print('Mean value >= STD =====> ', mean_i_18_modified_x_cross >= std_i_18_mo
print('-------------------------------------------------------')
print("Max value in i_18_modified_x_cross = ", i_18_modified_x_cross_max)
print("Min value in i_18_modified_x_cross = ", i_18_modified_x_cross_min)
print('Mode value in i_18_modified_x_cross = ', i_18_modified_x_cross_mode)
```

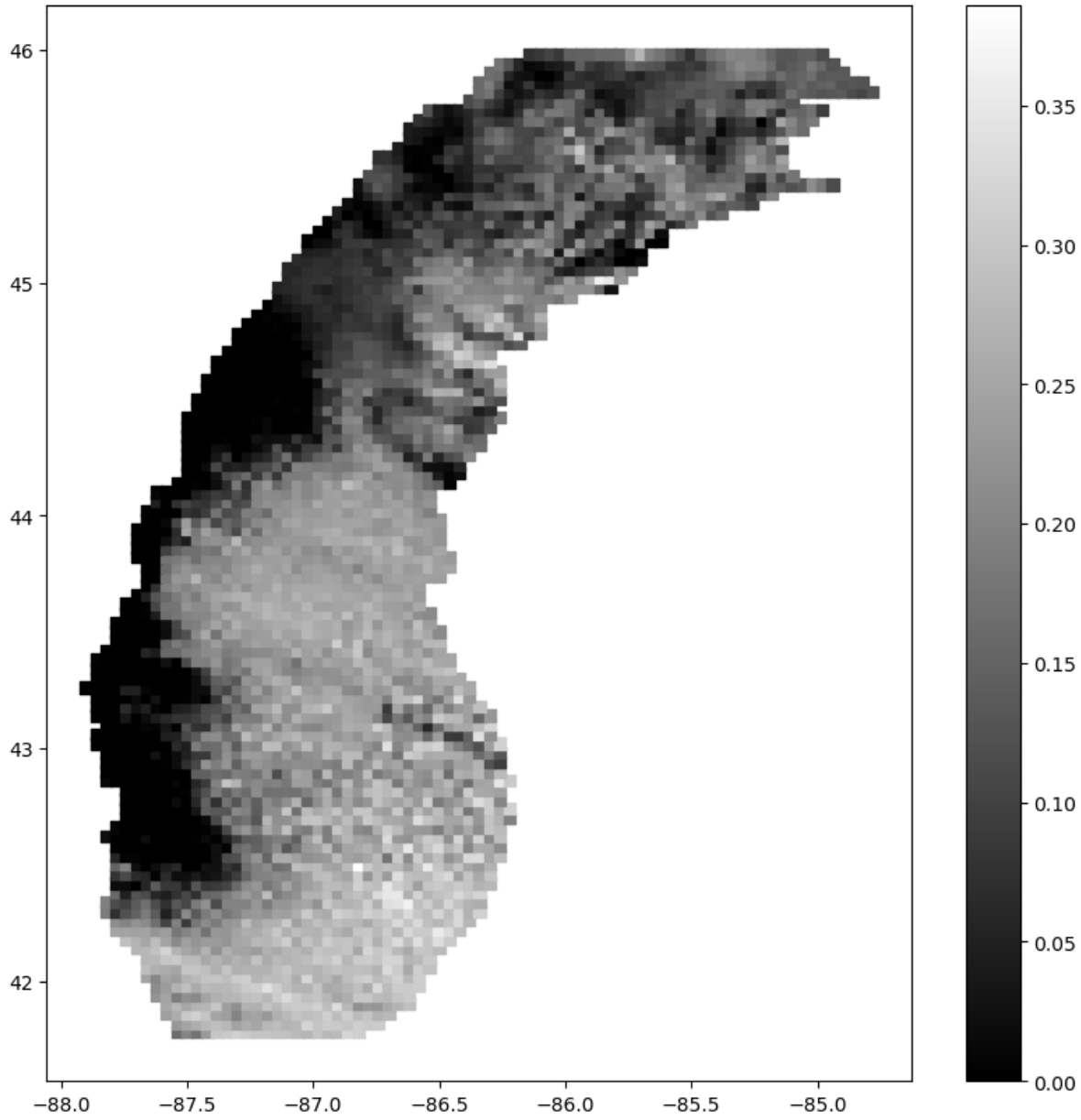
```
Number of nan values in i_18_modified_x_cross =  0
Number of 0 values in i_18_modified_x_cross =  342
Number of values in i_18_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_18_modified_x_cross =  0.1616859275220949
STD of 1-D values in i_18_modified_x_cross =  0.09701695408717351
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_18_modified_x_cross =  0.3858390139336873
Min value in i_18_modified_x_cross =  0.0
Mode value in i_18_modified_x_cross =  0.0
```

In [108…
```
fig = px.histogram(i_18_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [109…
```
# x_18, y_18, i_18 = df_18.longitude, df_18.latitude, df_18.value
plt.figure(figsize=(10, 10))
plt.scatter(x_18.values, y_18.values, c=i_18_modified_x_cross.values, cmap=c
plt.colorbar(orientation='vertical')
len(x_18), len(y_18), len(i_18_modified_x_cross)
```

Out[109]: (3599, 3599, 3599)



---

3. Case 3

`Duration` : 2016.12.11 1400-1600 UTC

image

> [Q]:
>
> In this case, the cloud was not visible at 1200 UTC, aka 7 AM CDT. However, at 1400 UTC, it becomes visible as the cloud covers the entire Lake Michigan area. And the intensity changes greatly in an interval of 15 mins. And by the time 1600, it reaches its peak intensity, where everything

> Do we consider 1400 as the beginning of the cloud?

> [A]:

> If there are no clouds the day before, @before sunset, then yes.

3.1 2016.12.11.1400

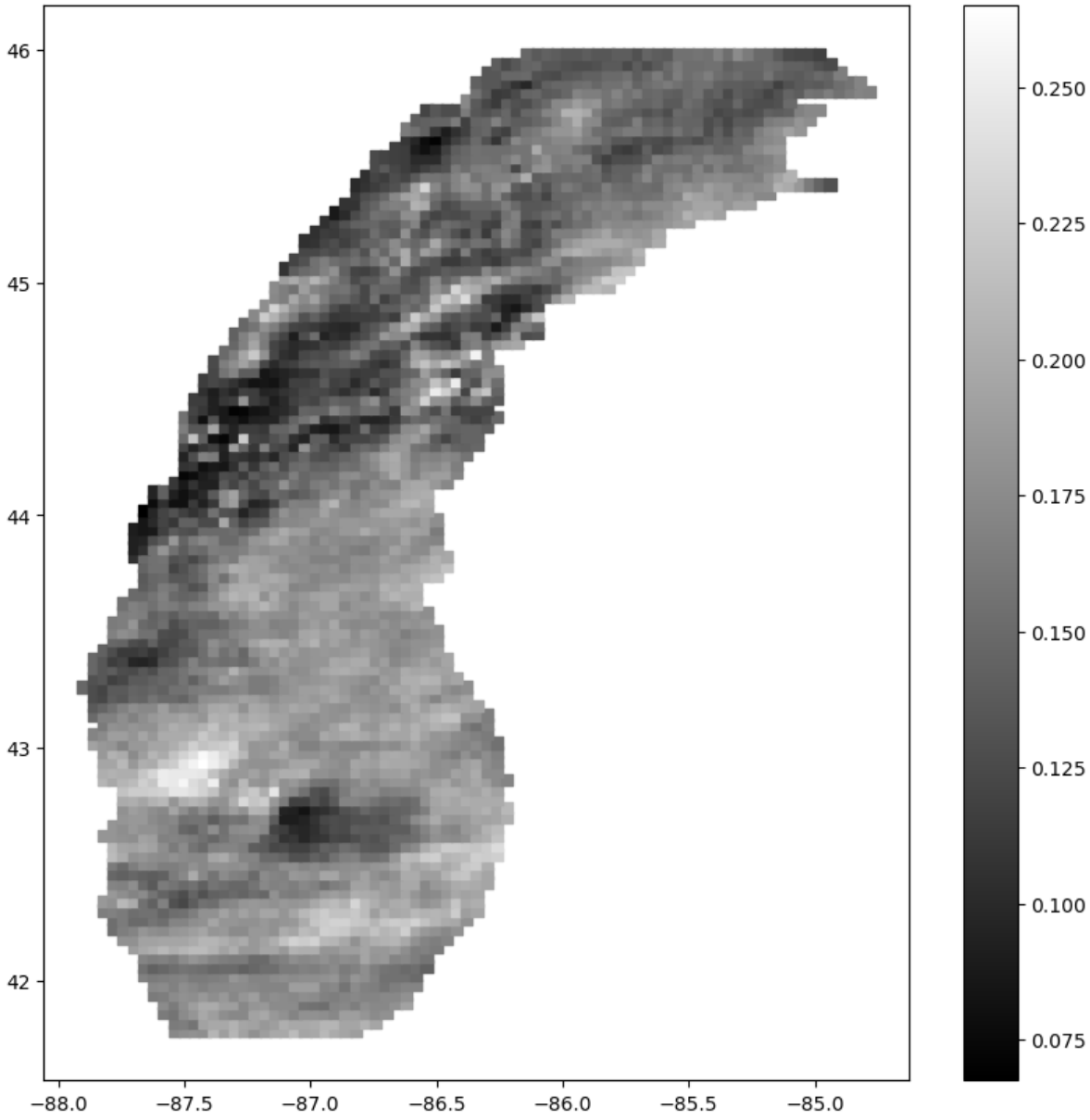
But before this, let's take a look at what happened at `1400`.

In [110…
```
df_20 = pd.read_csv(zone_0_folder_path + file_list[20])

print(file_list[20])
```

```
goes15.2016.12.11.1400.v01.nc-var1-t0.csv
```

In [111…
```
x_20, y_20, i_20 = df_20.longitude, df_20.latitude, df_20.value
plt.figure(figsize=(10, 10))
plt.scatter(x_20.values, y_20.values, c=i_20.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_20), len(y_20), len(i_20)
```

Out[111]: (3599, 3599, 3599)



In [112…

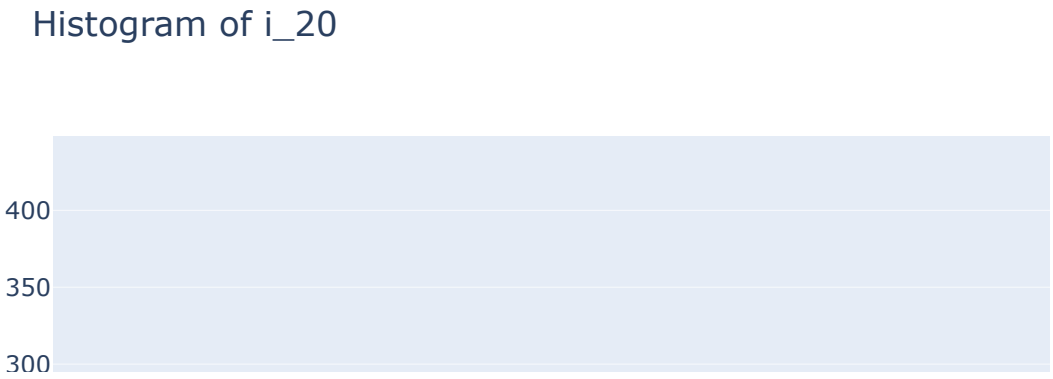
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_20 = np.nanmean(i_20.values)
std_i_20 = np.nanstd(i_20.values)

# Check number of nan values and 0s in i, just in case
num_nan_20 = i_20.isna().sum().sum()
num_zeros_20 = (i_20 == 0).sum().sum()

# Check min and max in i

i_20_max = i_20.max().max()
i_20_min = i_20.min().min()
i_20_mode = i_20.mode()[0]

print('Number of nan values in i_20 = ', num_nan_20)
print('Number of 0 values in i_20 = ', num_zeros_20)
print('                values in i_20 = ', len(i_20))
print('-------------------------------------------------')
```

```
print('Mean of values in i_20 = ', mean_i_20)
print('STD of 1-D values in i_20 = ', std_i_20)
print('Mean value >= STD =====> ', mean_i_20 >= std_i_20)
print('-------------------------------------------------------')
print("Max value in i_20 = ", i_20_max)
print("Min value in i_20 = ", i_20_min)
print('Mode value in i_20 = ', i_20_mode)
```

```
Number of nan values in i_20 =  0
Number of 0 values in i_20 =  0
Number of values in i_20 =  3599
-------------------------------------------------------
Mean of values in i_20 =  0.16365448466073906
STD of 1-D values in i_20 =  0.031171956557496307
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_20 =  0.265
Min value in i_20 =  0.067499995
Mode value in i_20 =  0.17999999
```

In [113…

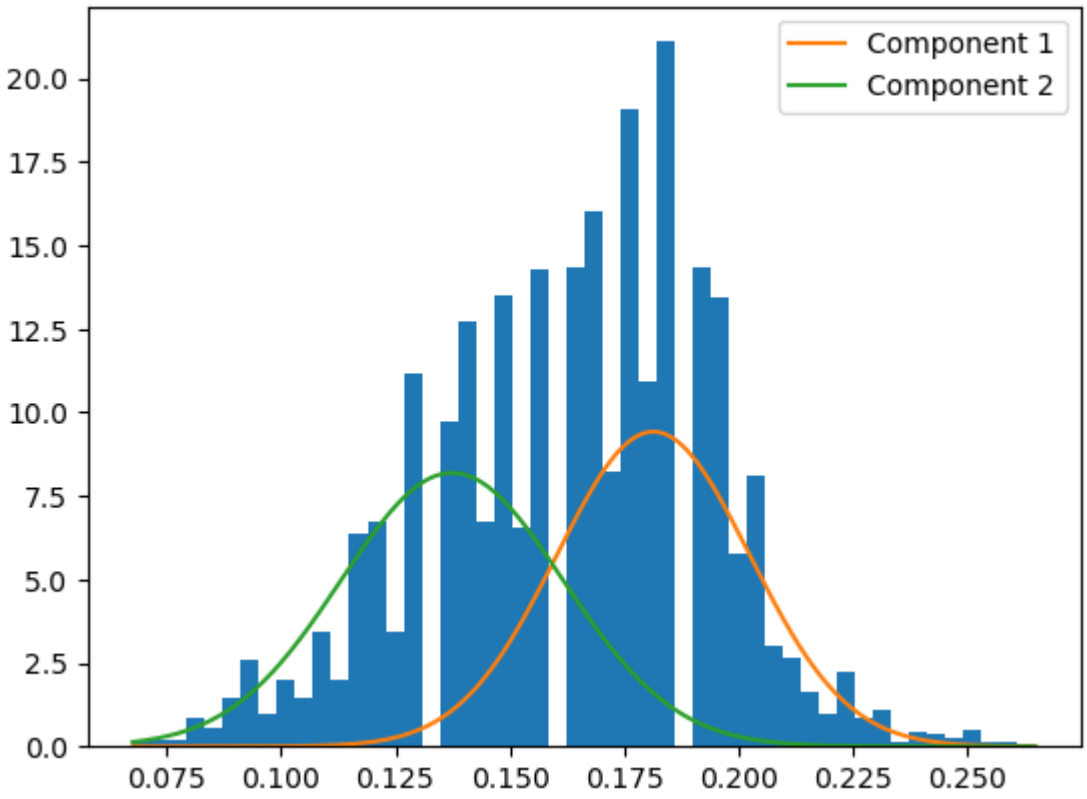
```
fig = px.histogram(i_20.values.flatten(), title="Histogram of i_20")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



In [114…

```
x_20 = np.array(i_20.sort_values()).reshape(-1, 1)

# x_20 = np.nan_to_num(x_20)
# x_20 = x_20.sort()
# x_20 = np.array(i_20).flatten()



# print(x_20.shape)
# x_20.sorted()

# Fit a mixture of two Gaussians to the data
gmm_20 = GaussianMixture(n_components=2).fit(x_20)

# Get the means and standard deviations of the two Gaussian components
mu1_20, mu2_20 = gmm_20.means_.flatten()
sigma1_20, sigma2_20 = np.sqrt(gmm_20.covariances_.flatten())

# Visualize the results
plt.hist(i_20, density=True, bins = 50)
# x_axis = i_20.unique()
x_axis_20 = np.linspace(min(i_20), max(i_20), 3000)
20, 0.5 * np.exp(-(x_axis_20 - mu1_20)**2 / (2 * sigma1_20**
plt.plot(x_axis_20, 0.5 * np.exp(-(x_axis_20 - mu2_20)**2 / (2 * sigma2_20**
```

```
plt.legend()
plt.show()
```



In [115…
```
print('The mean of the 1st normal distribution = ', mu1_20)
print('The mean of the 2nd normal distribution = ', mu2_20)
```

```
The mean of the 1st normal distribution =  0.18141234307184428
The mean of the 2nd normal distribution =  0.13739319958501184
```

In [116…
```
print('The std of the 1st normal distribution = ', sigma1_20)
print('The std of the 2nd normal distribution = ', sigma2_20)
```

```
The std of the 1st normal distribution =  0.021162216608848698
The std of the 2nd normal distribution =  0.024348930944025
```

In [117…
```
# Define the two normal distributions
mean1, std1 = mu1_20, sigma1_20
mean2, std2 = mu2_20, sigma2_20

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_20 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_20[0])
```



```
Cross point of two normal distributions: x = 0.15931044990914905
```

In [118…

```
# Define the two normal distributions
mean1, std1 = mu1_20, sigma1_20
mean2, std2 = mu2_20, sigma2_20

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_20 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_20[0])
```

```
Cross point of two normal distributions: x = 0.15931044990914905
```

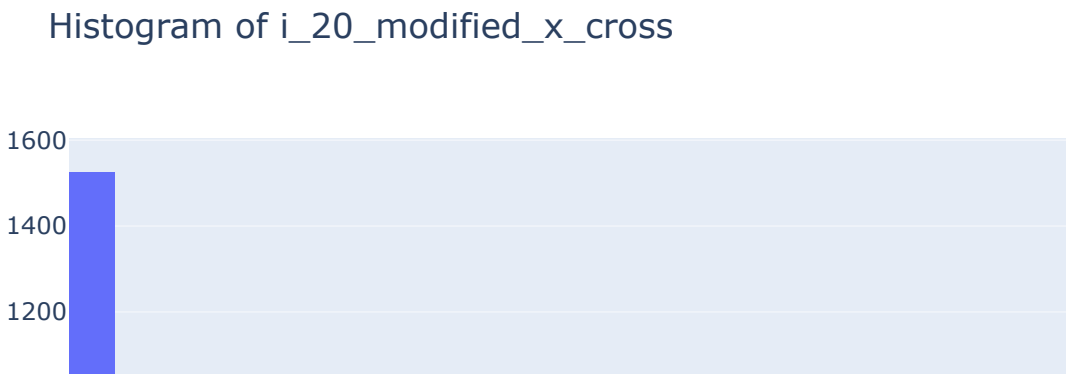
In [119…

```
# What if we reduce by x_cross

# If we were to reduce the values by x_cross value as mentioned above
i_20_modified_x_cross = i_20.apply(lambda x: max(0, x - x_cross_20[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_20_modified_x_cross = np.nanmean(i_20_modified_x_cross.values)
std_i_20_modified_x_cross = np.nanstd(i_20_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_20_modified_x_cross = i_20_modified_x_cross.isna().sum().sum()
num_zeros_20_modified_x_cross = (i_20_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_20_modified_x_cross_max = i_20_modified_x_cross.max().max()
i_20_modified_x_cross_min = i_20_modified_x_cross.min().min()
# i_20_modified_x_cross_min = 0
i_20_modified_x_cross_mode = i_20_modified_x_cross.mode()[0]

print('Number of nan values in i_20_modified_x_cross = ', num_nan_20_modifie
print('Number of 0 values in i_20_modified_x_cross = ', num_zeros_20_modifie
print('Number of values in i_20_modified_x_cross = ', len(i_20_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_20_modified_x_cross = ', mean_i_20_modified_x_cro
print('STD of 1-D values in i_20_modified_x_cross = ', std_i_20_modified_x_c
print('Mean value >= STD =====> ', mean_i_20_modified_x_cross >= std_i_20_mo
print('-------------------------------------------------------')
print("Max value in i_20_modified_x_cross = ", i_20_modified_x_cross_max)
print("Min value in i_20_modified_x_cross = ", i_20_modified_x_cross_min)
print('Mode value in i_20_modified_x_cross = ', i_20_modified_x_cross_mode)
```

```
Number of nan values in i_20_modified_x_cross =  0
Number of 0 values in i_20_modified_x_cross =  1525
Number of values in i_20_modified_x_cross =  3599
--------------------------------------------------
Mean of values in i_20_modified_x_cross =  0.015141044620290318
STD of 1-D values in i_20_modified_x_cross =  0.01825906357346775
Mean value >= STD =====>  False
--------------------------------------------------
Max value in i_20_modified_x_cross =  0.10568955009085096
Min value in i_20_modified_x_cross =  0.0
Mode value in i_20_modified_x_cross =  0.0
```

In [120…

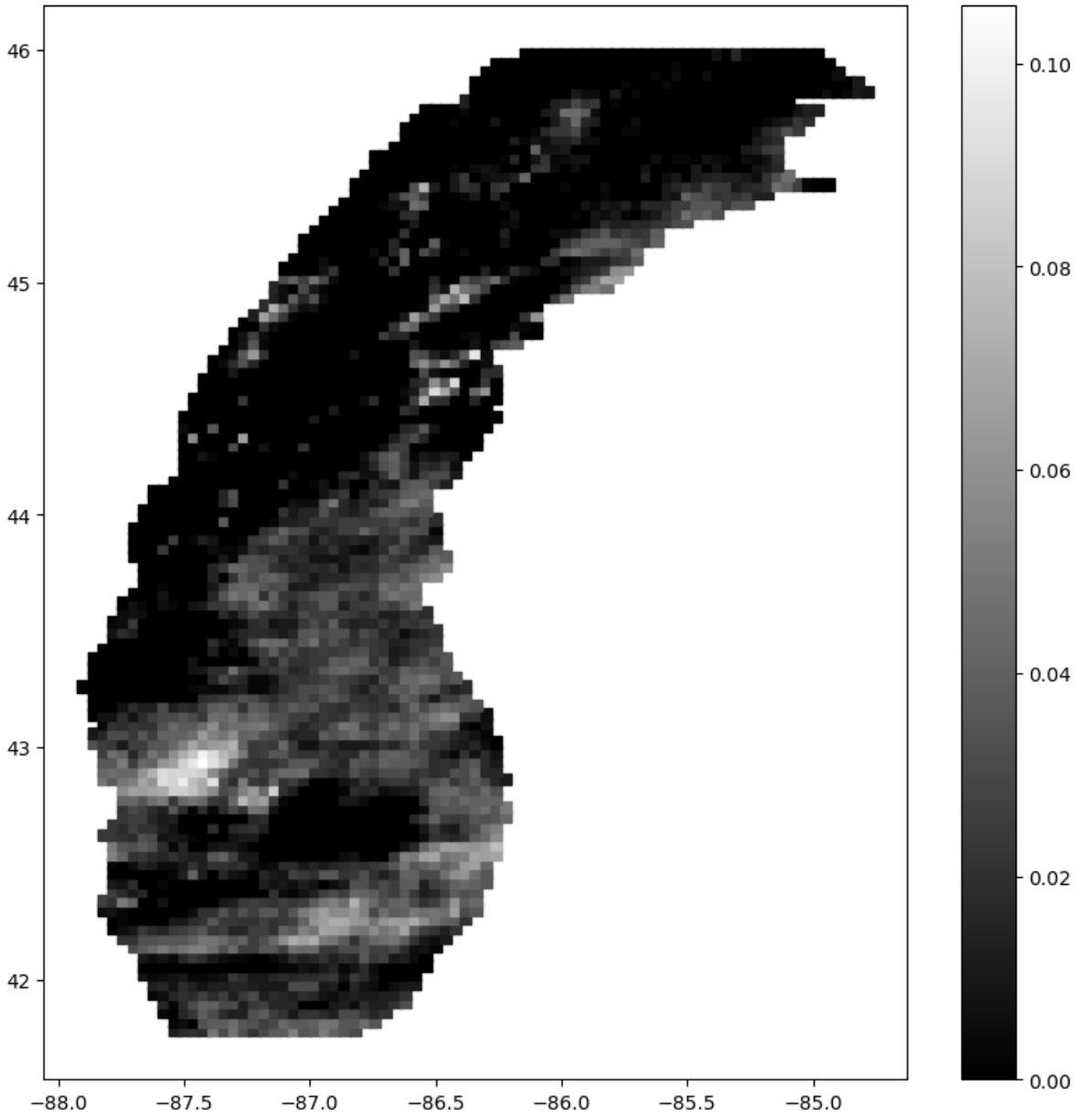
```
fig = px.histogram(i_20_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [121…

```
# x_20, y_20, i_20 = df_20.longitude, df_20.latitude, df_20.value
plt.figure(figsize=(10, 10))
plt.scatter(df_20.longitude, y_20.values, c=i_20_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_20), len(y_20), len(i_20_modified_x_cross)
```

Out[121]: (3599, 3599, 3599)



In [ ]:

In [ ]:

3.2 2016.12.11.1500

But before this, let's take a look at what happened at `1500`.

image

In [122…

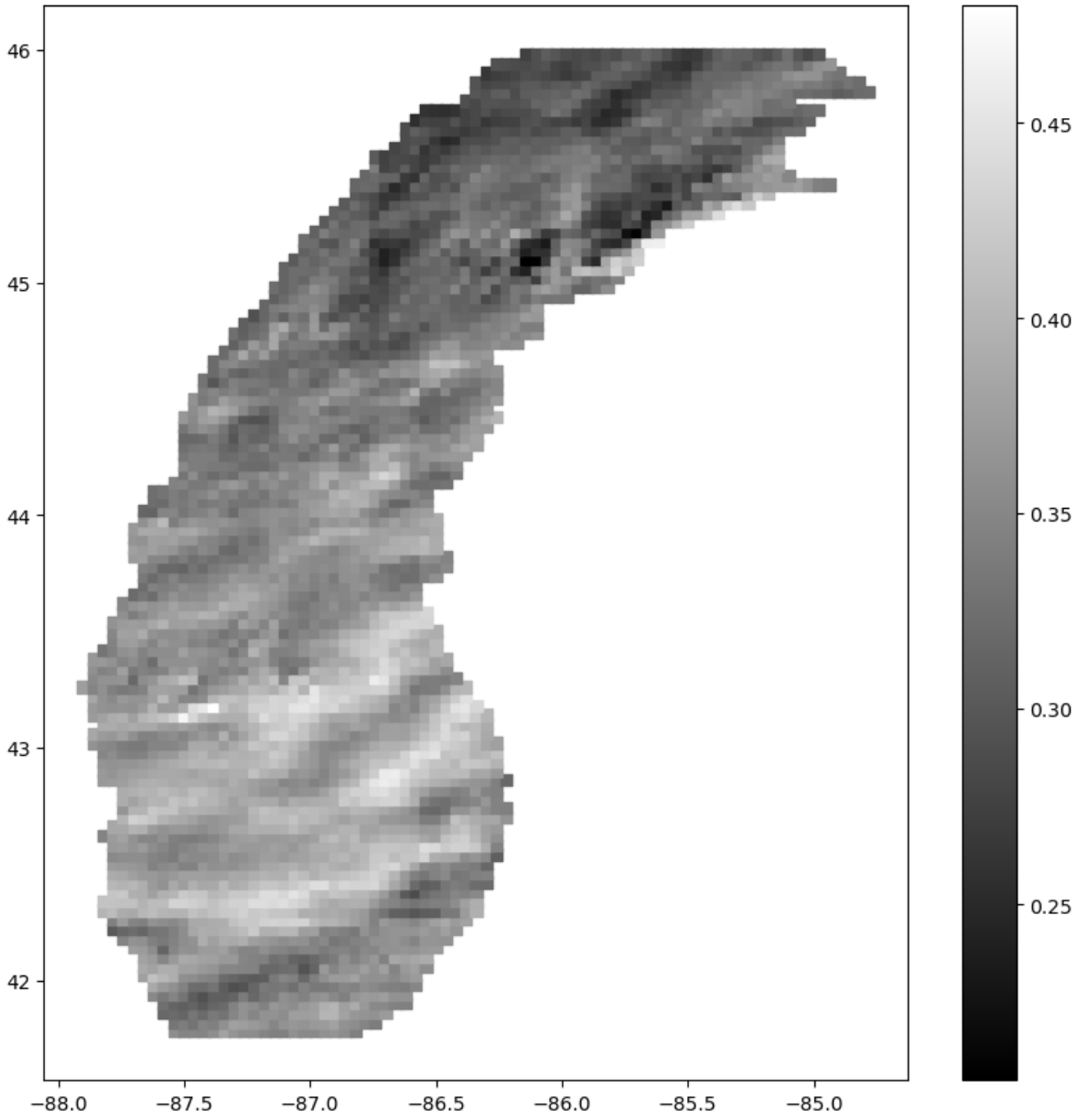
```
df_24 = pd.read_csv(zone_0_folder_path + file_list[24])

print(file_list[24])
```

```
1.1500.v01.nc-var1-t0.csv
```

In [123…

```
x_24, y_24, i_24 = df_24.longitude, df_24.latitude, df_24.value
plt.figure(figsize=(10, 10))
plt.scatter(x_24.values, y_24.values, c=i_24.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_24), len(y_24), len(i_24)
```

Out[123]: (3599, 3599, 3599)



In [124…

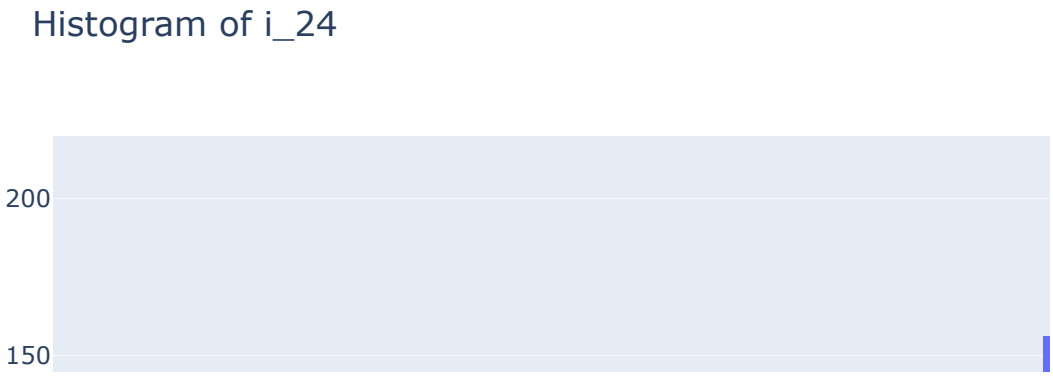
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_24 = np.nanmean(i_24.values)
std_i_24 = np.nanstd(i_24.values)

# Check number of nan values and 0s in i, just in case
num_nan_24 = i_24.isna().sum().sum()
num_zeros_24 = (i_24 == 0).sum().sum()

max in i
```

```
i_24_max = i_24.max().max()
i_24_min = i_24.min().min()
i_24_mode = i_24.mode()[0]

print('Number of nan values in i_24 = ', num_nan_24)
print('Number of 0 values in i_24 = ', num_zeros_24)
print('Number of values in i_24 = ', len(i_24))
print('-------------------------------------------------------')
print('Mean of values in i_24 = ', mean_i_24)
print('STD of 1-D values in i_24 = ', std_i_24)
print('Mean value >= STD =====> ', mean_i_24 >= std_i_24)
print('-------------------------------------------------------')
print("Max value in i_24 = ", i_24_max)
print("Min value in i_24 = ", i_24_min)
print('Mode value in i_24 = ', i_24_mode)
```

```
Number of nan values in i_24 =  0
Number of 0 values in i_24 =  0
Number of values in i_24 =  3599
-------------------------------------------------------
Mean of values in i_24 =  0.35145734378994165
STD of 1-D values in i_24 =  0.04130152211299466
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_24 =  0.48
Min value in i_24 =  0.205
Mode value in i_24 =  0.35
```

In [125…

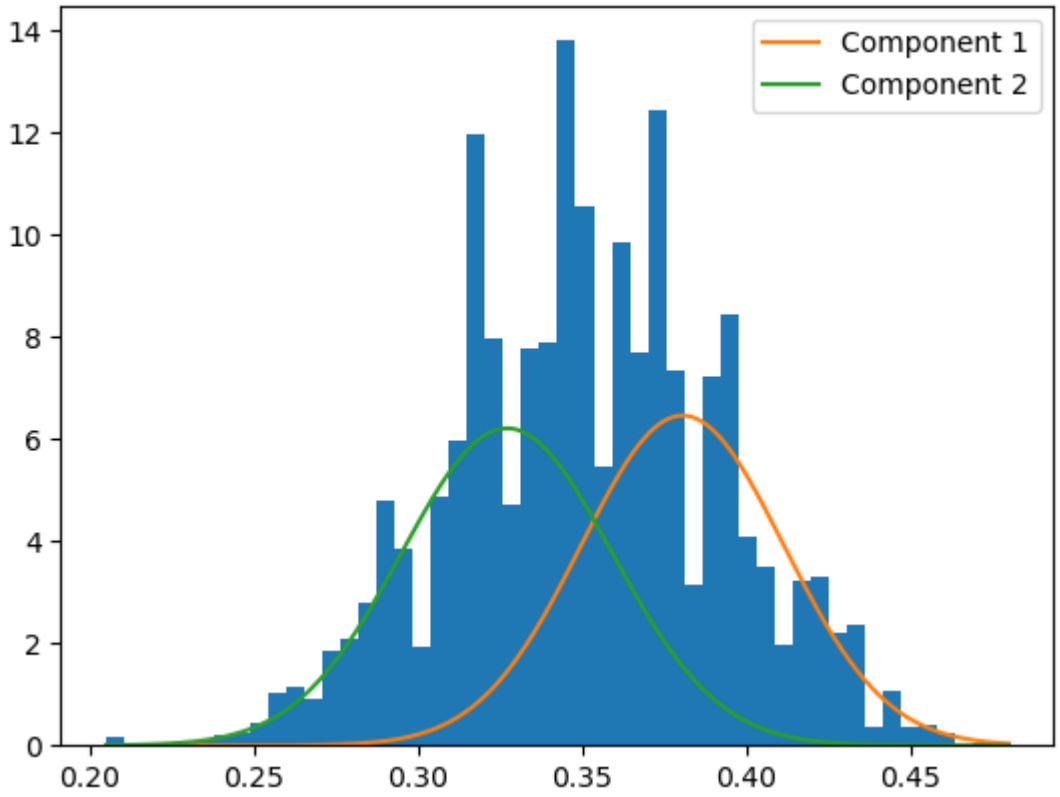
```
fig = px.histogram(i_24.values.flatten(), title="Histogram of i_24")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



It is interesting to see how there is only one distribution in this cloud, well, it is certain since the whole lake area is covered by the cloud.

In [126…

```
x_24 = np.array(i_24).reshape(-1, 1)
# print(x_24)
# Fit a mixture of two Gaussians to the data
gmm_24 = GaussianMixture(n_components=2).fit(x_24)

# Get the means and standard deviations of the two Gaussian components
mu1_24, mu2_24 = gmm_24.means_.flatten()
sigma1_24, sigma2_24 = np.sqrt(gmm_24.covariances_.flatten())

# Visualize the results
plt.hist(i_24, density=True, bins = 50)
# x_axis = i_24.unique()
x_axis_24 = np.linspace(min(i_24), max(i_24), 3000)
plt.plot(x_axis_24, 0.5 * np.exp(-(x_axis_24 - mu1_24)**2 / (2 * sigma1_24**
plt.plot(x_axis_24, 0.5 * np.exp(-(x_axis_24 - mu2_24)**2 / (2 * sigma2_24**
plt.legend()
```



In [127…
```
print('The mean of the 1st normal distribution = ', mu1_24)
print('The mean of the 2nd normal distribution = ', mu2_24)
```

```
The mean of the 1st normal distribution =  0.38058511512577226
The mean of the 2nd normal distribution =  0.32709499960979194
```

In [128…
```
print('The std of the 1st normal distribution = ', sigma1_24)
print('The std of the 2nd normal distribution = ', sigma2_24)
```

```
The std of the 1st normal distribution =  0.030895438009250785
The std of the 2nd normal distribution =  0.03213847086012528
```

In [129…
```
# Define the two normal distributions
mean1, std1 = mu1_24, sigma1_24
mean2, std2 = mu2_24, sigma2_24

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_24 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_24[0])
```

```
Cross point of two normal distributions: x = 0.3536356482183719
```

```
uce by x_cross
```

```
# If we were to reduce the values by x_cross value as mentioned above
i_24_modified_x_cross = i_24.apply(lambda x: max(0, x - x_cross_24[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_24_modified_x_cross = np.nanmean(i_24_modified_x_cross.values)
std_i_24_modified_x_cross = np.nanstd(i_24_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_24_modified_x_cross = i_24_modified_x_cross.isna().sum().sum()
num_zeros_24_modified_x_cross = (i_24_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_24_modified_x_cross_max = i_24_modified_x_cross.max().max()
i_24_modified_x_cross_min = i_24_modified_x_cross.min().min()
# i_24_modified_x_cross_min = 0
i_24_modified_x_cross_mode = i_24_modified_x_cross.mode()[0]

print('Number of nan values in i_24_modified_x_cross = ', num_nan_24_modifie
print('Number of 0 values in i_24_modified_x_cross = ', num_zeros_24_modifie
print('Number of values in i_24_modified_x_cross = ', len(i_24_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_24_modified_x_cross = ', mean_i_24_modified_x_cro
print('STD of 1-D values in i_24_modified_x_cross = ', std_i_24_modified_x_c
print('Mean value >= STD =====> ', mean_i_24_modified_x_cross >= std_i_24_mo
print('-------------------------------------------------------')
print("Max value in i_24_modified_x_cross = ", i_24_modified_x_cross_max)
print("Min value in i_24_modified_x_cross = ", i_24_modified_x_cross_min)
print('Mode value in i_24_modified_x_cross = ', i_24_modified_x_cross_mode)
```

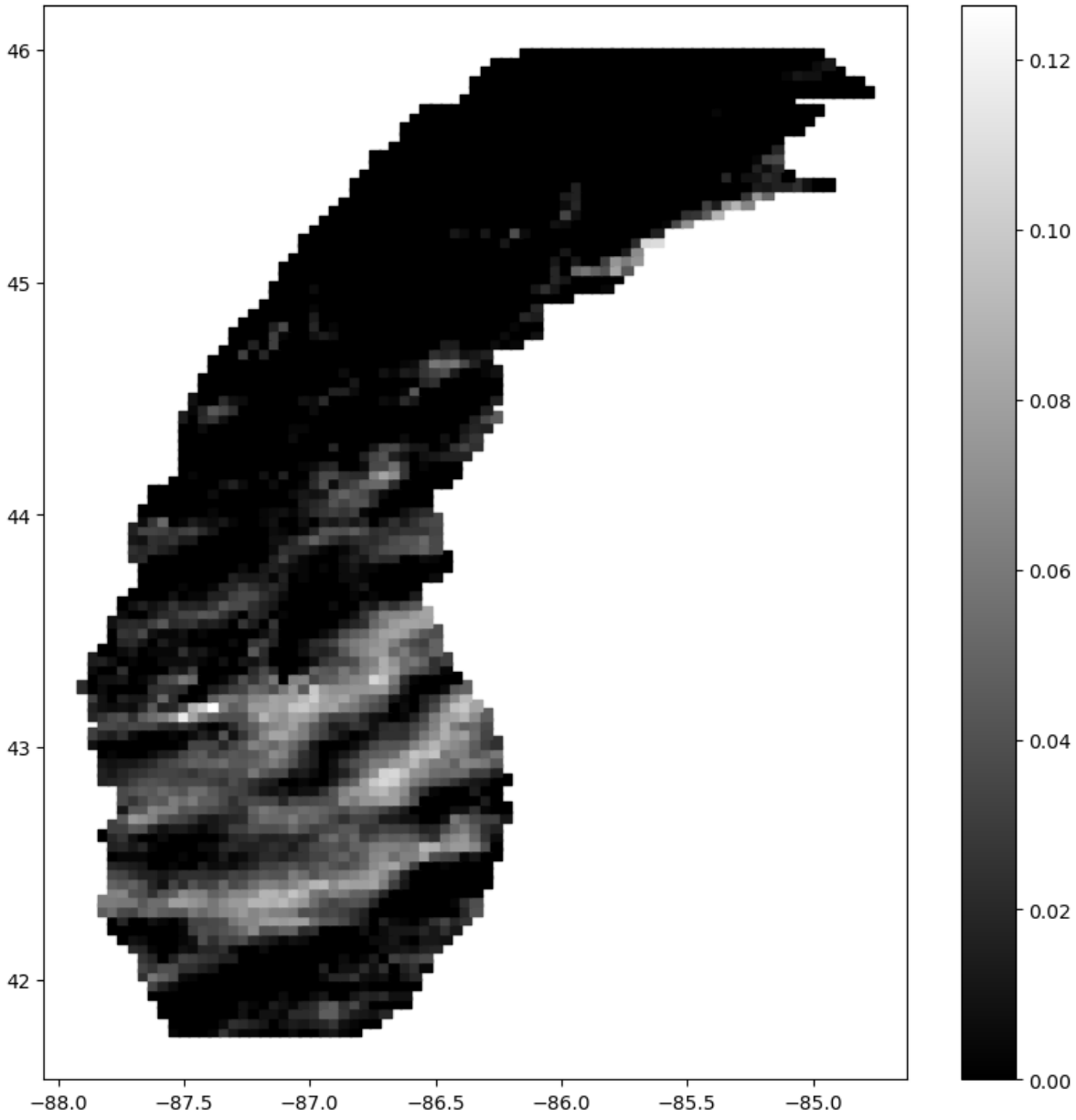
```
Number of nan values in i_24_modified_x_cross =  0
Number of 0 values in i_24_modified_x_cross =  1922
Number of values in i_24_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_24_modified_x_cross =  0.015541262883520512
STD of 1-D values in i_24_modified_x_cross =  0.02322435587148726
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_24_modified_x_cross =  0.12636435178162808
Min value in i_24_modified_x_cross =  0.0
Mode value in i_24_modified_x_cross =  0.0
```

In [131…

```
fig = px.histogram(i_24_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```

In [132…
```
# x_24, y_24, i_24 = df_24.longitude, df_24.latitude, df_24.value
plt.figure(figsize=(10, 10))
plt.scatter(df_24.longitude, y_24.values, c=i_24_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_24), len(y_24), len(i_24_modified_x_cross)
```

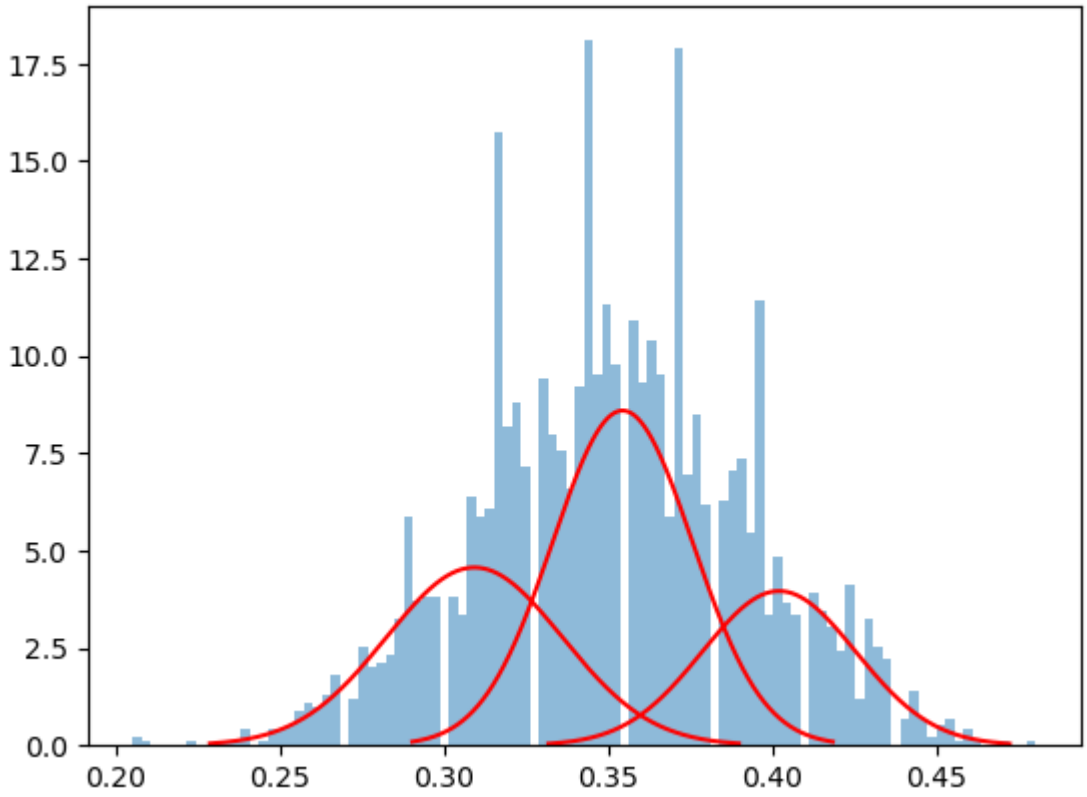
Out[132]: (3599, 3599, 3599)



In [133…

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Create a sample 1-D array
i_3N_24 = np.array(i_24).reshape(-1, 1)

# Set the number of components to 3
n_components = 3

# Define the Gaussian mixture model
gmm = GaussianMixture(n_components=n_components)

# Fit the model to the data
gmm.fit(i_3N_24.reshape(-1, 1))

# Get the mean and standard deviation for each component
s_.flatten()
```

```
stds = np.sqrt(gmm.covariances_.flatten())

# Plot the histogram and the fitted Gaussian curves
plt.hist(i_3N_24, bins=100, density=True, alpha=0.5)
for i in range(n_components):
    x = np.linspace(means[i] - 3 * stds[i], means[i] + 3 * stds[i], 3000)
    plt.plot(x, gmm.weights_[i] * np.exp(-0.5 * ((x - means[i]) / stds[i]) *
plt.show()
```



It does reveal the thickest part of the cloud above Lake Michigan.

3.2 2016.12.11.1500

But before this, let's take a look at what happened at `1600`.

image

Full-lit?

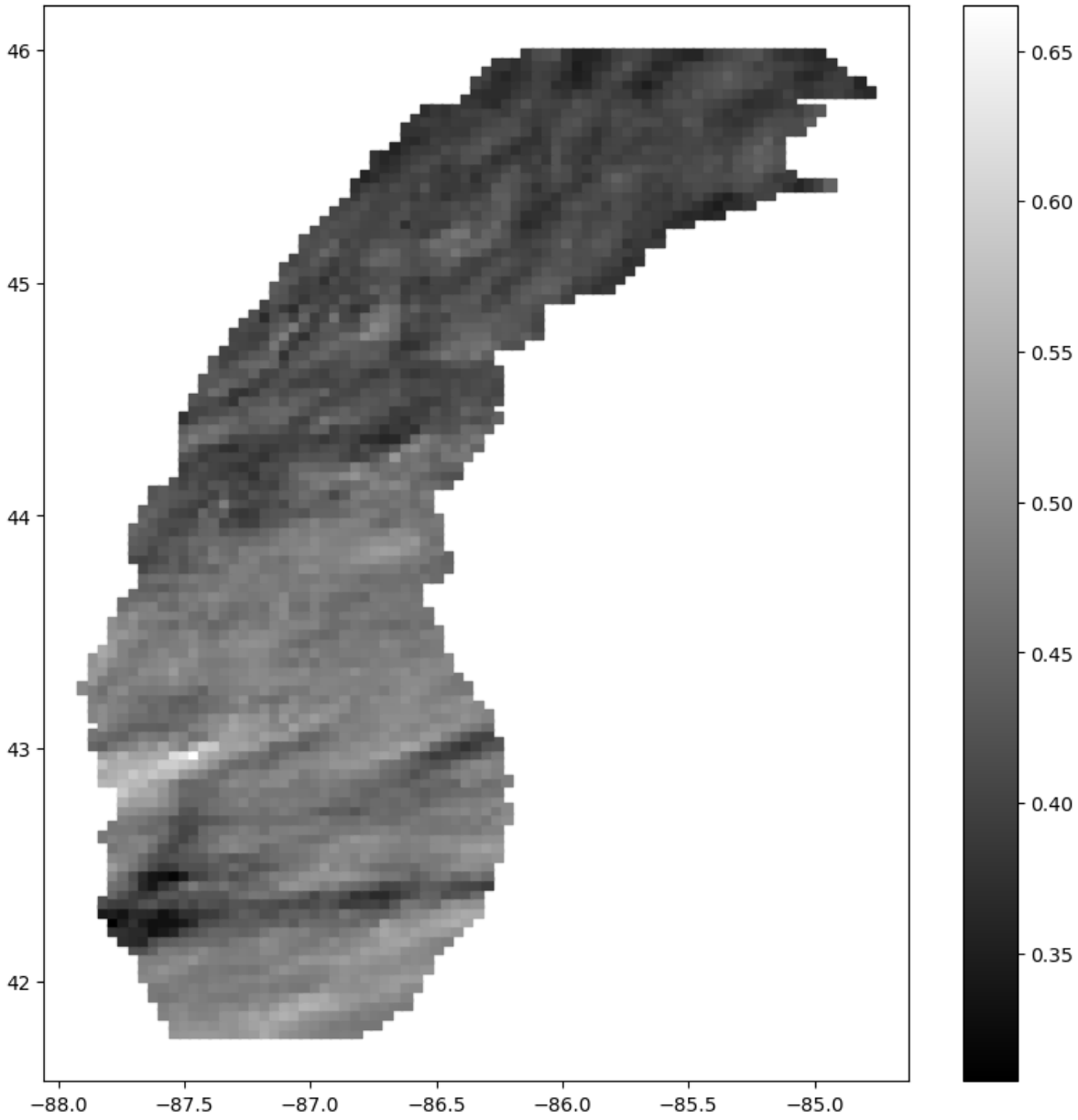
In [134…

```
df_27 = pd.read_csv(zone_0_folder_path + file_list[27])

print(file_list[27])

x_27, y_27, i_27 = df_27.longitude, df_27.latitude, df_27.value
plt.figure(figsize=(10, 10))
plt.scatter(x_27.values, y_27.values, c=i_27.values, cmap=cm.gray, marker='s
```

```
plt.colorbar(orientation='vertical')
len(x_27), len(y_27), len(i_27)
```

```
goes15.2016.12.11.1600.v01.nc-var1-t0.csv
```

Out[134]: (3599, 3599, 3599)



In [135…

```
# Calculate the mean and standard deviation of the 1-D array

mean_i_27 = np.nanmean(i_27.values)
std_i_27 = np.nanstd(i_27.values)

# Check number of nan values and 0s in i, just in case
num_nan_27 = i_27.isna().sum().sum()
num_zeros_27 = (i_27 == 0).sum().sum()

# Check min and max in i

max().max()
min().min()
```

```
i_27_mode = i_27.mode()[0]

print('Number of nan values in i_27 = ', num_nan_27)
print('Number of 0 values in i_27 = ', num_zeros_27)
print('Number of values in i_27 = ', len(i_27))
print('-----------------------------------------------------')
print('Mean of values in i_27 = ', mean_i_27)
print('STD of 1-D values in i_27 = ', std_i_27)
print('Mean value >= STD =====> ', mean_i_27 >= std_i_27)
print('-----------------------------------------------------')
print("Max value in i_27 = ", i_27_max)
print("Min value in i_27 = ", i_27_min)
print('Mode value in i_27 = ', i_27_mode)
```

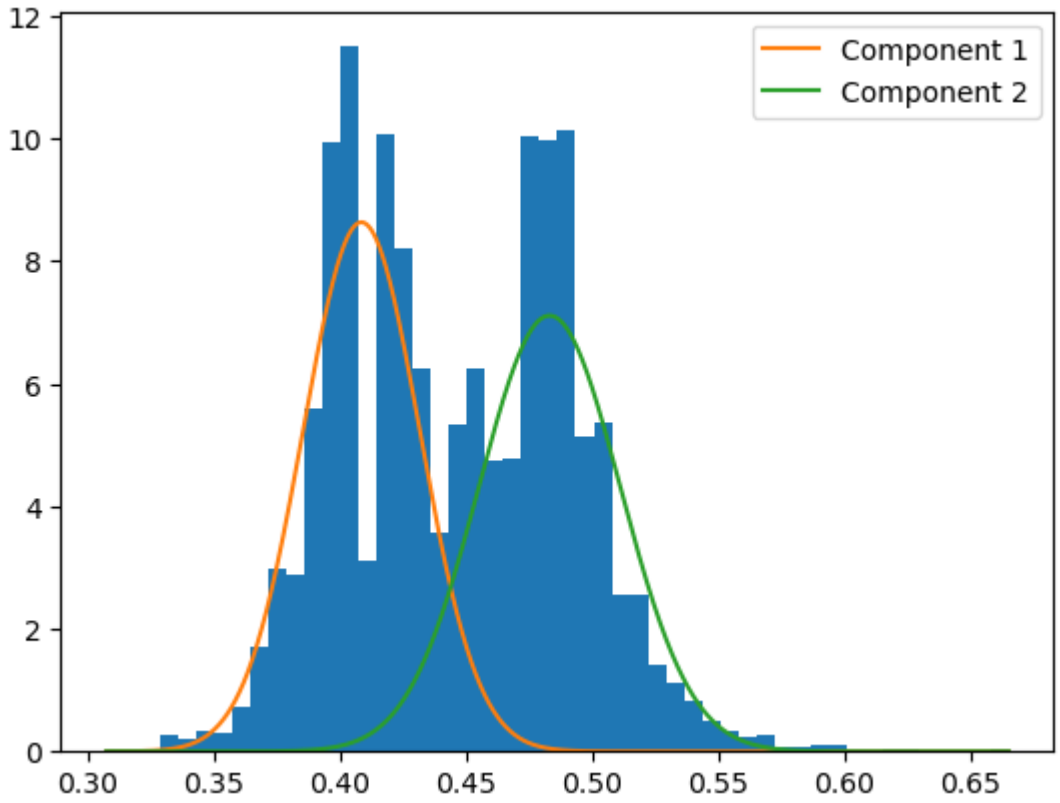
```
Number of nan values in i_27 =  0
Number of 0 values in i_27 =  0
Number of values in i_27 =  3599
-----------------------------------------------------
Mean of values in i_27 =  0.4470345856932481
STD of 1-D values in i_27 =  0.045271273389603475
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_27 =  0.66499996
Min value in i_27 =  0.3075
Mode value in i_27 =  0.405
```

In [136…

```
fig = px.histogram(i_27.values.flatten(), title="Histogram of i_27")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```

In [137…

```
# print(i_27)
x_27 = np.array(i_27).reshape(-1, 1)
# print(x_27)
# Fit a mixture of two Gaussians to the data
gmm_27 = GaussianMixture(n_components=2).fit(x_27)

# Get the means and standard deviations of the two Gaussian components
mu1_27, mu2_27 = gmm_27.means_.flatten()
sigma1_27, sigma2_27 = np.sqrt(gmm_27.covariances_.flatten())

# Visualize the results
plt.hist(i_27, density=True, bins = 50)
# x_axis = i_27.unique()
x_axis_27 = np.linspace(min(i_27), max(i_27), 3000)
plt.plot(x_axis_27, 0.5 * np.exp(-(x_axis_27 - mu1_27)**2 / (2 * sigma1_27**
plt.plot(x_axis_27, 0.5 * np.exp(-(x_axis_27 - mu2_27)**2 / (2 * sigma2_27**
plt.legend()
plt.show()
```



In [138…
```
print('The mean of the 1st normal distribution = ', mu1_27)
print('The mean of the 2nd normal distribution = ', mu2_27)
```

```
The mean of the 1st normal distribution =  0.40859393325502935
The mean of the 2nd normal distribution =  0.48310656202985924
```

In [139…
```
print('The std of the 1st normal distribution = ', sigma1_27)
print('The std of the 2nd normal distribution = ', sigma2_27)
```

```
The std of the 1st normal distribution =  0.02309588165241067
The std of the 2nd normal distribution =  0.028040302354565616
```

In [140…
```
# Define the two normal distributions
mean1, std1 = mu1_27, sigma1_27
mean2, std2 = mu2_27, sigma2_27

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_27 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_27[0])
```

```
Cross point of two normal distributions: x = 0.44392652913980124
```

4. Case 4

`Duration` : 2016.12.12 1700-1900 UTC

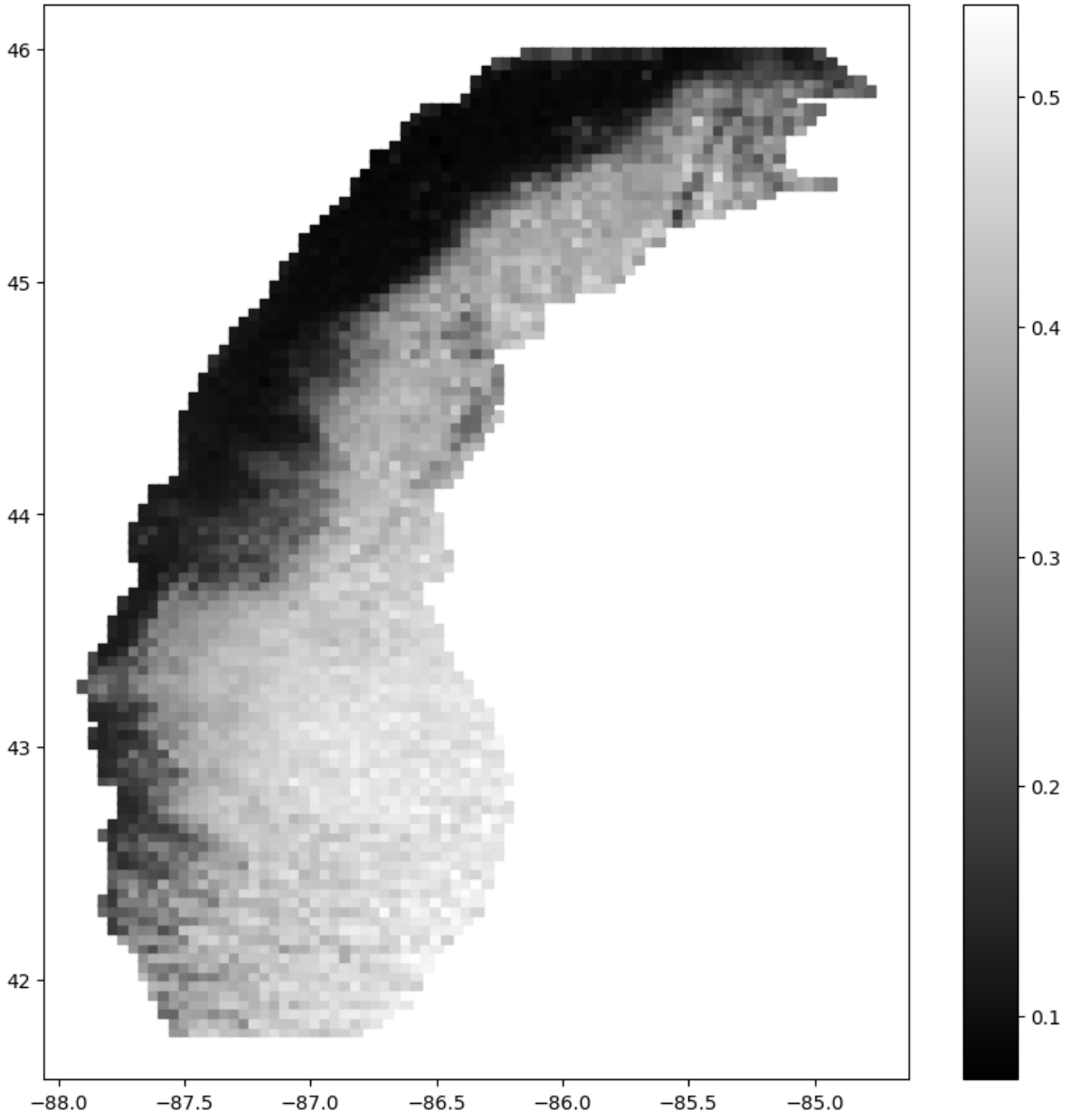
4.1 2016.12.11.1700

In [141…

```
df_28 = pd.read_csv(zone_0_folder_path + file_list[28])

print(file_list[28])

x_28, y_28, i_28 = df_28.longitude, df_28.latitude, df_28.value
plt.figure(figsize=(10, 10))
plt.scatter(x_28.values, y_28.values, c=i_28.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_28), len(y_28), len(i_28)
```

```
goes15.2016.12.12.1700.v01.nc-var1-t0.csv
```

Out[141]: (3599, 3599, 3599)



In [142…

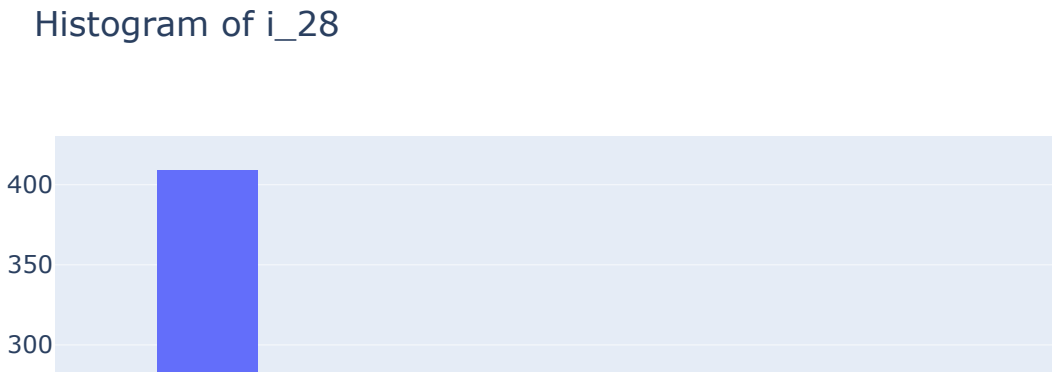
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_28 = np.nanmean(i_28.values)
std_i_28 = np.nanstd(i_28.values)

# Check number of nan values and 0s in i, just in case
num_nan_28 = i_28.isna().sum().sum()
num_zeros_28 = (i_28 == 0).sum().sum()

# Check min and max in i

i_28_max = i_28.max().max()
i_28_min = i_28.min().min()
i_28_mode = i_28.mode()[0]

print('Number of nan values in i_28 = ', num_nan_28)
print('Number of 0 values in i_28 = ', num_zeros_28)
print('Number of values in i_28 = ', len(i_28))
```

```
print('-----------------------------------------------------')
print('Mean of values in i_28 = ', mean_i_28)
print('STD of 1-D values in i_28 = ', std_i_28)
print('Mean value >= STD =====> ', mean_i_28 >= std_i_28)
print('-----------------------------------------------------')
print("Max value in i_28 = ", i_28_max)
print("Min value in i_28 = ", i_28_min)
print('Mode value in i_28 = ', i_28_mode)
```

```
Number of nan values in i_28 =  0
Number of 0 values in i_28 =  0
Number of values in i_28 =  3599
-----------------------------------------------------
Mean of values in i_28 =  0.32055014792386766
STD of 1-D values in i_28 =  0.14061549346140717
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_28 =  0.53999996
Min value in i_28 =  0.0725
Mode value in i_28 =  0.0925
```

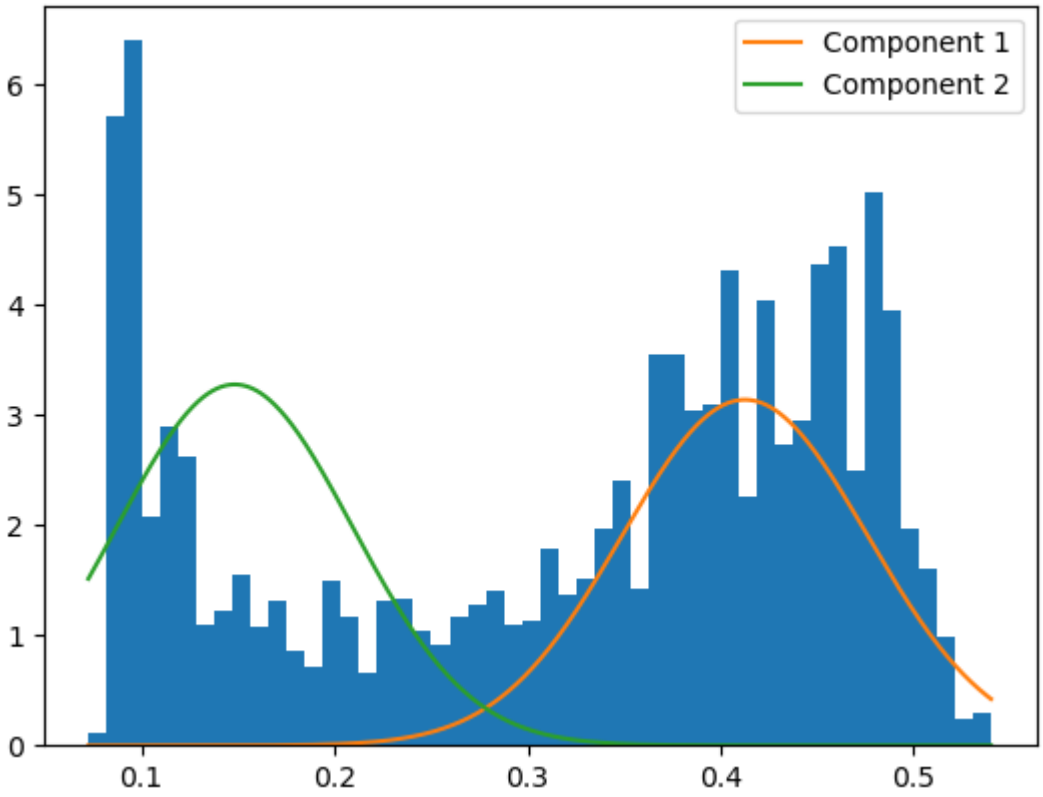
In [143…

```
fig = px.histogram(i_28.values.flatten(), title="Histogram of i_28")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



In [144…

```
x_28 = np.array(i_28).reshape(-1, 1)
# print(x_28)
# Fit a mixture of two Gaussians to the data
gmm_28 = GaussianMixture(n_components=2).fit(x_28)

# Get the means and standard deviations of the two Gaussian components
mu1_28, mu2_28 = gmm_28.means_.flatten()
sigma1_28, sigma2_28 = np.sqrt(gmm_28.covariances_.flatten())

# Visualize the results
plt.hist(i_28, density=True, bins = 50)
# x_axis = i_28.unique()
x_axis_28 = np.linspace(min(i_28), max(i_28), 3000)
plt.plot(x_axis_28, 0.5 * np.exp(-(x_axis_28 - mu1_28)**2 / (2 * sigma1_28**
plt.plot(x_axis_28, 0.5 * np.exp(-(x_axis_28 - mu2_28)**2 / (2 * sigma2_28**
plt.legend()
plt.show()
```



In [145…
```
print('The mean of the 1st normal distribution = ', mu1_28)
print('The mean of the 2nd normal distribution = ', mu2_28)
```

```
The mean of the 1st normal distribution =  0.4124905545437214
The mean of the 2nd normal distribution =  0.14826611939758239
```

In [146…
```
print('The std of the 1st normal distribution = ', sigma1_28)
print('The std of the 2nd normal distribution = ', sigma2_28)
```

```
The std of the 1st normal distribution =  0.06364359353742242
The std of the 2nd normal distribution =  0.06095293309437641
```

In [147…
```
# Define the two normal distributions
mean1, std1 = mu1_28, sigma1_28
mean2, std2 = mu2_28, sigma2_28

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_28 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_28[0])
```

```
Cross point of two normal distributions: x = 0.278159510337084
```

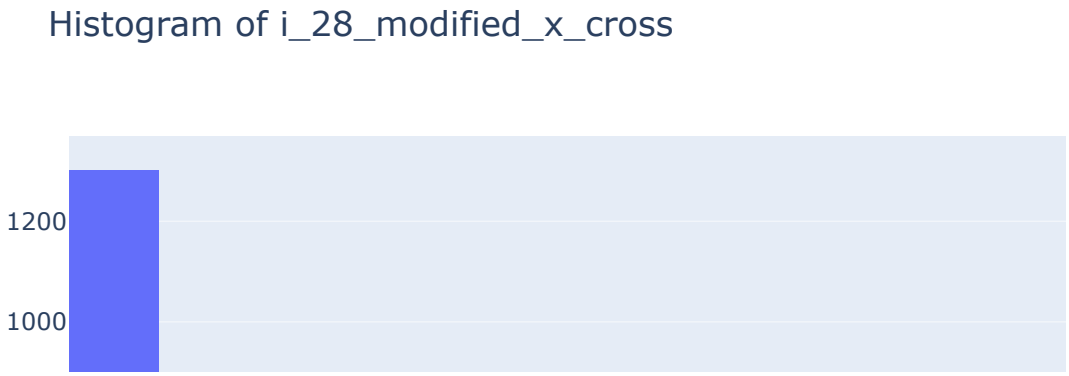
```
uce by x_cross
```

```
# If we were to reduce the values by x_cross value as mentioned above
i_28_modified_x_cross = i_28.apply(lambda x: max(0, x - x_cross_28[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_28_modified_x_cross = np.nanmean(i_28_modified_x_cross.values)
std_i_28_modified_x_cross = np.nanstd(i_28_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_28_modified_x_cross = i_28_modified_x_cross.isna().sum().sum()
num_zeros_28_modified_x_cross = (i_28_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_28_modified_x_cross_max = i_28_modified_x_cross.max().max()
i_28_modified_x_cross_min = i_28_modified_x_cross.min().min()
# i_28_modified_x_cross_min = 0
i_28_modified_x_cross_mode = i_28_modified_x_cross.mode()[0]

print('Number of nan values in i_28_modified_x_cross = ', num_nan_28_modifie
print('Number of 0 values in i_28_modified_x_cross = ', num_zeros_28_modifie
print('Number of values in i_28_modified_x_cross = ', len(i_28_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_28_modified_x_cross = ', mean_i_28_modified_x_cro
print('STD of 1-D values in i_28_modified_x_cross = ', std_i_28_modified_x_c
print('Mean value >= STD =====> ', mean_i_28_modified_x_cross >= std_i_28_mo
print('-------------------------------------------------------')
print("Max value in i_28_modified_x_cross = ", i_28_modified_x_cross_max)
print("Min value in i_28_modified_x_cross = ", i_28_modified_x_cross_min)
print('Mode value in i_28_modified_x_cross = ', i_28_modified_x_cross_mode)
```

```
Number of nan values in i_28_modified_x_cross =  0
Number of 0 values in i_28_modified_x_cross =  1277
Number of values in i_28_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_28_modified_x_cross =  0.08805601613984186
STD of 1-D values in i_28_modified_x_cross =  0.08154422428493104
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_28_modified_x_cross =  0.2618404496629159
Min value in i_28_modified_x_cross =  0.0
Mode value in i_28_modified_x_cross =  0.0
```

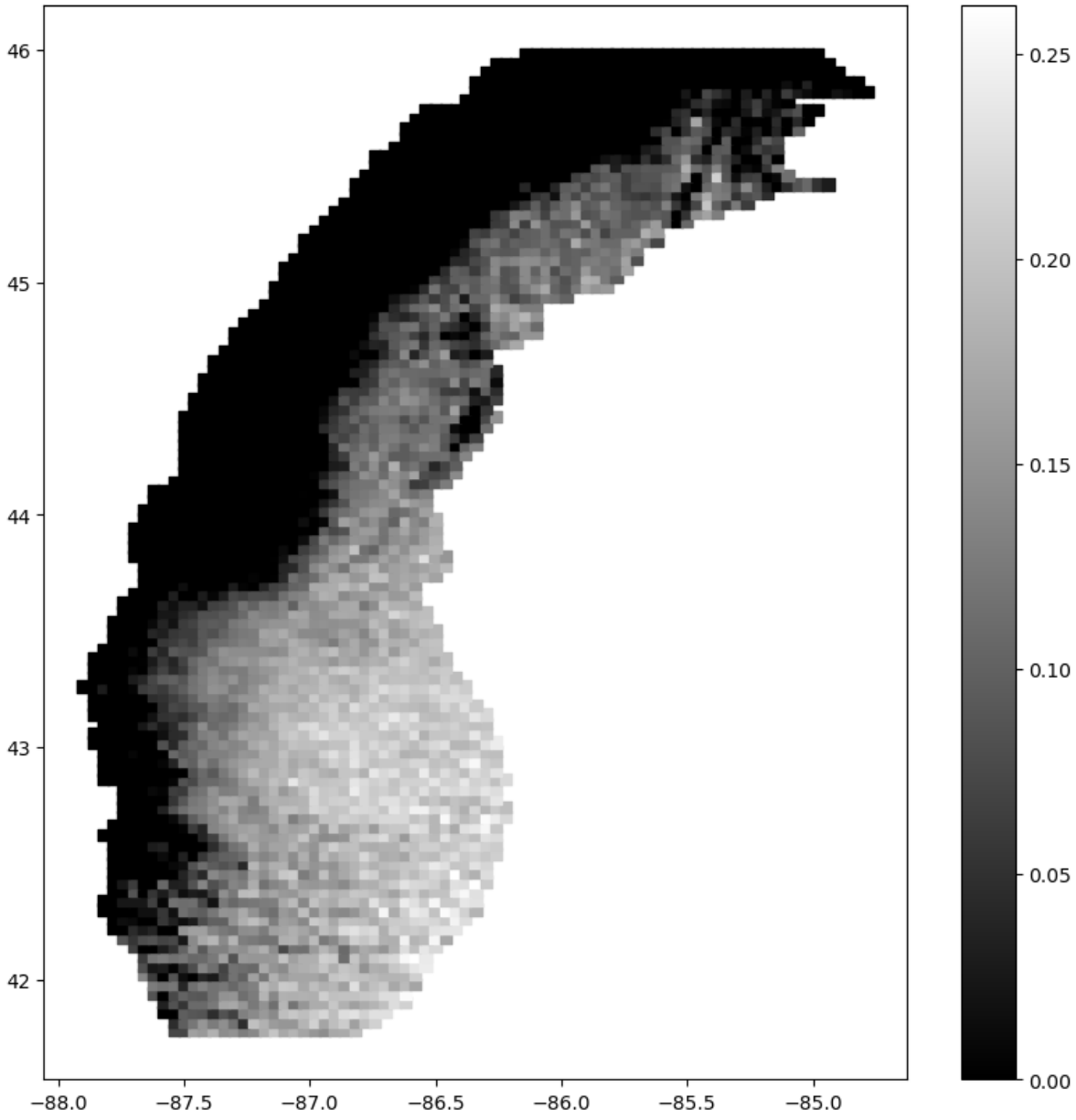
In [149…

```
fig = px.histogram(i_28_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [150…
```
# x_28, y_28, i_28 = df_28.longitude, df_28.latitude, df_28.value
plt.figure(figsize=(10, 10))
plt.scatter(df_28.longitude, y_28.values, c=i_28_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_28), len(y_28), len(i_28_modified_x_cross)
```

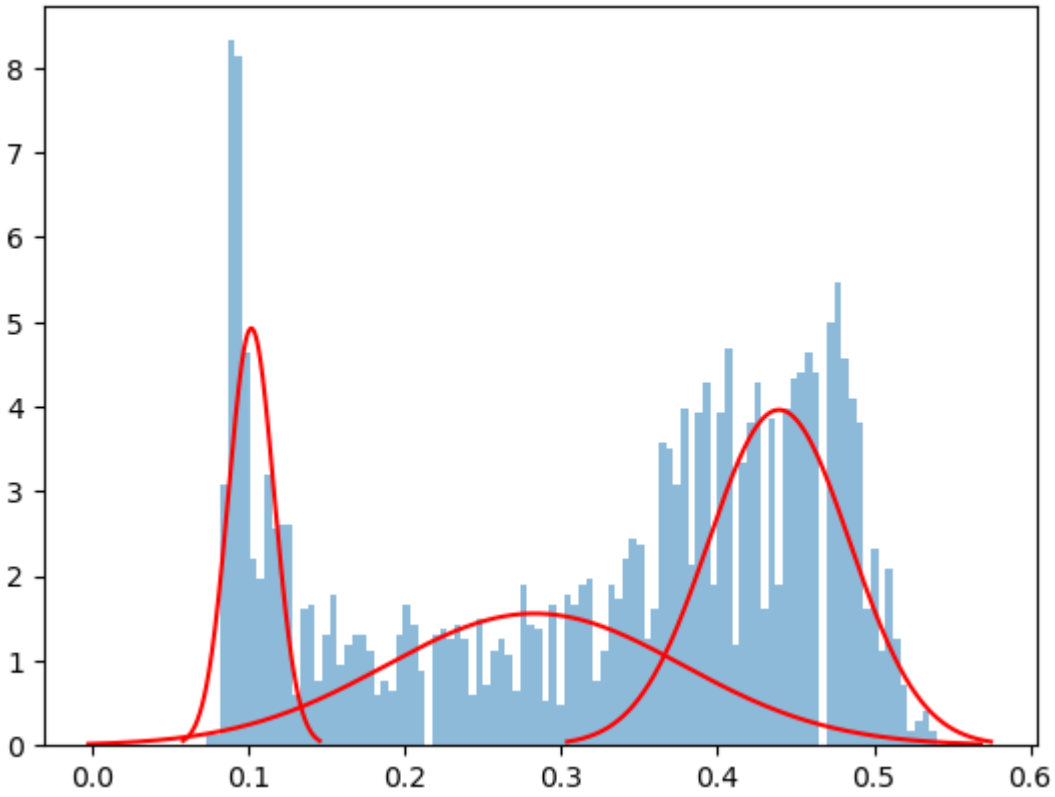
Out[150]: (3599, 3599, 3599)



In [151…

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Create a sample 1-D array
i_3N_28 = np.array(i_28).reshape(-1, 1)

# Set the number of components to 3
n_components = 3

# Define the Gaussian mixture model
gmm = GaussianMixture(n_components=n_components)

# Fit the model to the data
gmm.fit(i_3N_28.reshape(-1, 1))

# Get the mean and standard deviation for each component
s_.flatten()
```

```
stds = np.sqrt(gmm.covariances_.flatten())

# Plot the histogram and the fitted Gaussian curves
plt.hist(i_3N_28, bins=100, density=True, alpha=0.5)
for i in range(n_components):
    x = np.linspace(means[i] - 3 * stds[i], means[i] + 3 * stds[i], 3000)
    plt.plot(x, gmm.weights_[i] * np.exp(-0.5 * ((x - means[i]) / stds[i]) *
plt.show()
```



That is pretty neat in this case.

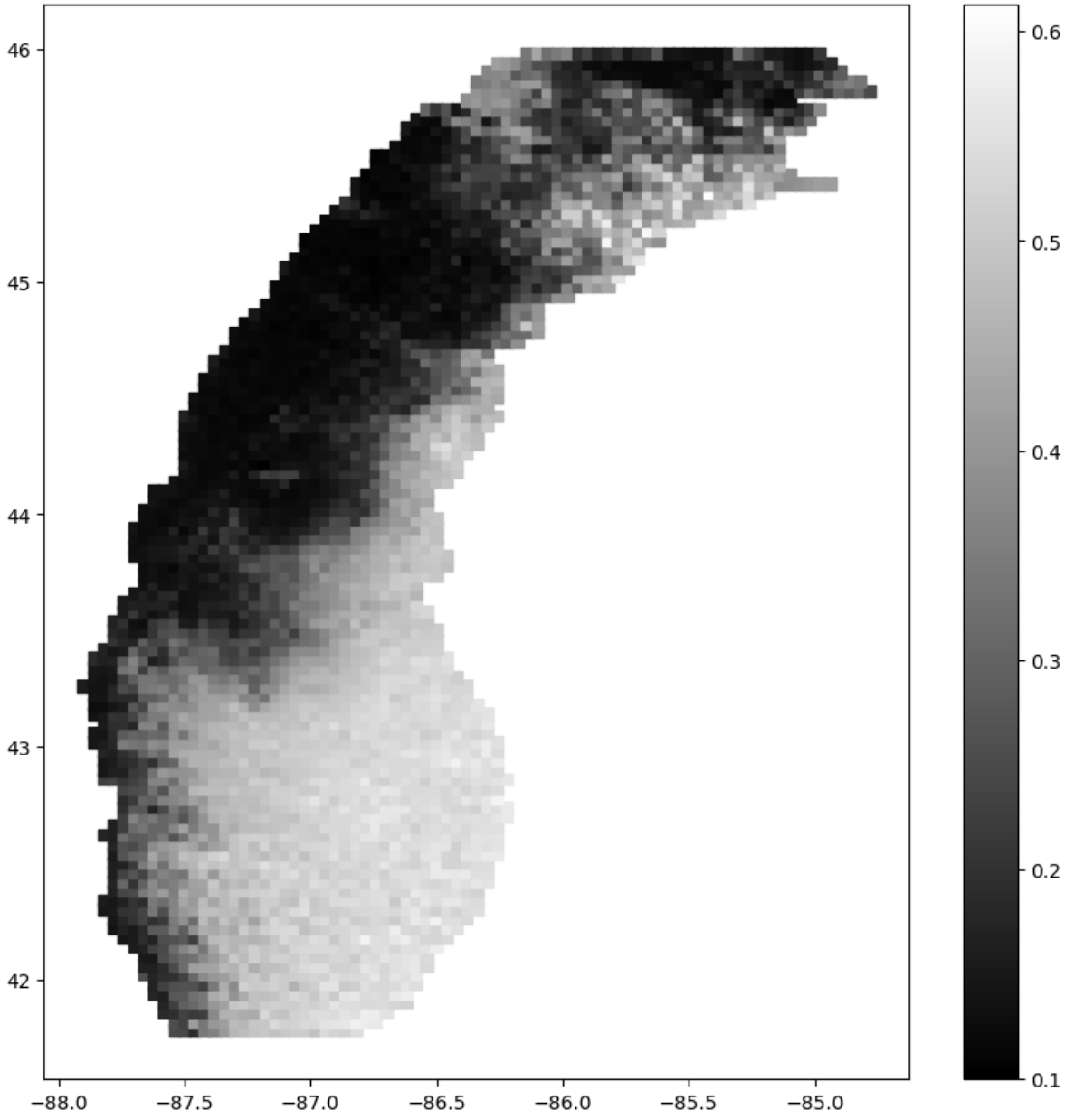
4.2 2016.12.12.1900

In [152…
```
df_35 = pd.read_csv(zone_0_folder_path + file_list[35])

print(file_list[35])
```

```
goes15.2016.12.12.1900.v01.nc-var1-t0.csv
```

In [153…
```
x_35, y_35, i_35 = df_35.longitude, df_35.latitude, df_35.value
plt.figure(figsize=(10, 10))
plt.scatter(x_35.values, y_35.values, c=i_35.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_35), len(y_35), len(i_35)
```

Out[153]: (3599, 3599, 3599)



In [154…

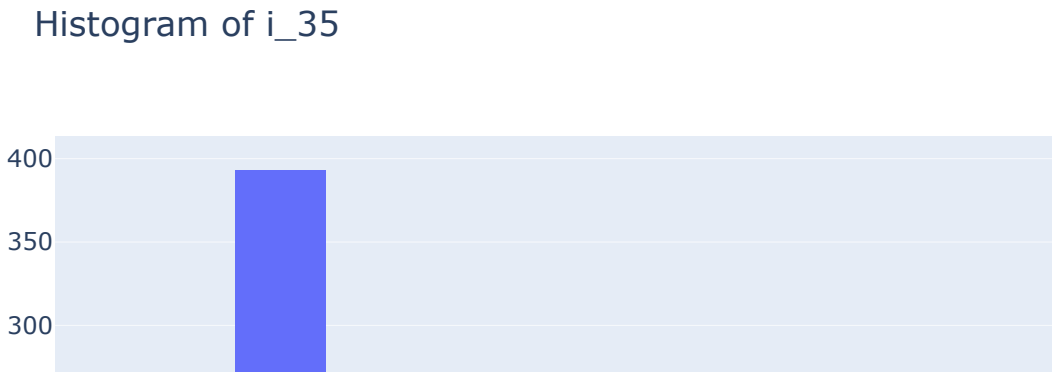
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_35 = np.nanmean(i_35.values)
std_i_35 = np.nanstd(i_35.values)

# Check number of nan values and 0s in i, just in case
num_nan_35 = i_35.isna().sum().sum()
num_zeros_35 = (i_35 == 0).sum().sum()

# Check min and max in i

i_35_max = i_35.max().max()
i_35_min = i_35.min().min()
i_35_mode = i_35.mode()[0]

print('Number of nan values in i_35 = ', num_nan_35)
print('Number of 0 values in i_35 = ', num_zeros_35)
print('Number of values in i_35 = ', len(i_35))
```

```
print('-----------------------------------------------------')
print('Mean of values in i_35 = ', mean_i_35)
print('STD of 1-D values in i_35 = ', std_i_35)
print('Mean value >= STD =====> ', mean_i_35 >= std_i_35)
print('-----------------------------------------------------')
print("Max value in i_35 = ", i_35_max)
print("Min value in i_35 = ", i_35_min)
print('Mode value in i_35 = ', i_35_mode)
```

```
Number of nan values in i_35 =  0
Number of 0 values in i_35 =  0
Number of values in i_35 =  3599
-----------------------------------------------------
Mean of values in i_35 =  0.3271360042017227
STD of 1-D values in i_35 =  0.16008814620554082
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_35 =  0.6125
Min value in i_35 =  0.099999994
Mode value in i_35 =  0.1175
```

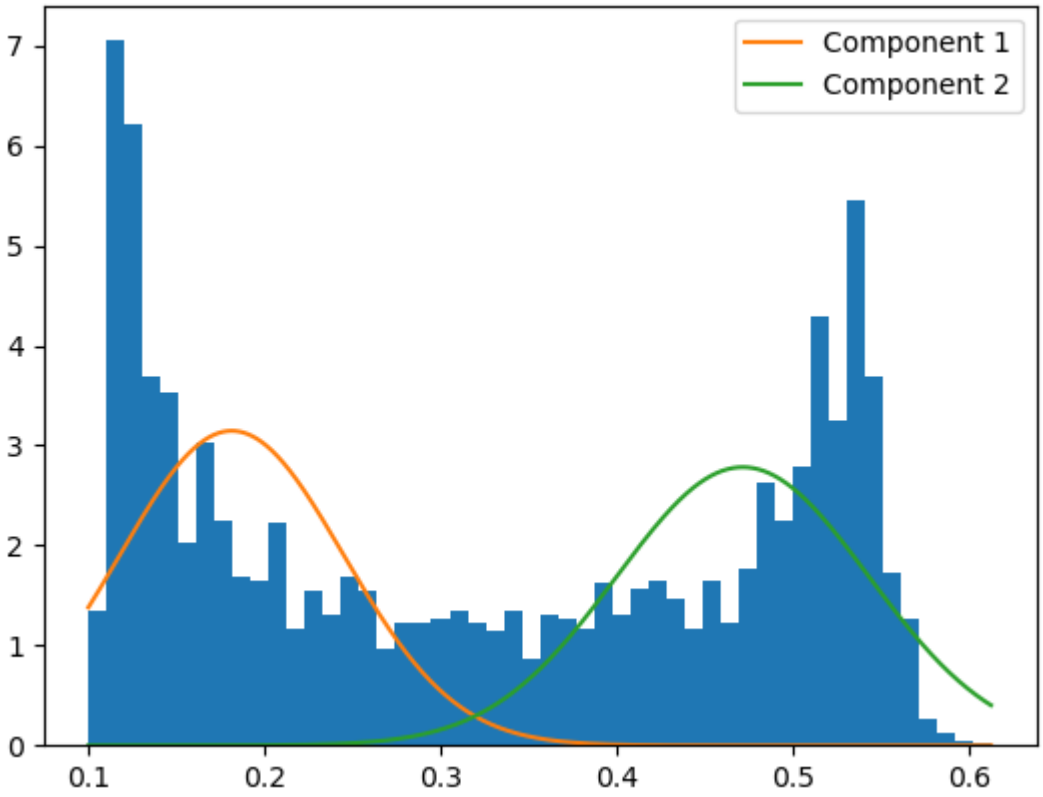
In [155…

```
fig = px.histogram(i_35.values.flatten(), title="Histogram of i_35")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



In [156…

```
x_35 = np.array(i_35).reshape(-1, 1)
# print(x_35)
# Fit a mixture of two Gaussians to the data
gmm_35 = GaussianMixture(n_components=2).fit(x_35)

# Get the means and standard deviations of the two Gaussian components
mu1_35, mu2_35 = gmm_35.means_.flatten()
sigma1_35, sigma2_35 = np.sqrt(gmm_35.covariances_.flatten())

# Visualize the results
plt.hist(i_35, density=True, bins = 50)
# x_axis = i_35.unique()
x_axis_35 = np.linspace(min(i_35), max(i_35), 3000)
plt.plot(x_axis_35, 0.5 * np.exp(-(x_axis_35 - mu1_35)**2 / (2 * sigma1_35**
plt.plot(x_axis_35, 0.5 * np.exp(-(x_axis_35 - mu2_35)**2 / (2 * sigma2_35**
plt.legend()
plt.show()
```



In [157…
```
print('The mean of the 1st normal distribution = ', mu1_35)
print('The mean of the 2nd normal distribution = ', mu2_35)
```

```
The mean of the 1st normal distribution =  0.1813680976581403
The mean of the 2nd normal distribution =  0.4715291233620202
```

In [158…
```
print('The std of the 1st normal distribution = ', sigma1_35)
print('The std of the 2nd normal distribution = ', sigma2_35)
```

```
The std of the 1st normal distribution =  0.06340231789504572
The std of the 2nd normal distribution =  0.07167645572953871
```

In [159…
```
# Define the two normal distributions
mean1, std1 = mu1_35, sigma1_35
mean2, std2 = mu2_35, sigma2_35

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_35 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_35[0])
```

```
Cross point of two normal distributions: x = 0.31948137307922875
```

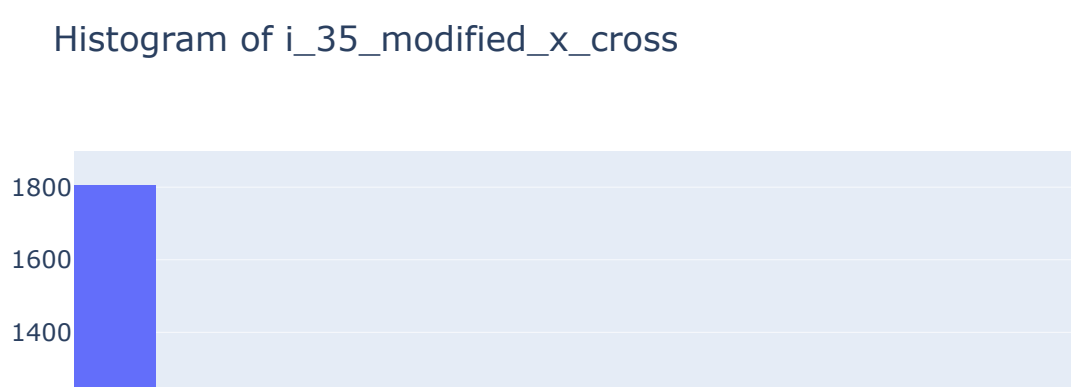
```
uce by x_cross
```

```
# If we were to reduce the values by x_cross value as mentioned above
i_35_modified_x_cross = i_35.apply(lambda x: max(0, x - x_cross_35[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_35_modified_x_cross = np.nanmean(i_35_modified_x_cross.values)
std_i_35_modified_x_cross = np.nanstd(i_35_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_35_modified_x_cross = i_35_modified_x_cross.isna().sum().sum()
num_zeros_35_modified_x_cross = (i_35_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_35_modified_x_cross_max = i_35_modified_x_cross.max().max()
i_35_modified_x_cross_min = i_35_modified_x_cross.min().min()
# i_35_modified_x_cross_min = 0
i_35_modified_x_cross_mode = i_35_modified_x_cross.mode()[0]

print('Number of nan values in i_35_modified_x_cross = ', num_nan_35_modifie
print('Number of 0 values in i_35_modified_x_cross = ', num_zeros_35_modifie
print('Number of values in i_35_modified_x_cross = ', len(i_35_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_35_modified_x_cross = ', mean_i_35_modified_x_cro
print('STD of 1-D values in i_35_modified_x_cross = ', std_i_35_modified_x_c
print('Mean value >= STD =====> ', mean_i_35_modified_x_cross >= std_i_35_mo
print('-------------------------------------------------------')
print("Max value in i_35_modified_x_cross = ", i_35_modified_x_cross_max)
print("Min value in i_35_modified_x_cross = ", i_35_modified_x_cross_min)
print('Mode value in i_35_modified_x_cross = ', i_35_modified_x_cross_mode)
```

```
Number of nan values in i_35_modified_x_cross =  0
Number of 0 values in i_35_modified_x_cross =  1785
Number of values in i_35_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_35_modified_x_cross =  0.07689240706148348
STD of 1-D values in i_35_modified_x_cross =  0.09100410769041996
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_35_modified_x_cross =  0.2930186269207713
Min value in i_35_modified_x_cross =  0.0
Mode value in i_35_modified_x_cross =  0.0
```

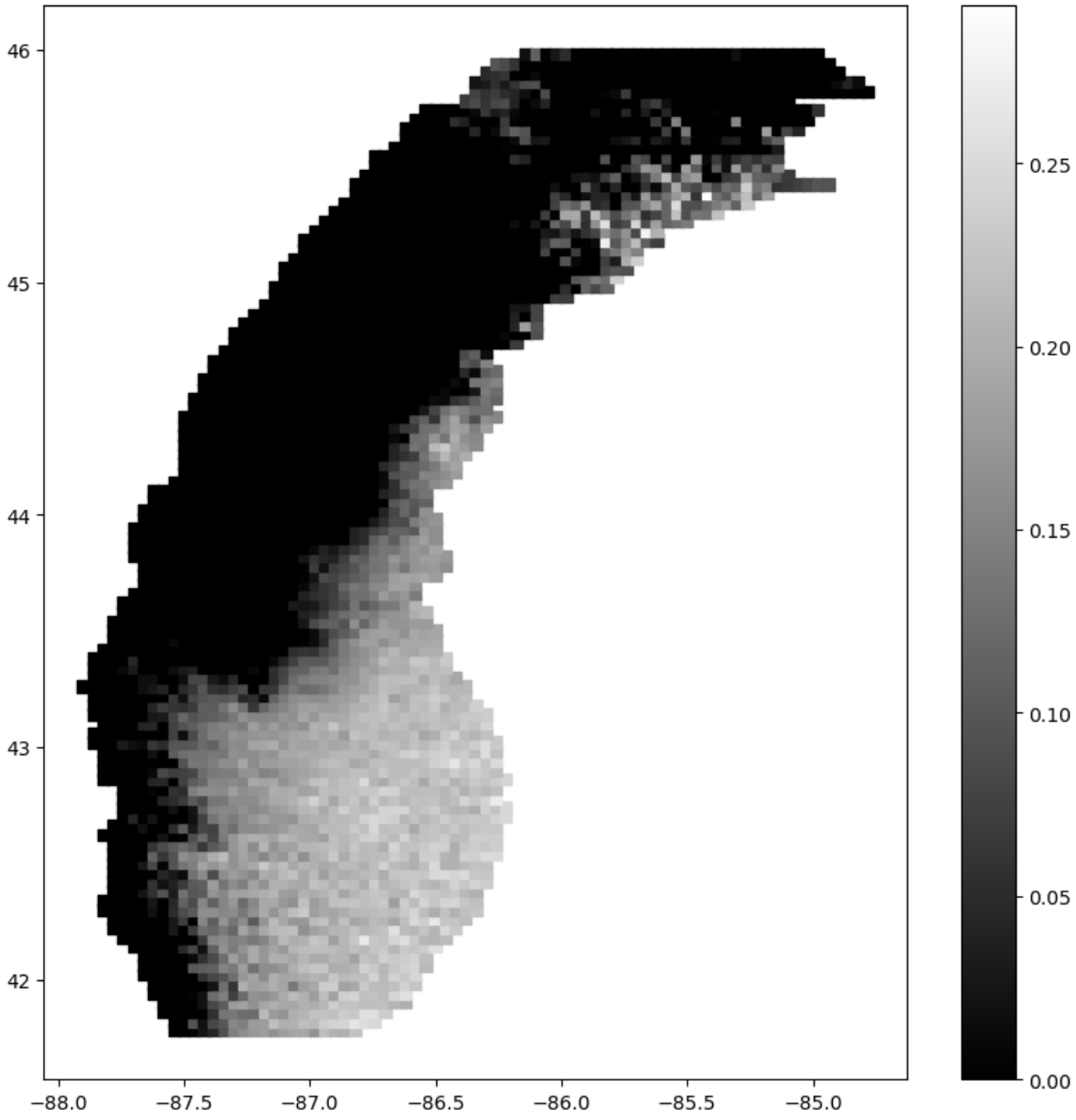
In [161…

```
fig = px.histogram(i_35_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [162…
```
# x_35, y_35, i_35 = df_35.longitude, df_35.latitude, df_35.value
plt.figure(figsize=(10, 10))
plt.scatter(df_35.longitude, y_35.values, c=i_35_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_35), len(y_35), len(i_35_modified_x_cross)
```

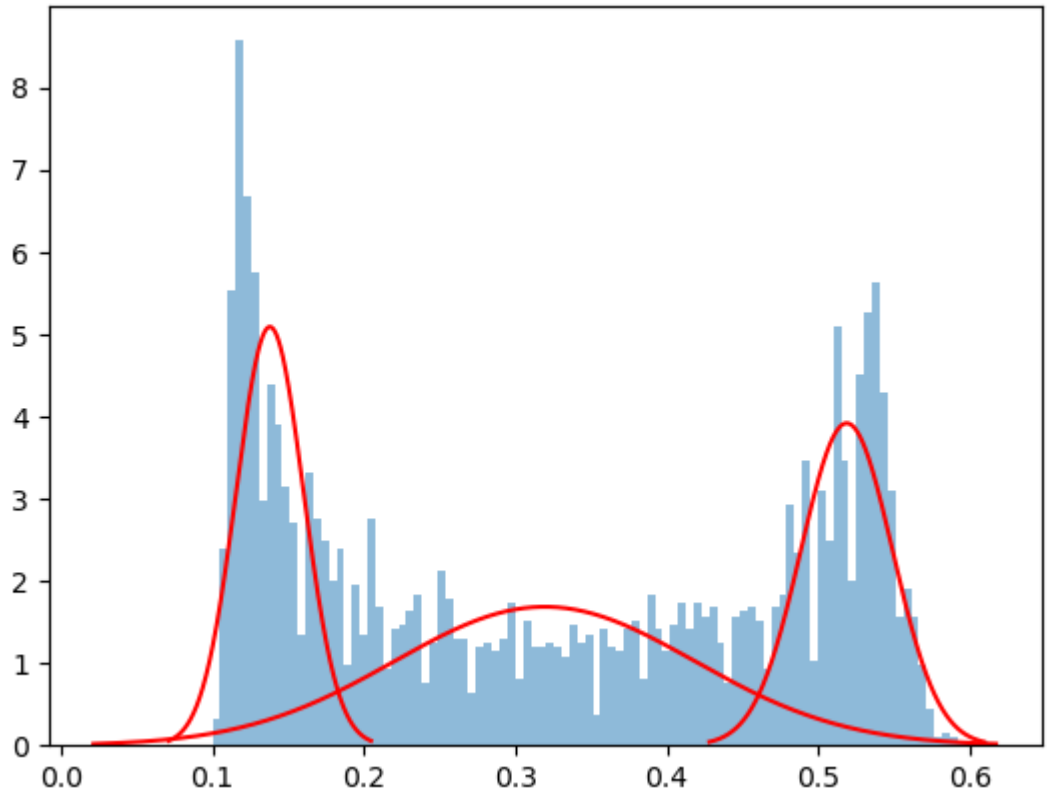
Out[162]: (3599, 3599, 3599)



In [163…

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Create a sample 1-D array
i_3N_35 = np.array(i_35).reshape(-1, 1)

# Set the number of components to 3
n_components = 3

# Define the Gaussian mixture model
gmm = GaussianMixture(n_components=n_components)

# Fit the model to the data
gmm.fit(i_3N_35.reshape(-1, 1))

# Get the mean and standard deviation for each component
s_.flatten()
```

```
stds = np.sqrt(gmm.covariances_.flatten())

# Plot the histogram and the fitted Gaussian curves
plt.hist(i_3N_35, bins=100, density=True, alpha=0.5)
for i in range(n_components):
    x = np.linspace(means[i] - 3 * stds[i], means[i] + 3 * stds[i], 3000)
    plt.plot(x, gmm.weights_[i] * np.exp(-0.5 * ((x - means[i]) / stds[i]) *
plt.show()
```



---

5. Case 5

`Duration` : 2016.12.13 1400-1900 UTC

image

[Q]:

> In this case, do we consider 1400 as the start of the cloud? The intensity was kinda low. And the interesting part of the case is that the SLAP (short-lake axis parallel) cloud is hidden under a large piece of cloud, where it has higher intensity value and altitude/height. And at 1700, the cloud still exists, but the overall coverage is under 50%.

[A]:

> That's a good point. We may have really intense clouds near Chicago, and none far away. So we need a better way to compute threshold images: Instead of considering the entire area of lake michigan and avergaing all pixels over lake michigan, break down the lake into 4 horizontal slices, and compute overall pixel intensity in each slice. If any slice is above threshold, consider that clouds.

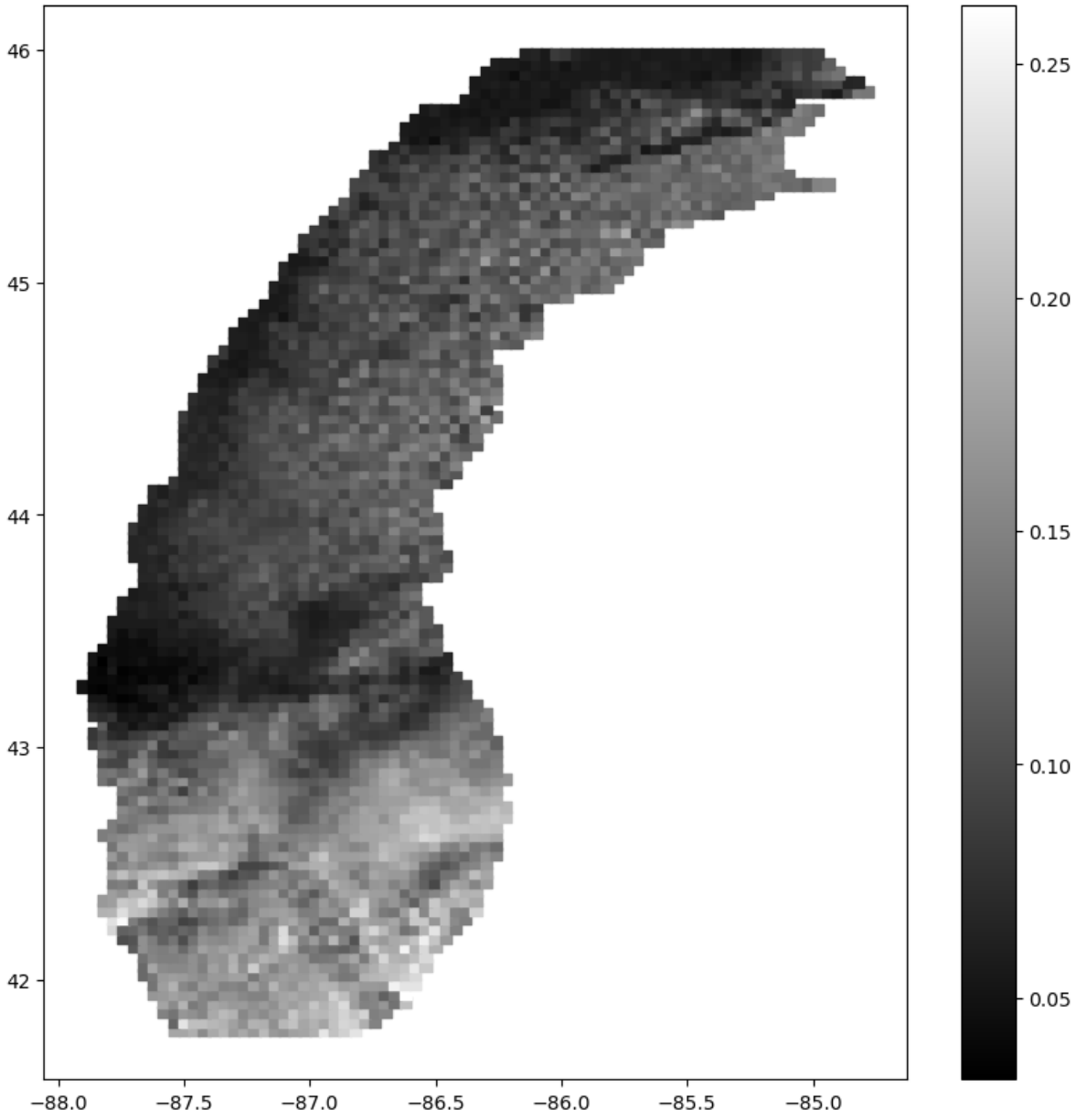
5.1 2016.12.13.1400

In [164…

```
df_36 = pd.read_csv(zone_0_folder_path + file_list[36])

print(file_list[36])

x_36, y_36, i_36 = df_36.longitude, df_36.latitude, df_36.value
plt.figure(figsize=(10, 10))
plt.scatter(x_36.values, y_36.values, c=i_36.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_36), len(y_36), len(i_36)
```

```
goes15.2016.12.13.1400.v01.nc-var1-t0.csv
```

Out[164]: (3599, 3599, 3599)



In [165…

```
# Calculate the mean and standard deviation of the 1-D array

mean_i_36 = np.nanmean(i_36.values)
std_i_36 = np.nanstd(i_36.values)

# Check number of nan values and 0s in i, just in case
num_nan_36 = i_36.isna().sum().sum()
num_zeros_36 = (i_36 == 0).sum().sum()

# Check min and max in i

i_36_max = i_36.max().max()
i_36_min = i_36.min().min()
i_36_mode = i_36.mode()[0]

print('Number of nan values in i_36 = ', num_nan_36)
print('Number of 0 values in i_36 = ', num_zeros_36)
print('values in i_36 = ', len(i_36))
```

```
print('-----------------------------------------------------')
print('Mean of values in i_36 = ', mean_i_36)
print('STD of 1-D values in i_36 = ', std_i_36)
print('Mean value >= STD =====> ', mean_i_36 >= std_i_36)
print('-----------------------------------------------------')
print("Max value in i_36 = ", i_36_max)
print("Min value in i_36 = ", i_36_min)
print('Mode value in i_36 = ', i_36_mode)
```

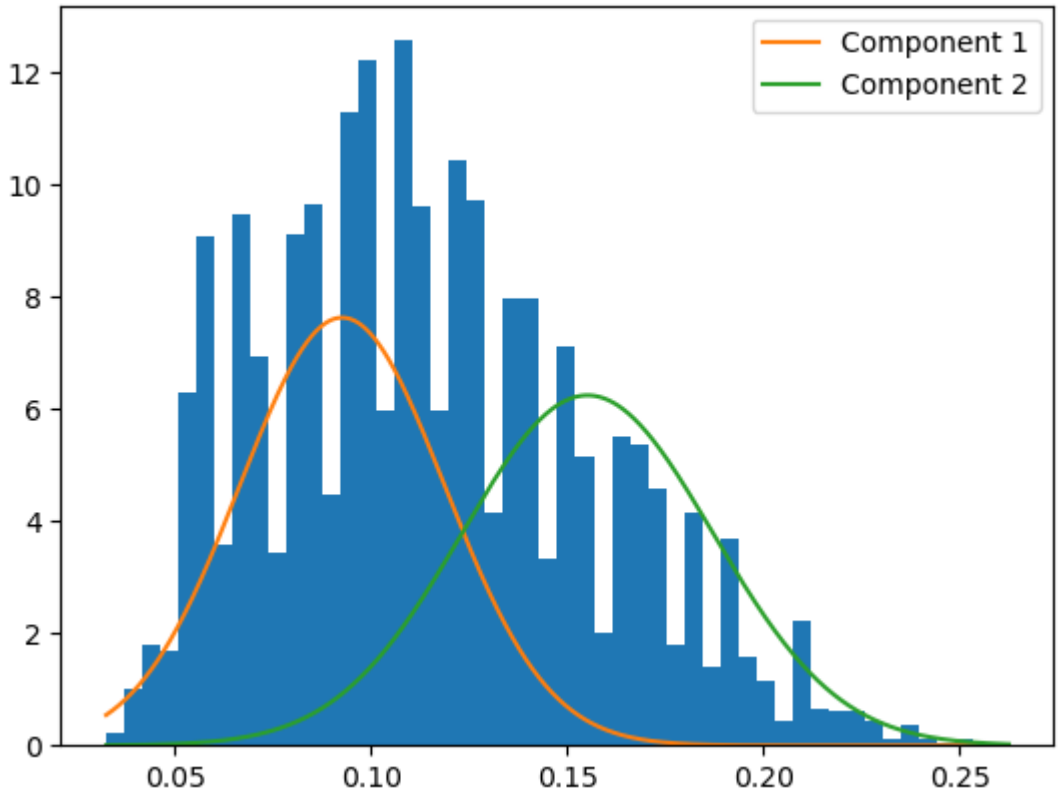
```
Number of nan values in i_36 =  0
Number of 0 values in i_36 =  0
Number of values in i_36 =  3599
-----------------------------------------------------
Mean of values in i_36 =  0.1152931351097527
STD of 1-D values in i_36 =  0.0413362288311556
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_36 =  0.2625
Min value in i_36 =  0.0325
Mode value in i_36 =  0.107499994
```

In [166…

```
fig = px.histogram(i_36.values.flatten(), title="Histogram of i_36")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```

In [167…

```
# print(i_36)
x_36 = np.array(i_36).reshape(-1, 1)
# print(x_36)
# Fit a mixture of two Gaussians to the data
gmm_36 = GaussianMixture(n_components=2).fit(x_36)

# Get the means and standard deviations of the two Gaussian components
mu1_36, mu2_36 = gmm_36.means_.flatten()
sigma1_36, sigma2_36 = np.sqrt(gmm_36.covariances_.flatten())

# Visualize the results
plt.hist(i_36, density=True, bins = 50)
# x_axis = i_36.unique()
x_axis_36 = np.linspace(min(i_36), max(i_36), 3000)
plt.plot(x_axis_36, 0.5 * np.exp(-(x_axis_36 - mu1_36)**2 / (2 * sigma1_36**
plt.plot(x_axis_36, 0.5 * np.exp(-(x_axis_36 - mu2_36)**2 / (2 * sigma2_36**
plt.legend()
plt.show()
```



In [168…
```
print('The mean of the 1st normal distribution = ', mu1_36)
print('The mean of the 2nd normal distribution = ', mu2_36)
```

```
The mean of the 1st normal distribution =  0.09262566584941169
The mean of the 2nd normal distribution =  0.15514761354036213
```

In [169…
```
print('The std of the 1st normal distribution = ', sigma1_36)
print('The std of the 2nd normal distribution = ', sigma2_36)
```

```
The std of the 1st normal distribution =  0.026152971433952437
The std of the 2nd normal distribution =  0.03195816754323018
```

In [170…
```
# Define the two normal distributions
mean1, std1 = mu1_36, sigma1_36
mean2, std2 = mu2_36, sigma2_36

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_36 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_36[0])
```

```
Cross point of two normal distributions: x = 0.12342077776329215
```

```
uce by x_cross
```

```
# If we were to reduce the values by x_cross value as mentioned above
i_36_modified_x_cross = i_36.apply(lambda x: max(0, x - x_cross_36[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_36_modified_x_cross = np.nanmean(i_36_modified_x_cross.values)
std_i_36_modified_x_cross = np.nanstd(i_36_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_36_modified_x_cross = i_36_modified_x_cross.isna().sum().sum()
num_zeros_36_modified_x_cross = (i_36_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_36_modified_x_cross_max = i_36_modified_x_cross.max().max()
i_36_modified_x_cross_min = i_36_modified_x_cross.min().min()
# i_36_modified_x_cross_min = 0
i_36_modified_x_cross_mode = i_36_modified_x_cross.mode()[0]

print('Number of nan values in i_36_modified_x_cross = ', num_nan_36_modifie
print('Number of 0 values in i_36_modified_x_cross = ', num_zeros_36_modifie
print('Number of values in i_36_modified_x_cross = ', len(i_36_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_36_modified_x_cross = ', mean_i_36_modified_x_cro
print('STD of 1-D values in i_36_modified_x_cross = ', std_i_36_modified_x_c
print('Mean value >= STD =====> ', mean_i_36_modified_x_cross >= std_i_36_mo
print('-------------------------------------------------------')
print("Max value in i_36_modified_x_cross = ", i_36_modified_x_cross_max)
print("Min value in i_36_modified_x_cross = ", i_36_modified_x_cross_min)
print('Mode value in i_36_modified_x_cross = ', i_36_modified_x_cross_mode)
```

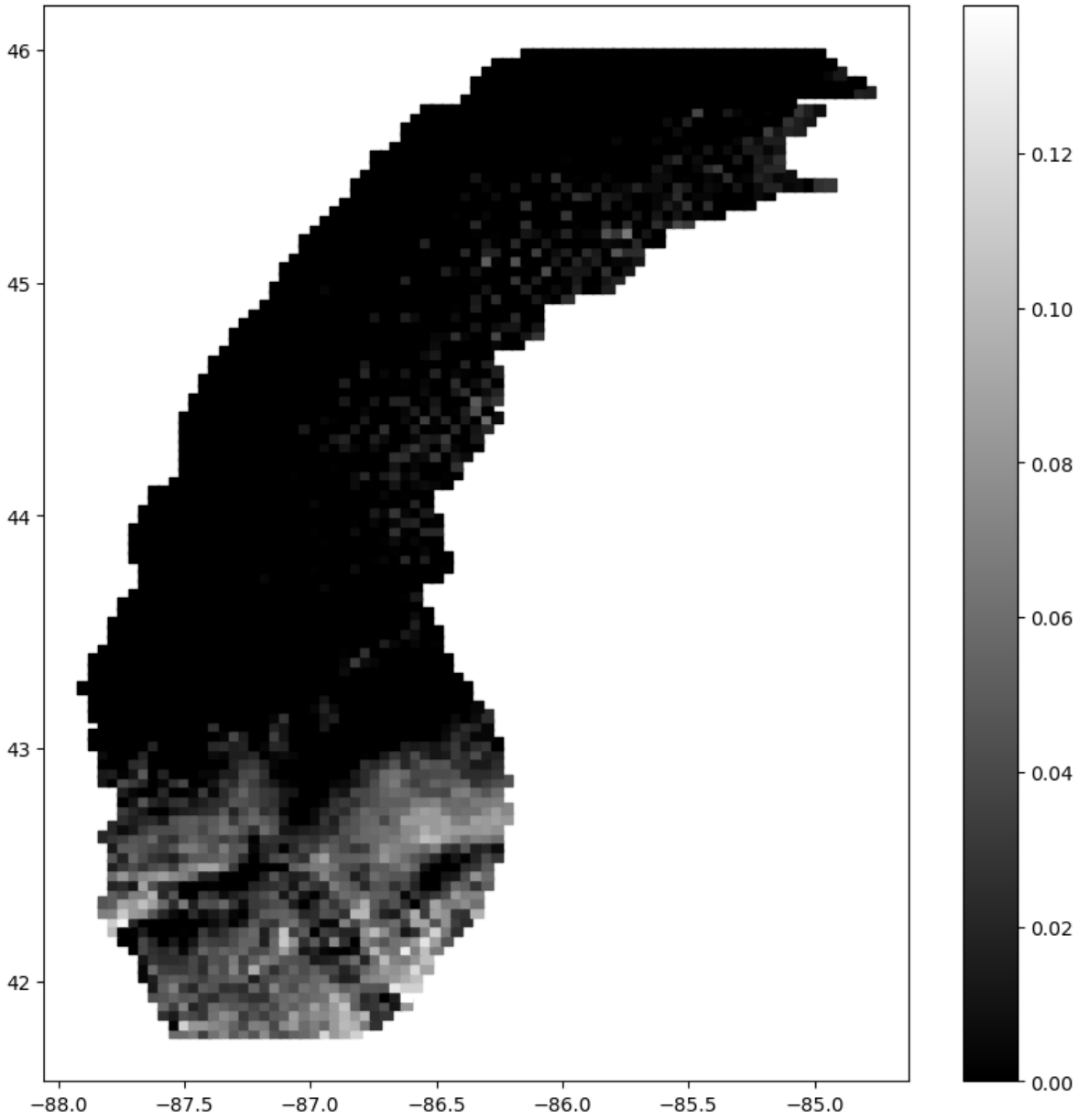
```
Number of nan values in i_36_modified_x_cross =  0
Number of 0 values in i_36_modified_x_cross =  2233
Number of values in i_36_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_36_modified_x_cross =  0.013356408509403426
STD of 1-D values in i_36_modified_x_cross =  0.023504364220541824
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_36_modified_x_cross =  0.13907922223670788
Min value in i_36_modified_x_cross =  0.0
Mode value in i_36_modified_x_cross =  0.0
```

In [172…

```
fig = px.histogram(i_36_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```

In [173…
```
plt.figure(figsize=(10, 10))
plt.scatter(df_36.longitude, y_36.values, c=i_36_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_36), len(y_36), len(i_36_modified_x_cross)
```

Out[173]: (3599, 3599, 3599)



As shown above, it does eliminate the upper cloud/another type of cloud in this case.

5.2 2016.12.13.1500

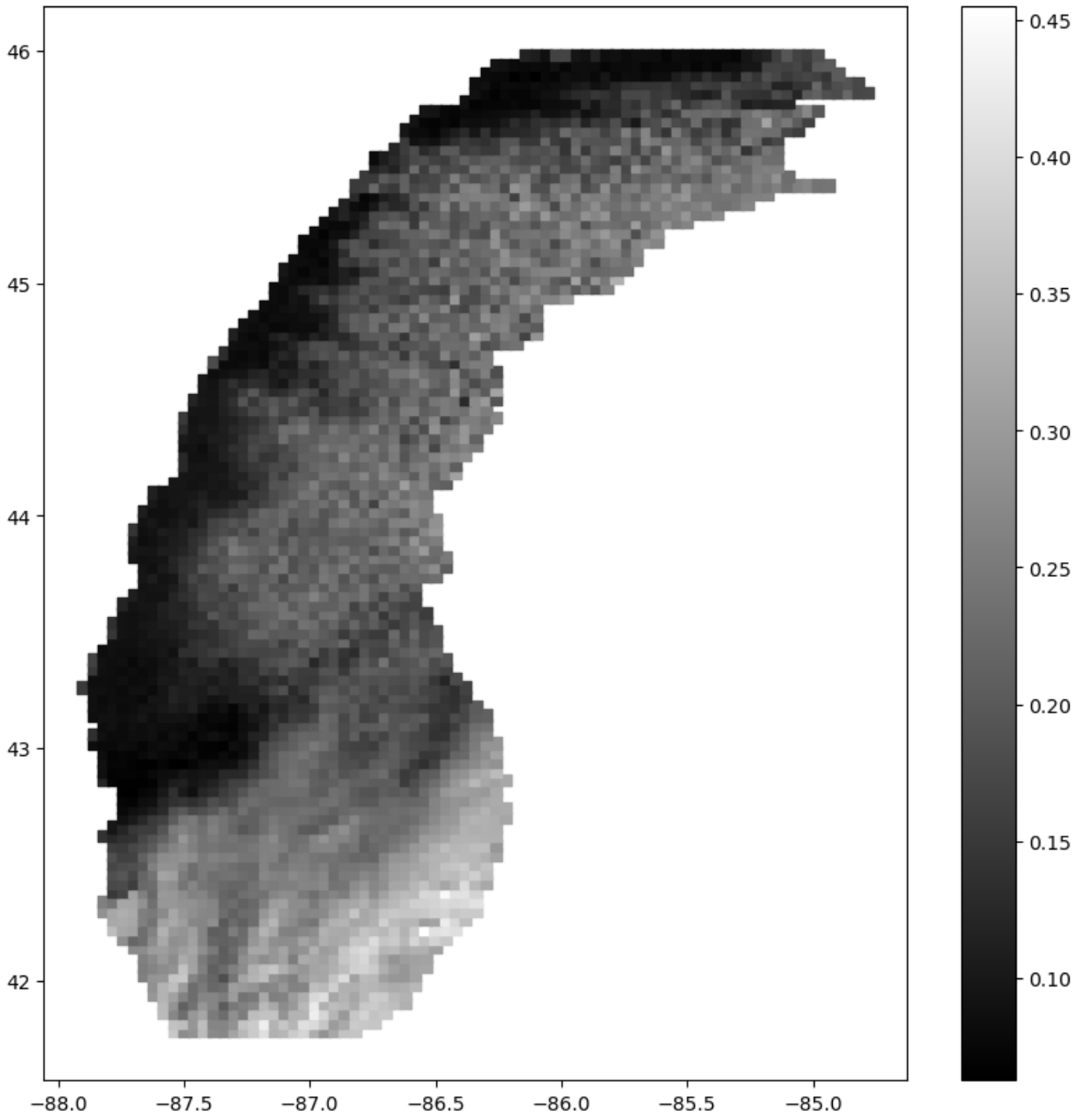
image

In [174…

```
df_40 = pd.read_csv(zone_0_folder_path + file_list[40])

print(file_list[40])


x_40, y_40, i_40 = df_40.longitude, df_40.latitude, df_40.value
plt.figure(figsize=(10, 10))
plt.scatter(x_40.values, y_40.values, c=i_40.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
_40), len(i_40)
```

```
goes15.2016.12.13.1500.v01.nc-var1-t0.csv
```

Out[174]: (3599, 3599, 3599)



In [175…

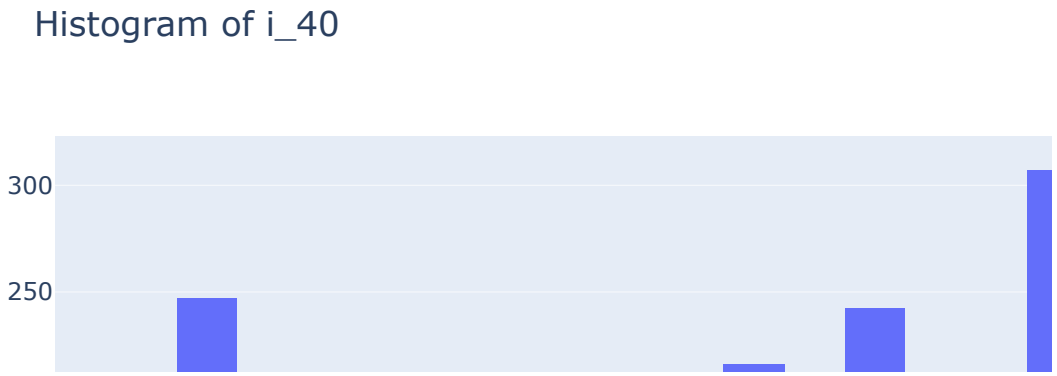
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_40 = np.nanmean(i_40.values)
std_i_40 = np.nanstd(i_40.values)

# Check number of nan values and 0s in i, just in case
num_nan_40 = i_40.isna().sum().sum()
num_zeros_40 = (i_40 == 0).sum().sum()

# Check min and max in i

i_40_max = i_40.max().max()
i_40_min = i_40.min().min()
i_40_mode = i_40.mode()[0]
print('Number of nan values in i_40 = ', num_nan_40)
```

```
print('Number of 0 values in i_40 = ', num_zeros_40)
print('Number of values in i_40 = ', len(i_40))
print('-----------------------------------------------------')
print('Mean of values in i_40 = ', mean_i_40)
print('STD of 1-D values in i_40 = ', std_i_40)
print('Mean value >= STD =====> ', mean_i_40 >= std_i_40)
print('-----------------------------------------------------')
print("Max value in i_40 = ", i_40_max)
print("Min value in i_40 = ", i_40_min)
print('Mode value in i_40 = ', i_40_mode)
```

```
Number of nan values in i_40 =  0
Number of 0 values in i_40 =  0
Number of values in i_40 =  3599
-----------------------------------------------------
Mean of values in i_40 =  0.20060572086551823
STD of 1-D values in i_40 =  0.07618477712199703
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_40 =  0.45499998
Min value in i_40 =  0.0625
Mode value in i_40 =  0.22999999
```

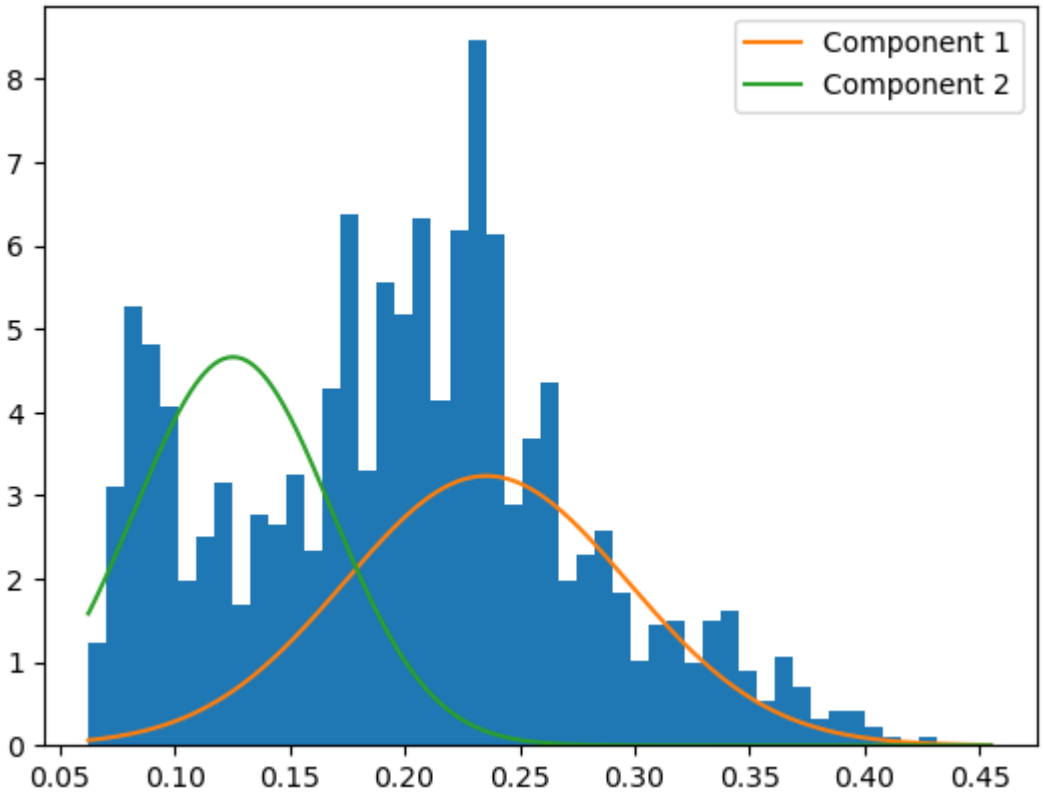
In [176…

```
fig = px.histogram(i_40.values.flatten(), title="Histogram of i_40")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



In [177…

```
# print(i_40)
x_40 = np.array(i_40).reshape(-1, 1)
# print(x_40)
# Fit a mixture of two Gaussians to the data
gmm_40 = GaussianMixture(n_components=2).fit(x_40)

# Get the means and standard deviations of the two Gaussian components
mu1_40, mu2_40 = gmm_40.means_.flatten()
sigma1_40, sigma2_40 = np.sqrt(gmm_40.covariances_.flatten())

# Visualize the results
plt.hist(i_40, density=True, bins = 50)
# x_axis = i_40.unique()
x_axis_40 = np.linspace(min(i_40), max(i_40), 3000)
plt.plot(x_axis_40, 0.5 * np.exp(-(x_axis_40 - mu1_40)**2 / (2 * sigma1_40**
plt.plot(x_axis_40, 0.5 * np.exp(-(x_axis_40 - mu2_40)**2 / (2 * sigma2_40**
plt.legend()
plt.show()
```



In [178…
```
print('The mean of the 1st normal distribution = ', mu1_40)
print('The mean of the 2nd normal distribution = ', mu2_40)
```

```
The mean of the 1st normal distribution =  0.2355651929051037
The mean of the 2nd normal distribution =  0.12538955067206162
```

In [179…
```
print('The std of the 1st normal distribution = ', sigma1_40)
print('The std of the 2nd normal distribution = ', sigma2_40)
```

```
The std of the 1st normal distribution =  0.061649202008141675
The std of the 2nd normal distribution =  0.04278847449381346
```

In [180…
```
# Define the two normal distributions
mean1, std1 = mu1_40, sigma1_40
mean2, std2 = mu2_40, sigma2_40

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_40 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_40[0])
```

```
Cross point of two normal distributions: x = 0.1790276983692482
```

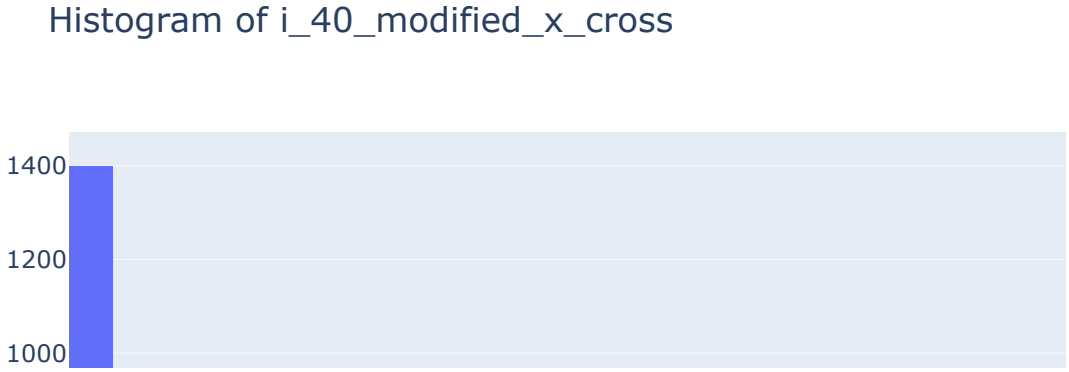
```
uce by x_cross
```

```
# If we were to reduce the values by x_cross value as mentioned above
i_40_modified_x_cross = i_40.apply(lambda x: max(0, x - x_cross_40[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_40_modified_x_cross = np.nanmean(i_40_modified_x_cross.values)
std_i_40_modified_x_cross = np.nanstd(i_40_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_40_modified_x_cross = i_40_modified_x_cross.isna().sum().sum()
num_zeros_40_modified_x_cross = (i_40_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_40_modified_x_cross_max = i_40_modified_x_cross.max().max()
i_40_modified_x_cross_min = i_40_modified_x_cross.min().min()
# i_40_modified_x_cross_min = 0
i_40_modified_x_cross_mode = i_40_modified_x_cross.mode()[0]

print('Number of nan values in i_40_modified_x_cross = ', num_nan_40_modifie
print('Number of 0 values in i_40_modified_x_cross = ', num_zeros_40_modifie
print('Number of values in i_40_modified_x_cross = ', len(i_40_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_40_modified_x_cross = ', mean_i_40_modified_x_cro
print('STD of 1-D values in i_40_modified_x_cross = ', std_i_40_modified_x_c
print('Mean value >= STD =====> ', mean_i_40_modified_x_cross >= std_i_40_mo
print('-------------------------------------------------------')
print("Max value in i_40_modified_x_cross = ", i_40_modified_x_cross_max)
print("Min value in i_40_modified_x_cross = ", i_40_modified_x_cross_min)
print('Mode value in i_40_modified_x_cross = ', i_40_modified_x_cross_mode)
```

```
Number of nan values in i_40_modified_x_cross =  0
Number of 0 values in i_40_modified_x_cross =  1362
Number of values in i_40_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_40_modified_x_cross =  0.04291609627896409
STD of 1-D values in i_40_modified_x_cross =  0.05254965805238171
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_40_modified_x_cross =  0.2759722816307518
Min value in i_40_modified_x_cross =  0.0
Mode value in i_40_modified_x_cross =  0.0
```

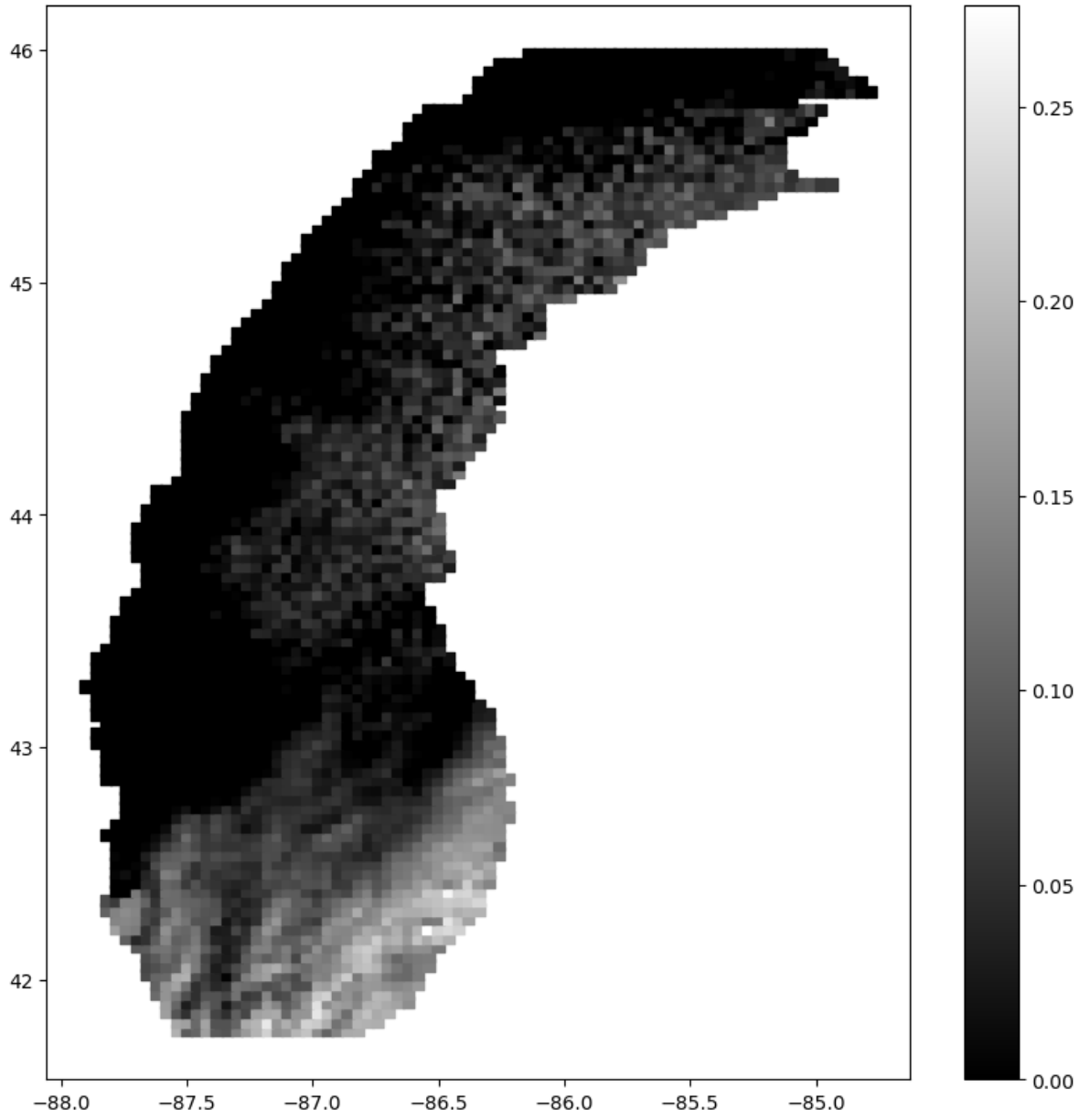
In [182…

```
fig = px.histogram(i_40_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [183…
```
# x_40, y_40, i_40 = df_40.longitude, df_40.latitude, df_40.value
plt.figure(figsize=(10, 10))
plt.scatter(df_40.longitude, y_40.values, c=i_40_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_40), len(y_40), len(i_40_modified_x_cross)
```

Out[183]: (3599, 3599, 3599)



Same effect as shown above.

5.3 2016.12.13.1600 Different now!

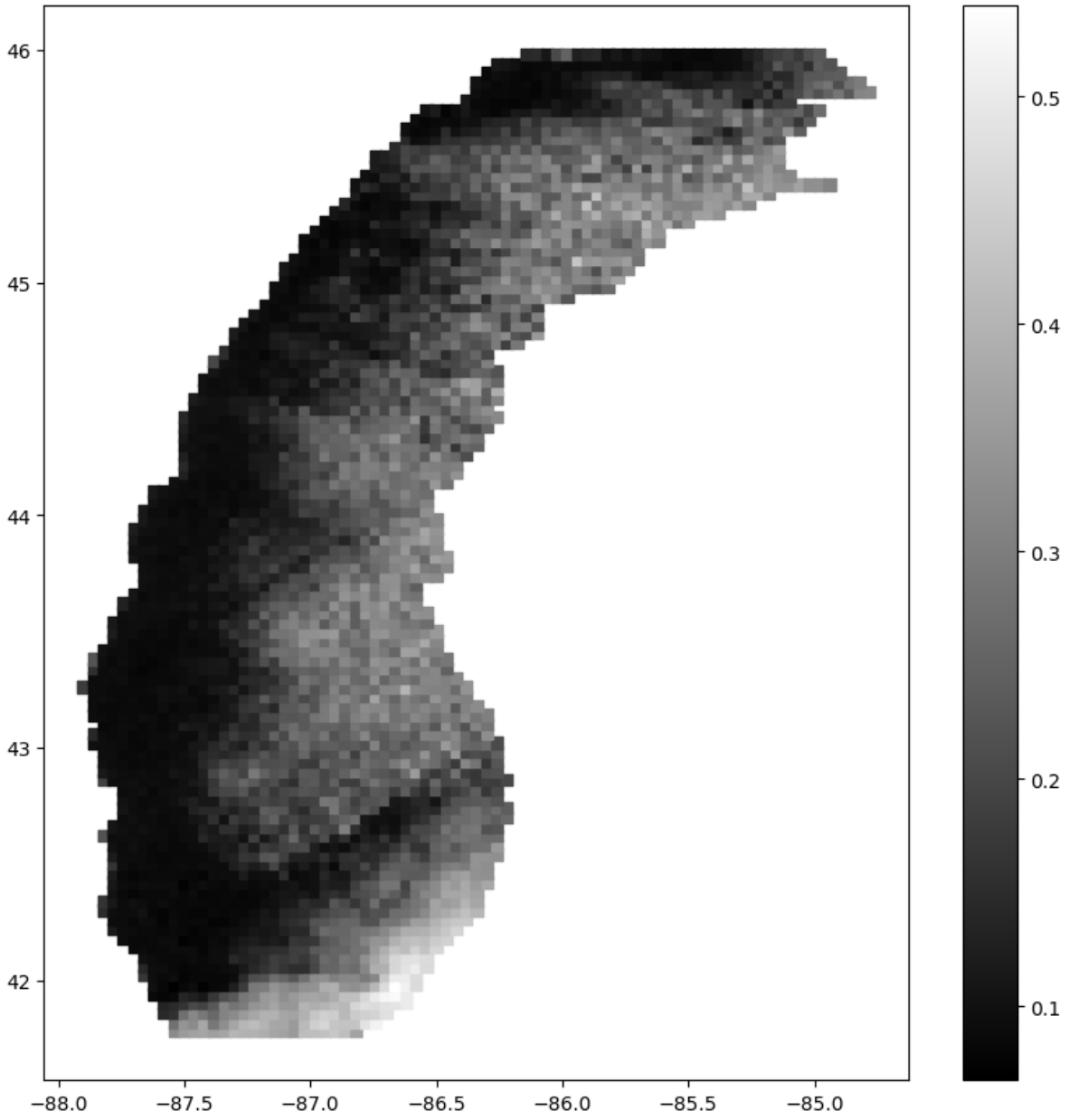
image

In [184…

```
df_43 = pd.read_csv(zone_0_folder_path + file_list[43])

print(file_list[43])


x_43, y_43, i_43 = df_43.longitude, df_43.latitude, df_43.value
plt.figure(figsize=(10, 10))
plt.scatter(x_43.values, y_43.values, c=i_43.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
_43), len(i_43)
```

goes15.2016.12.13.1600.v01.nc-var1-t0.csv

Out[184]: (3599, 3599, 3599)



In [185…

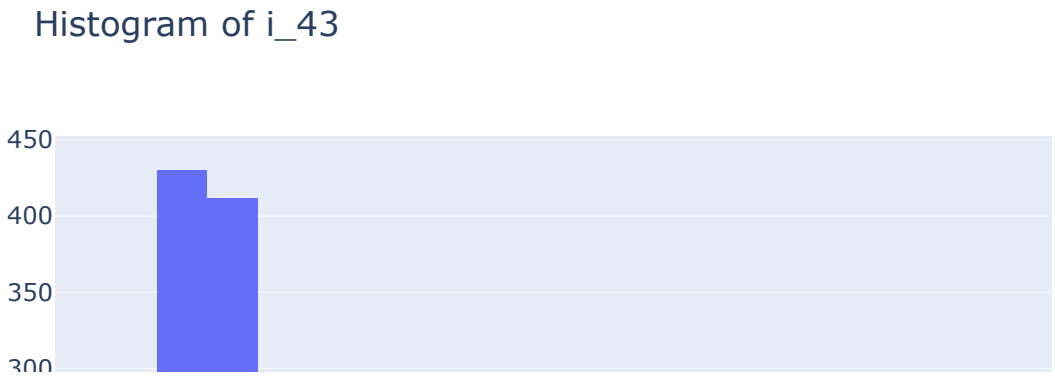
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_43 = np.nanmean(i_43.values)
std_i_43 = np.nanstd(i_43.values)

# Check number of nan values and 0s in i, just in case
num_nan_43 = i_43.isna().sum().sum()
num_zeros_43 = (i_43 == 0).sum().sum()

# Check min and max in i

i_43_max = i_43.max().max()
i_43_min = i_43.min().min()
i_43_mode = i_43.mode()[0]
```

```
print('Number of nan values in i_43 = ', num_nan_43)
print('Number of 0 values in i_43 = ', num_zeros_43)
print('Number of values in i_43 = ', len(i_43))
print('-----------------------------------------------------')
print('Mean of values in i_43 = ', mean_i_43)
print('STD of 1-D values in i_43 = ', std_i_43)
print('Mean value >= STD =====> ', mean_i_43 >= std_i_43)
print('-----------------------------------------------------')
print("Max value in i_43 = ", i_43_max)
print("Min value in i_43 = ", i_43_min)
print('Mode value in i_43 = ', i_43_mode)
```

```
Number of nan values in i_43 =  0
Number of 0 values in i_43 =  0
Number of values in i_43 =  3599
-----------------------------------------------------
Mean of values in i_43 =  0.19755626260850237
STD of 1-D values in i_43 =  0.09487295191735971
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_43 =  0.53999996
Min value in i_43 =  0.067499995
Mode value in i_43 =  0.089999996
```

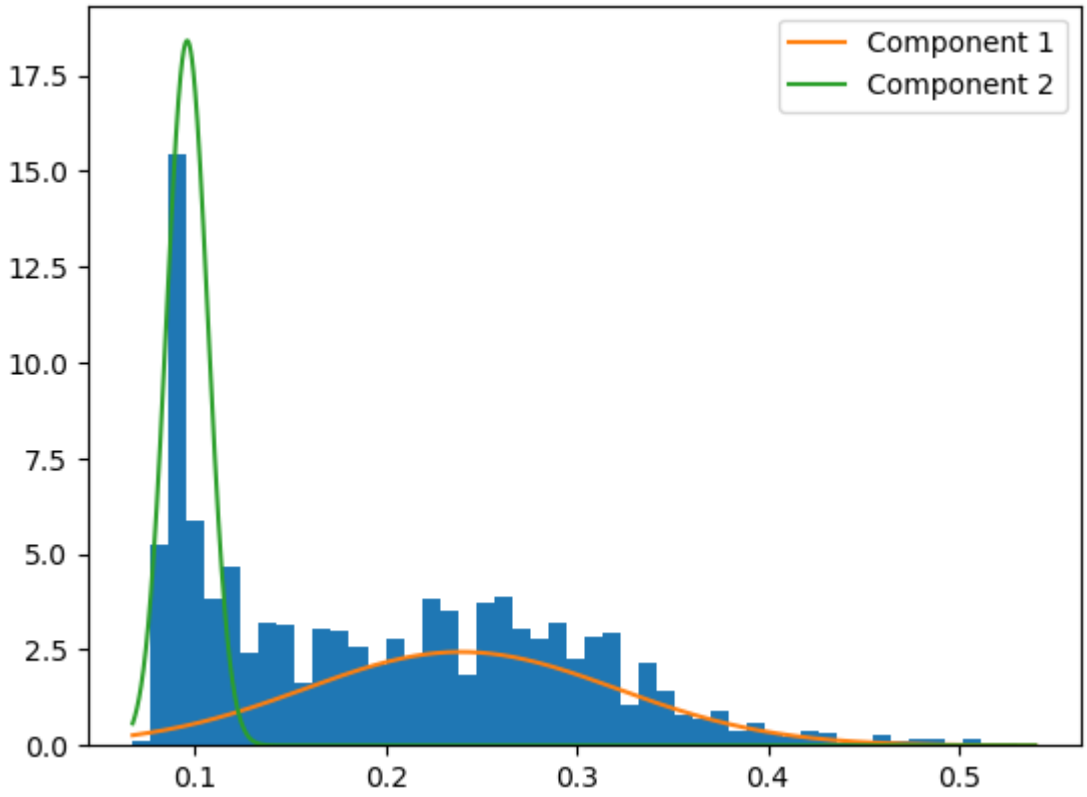
In [186…

```
fig = px.histogram(i_43.values.flatten(), title="Histogram of i_43")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



In [187…

```
# print(i_43)
x_43 = np.array(i_43).reshape(-1, 1)
# print(x_43)
# Fit a mixture of two Gaussians to the data
gmm_43 = GaussianMixture(n_components=2).fit(x_43)

# Get the means and standard deviations of the two Gaussian components
mu1_43, mu2_43 = gmm_43.means_.flatten()
sigma1_43, sigma2_43 = np.sqrt(gmm_43.covariances_.flatten())

# Visualize the results
plt.hist(i_43, density=True, bins = 50)
# x_axis = i_43.unique()
x_axis_43 = np.linspace(min(i_43), max(i_43), 3000)
plt.plot(x_axis_43, 0.5 * np.exp(-(x_axis_43 - mu1_43)**2 / (2 * sigma1_43**
plt.plot(x_axis_43, 0.5 * np.exp(-(x_axis_43 - mu2_43)**2 / (2 * sigma2_43**
plt.legend()
plt.show()
```



In [188…
```
print('The mean of the 1st normal distribution = ', mu1_43)
print('The mean of the 2nd normal distribution = ', mu2_43)
```

```
The mean of the 1st normal distribution =  0.23937315627912026
The mean of the 2nd normal distribution =  0.0960321172381082
```

In [189…
```
print('The std of the 1st normal distribution = ', sigma1_43)
print('The std of the 2nd normal distribution = ', sigma2_43)
```

```
The std of the 1st normal distribution =  0.0816529664325025
The std of the 2nd normal distribution =  0.010838516650394798
```

In [190…
```
# Define the two normal distributions
mean1, std1 = mu1_43, sigma1_43
mean2, std2 = mu2_43, sigma2_43

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_43 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_43[0])
```

```
Cross point of two normal distributions: x = 0.12275422653310832
```

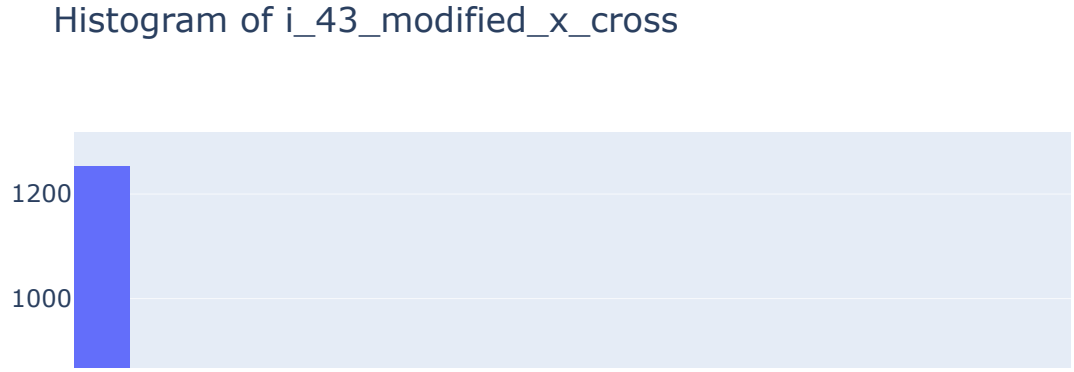
```
uce by x_cross
```

```
# If we were to reduce the values by x_cross value as mentioned above
i_43_modified_x_cross = i_43.apply(lambda x: max(0, x - x_cross_43[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_43_modified_x_cross = np.nanmean(i_43_modified_x_cross.values)
std_i_43_modified_x_cross = np.nanstd(i_43_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_43_modified_x_cross = i_43_modified_x_cross.isna().sum().sum()
num_zeros_43_modified_x_cross = (i_43_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_43_modified_x_cross_max = i_43_modified_x_cross.max().max()
i_43_modified_x_cross_min = i_43_modified_x_cross.min().min()
# i_43_modified_x_cross_min = 0
i_43_modified_x_cross_mode = i_43_modified_x_cross.mode()[0]

print('Number of nan values in i_43_modified_x_cross = ', num_nan_43_modifie
print('Number of 0 values in i_43_modified_x_cross = ', num_zeros_43_modifie
print('Number of values in i_43_modified_x_cross = ', len(i_43_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_43_modified_x_cross = ', mean_i_43_modified_x_cro
print('STD of 1-D values in i_43_modified_x_cross = ', std_i_43_modified_x_c
print('Mean value >= STD =====> ', mean_i_43_modified_x_cross >= std_i_43_mo
print('-------------------------------------------------------')
print("Max value in i_43_modified_x_cross = ", i_43_modified_x_cross_max)
print("Min value in i_43_modified_x_cross = ", i_43_modified_x_cross_min)
print('Mode value in i_43_modified_x_cross = ', i_43_modified_x_cross_mode)
```

```
Number of nan values in i_43_modified_x_cross =  0
Number of 0 values in i_43_modified_x_cross =  1196
Number of values in i_43_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_43_modified_x_cross =  0.08336387457931113
STD of 1-D values in i_43_modified_x_cross =  0.08592222797325504
Mean value >= STD =====>  False
-------------------------------------------------------
Max value in i_43_modified_x_cross =  0.41724573346689164
Min value in i_43_modified_x_cross =  0.0
Mode value in i_43_modified_x_cross =  0.0
```

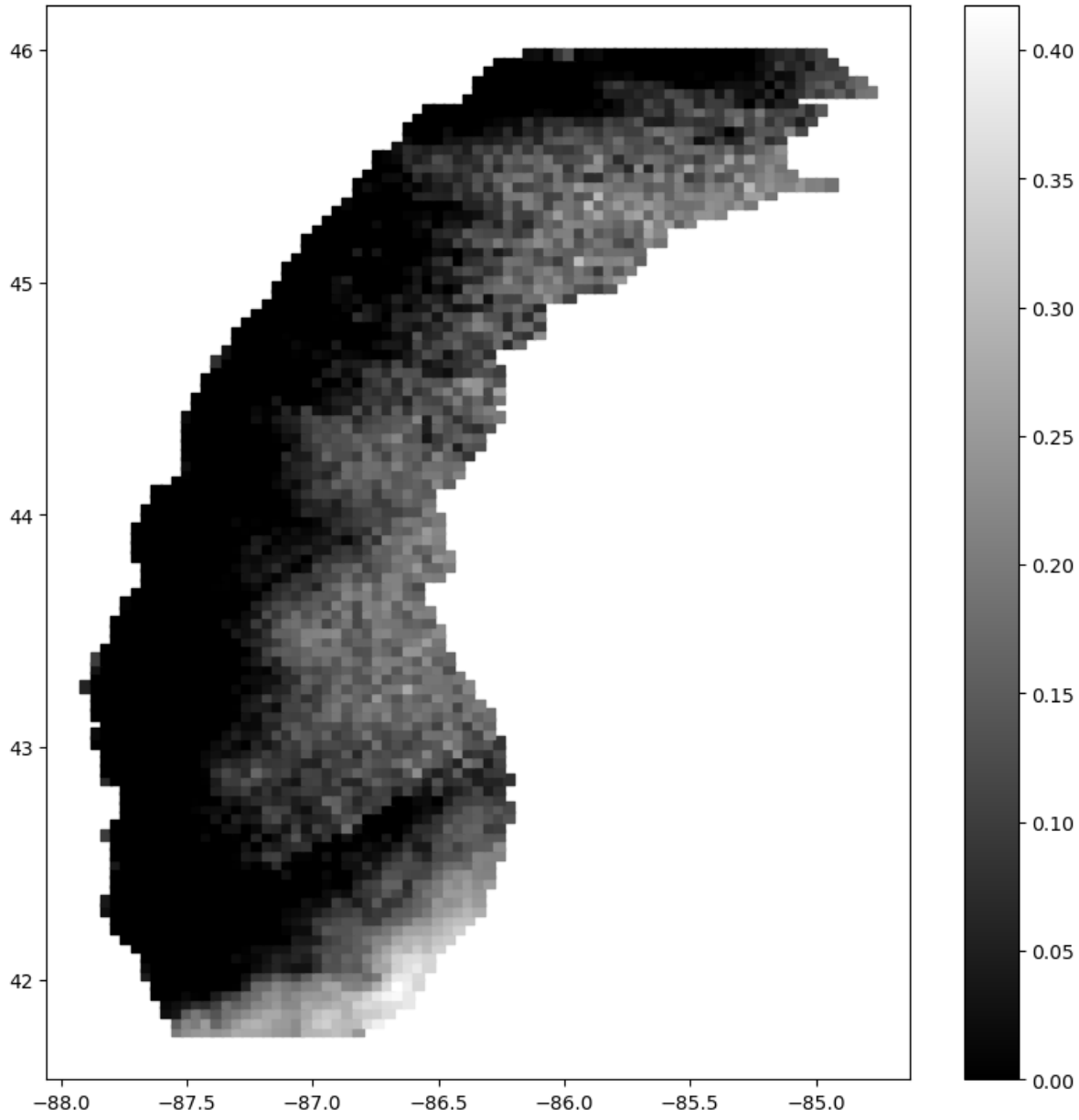
In [192…

```
fig = px.histogram(i_43_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```



In [193…
```
# x_43, y_43, i_43 = df_43.longitude, df_43.latitude, df_43.value
plt.figure(figsize=(10, 10))
plt.scatter(df_43.longitude, y_43.values, c=i_43_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_43), len(y_43), len(i_43_modified_x_cross)
```

Out[193]: (3599, 3599, 3599)



5.4 2016.12.13.1700 Different now!

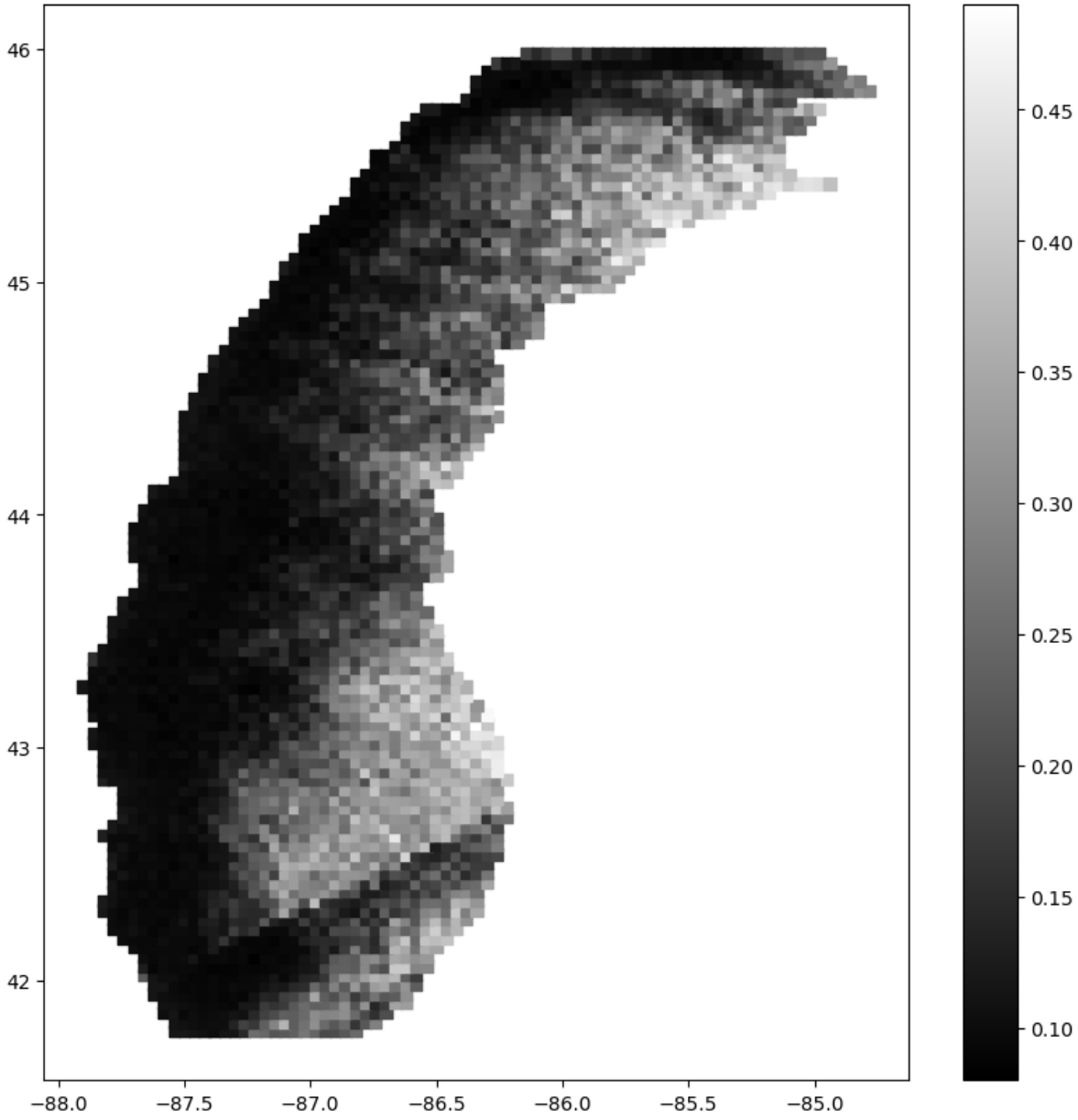
image

In [194…

```
df_47 = pd.read_csv(zone_0_folder_path + file_list[47])

print(file_list[47])


x_47, y_47, i_47 = df_47.longitude, df_47.latitude, df_47.value
plt.figure(figsize=(10, 10))
plt.scatter(x_47.values, y_47.values, c=i_47.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_47), len(y_47), len(i_47)
```


goes15.2016.12.13.1700.v01.nc-var1-t0.csv

Out[194]: (3599, 3599, 3599)



In [195…

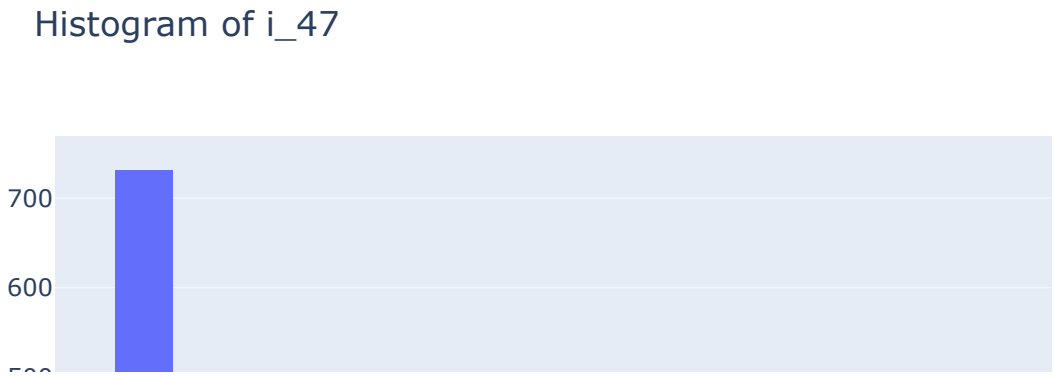
```
# Calculate the mean and standard deviation of the 1-D array

mean_i_47 = np.nanmean(i_47.values)
std_i_47 = np.nanstd(i_47.values)

# Check number of nan values and 0s in i, just in case
num_nan_47 = i_47.isna().sum().sum()
num_zeros_47 = (i_47 == 0).sum().sum()

# Check min and max in i

i_47_max = i_47.max().max()
i_47_min = i_47.min().min()
i_47_mode = i_47.mode()[0]

print('Number of nan values in i_47 = ', num_nan_47)
print('Number of 0 values in i_47 = ', num_zeros_47)
```

```
print('Number of values in i_47 = ', len(i_47))
print('-------------------------------------------------------')
print('Mean of values in i_47 = ', mean_i_47)
print('STD of 1-D values in i_47 = ', std_i_47)
print('Mean value >= STD =====> ', mean_i_47 >= std_i_47)
print('-------------------------------------------------------')
print("Max value in i_47 = ", i_47_max)
print("Min value in i_47 = ", i_47_min)
print('Mode value in i_47 = ', i_47_mode)
```

```
Number of nan values in i_47 =  0
Number of 0 values in i_47 =  0
Number of values in i_47 =  3599
-------------------------------------------------------
Mean of values in i_47 =  0.18886079183884413
STD of 1-D values in i_47 =  0.09692560185913805
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_47 =  0.48999998
Min value in i_47 =  0.08
Mode value in i_47 =  0.0975
```

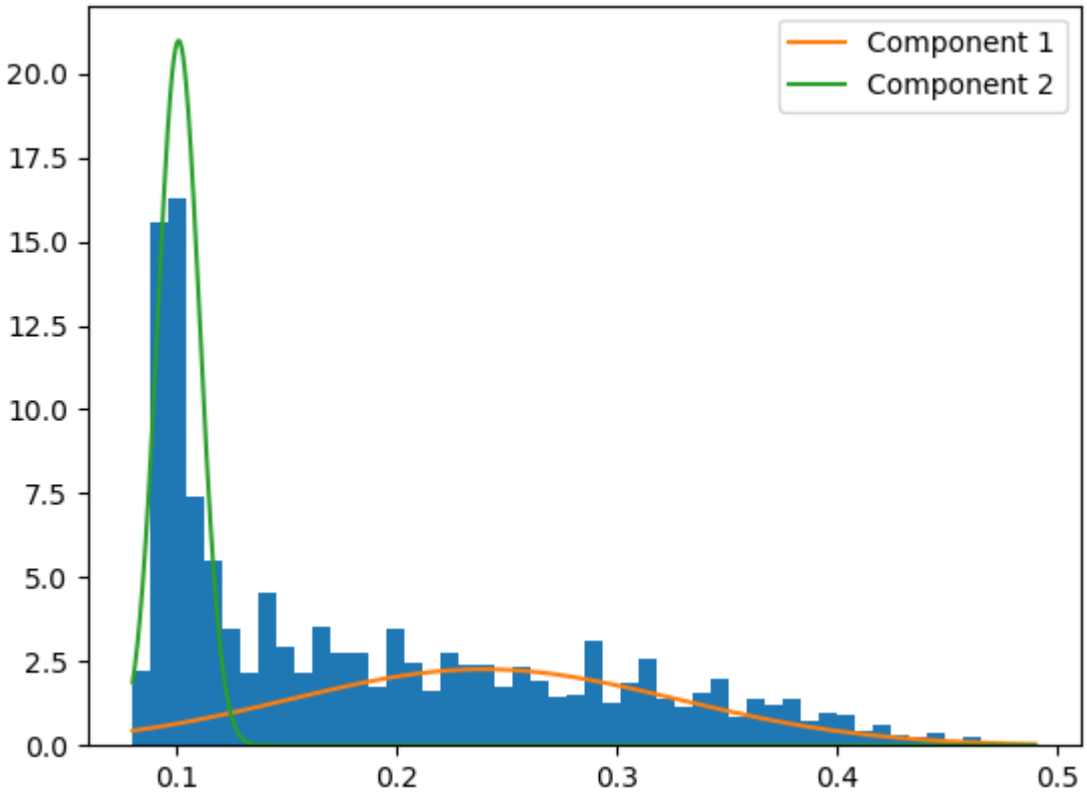
In [196…

```
fig = px.histogram(i_47.values.flatten(), title="Histogram of i_47")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```



In [197…

```
x_47 = np.array(i_47).reshape(-1, 1)
# print(x_47)
# Fit a mixture of two Gaussians to the data
gmm_47 = GaussianMixture(n_components=2).fit(x_47)

# Get the means and standard deviations of the two Gaussian components
mu1_47, mu2_47 = gmm_47.means_.flatten()
sigma1_47, sigma2_47 = np.sqrt(gmm_47.covariances_.flatten())

# Visualize the results
plt.hist(i_47, density=True, bins = 50)
# x_axis = i_47.unique()
x_axis_47 = np.linspace(min(i_47), max(i_47), 3000)
plt.plot(x_axis_47, 0.5 * np.exp(-(x_axis_47 - mu1_47)**2 / (2 * sigma1_47**
plt.plot(x_axis_47, 0.5 * np.exp(-(x_axis_47 - mu2_47)**2 / (2 * sigma2_47**
plt.legend()
plt.show()
```



In [198…
```
print('The mean of the 1st normal distribution = ', mu1_47)
print('The mean of the 2nd normal distribution = ', mu2_47)
```

```
The mean of the 1st normal distribution =  0.23945043894449275
The mean of the 2nd normal distribution =  0.10089440428514834
```

In [199…
```
print('The std of the 1st normal distribution = ', sigma1_47)
print('The std of the 2nd normal distribution = ', sigma2_47)
```

```
The std of the 1st normal distribution =  0.08796291204889661
The std of the 2nd normal distribution =  0.009509800403775736
```

In [200…
```
# Define the two normal distributions
mean1, std1 = mu1_47, sigma1_47
mean2, std2 = mu2_47, sigma2_47

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_47 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_47[0])
```

```
Cross point of two normal distributions: x = -1.8361746151970342
```

```
/home/jupyter-doggo/.local/lib/python3.9/site-packages/scipy/optimize/_minp
ack_py.py:178: RuntimeWarning:

The number of calls to function has reached maxfev = 400.
```

In [201…

```python
# What if we reduce by x_cross

# If we were to reduce the values by x_cross value as mentioned above
i_47_modified_x_cross = i_47.apply(lambda x: max(0, x - x_cross_47[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_47_modified_x_cross = np.nanmean(i_47_modified_x_cross.values)
std_i_47_modified_x_cross = np.nanstd(i_47_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_47_modified_x_cross = i_47_modified_x_cross.isna().sum().sum()
num_zeros_47_modified_x_cross = (i_47_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_47_modified_x_cross_max = i_47_modified_x_cross.max().max()
i_47_modified_x_cross_min = i_47_modified_x_cross.min().min()
# i_47_modified_x_cross_min = 0
i_47_modified_x_cross_mode = i_47_modified_x_cross.mode()[0]

print('Number of nan values in i_47_modified_x_cross = ', num_nan_47_modifie
print('Number of 0 values in i_47_modified_x_cross = ', num_zeros_47_modifie
print('Number of values in i_47_modified_x_cross = ', len(i_47_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_47_modified_x_cross = ', mean_i_47_modified_x_cro
print('STD of 1-D values in i_47_modified_x_cross = ', std_i_47_modified_x_c
print('Mean value >= STD =====> ', mean_i_47_modified_x_cross >= std_i_47_mo
print('-------------------------------------------------------')
print("Max value in i_47_modified_x_cross = ", i_47_modified_x_cross_max)
print("Min value in i_47_modified_x_cross = ", i_47_modified_x_cross_min)
print('Mode value in i_47_modified_x_cross = ', i_47_modified_x_cross_mode)
```

```
Number of nan values in i_47_modified_x_cross =  0
Number of 0 values in i_47_modified_x_cross =  0
Number of values in i_47_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_47_modified_x_cross =  2.0250354070358783
STD of 1-D values in i_47_modified_x_cross =  0.09692560185913805
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_47_modified_x_cross =  2.3261745951970343
Min value in i_47_modified_x_cross =  1.9161746151970342
Mode value in i_47_modified_x_cross =  1.933674615197034
```

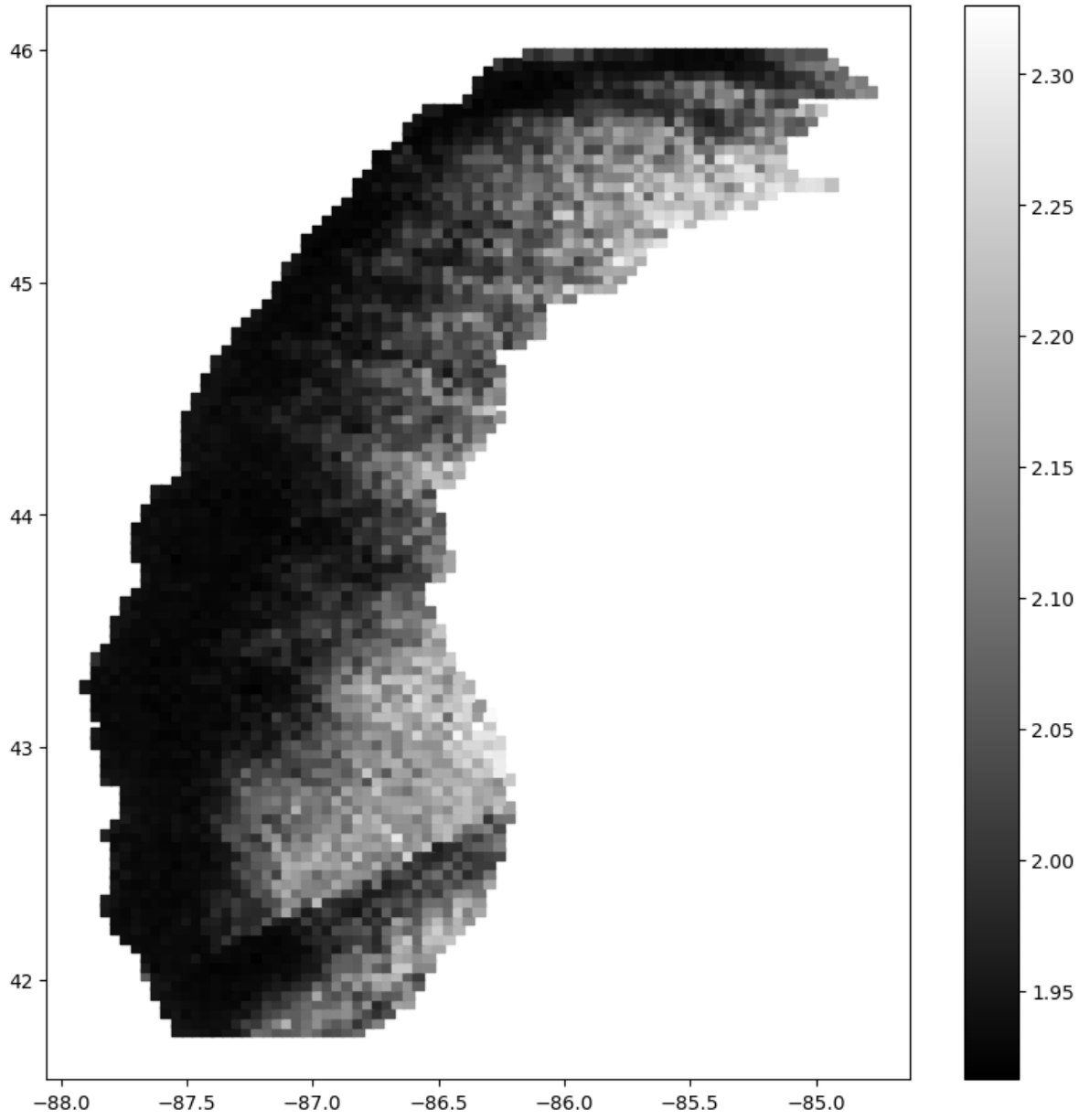
In [202…

```python
fig = px.histogram(i_47_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```

In [203…

```
# x_47, y_47, i_47 = df_47.longitude, df_47.latitude, df_47.value
plt.figure(figsize=(10, 10))
plt.scatter(df_47.longitude, y_47.values, c=i_47_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_47), len(y_47), len(i_47_modified_x_cross)
```

Out[203]: (3599, 3599, 3599)



---

6. Case 6

`Duration` : 2016.12.15 1700-1900 UTC

This is a perfect case of `SLAP` .

6.1 2016.12.15.1900

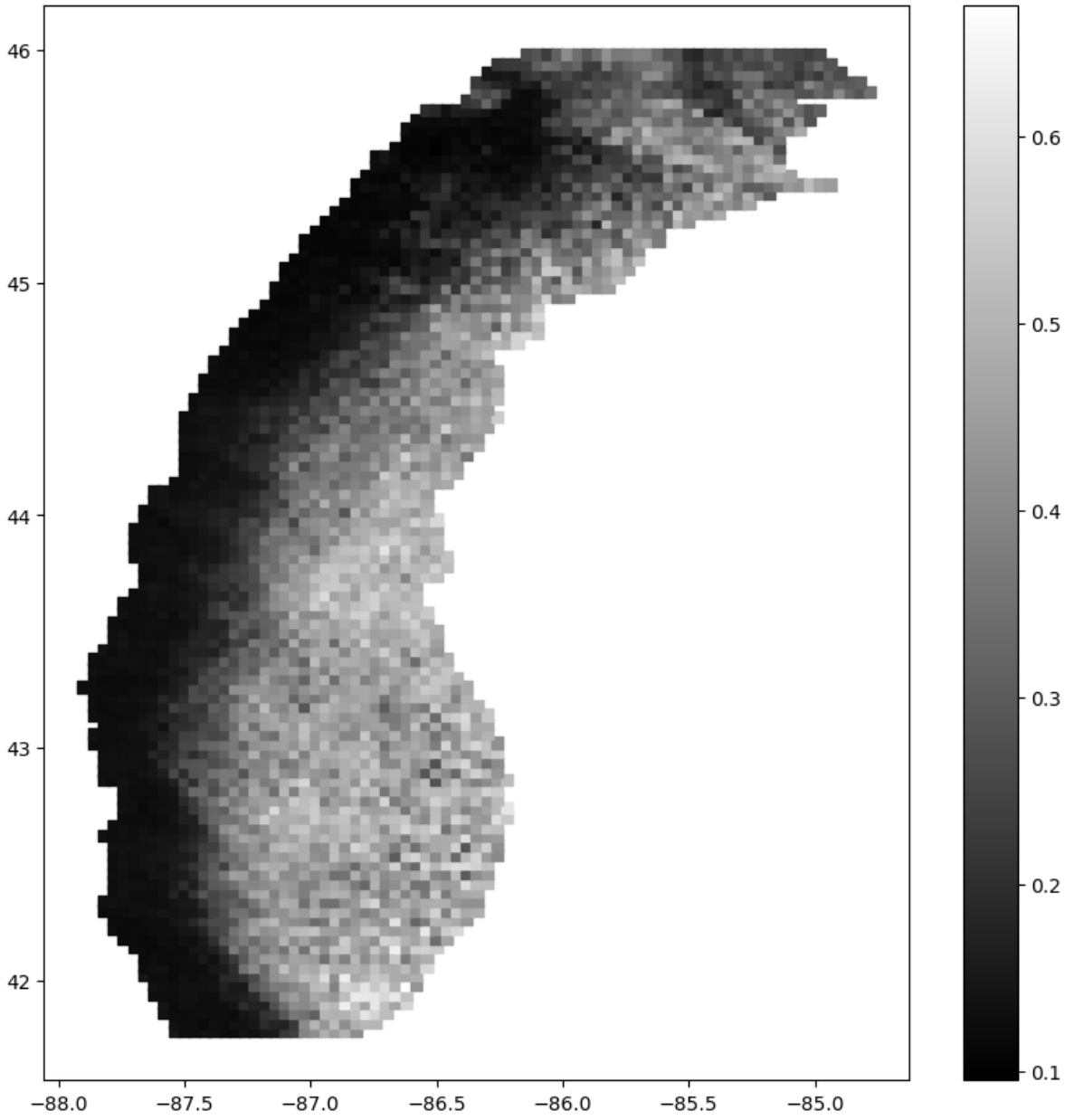
In [204…

```
df_50 = pd.read_csv(zone_0_folder_path + file_list[50])

print(file_list[50])
```

```
x_50, y_50, i_50 = df_50.longitude, df_50.latitude, df_50.value
plt.figure(figsize=(10, 10))
plt.scatter(x_50.values, y_50.values, c=i_50.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_50), len(y_50), len(i_50)
```

```
goes15.2016.12.15.1900.v01.nc-var1-t0.csv
```

Out[204]: (3599, 3599, 3599)



In [205…

```
# Calculate the mean and standard deviation of the 1-D array

mean_i_50 = np.nanmean(i_50.values)
std_i_50 = np.nanstd(i_50.values)

# Check number of nan values and 0s in i, just in case
num_nan_50 = i_50.isna().sum().sum()
num_zeros_50 = (i_50 == 0).sum().sum()

# Check min and max in i
```

```
i_50_max = i_50.max().max()
i_50_min = i_50.min().min()
i_50_mode = i_50.mode()[0]

print('Number of nan values in i_50 = ', num_nan_50)
print('Number of 0 values in i_50 = ', num_zeros_50)
print('Number of values in i_50 = ', len(i_50))
print('-------------------------------------------------------')
print('Mean of values in i_50 = ', mean_i_50)
print('STD of 1-D values in i_50 = ', std_i_50)
print('Mean value >= STD =====> ', mean_i_50 >= std_i_50)
print('-------------------------------------------------------')
print("Max value in i_50 = ", i_50_max)
print("Min value in i_50 = ", i_50_min)
print('Mode value in i_50 = ', i_50_mode)
```

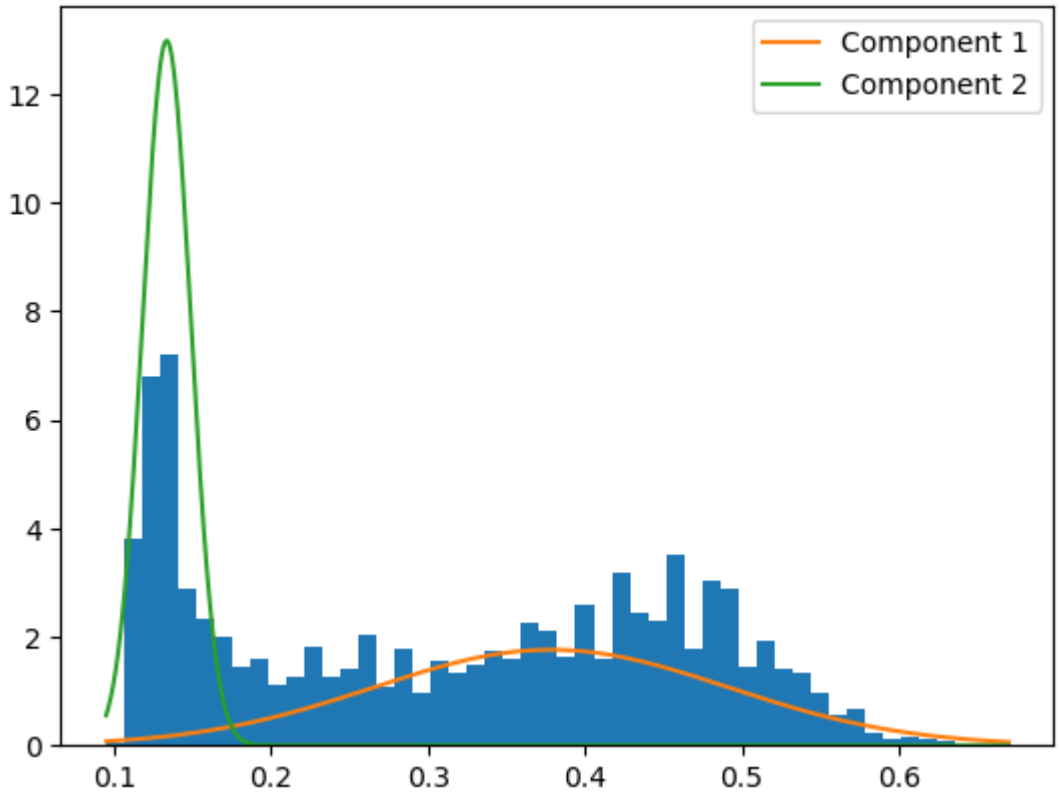
```
Number of nan values in i_50 =  0
Number of 0 values in i_50 =  0
Number of values in i_50 =  3599
-------------------------------------------------------
Mean of values in i_50 =  0.31333355981772715
STD of 1-D values in i_50 =  0.14542906704951197
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_50 =  0.66999996
Min value in i_50 =  0.095
Mode value in i_50 =  0.13499999
```

In [206…

```
fig = px.histogram(i_50.values.flatten(), title="Histogram of i_50")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```

In [207…

```
x_50 = np.array(i_50).reshape(-1, 1)
# print(x_50)
# Fit a mixture of two Gaussians to the data
gmm_50 = GaussianMixture(n_components=2).fit(x_50)

# Get the means and standard deviations of the two Gaussian components
mu1_50, mu2_50 = gmm_50.means_.flatten()
sigma1_50, sigma2_50 = np.sqrt(gmm_50.covariances_.flatten())

# Visualize the results
plt.hist(i_50, density=True, bins = 50)
# x_axis = i_50.unique()
x_axis_50 = np.linspace(min(i_50), max(i_50), 3000)
plt.plot(x_axis_50, 0.5 * np.exp(-(x_axis_50 - mu1_50)**2 / (2 * sigma1_50**
plt.plot(x_axis_50, 0.5 * np.exp(-(x_axis_50 - mu2_50)**2 / (2 * sigma2_50**
plt.legend()
plt.show()
```



In [208…
```
print('The mean of the 1st normal distribution = ', mu1_50)
print('The mean of the 2nd normal distribution = ', mu2_50)
```

```
The mean of the 1st normal distribution =  0.3784292286749542
The mean of the 2nd normal distribution =  0.13357064544269795
```

In [209…
```
print('The std of the 1st normal distribution = ', sigma1_50)
print('The std of the 2nd normal distribution = ', sigma2_50)
```

```
The std of the 1st normal distribution =  0.11307081045481804
The std of the 2nd normal distribution =  0.015358959433278753
```

In [210…
```
# Define the two normal distributions
mean1, std1 = mu1_50, sigma1_50
mean2, std2 = mu2_50, sigma2_50

dist1 = norm(mean1, std1)
dist2 = norm(mean2, std2)

# Define a function to find the difference between the two distributions
def diff(x):
    return dist1.pdf(x) - dist2.pdf(x)

# Use fsolve to find the x value where the difference is zero
x_cross_50 = fsolve(diff, x0=(mean1+mean2)/2)

print("Cross point of two normal distributions: x =", x_cross_50[0])
```

```
Cross point of two normal distributions: x = 0.17487845980647534
```

```
uce by x_cross
```

```
# If we were to reduce the values by x_cross value as mentioned above
i_50_modified_x_cross = i_50.apply(lambda x: max(0, x - x_cross_50[0]))

# Calculate the mean and standard deviation of the 1-D array

mean_i_50_modified_x_cross = np.nanmean(i_50_modified_x_cross.values)
std_i_50_modified_x_cross = np.nanstd(i_50_modified_x_cross.values)

# Check number of nan values and 0s in i, just in case
num_nan_50_modified_x_cross = i_50_modified_x_cross.isna().sum().sum()
num_zeros_50_modified_x_cross = (i_50_modified_x_cross == 0).sum().sum()

# Check min and max in i

i_50_modified_x_cross_max = i_50_modified_x_cross.max().max()
i_50_modified_x_cross_min = i_50_modified_x_cross.min().min()
# i_50_modified_x_cross_min = 0
i_50_modified_x_cross_mode = i_50_modified_x_cross.mode()[0]

print('Number of nan values in i_50_modified_x_cross = ', num_nan_50_modifie
print('Number of 0 values in i_50_modified_x_cross = ', num_zeros_50_modifie
print('Number of values in i_50_modified_x_cross = ', len(i_50_modified_x_cr
print('-------------------------------------------------------')
print('Mean of values in i_50_modified_x_cross = ', mean_i_50_modified_x_cro
print('STD of 1-D values in i_50_modified_x_cross = ', std_i_50_modified_x_c
print('Mean value >= STD =====> ', mean_i_50_modified_x_cross >= std_i_50_mo
print('-------------------------------------------------------')
print("Max value in i_50_modified_x_cross = ", i_50_modified_x_cross_max)
print("Min value in i_50_modified_x_cross = ", i_50_modified_x_cross_min)
print('Mode value in i_50_modified_x_cross = ', i_50_modified_x_cross_mode)
```

```
Number of nan values in i_50_modified_x_cross =  0
Number of 0 values in i_50_modified_x_cross =  1020
Number of values in i_50_modified_x_cross =  3599
-------------------------------------------------------
Mean of values in i_50_modified_x_cross =  0.1500127635618505
STD of 1-D values in i_50_modified_x_cross =  0.13143786069845906
Mean value >= STD =====>  True
-------------------------------------------------------
Max value in i_50_modified_x_cross =  0.49512150019352463
Min value in i_50_modified_x_cross =  0.0
Mode value in i_50_modified_x_cross =  0.0
```

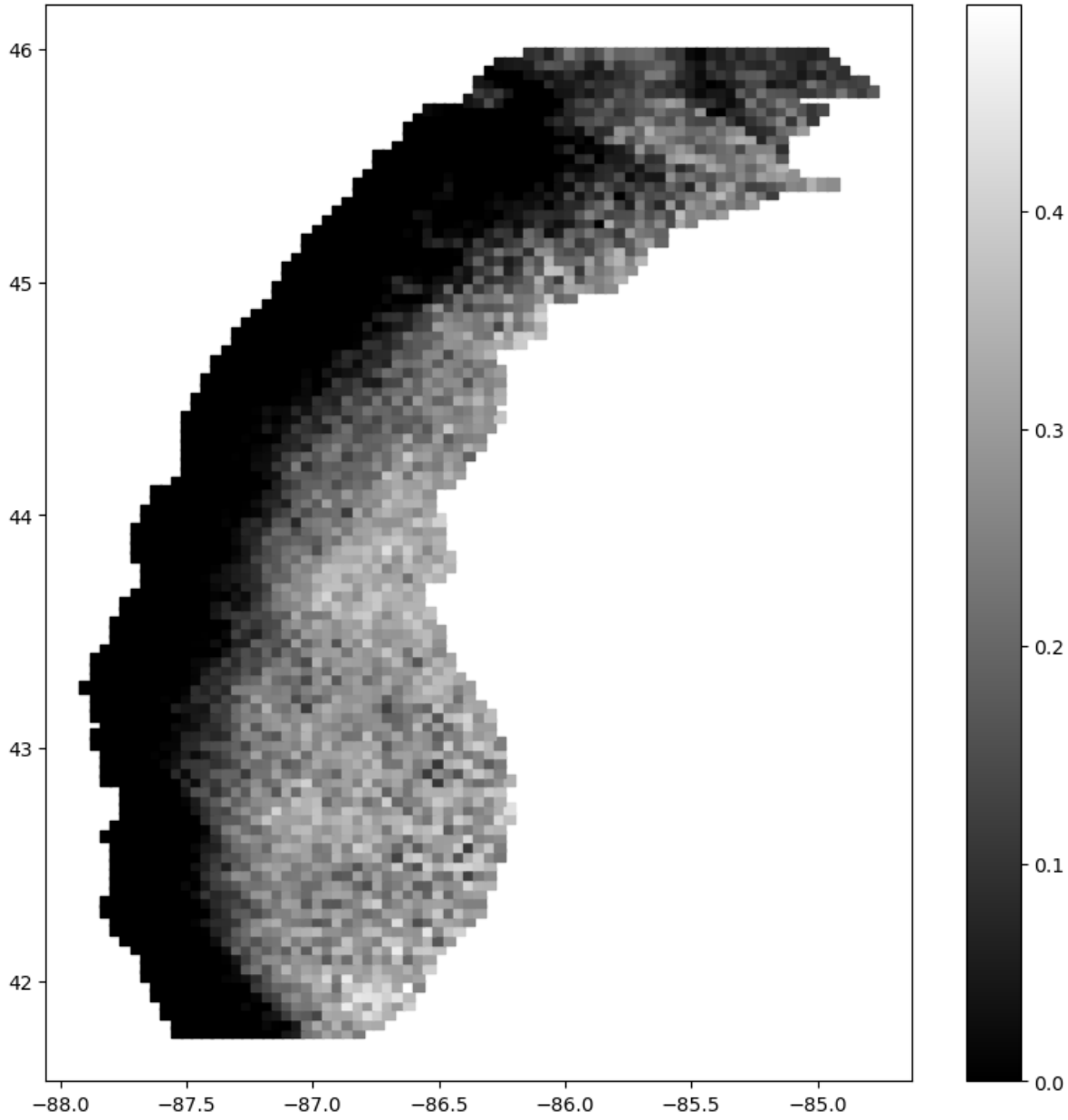
In [212…

```
fig = px.histogram(i_50_modified_x_cross.values.flatten(), title="Histogram
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")
```

In [213…

```
# x_50, y_50, i_50 = df_50.longitude, df_50.latitude, df_50.value
plt.figure(figsize=(10, 10))
plt.scatter(df_50.longitude, y_50.values, c=i_50_modified_x_cross.values, cm
plt.colorbar(orientation='vertical')
len(x_50), len(y_50), len(i_50_modified_x_cross)
```

Out[213]: (3599, 3599, 3599)



Since the cloud is rather uniformly formed, so it preserve the major part of the cloud rather nicely.

---

7. Case 7

`Duration` : 2016.12.19 1400-1700 UTC

7.1 2016.12.19.1400

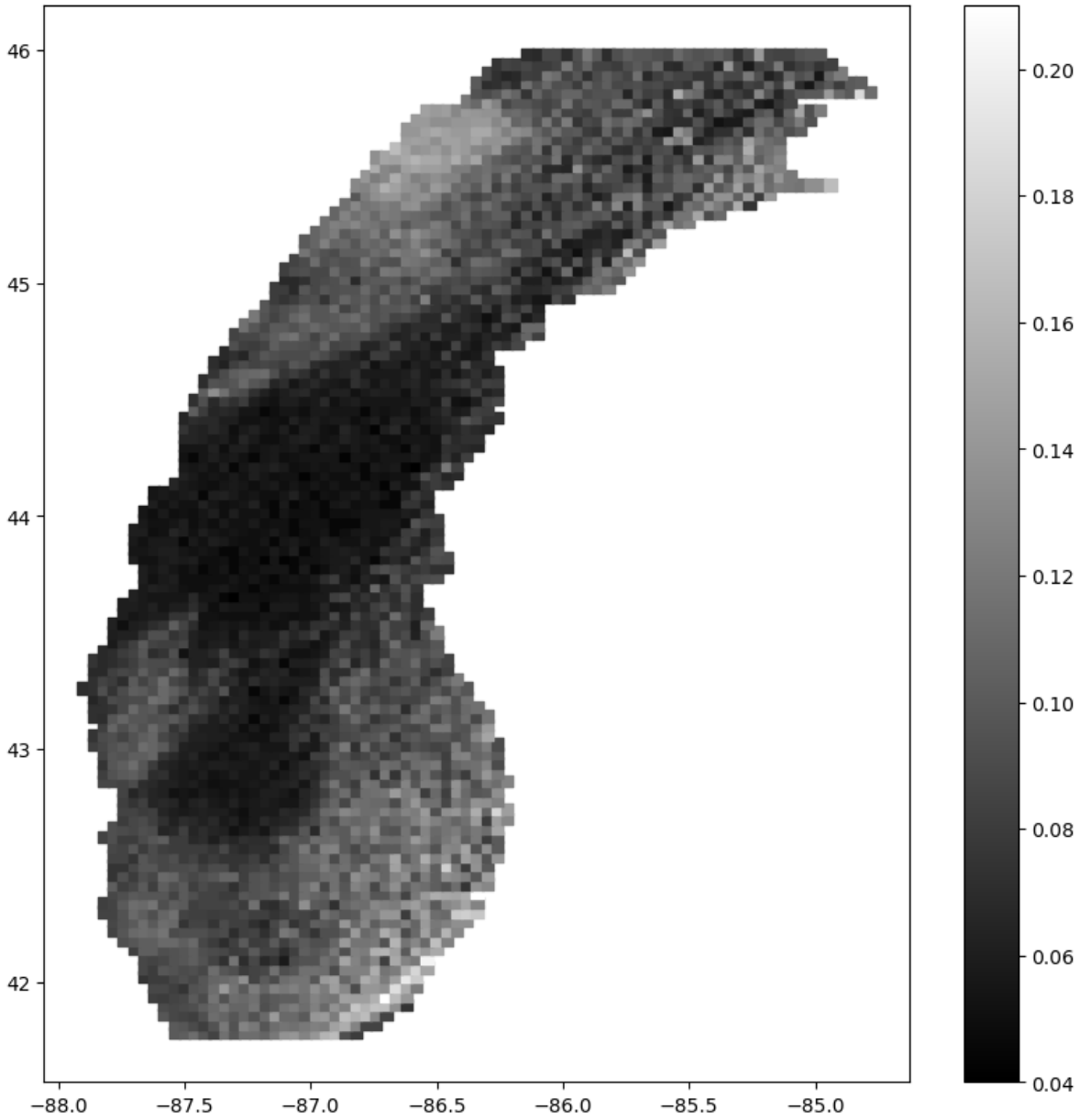
In [214…

```
df_51 = pd.read_csv(zone_0_folder_path + file_list[51])
```

```
print(file_list[51])


x_51, y_51, i_51 = df_51.longitude, df_51.latitude, df_51.value
plt.figure(figsize=(10, 10))
plt.scatter(x_51.values, y_51.values, c=i_51.values, cmap=cm.gray, marker='s
plt.colorbar(orientation='vertical')
len(x_51), len(y_51), len(i_51)
```

```
goes15.2016.12.19.1400.v01.nc-var1-t0.csv
```

Out[214]: (3599, 3599, 3599)



In [215…

```
# Calculate the mean and standard deviation of the 1-D array

mean_i_51 = np.nanmean(i_51.values)
std_i_51 = np.nanstd(i_51.values)

# Check number of nan values and 0s in i, just in case
num_nan_51 = i_51.isna().sum().sum()
```

```
num_zeros_51 = (i_51 == 0).sum().sum()

# Check min and max in i

i_51_max = i_51.max().max()
i_51_min = i_51.min().min()
i_51_mode = i_51.mode()[0]

print('Number of nan values in i_51 = ', num_nan_51)
print('Number of 0 values in i_51 = ', num_zeros_51)
print('Number of values in i_51 = ', len(i_51))
print('-----------------------------------------------------')
print('Mean of values in i_51 = ', mean_i_51)
print('STD of 1-D values in i_51 = ', std_i_51)
print('Mean value >= STD =====> ', mean_i_51 >= std_i_51)
print('-----------------------------------------------------')
print("Max value in i_51 = ", i_51_max)
print("Min value in i_51 = ", i_51_min)
print('Mode value in i_51 = ', i_51_mode)
```

```
Number of nan values in i_51 =  0
Number of 0 values in i_51 =  0
Number of values in i_51 =  3599
-----------------------------------------------------
Mean of values in i_51 =  0.08729855367963323
STD of 1-D values in i_51 =  0.0272107608229684
Mean value >= STD =====>  True
-----------------------------------------------------
Max value in i_51 =  0.21
Min value in i_51 =  0.04
Mode value in i_51 =  0.055
```

In [216…

```
fig = px.histogram(i_51.values.flatten(), title="Histogram of i_51")
fig.update_xaxes(title_text="Value")
fig.update_yaxes(title_text="Count")

# Show the plot
fig.show()
```

In [217…

```
# print(i_51)
x_51 = np.array(i_51).reshape(-1, 1)
# print(x_51)
# Fit a mixture of two Gaussians to the data
gmm_51 = GaussianMixture(n_components=2).fit(x_51)

# Get the means and standard deviations of the two Gaussian components
mu1_51, mu2_51 = gmm_51
```