p_q_frames

January 14, 2025

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import h5py
```

[2]:
```
import h5py
import pandas as pd

# Path to the HDF5 file
file_path = 'misc/FRT/Run05/misc_05.hdf5'

# The key you want to access
key_to_access = 'InIceDSTPulses'

# Open the HDF5 file and save data to a Pandas DataFrame
with h5py.File(file_path, 'r') as hdf:
    if key_to_access in hdf:
        # Access the data for the specific key
        data = hdf[key_to_access][:]

        # Convert the data into a Pandas DataFrame
        df = pd.DataFrame(data)

        print(f"Data for key '{key_to_access}' has been saved to a DataFrame.")
        print(df.head())  # Display the first few rows of the DataFrame
    else:
        print(f"Key '{key_to_access}' not found in the HDF5 file.")
```

```
Data for key 'InIceDSTPulses' has been saved to a DataFrame.
   Run  Event  SubEvent  SubEventStream  exists  string  om  pmt  \
0    5      0         0               1       1       1   1    0
1    5      0         0               1       1       1   1    0
2    5      0         0               1       1       1   1    0
3    5      0         0               1       1       1   1    0
4    5      0         0               1       1       1   2    0

   vector_index       time  width  charge
0             0  6202895.0    8.0   1.625
```

```
1              1  6205476.0    8.0   1.225
2              2  6215433.0    8.0   0.875
3              3  6293145.0    8.0   1.275
4              0  3679262.0    8.0   0.625
```

[3]:
```python
df_q = df
df_q_sorted_t = df_q.sort_values(by='time')
```

[4]:
```python
df_q.head(20)
```

[4]:
```
    Run  Event  SubEvent  SubEventStream  exists  string  om  pmt  \
0     5      0         0               1       1       1   1    0
1     5      0         0               1       1       1   1    0
2     5      0         0               1       1       1   1    0
3     5      0         0               1       1       1   1    0
4     5      0         0               1       1       1   2    0
5     5      0         0               1       1       1   2    0
6     5      0         0               1       1       1   2    0
7     5      0         0               1       1       1   2    0
8     5      0         0               1       1       1   2    0
9     5      0         0               1       1       1   2    0
10    5      0         0               1       1       1   2    0
11    5      0         0               1       1       1   2    0
12    5      0         0               1       1       1   2    0
13    5      0         0               1       1       1   2    0
14    5      0         0               1       1       1   2    0
15    5      0         0               1       1       1   2    0
16    5      0         0               1       1       1   2    0
17    5      0         0               1       1       1   2    0
18    5      0         0               1       1       1   3    0
19    5      0         0               1       1       1   3    0

    vector_index       time  width  charge
0              0  6202895.0    8.0   1.625
1              1  6205476.0    8.0   1.225
2              2  6215433.0    8.0   0.875
3              3  6293145.0    8.0   1.275
4              0  3679262.0    8.0   0.625
5              1  3708294.0    8.0   1.225
6              2  3795625.0    8.0   1.275
7              3  7088763.0    8.0   1.025
8              4  8219540.0    8.0   2.875
9              5  8219571.0    8.0   1.075
10             6  8222390.0    8.0   0.875
11             7  8234734.0    8.0   1.725
12             8  8250215.0    8.0   0.975
13             9  8250240.0    8.0   0.425
```

```
14            10  8255859.0    8.0   1.225
15            11  8275678.0    8.0   0.975
16            12  8311009.0    8.0   1.475
17            13  8387197.0    8.0   0.725
18             0  1950739.0    8.0   1.175
19             1  3090267.0    8.0   0.425
```

[5]:
```
df_q_sorted_t.head()
```

[5]:
```
           Run  Event  SubEvent  SubEventStream  exists  string  om  pmt  \
298353       5      9         0               1       1      64   4    0
125118736    5   4058         0               1       1       6   7    0
11977842     5    387         0               1       1      85  30    0
125958689    5   4085         0               1       1      26   8    0
201561330    5   6536         0               1       1      26  58    0

           vector_index   time  width  charge
298353                0 -141.0    8.0   0.425
125118736             0 -140.0    8.0   0.425
11977842              0 -136.0    8.0   0.125
125958689             0 -136.0    8.0   0.825
201561330             0 -135.0    8.0   0.825
```

[6]:
```
df_p = pd.read_csv('SplitInIceDSTPulses.csv')
df_p.head()
```

[6]:
```
      Run   Event  SubEvent  SubEventStream  exists  string  om  pmt  \
0  127870  804202         0               0       1       1  56    0
1  127870  804202         0               0       1      12   3    0
2  127870  804202         0               0       1      12   3    0
3  127870  804202         0               0       1      19  14    0
4  127870  804202         0               0       1      19  14    0

   vector_index      time  width  charge
0             0  442884.0    8.0   1.525
1             0  438023.0    8.0   0.675
2             1  444341.0    8.0   1.025
3             0  441483.0    8.0   0.475
4             1  444915.0    8.0   0.475
```

[7]:
```
df_p_sorted_t = df_p.sort_values(by='time')
```

[8]:
```
df_p_sorted_t.head()
```

[8]:
```
            Run     Event  SubEvent  SubEventStream  exists  string  om  pmt  \
2910885  127938  46017518         0               0       1      28  22    0
7522769  127910   7299274         0               0       1      60   9    0
```

```
5088931  127897  40763812         0              0        1       37  41     0
5899629  127874  53123243         0              0        1       36  51     0
3965039  127886   5670183         0              0        1        5  15     0

         vector_index    time  width  charge
2910885             0  5712.0    8.0   1.125
7522769             0  5715.0    8.0   1.375
5088931             0  5727.0    8.0   0.825
5899629             0  5728.0    8.0   0.625
3965039             0  5729.0    8.0   0.975
```

[9]:
```python
df_p_sorted_E = df_p.sort_values(by='Event')
```

[10]:
```python
sub_df_p = df_p.groupby(['Event', 'SubEvent'], as_index=False).agg({'charge':
 ↪'sum'})
```

[11]:
```python
sub_df_q = df_q.groupby(['Event', 'SubEvent'], as_index=False).agg({'charge':
 ↪'sum'})
```

[12]:
```python
sub_df_q
```

[12]:
```
      Event  SubEvent        charge
0         0         0  30702.500032
1         1         0  30104.575021
2         2         0  30709.400017
3         3         0  30975.600008
4         4         0  31244.875028
…       …         …             …
8888   8888         0  31024.425021
8889   8889         0  30881.125024
8890   8890         0  30935.450009
8891   8891         0  31811.050024
8892   8892         0  30577.950019

[8893 rows x 3 columns]
```

[13]:
```python
df_p_sorted_E.tail(20)
```

[13]:
```
           Run     Event  SubEvent  SubEventStream  exists  string  om  pmt  \
200118  127870  76526981        14               0       1      81  43    0
200119  127870  76526981        14               0       1      81  43    0
199012  127870  76526981         0               0       1      14  59    0
199013  127870  76526981         0               0       1      14  60    0
199046  127870  76526981         0               0       1      31  54    0
199047  127870  76526981         0               0       1      35  30    0
199048  127870  76526981         0               0       1      36   3    0
199017  127870  76526981         0               0       1      15  57    0
```

```
199018  127870  76526981         0             0        1        17   39     0
199019  127870  76526981         0             0        1        22   19     0
199020  127870  76526981         0             0        1        22   45     0
199021  127870  76526981         0             0        1        22   50     0
199022  127870  76526981         0             0        1        22   53     0
199023  127870  76526981         0             0        1        22   54     0
199024  127870  76526981         0             0        1        22   54     0
199025  127870  76526981         0             0        1        22   56     0
199026  127870  76526981         0             0        1        22   56     0
200120  127870  76526981        14             0        1        81   58     0
199015  127870  76526981         0             0        1        14   60     0
199016  127870  76526981         0             0        1        14   60     0

        vector_index       time  width  charge
200118             0  9908017.0    8.0   1.075
200119             1  9916423.0    8.0   0.975
199012             1   997603.0    1.0   1.075
199013             0   997496.0    1.0   0.125
199046             1   996768.0    8.0   0.325
199047             0  1000654.0    8.0   1.175
199048             0   999651.0    8.0   1.775
199017             0   998182.0    8.0   0.775
199018             0   999068.0    8.0   0.875
199019             0   996053.0    8.0   0.825
199020             0   997827.0    8.0   1.025
199021             0   999107.0    8.0   1.275
199022             0   997210.0    1.0   1.325
199023             0   997125.0    1.0   0.825
199024             1   997408.0    1.0   0.675
199025             0   997104.0    1.0   1.825
199026             1   997121.0    1.0   0.275
200120             0  9913223.0    8.0   0.875
199015             2   997576.0    1.0   1.475
199016             3   997634.0    1.0   1.075
```

[14]:
```python
df_q_sorted_E = df_q.sort_values(by='Event')
```

[15]:
```python
df_q_sorted_E.head(20)
```

[15]:
```
     Run  Event  SubEvent  SubEventStream  exists  string  om  pmt  \
18     5      0         0               1       1       1   3    0
17     5      0         0               1       1       1   2    0
16     5      0         0               1       1       1   2    0
15     5      0         0               1       1       1   2    0
14     5      0         0               1       1       1   2    0
13     5      0         0               1       1       1   2    0
220    5      0         0               1       1       1  27    0
```

219	5	0	0	1	1	1	27	0
218	5	0	0	1	1	1	27	0
217	5	0	0	1	1	1	27	0
216	5	0	0	1	1	1	27	0
215	5	0	0	1	1	1	27	0
214	5	0	0	1	1	1	27	0
213	5	0	0	1	1	1	27	0
212	5	0	0	1	1	1	27	0
211	5	0	0	1	1	1	26	0
223	5	0	0	1	1	1	28	0
222	5	0	0	1	1	1	27	0
221	5	0	0	1	1	1	27	0
204	5	0	0	1	1	1	26	0

	vector_index	time	width	charge
18	0	1950739.0	8.0	1.175
17	13	8387197.0	8.0	0.725
16	12	8311009.0	8.0	1.475
15	11	8275678.0	8.0	0.975
14	10	8255859.0	8.0	1.225
13	9	8250240.0	8.0	0.425
220	8	9358267.0	8.0	0.775
219	7	9331380.0	8.0	0.225
218	6	9313630.0	8.0	0.325
217	5	9313592.0	8.0	1.175
216	4	2889292.0	8.0	0.625
215	3	2347160.0	8.0	0.975
214	2	2290330.0	8.0	0.675
213	1	2286354.0	8.0	0.225
212	0	307181.0	8.0	1.275
211	9	3692237.0	8.0	0.425
223	0	4137708.0	8.0	1.325
222	10	9367811.0	8.0	0.325
221	9	9363961.0	8.0	0.475
204	2	664869.0	8.0	0.475

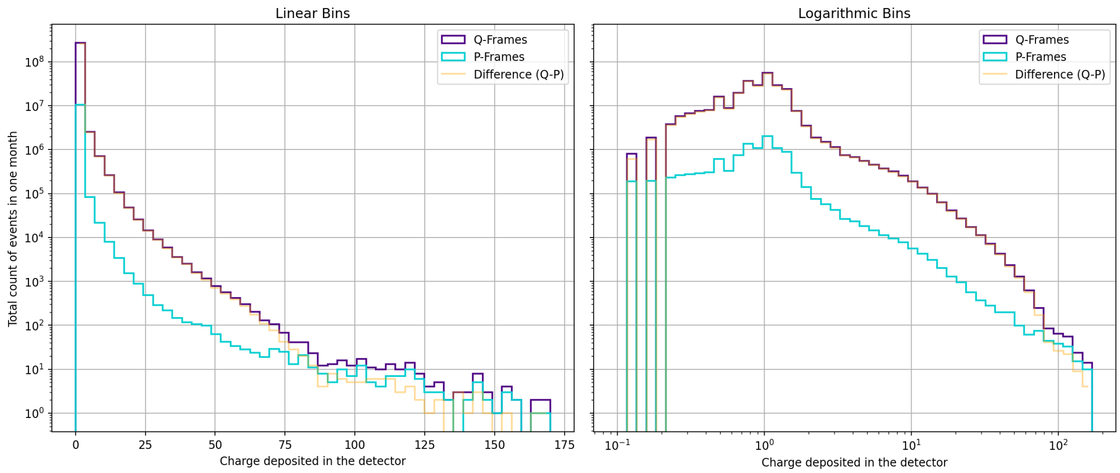
[16]:
```
bins= np.linspace(0,170,50)
bins_log = np.logspace(-1, np.log10(170), 50)
```

[17]:
```
df_diff = df_q['charge'] - df_p['charge']
```

[18]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define bins
bins = np.linspace(0, 170, 50)
bins_log = np.logspace(-1, np.log10(170), 50)
```

```
# Calculate histogram counts for linear bins
counts_q, _ = np.histogram(df_q['charge'], bins=bins)
counts_p, _ = np.histogram(df_p['charge'], bins=bins)
counts_diff_linear = counts_q - counts_p  # Difference in counts for linear bins

# Calculate bin centers for linear bins
bin_centers_linear = (bins[:-1] + bins[1:]) / 2

# Calculate histogram counts for logarithmic bins
counts_q_log, _ = np.histogram(df_q['charge'], bins=bins_log)
counts_p_log, _ = np.histogram(df_p['charge'], bins=bins_log)
counts_diff_log = counts_q_log - counts_p_log  # Difference in counts for␣
↪logarithmic bins

# Calculate bin centers for logarithmic bins
bin_centers_log = (bins_log[:-1] + bins_log[1:]) / 2

# Create a subplot
fig, axs = plt.subplots(1, 2, figsize=(14, 6), sharey=True, dpi=170)

# Linear bins: Plot Q-Frames, P-Frames, and the difference
axs[0].hist(df_q['charge'], bins=bins, histtype='step', linewidth=1.5,␣
↪color='indigo', log=False, label='Q-Frames')
axs[0].hist(df_p['charge'], bins=bins, histtype='step', linewidth=1.5,␣
↪color='darkturquoise', log=False, label='P-Frames')
axs[0].step(bin_centers_linear, counts_diff_linear, where='mid',␣
↪color='orange', label='Difference (Q-P)', linewidth=1.5, alpha= 0.4)
axs[0].grid(True)
axs[0].legend()
axs[0].set_xlabel('Charge deposited in the detector')
axs[0].set_ylabel('Total count of events in one month')
axs[0].set_title('Linear Bins')

# Logarithmic bins: Plot Q-Frames, P-Frames, and the difference
axs[1].hist(df_q['charge'], bins=bins_log, histtype='step', linewidth=1.5,␣
↪color='indigo', log=True, label='Q-Frames')
axs[1].hist(df_p['charge'], bins=bins_log, histtype='step', linewidth=1.5,␣
↪color='darkturquoise', log=True, label='P-Frames')
axs[1].step(bin_centers_log, counts_diff_log, where='mid', color='orange',␣
↪label='Difference (Q-P)', linewidth=1.5, alpha= 0.4)
axs[1].set_xscale('log')
axs[1].grid(True)
axs[1].legend()
axs[1].set_xlabel('Charge deposited in the detector')
axs[1].set_title('Logarithmic Bins')
```

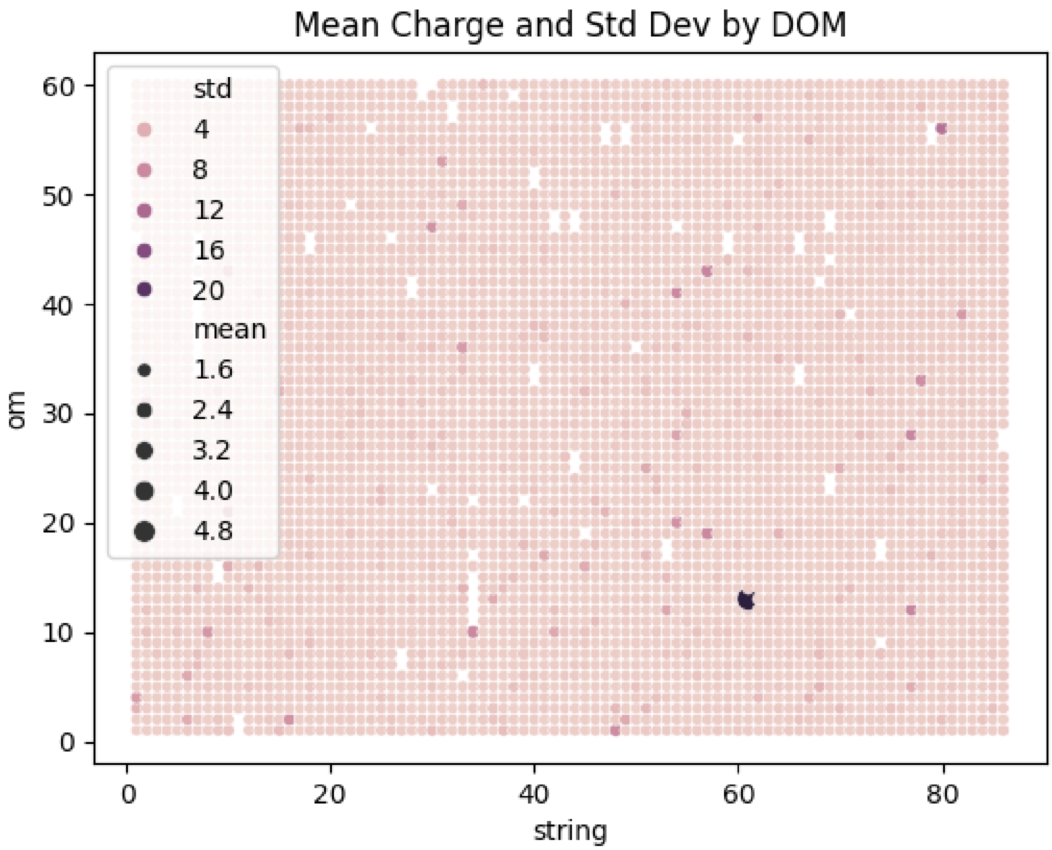
```
# Adjust layout
plt.tight_layout()
plt.savefig('plots/q_p_comp.pdf', format='pdf')
plt.show()
```



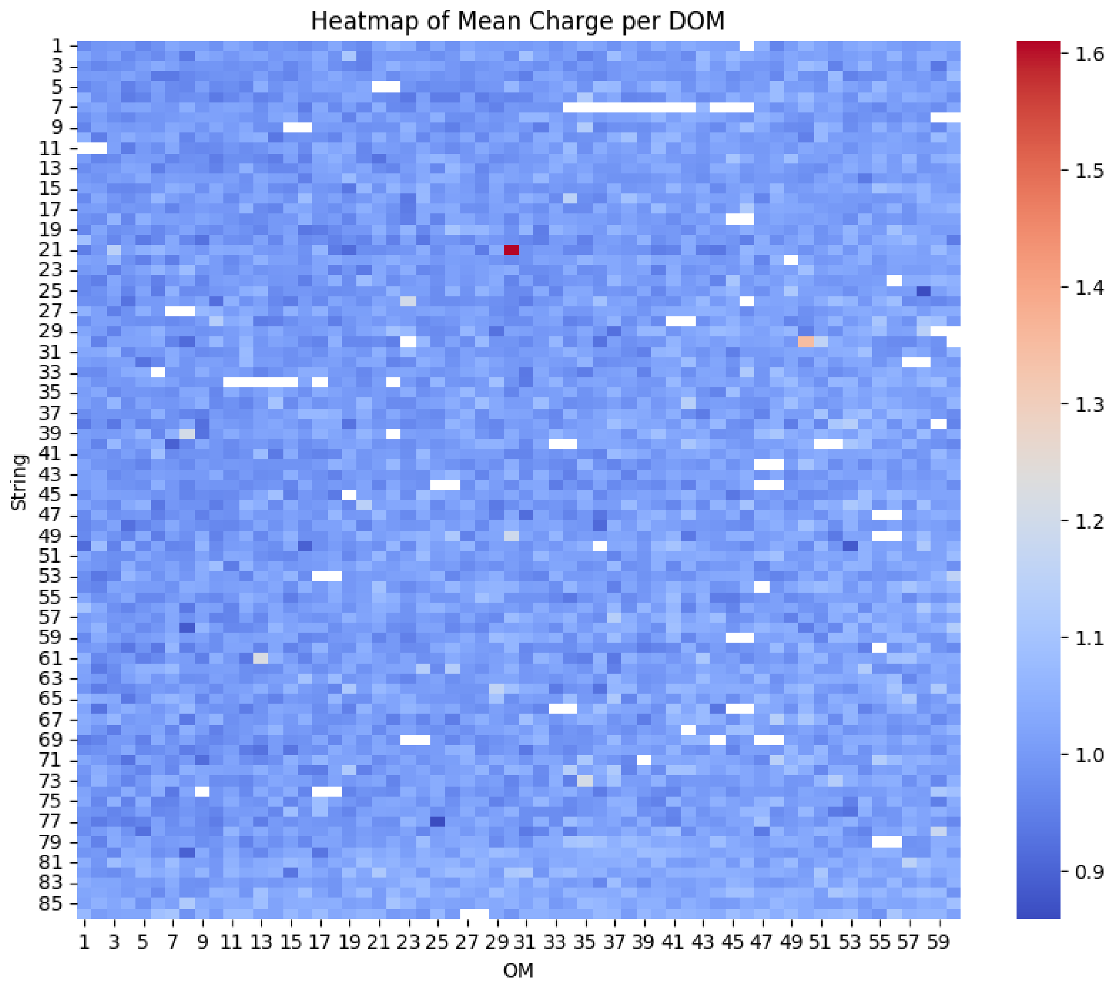
[19]:
```
df_q_stats = df_q.groupby(['string', 'om'])['charge'].agg(['mean', 'std',␣
↪'count', 'skew']).reset_index()
```

[20]:
```
df_p_stats = df_p.groupby(['string', 'om'])['charge'].agg(['mean', 'std',␣
↪'count', 'skew']).reset_index()
```

[21]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.scatterplot(data=df_p_stats, x='string', y='om', size='mean', hue='std')
plt.title("Mean Charge and Std Dev by DOM")
plt.show()
```

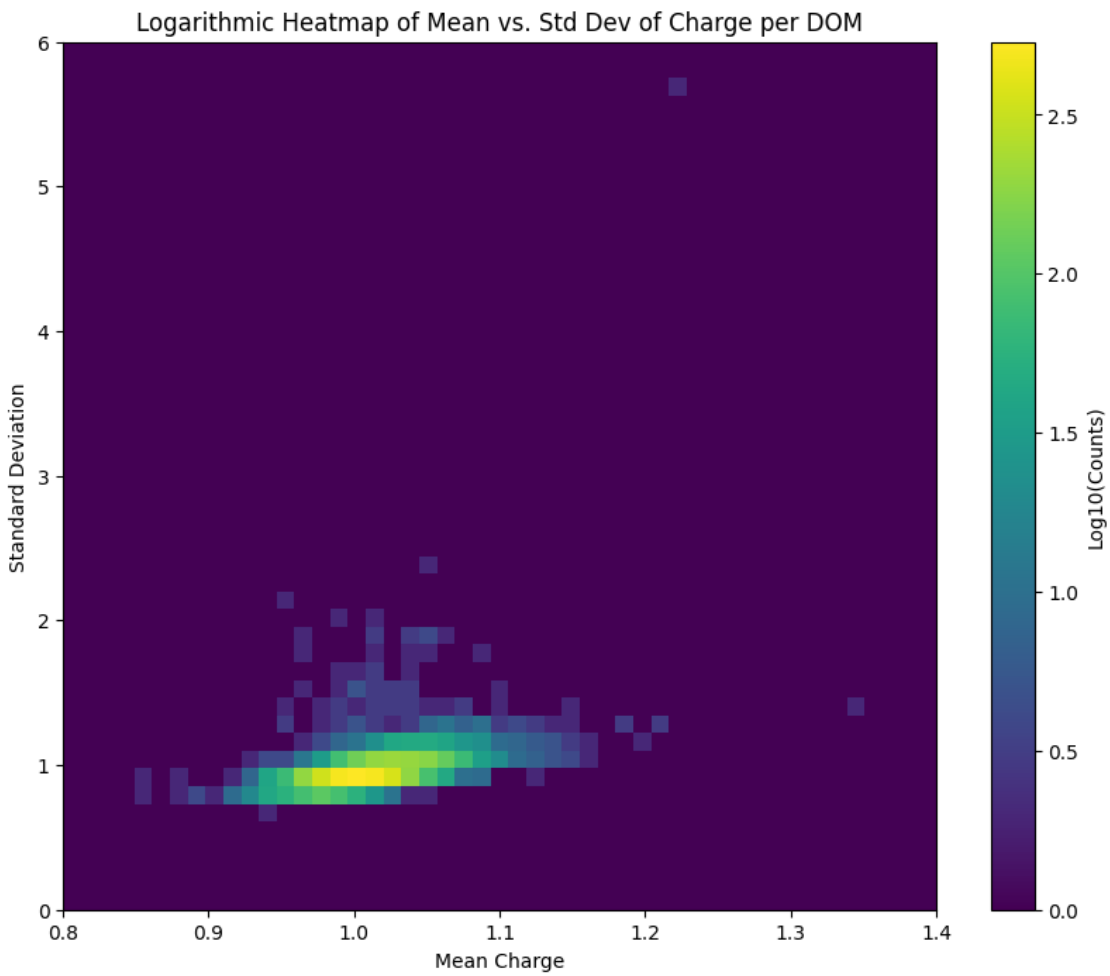


[22]:
```
# Pivot the data for the heatmap
heatmap_data = df_q_stats.pivot(index='string', columns='om', values='mean')

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_data, cmap='coolwarm', annot=False)
plt.title("Heatmap of Mean Charge per DOM")
plt.xlabel("OM")
plt.ylabel("String")
plt.show()
```




```
[23]: import numpy as np
import matplotlib.pyplot as plt

# Define bins for mean and std
mean_bins = np.linspace(0.8, 1.4, 50)  # Adjust bin count as needed
std_bins = np.linspace(0, 6, 50)

# Create a 2D histogram
heatmap, xedges, yedges = np.histogram2d(df_q_stats['mean'], df_q_stats['std'],␣
↪bins=[mean_bins, std_bins])

# Apply logarithmic transformation (add a small value to avoid log(0))
log_heatmap = np.log10(heatmap + 1)

# Plot the heatmap
plt.figure(figsize=(10, 8))
```

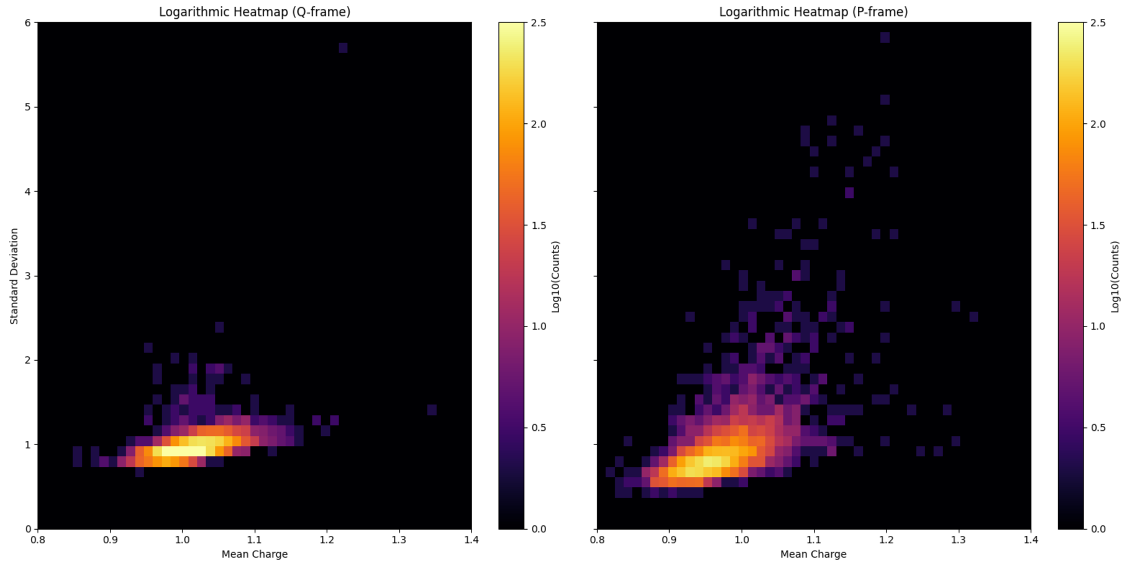
```
plt.imshow(log_heatmap.T, origin='lower', aspect='auto', extent=[mean_bins[0],␣
 ↪mean_bins[-1], std_bins[0], std_bins[-1]], cmap='viridis')
plt.colorbar(label="Log10(Counts)")
plt.xlabel("Mean Charge")
plt.ylabel("Standard Deviation")
plt.title("Logarithmic Heatmap of Mean vs. Std Dev of Charge per DOM")
plt.show()
```



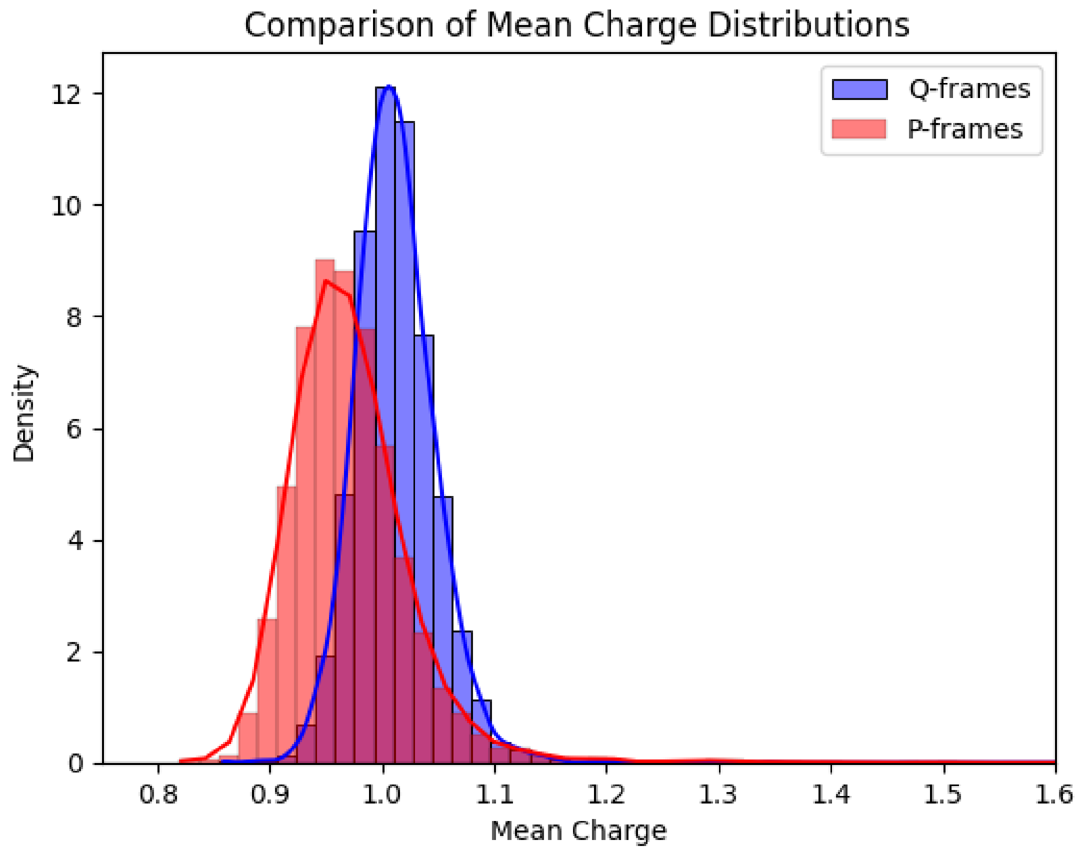
[24]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define bins for mean and std
mean_bins = np.linspace(0.8, 1.4, 50)  # Adjust bin count as needed
std_bins = np.linspace(0, 6, 50)

# Create 2D histograms for Q-frame and P-frame data
```

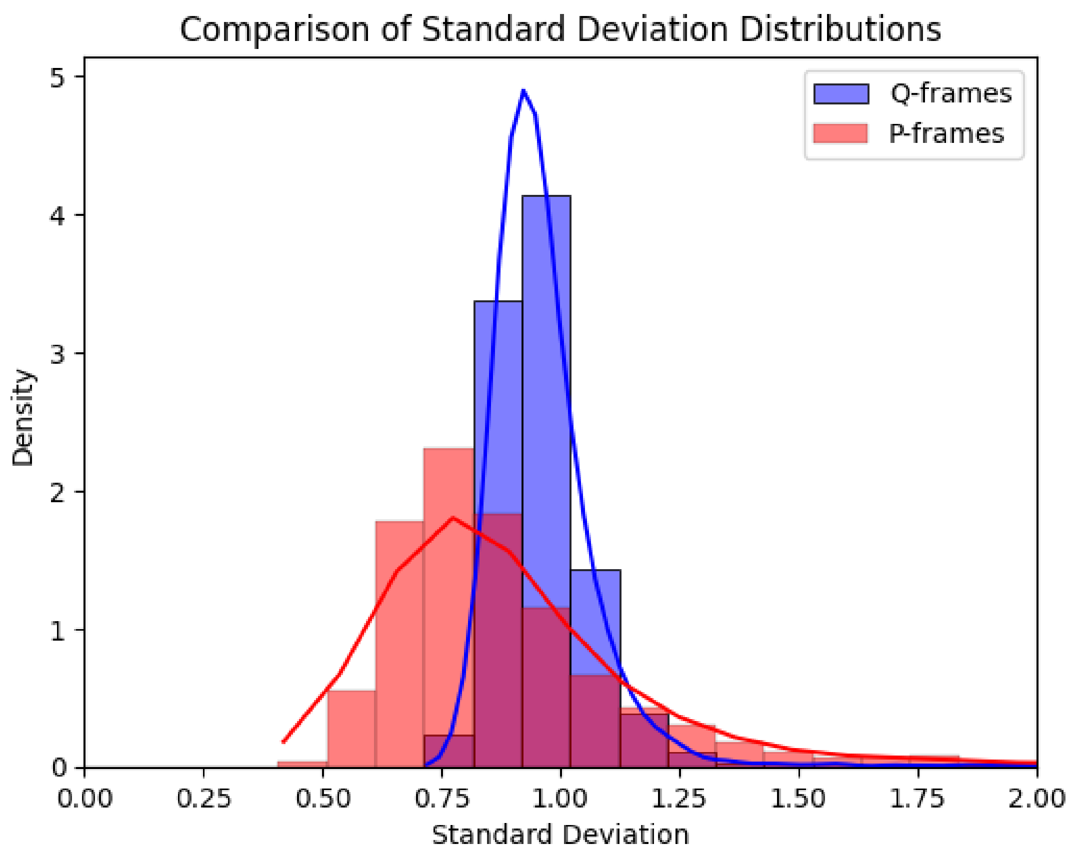
```python
heatmap_q, xedges, yedges = np.histogram2d(df_q_stats['mean'],␣
↪df_q_stats['std'], bins=[mean_bins, std_bins])
heatmap_p, _, _ = np.histogram2d(df_p_stats['mean'], df_p_stats['std'],␣
↪bins=[mean_bins, std_bins])

# Apply logarithmic transformation
log_heatmap_q = np.log10(heatmap_q + 1)
log_heatmap_p = np.log10(heatmap_p + 1)

vmin = 0  # Minimum value for color scale (Log10(Counts))
vmax = 2.5  # Maximum value (adjust as needed)
# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(16, 8), sharey=True)


# Plot the Q-frame heatmap
im1 = axes[0].imshow(
    log_heatmap_q.T,
    origin='lower',
    aspect='auto',
    extent=[mean_bins[0], mean_bins[-1], std_bins[0], std_bins[-1]],
    cmap='inferno',
    vmin=vmin,
    vmax=vmax
)
axes[0].set_title("Logarithmic Heatmap (Q-frame)")
axes[0].set_xlabel("Mean Charge")
axes[0].set_ylabel("Standard Deviation")

fig.colorbar(im1, ax=axes[0], label="Log10(Counts)")

# Plot the P-frame heatmap
im2 = axes[1].imshow(
    log_heatmap_p.T,
    origin='lower',
    aspect='auto',
    extent=[mean_bins[0], mean_bins[-1], std_bins[0], std_bins[-1]],
    cmap='inferno',
    vmin=vmin,
    vmax=vmax
)
axes[1].set_title("Logarithmic Heatmap (P-frame)")
axes[1].set_xlabel("Mean Charge")
fig.colorbar(im2, ax=axes[1], label="Log10(Counts)")

# Adjust layout
plt.tight_layout()
```

```
plt.show()
```



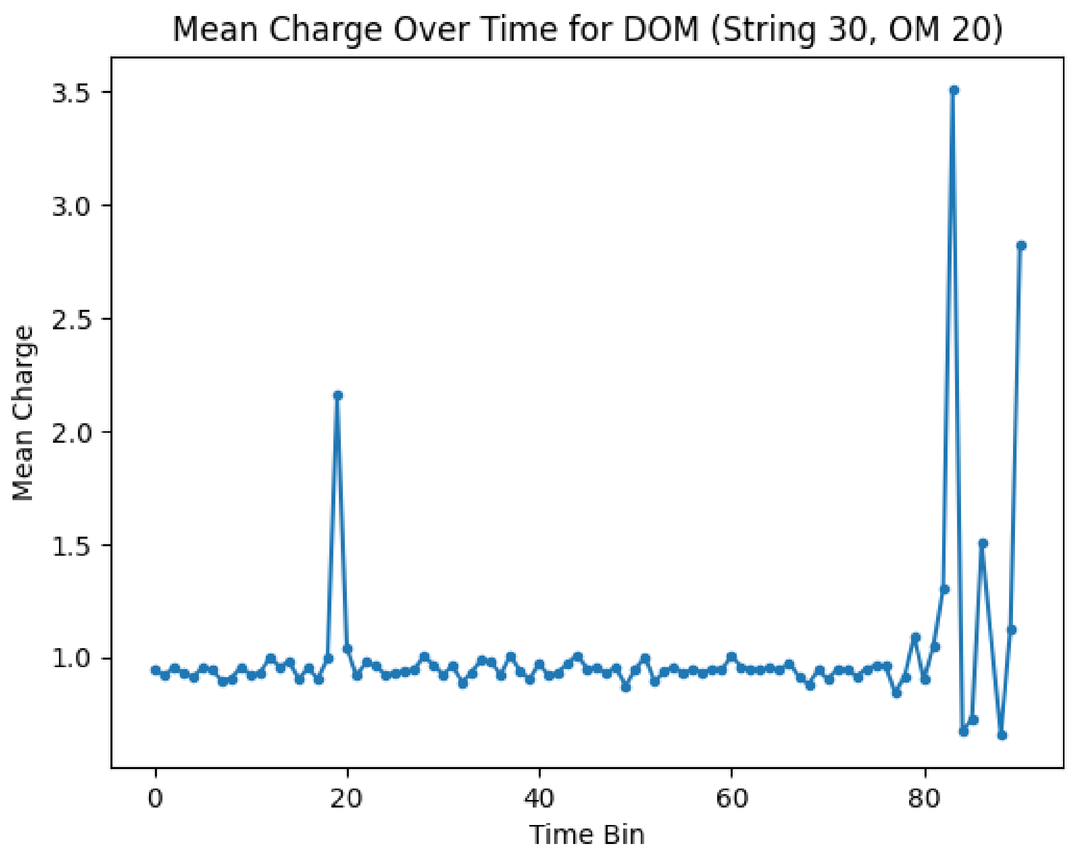
[25]:
```
import seaborn as sns
bins_mean= np.linspace(0.75,1.6,50)
bins_std = np.linspace(0,5,50)
# Compare mean charge distributions
sns.histplot(df_q_stats['mean'], bins=bins_mean, color='blue',␣
↪label='Q-frames', kde=True, stat='density')
sns.histplot(df_p_stats['mean'], bins=bins_mean, color='red', label='P-frames',␣
↪kde=True, stat='density')
plt.xlabel("Mean Charge")
plt.ylabel("Density")
plt.title("Comparison of Mean Charge Distributions")
plt.xlim(0.75, 1.6)
plt.legend()
plt.show()

# Compare standard deviation distributions
sns.histplot(df_q_stats['std'], bins=bins_std, color='blue', label='Q-frames',␣
↪kde=True, stat='density')
sns.histplot(df_p_stats['std'], bins=bins_std, color='red', label='P-frames',␣
↪kde=True, stat='density')
plt.xlabel("Standard Deviation")
plt.ylabel("Density")
plt.title("Comparison of Standard Deviation Distributions")
plt.xlim(0, 2)
plt.legend()
plt.show()
```


Comparison of Mean Charge Distributions
Q-frames
P-frames
Density
Mean Charge
0
2
4
6
8
10
12
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
1.6

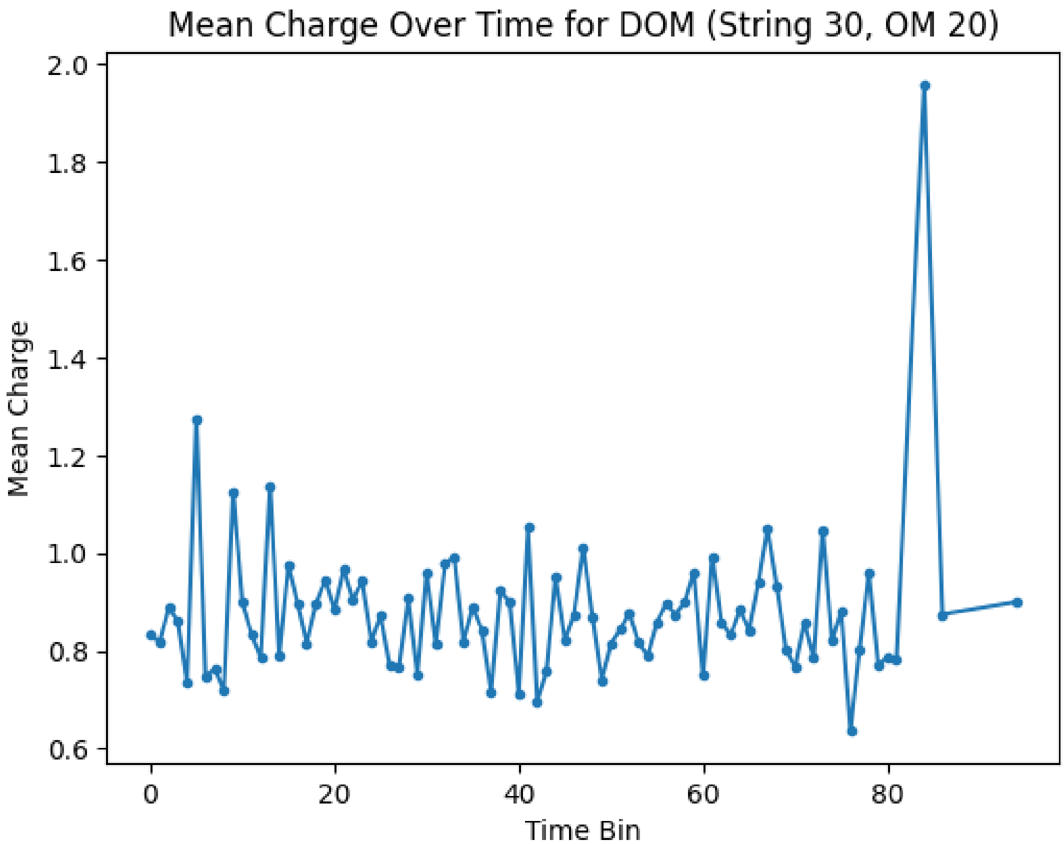


[26]:
```
df_q['time_bin'] = pd.cut(df_q['time'], bins=100)
time_q_stats = df_q.groupby(['time_bin', 'string', 'om'],␣
↪observed=True)['charge'].mean().reset_index()

df_p['time_bin'] = pd.cut(df_p['time'], bins=100)
time_p_stats = df_p.groupby(['time_bin', 'string', 'om'],␣
↪observed=True)['charge'].mean().reset_index()
```

[27]:
```
q_dom_data = time_q_stats[(time_q_stats['string'] == 30) & (time_q_stats['om']␣
↪== 20)]

# Plot mean charge over time for the selected DOM
plt.plot(q_dom_data['time_bin'].cat.codes, q_dom_data['charge'], marker='.',␣
↪linestyle='-')
plt.xlabel("Time Bin")
plt.ylabel("Mean Charge")
plt.title("Mean Charge Over Time for DOM (String 30, OM 20)")
plt.show()
```



[28]:
```
p_dom_data = time_p_stats[(time_p_stats['string'] == 30) & (time_p_stats['om']␣
↪== 20)]

# Plot mean charge over time for the selected DOM
plt.plot(p_dom_data['time_bin'].cat.codes, p_dom_data['charge'], marker='.',␣
↪linestyle='-')
plt.xlabel("Time Bin")
plt.ylabel("Mean Charge")
plt.title("Mean Charge Over Time for DOM (String 30, OM 20)")
plt.show()
```



[29]:
```
# # Identify high-mean/std DOMs in P-frames
# high_dom_p = df_p_stats[(df_p_stats['mean'] > 1.2) & (df_p_stats['std'] > 2)]

# # Analyze their time-series in P-frames
# for _, row in high_dom_p.iterrows():
#     dom_data = time_p_stats[
#         (time_p_stats['string'] == row['string']) &
#         (time_p_stats['om'] == row['om'])
#     ]
#     plt.plot(dom_data['time_bin'].cat.codes, dom_data['charge'])
#     plt.title(f"Time-Series for DOM {row['string']}-{row['om']} (P-frame)")
#     plt.xlabel("Time Bin")
#     plt.ylabel("Charge")
#     plt.show()
```

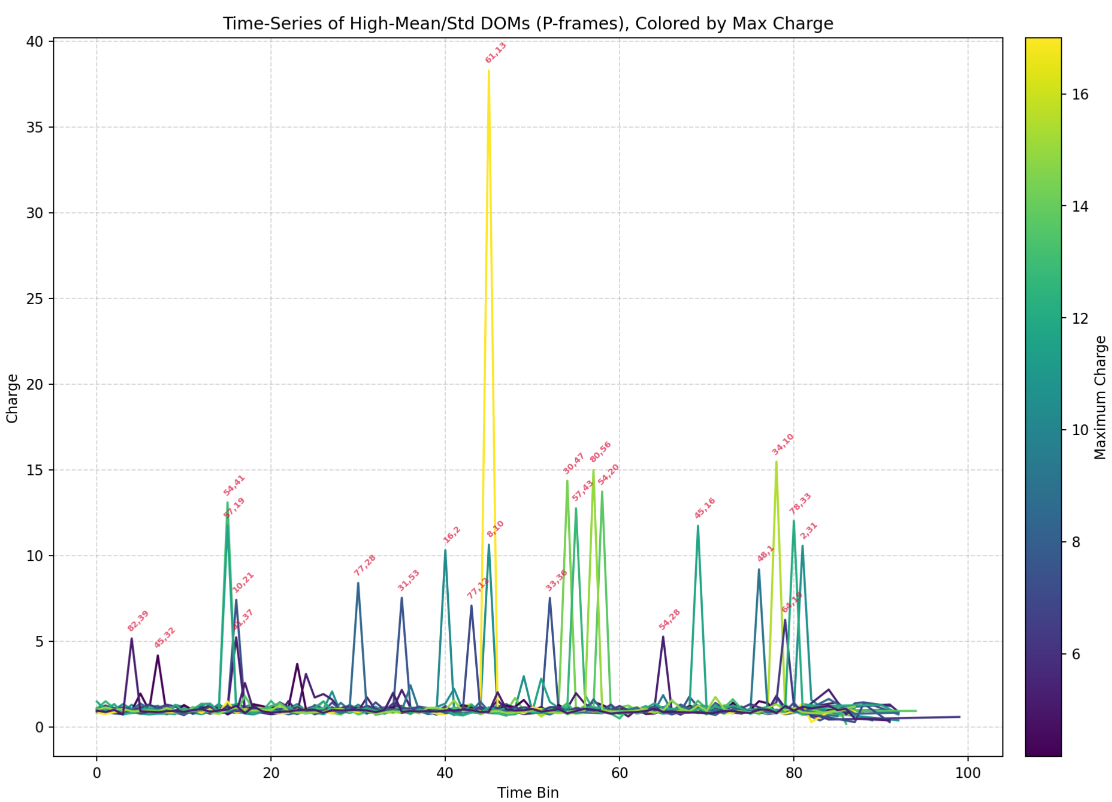
[38]:
```
# import matplotlib.pyplot as plt
# import matplotlib.cm as cm
# import numpy as np

# # Identify high-mean/std DOMs in P-frames
```

```
# high_dom_p = df_p_stats[(df_p_stats['mean'] > 1.2) & (df_p_stats['std'] > 2)]

# # Calculate maximum charge for each DOM
# dom_max_charges = []
# for _, row in high_dom_p.iterrows():
#     dom_data = time_p_stats[
#         (time_p_stats['string'] == row['string']) &
#         (time_p_stats['om'] == row['om'])
#     ]
#     dom_max_charges.append(dom_data['charge'].max())

# # Normalize the maximum charges to [0, 1] for the colormap
# norm = plt.Normalize(min(dom_max_charges), 17)
# colors = cm.viridis(norm(dom_max_charges))  # Use the 'viridis' colormap

# # Create the plot
# fig, ax = plt.subplots(figsize=(12, 8))  # Explicitly create Axes for plotting
# for i, (index, row) in enumerate(high_dom_p.iterrows()):
#     dom_data = time_p_stats[
#         (time_p_stats['string'] == row['string']) &
#         (time_p_stats['om'] == row['om'])
#     ]
#     ax.plot(
#         dom_data['time_bin'].cat.codes,  # Convert time bins to integer codes␣
↪for plotting
#         dom_data['charge'],
#         label=f"{int(row['string'])},{int(row['om'])}",
#         color=colors[i]
#     )

# # Add a colorbar explicitly linked to the figure and normalized colormap
# sm = plt.cm.ScalarMappable(cmap='viridis', norm=norm)
# sm.set_array([])  # Empty array as ScalarMappable is only used for the␣
↪colorbar
# #cbar = fig.colorbar(sm, ax=ax, pad=0.02)
# #cbar.set_label('Maximum Charge')

# # Add labels, title, and legend
# ax.set_title("Time-Series of High-Mean/Std DOMs (P-frames), Colored by Max␣
↪Charge")
# ax.set_xlabel("Time Bin")
# ax.set_ylabel("Charge")
# #ax.legend(loc='upper left', bbox_to_anchor=(1, 1))  # Place the legend␣
↪outside the plot
# ax.grid(True, linestyle='--', alpha=0.5)

# # Adjust layout for better visibility
```

```
# plt.tight_layout()
# plt.show()
```

[31]:
```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

# Identify high-mean/std DOMs in P-frames
high_dom_p = df_p_stats[(df_p_stats['mean'] > 1.2) & (df_p_stats['std'] > 2)]

# Calculate maximum charge for each DOM
dom_max_charges = []
for _, row in high_dom_p.iterrows():
    dom_data = time_p_stats[
        (time_p_stats['string'] == row['string']) &
        (time_p_stats['om'] == row['om'])
    ]
    dom_max_charges.append(dom_data['charge'].max())

# Normalize the maximum charges to [0, 1] for the colormap
norm = plt.Normalize(min(dom_max_charges), 17)
colors = cm.viridis(norm(dom_max_charges))  # Use the 'viridis' colormap

# Create the plot
fig, ax = plt.subplots(figsize=(12, 8), dpi= 170)
for i, (index, row) in enumerate(high_dom_p.iterrows()):
    dom_data = time_p_stats[
        (time_p_stats['string'] == row['string']) &
        (time_p_stats['om'] == row['om'])
    ]

    # Extract data for plotting
    x_values = dom_data['time_bin'].cat.codes
    y_values = dom_data['charge']

    # Plot the time-series
    ax.plot(x_values, y_values, color=colors[i])

    # Annotate at the peak
    peak_index = y_values.idxmax()  # Index of the maximum charge
    peak_x = x_values[peak_index]
    peak_y = y_values[peak_index]
    ax.annotate(
        f"{int(row['string'])},{int(row['om'])}",  # Annotation text
        (peak_x, peak_y),  # Position of the annotation
        textcoords="offset points",  # Offset to prevent overlap
        xytext=(5, 5),  # Offset values (x, y) in points
```

```
        fontsize=6,  # Small font size
        color="crimson",
        alpha=0.7,
        weight= 'bold',  # Set text color to red
        ha="center",  # Center align the text
        rotation=45  # Rotate the text by 45° upwards
    )

# Add a colorbar to indicate the maximum charge values
sm = plt.cm.ScalarMappable(cmap='viridis', norm=norm)
sm.set_array([])  # Empty array as ScalarMappable is only used for the colorbar
cbar = fig.colorbar(sm, ax=ax, pad=0.02)
cbar.set_label('Maximum Charge')

# Add labels, title, and grid
ax.set_title("Time-Series of High-Mean/Std DOMs (P-frames), Colored by Max Charge")
ax.set_xlabel("Time Bin")
ax.set_ylabel("Charge")
ax.grid(True, linestyle='--', alpha=0.5)

# Adjust layout for better visibility
plt.tight_layout()
plt.show()
```



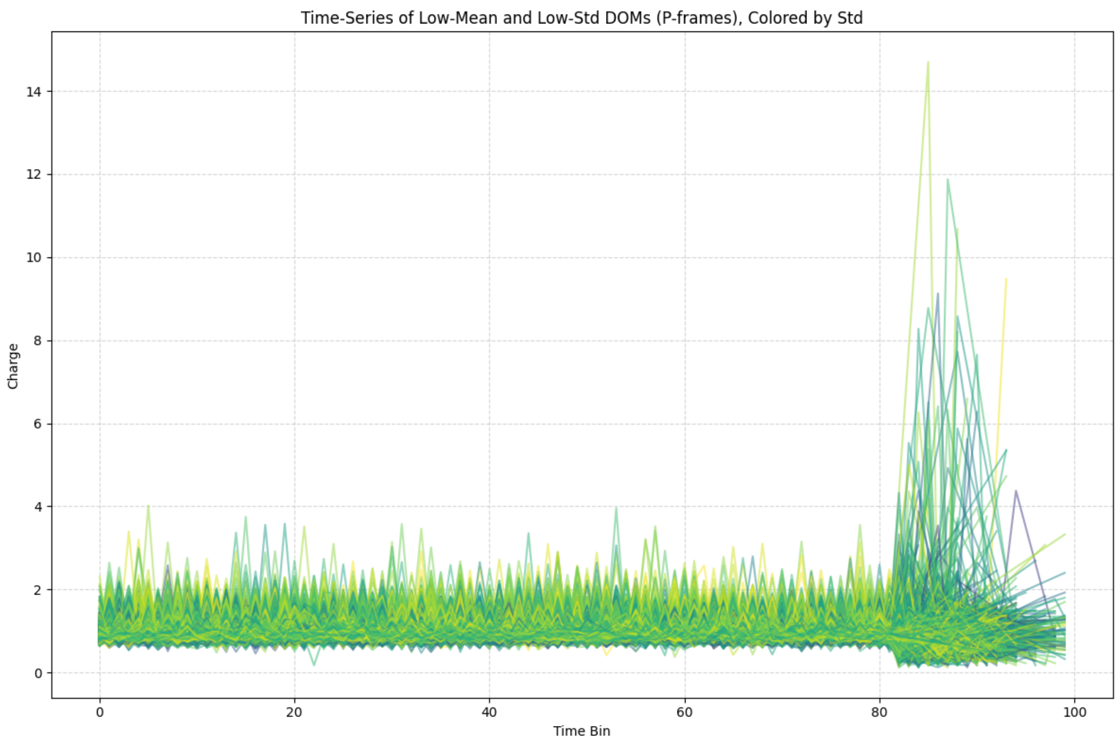
[ ]:

```python

```

[32]:

```python
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

# Create a subset for low-mean and low-std DOMs
low_dom_p = df_p_stats[(df_p_stats['mean'] < 1) & (df_p_stats['std'] < 1)]

# Normalize the `std` column for the colormap
norm = plt.Normalize(low_dom_p['std'].min(), low_dom_p['std'].max())
colors = cm.viridis(norm(low_dom_p['std']))  # Use the 'viridis' colormap

# Create the plot
fig, ax = plt.subplots(figsize=(12, 8))

# Loop through each DOM in the subset
for i, (index, row) in enumerate(low_dom_p.iterrows()):
    dom_data = time_p_stats[
        (time_p_stats['string'] == row['string']) &
        (time_p_stats['om'] == row['om'])
    ]

    # Extract data for plotting
    x_values = dom_data['time_bin'].cat.codes
    y_values = dom_data['charge']

    # Plot the time-series with colormap
    ax.plot(
        x_values,
        y_values,
        linestyle='-',
        color=colors[i],  # Use the colormap color
        alpha=0.5
    )

# # Add a colorbar to indicate the mapped property (`std`)
# sm = plt.cm.ScalarMappable(cmap='inferno', norm=norm)
# sm.set_array([])  # Empty array for ScalarMappable
# #cbar = fig.colorbar(sm, ax=ax, pad=0.02)
# cbar.set_label('Standard Deviation')

# Add labels, title, and grid
```

```
ax.set_title("Time-Series of Low-Mean and Low-Std DOMs (P-frames), Colored by␣
↪Std")
ax.set_xlabel("Time Bin")
ax.set_ylabel("Charge")
ax.grid(True, linestyle='--', alpha=0.5)

# Adjust layout for better visibility
plt.tight_layout()
plt.show()
```



[33]:
```
# Filter the time_p_stats DataFrame to include only time bins <= 80
filtered_time_p_stats = time_p_stats[time_p_stats['time_bin'].cat.codes <= 80]

# Create a subset for low-mean and low-std DOMs
low_dom_p = df_p_stats[(df_p_stats['mean'] < 1) & (df_p_stats['std'] < 0.5)]

norm = plt.Normalize(low_dom_p['std'].min(), low_dom_p['std'].max())
colors = cm.viridis(norm(low_dom_p['std']))  # Use the 'viridis' colormap

# Create the plot for filtered data
fig, ax = plt.subplots(figsize=(12, 8))

# Loop through each DOM in the subset
```

```
for i, (index, row) in enumerate(low_dom_p.iterrows()):
    dom_data = filtered_time_p_stats[
        (filtered_time_p_stats['string'] == row['string']) &
        (filtered_time_p_stats['om'] == row['om'])
    ]

    # Extract data for plotting
    x_values = dom_data['time_bin'].cat.codes
    y_values = dom_data['charge']

    # Plot the time-series with colormap
    ax.plot(
        x_values,
        y_values,
        linestyle='-',
        color=colors[i],  # Use gray color to signify these signals
        alpha=0.7
    )

# Add labels, title, and grid
ax.set_title("Time-Series of Low-Mean and Low-Std DOMs (P-frames), Time Bin <=␣
 ↪80")
ax.set_xlabel("Time Bin")
ax.set_ylabel("Charge")
ax.grid(True, linestyle='--', alpha=0.5)

# Adjust layout for better visibility
plt.tight_layout()
plt.show()
```

[34]:
```
time_bin_counts = time_p_stats['time_bin'].value_counts().sort_index()
#print(time_bin_counts)
# Explicitly set `observed=False` to retain current behavior
dom_counts = time_p_stats.groupby('time_bin', observed=False)['string'].
    ↪nunique()
print(dom_counts)
```

```
time_bin
(-6518.542, 128017.42]        86
(128017.42, 250322.84]        86
(250322.84, 372628.26]        86
(372628.26, 494933.68]        86
(494933.68, 617239.1]         86
                              ..
(11624726.9, 11747032.32]     27
(11747032.32, 11869337.74]    47
(11869337.74, 11991643.16]    31
(11991643.16, 12113948.58]    24
(12113948.58, 12236254.0]     42
Name: string, Length: 100, dtype: int64
```

[35]:
```
# plt.scatter(df_p_stats['string'], df_p_stats['om'], c=df_p_stats['mean'],␣
    ↪cmap='viridis', s=100)
```

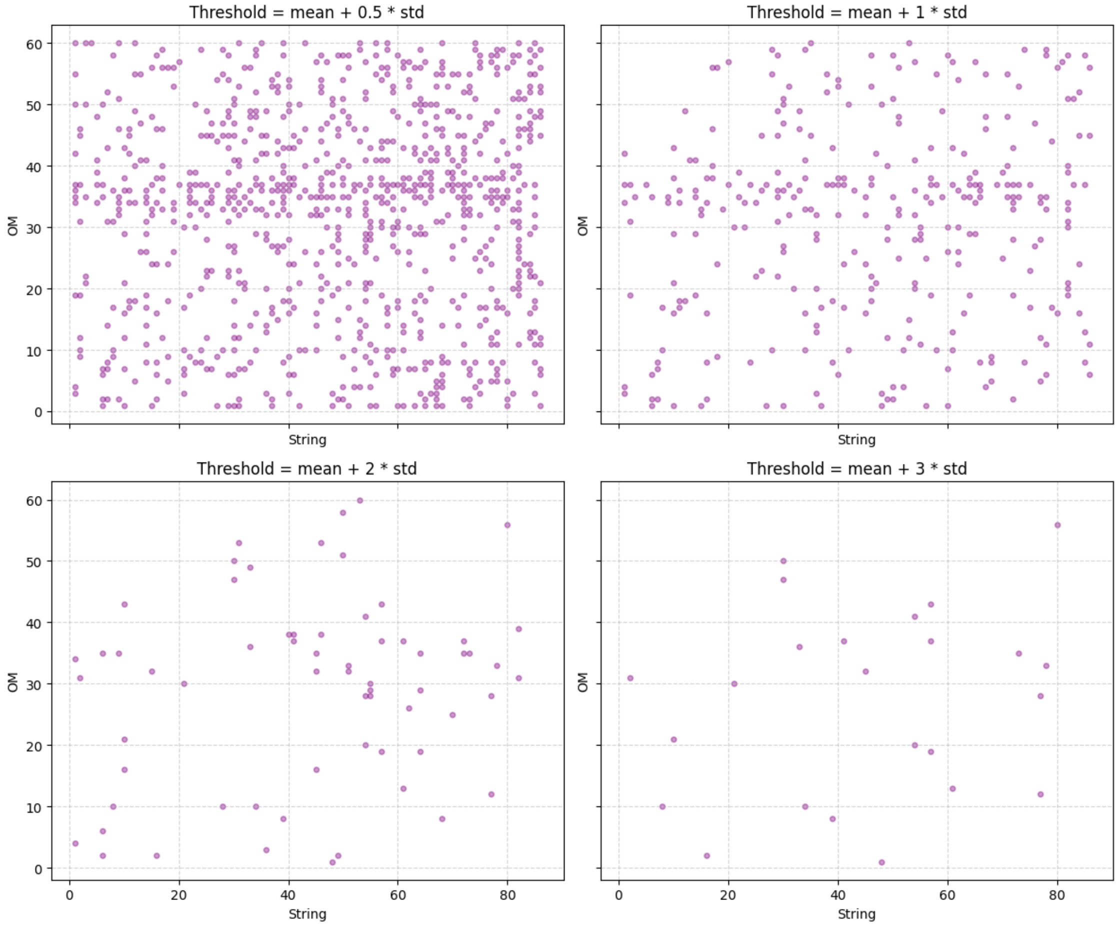
```
# plt.colorbar(label="Mean Charge")
# plt.xlabel("String")
# plt.ylabel("OM")
# plt.title("Spatial Distribution of Mean Charge (P-frames)")
# plt.show()
```

[36]:
```
import matplotlib.pyplot as plt

# Calculate the threshold: mean of the mean values + 1 standard deviation
mean_mean = df_p_stats['mean'].mean()
std_mean = df_p_stats['mean'].std()
threshold = (mean_mean + 0.5*std_mean)
print(f"Threshold for high mean charge: {threshold}")

# Filter for high-mean charge DOMs in P-frames
high_mean_p = df_p_stats[df_p_stats['mean'] > threshold]

# # Scatter plot of high-mean charge DOMs
# plt.figure(figsize=(10, 8))
# plt.scatter(
#     high_mean_p['string'],
#     high_mean_p['om'],
#     color='purple',  # All points in the same color
#     s=15,
#     alpha=0.7
# )
# plt.title("Scatter Plot of High-Mean Charge DOMs (P-frames)")
# plt.xlabel("String")
# plt.ylabel("OM")
# plt.grid(True, linestyle='--', alpha=0.5)  # Optional: Add a grid for better␣
↪visualization
# plt.show()
```

```
Threshold for high mean charge: 1.0112156242454493
```

[37]:
```
import matplotlib.pyplot as plt

# Calculate the mean and standard deviation of the mean values
mean_mean = df_p_stats['mean'].mean()
std_mean = df_p_stats['mean'].std()

# Define thresholds with x = [0.5, 1, 1.5, 2]
x_values = [0.5, 1, 2, 3]
thresholds = [mean_mean + x * std_mean for x in x_values]

# Create the 2x2 subplot figure
fig, axes = plt.subplots(2, 2, figsize=(12, 10), sharex=True, sharey=True)
```

```python
# Flatten the axes array for easier iteration
axes = axes.flatten()

# Generate scatter plots for each threshold
for i, threshold in enumerate(thresholds):
    # Filter for high-mean charge DOMs
    high_mean_p = df_p_stats[df_p_stats['mean'] > threshold]

    # Scatter plot
    axes[i].scatter(
        high_mean_p['string'],
        high_mean_p['om'],
        color='purple',  # All points in the same color
        s=15,
        alpha=0.4
    )
    axes[i].set_title(f"Threshold = mean + {x_values[i]} * std")
    axes[i].set_xlabel("String")
    axes[i].set_ylabel("OM")
    axes[i].grid(True, linestyle='--', alpha=0.5)

# Adjust layout
plt.tight_layout()
plt.show()
```



[39]:
```
import numpy as np
import pandas as pd

# Parameters for defining clusters
charge_threshold = 1.2
min_dom_count = 3  # Minimum number of DOMs for a cluster

# Function to check spatial proximity
def are_doms_spatially_adjacent(dom1, dom2):
    return abs(dom1['string'] - dom2['string']) <= 1 and abs(dom1['om'] -␣
 ↪dom2['om']) <= 1

# Search for clusters
clusters = []
for time_bin, group in time_p_stats.groupby('time_bin'):
    # Filter for DOMs with charge above the threshold
    significant_doms = group[group['charge'] > charge_threshold]
```

```python
    # Skip if not enough DOMs for a cluster
    if len(significant_doms) < min_dom_count:
        continue

    # Check spatial proximity
    cluster_candidates = []
    for _, dom in significant_doms.iterrows():
        # Check proximity to other DOMs
        neighboring_doms = significant_doms[
            significant_doms.apply(lambda x: are_doms_spatially_adjacent(dom,␣
↪x), axis=1)
        ]
        if len(neighboring_doms) >= min_dom_count:
            cluster_candidates.append(neighboring_doms)

    # Add cluster candidates to the list
    clusters.append({
        'time_bin': time_bin,
        'clusters': cluster_candidates
    })

# Output the identified clusters
print(f"Identified {len(clusters)} clusters:")
for cluster in clusters:
    print(f"Time Bin {cluster['time_bin']}: {len(cluster['clusters'])}␣
↪clusters")
```

```
/tmp/ipykernel_41155/2894619641.py:14: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future version of
pandas. Pass observed=False to retain current behavior or observed=True to adopt
the future default and silence this warning.
  for time_bin, group in time_p_stats.groupby('time_bin'):
```

```
Identified 100 clusters:
Time Bin (-6518.542, 128017.42]: 40 clusters
Time Bin (128017.42, 250322.84]: 48 clusters
Time Bin (250322.84, 372628.26]: 31 clusters
Time Bin (372628.26, 494933.68]: 34 clusters
Time Bin (494933.68, 617239.1]: 44 clusters
Time Bin (617239.1, 739544.52]: 54 clusters
Time Bin (739544.52, 861849.94]: 61 clusters
Time Bin (861849.94, 984155.36]: 83 clusters
Time Bin (984155.36, 1106460.78]: 26 clusters
Time Bin (1106460.78, 1228766.2]: 50 clusters
Time Bin (1228766.2, 1351071.62]: 24 clusters
Time Bin (1351071.62, 1473377.04]: 43 clusters
Time Bin (1473377.04, 1595682.46]: 44 clusters
Time Bin (1595682.46, 1717987.88]: 40 clusters
```

```
Time Bin (1717987.88, 1840293.3]: 29 clusters
Time Bin (1840293.3, 1962598.72]: 37 clusters
Time Bin (1962598.72, 2084904.14]: 63 clusters
Time Bin (2084904.14, 2207209.56]: 59 clusters
Time Bin (2207209.56, 2329514.98]: 57 clusters
Time Bin (2329514.98, 2451820.4]: 52 clusters
Time Bin (2451820.4, 2574125.82]: 53 clusters
Time Bin (2574125.82, 2696431.24]: 26 clusters
Time Bin (2696431.24, 2818736.66]: 50 clusters
Time Bin (2818736.66, 2941042.08]: 45 clusters
Time Bin (2941042.08, 3063347.5]: 29 clusters
Time Bin (3063347.5, 3185652.92]: 64 clusters
Time Bin (3185652.92, 3307958.34]: 49 clusters
Time Bin (3307958.34, 3430263.76]: 51 clusters
Time Bin (3430263.76, 3552569.18]: 36 clusters
Time Bin (3552569.18, 3674874.6]: 47 clusters
Time Bin (3674874.6, 3797180.02]: 74 clusters
Time Bin (3797180.02, 3919485.44]: 42 clusters
Time Bin (3919485.44, 4041790.86]: 82 clusters
Time Bin (4041790.86, 4164096.28]: 40 clusters
Time Bin (4164096.28, 4286401.7]: 119 clusters
Time Bin (4286401.7, 4408707.12]: 47 clusters
Time Bin (4408707.12, 4531012.54]: 34 clusters
Time Bin (4531012.54, 4653317.96]: 58 clusters
Time Bin (4653317.96, 4775623.38]: 62 clusters
Time Bin (4775623.38, 4897928.8]: 71 clusters
Time Bin (4897928.8, 5020234.22]: 43 clusters
Time Bin (5020234.22, 5142539.64]: 38 clusters
Time Bin (5142539.64, 5264845.06]: 39 clusters
Time Bin (5264845.06, 5387150.48]: 40 clusters
Time Bin (5387150.48, 5509455.9]: 40 clusters
Time Bin (5509455.9, 5631761.32]: 49 clusters
Time Bin (5631761.32, 5754066.74]: 36 clusters
Time Bin (5754066.74, 5876372.16]: 52 clusters
Time Bin (5876372.16, 5998677.58]: 60 clusters
Time Bin (5998677.58, 6120983.0]: 42 clusters
Time Bin (6120983.0, 6243288.42]: 34 clusters
Time Bin (6243288.42, 6365593.84]: 59 clusters
Time Bin (6365593.84, 6487899.26]: 64 clusters
Time Bin (6487899.26, 6610204.68]: 36 clusters
Time Bin (6610204.68, 6732510.1]: 41 clusters
Time Bin (6732510.1, 6854815.52]: 43 clusters
Time Bin (6854815.52, 6977120.94]: 34 clusters
Time Bin (6977120.94, 7099426.36]: 49 clusters
Time Bin (7099426.36, 7221731.78]: 35 clusters
Time Bin (7221731.78, 7344037.2]: 44 clusters
Time Bin (7344037.2, 7466342.62]: 40 clusters
Time Bin (7466342.62, 7588648.04]: 65 clusters
```

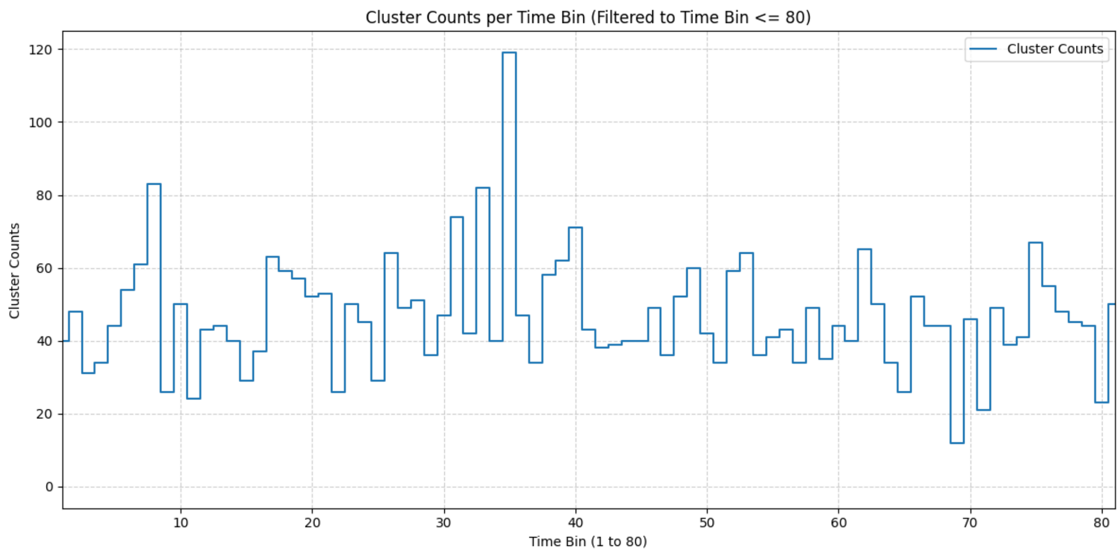
```
Time Bin (7588648.04, 7710953.46]: 50 clusters
Time Bin (7710953.46, 7833258.88]: 34 clusters
Time Bin (7833258.88, 7955564.3]: 26 clusters
Time Bin (7955564.3, 8077869.72]: 52 clusters
Time Bin (8077869.72, 8200175.14]: 44 clusters
Time Bin (8200175.14, 8322480.56]: 44 clusters
Time Bin (8322480.56, 8444785.98]: 12 clusters
Time Bin (8444785.98, 8567091.4]: 46 clusters
Time Bin (8567091.4, 8689396.82]: 21 clusters
Time Bin (8689396.82, 8811702.24]: 49 clusters
Time Bin (8811702.24, 8934007.66]: 39 clusters
Time Bin (8934007.66, 9056313.08]: 41 clusters
Time Bin (9056313.08, 9178618.5]: 67 clusters
Time Bin (9178618.5, 9300923.92]: 55 clusters
Time Bin (9300923.92, 9423229.34]: 48 clusters
Time Bin (9423229.34, 9545534.76]: 45 clusters
Time Bin (9545534.76, 9667840.18]: 44 clusters
Time Bin (9667840.18, 9790145.6]: 23 clusters
Time Bin (9790145.6, 9912451.02]: 50 clusters
Time Bin (9912451.02, 10034756.44]: 36 clusters
Time Bin (10034756.44, 10157061.86]: 33 clusters
Time Bin (10157061.86, 10279367.28]: 21 clusters
Time Bin (10279367.28, 10401672.7]: 22 clusters
Time Bin (10401672.7, 10523978.12]: 37 clusters
Time Bin (10523978.12, 10646283.54]: 31 clusters
Time Bin (10646283.54, 10768588.96]: 17 clusters
Time Bin (10768588.96, 10890894.38]: 12 clusters
Time Bin (10890894.38, 11013199.8]: 1 clusters
Time Bin (11013199.8, 11135505.22]: 0 clusters
Time Bin (11135505.22, 11257810.64]: 6 clusters
Time Bin (11257810.64, 11380116.06]: 5 clusters
Time Bin (11380116.06, 11502421.48]: 0 clusters
Time Bin (11502421.48, 11624726.9]: 0 clusters
Time Bin (11624726.9, 11747032.32]: 0 clusters
Time Bin (11747032.32, 11869337.74]: 0 clusters
Time Bin (11869337.74, 11991643.16]: 0 clusters
Time Bin (11991643.16, 12113948.58]: 0 clusters
Time Bin (12113948.58, 12236254.0]: 0 clusters
```

[41]:

```
import matplotlib.pyplot as plt

# Filter time bins <= 80
filtered_time_p_stats = time_p_stats[time_p_stats['time_bin'].cat.codes <= 80]

# Recompute cluster counts for filtered data
clusters_per_time_bin = []
for time_bin, group in filtered_time_p_stats.groupby('time_bin'):
```

```
    # Filter for significant DOMs in the group
    significant_doms = group[group['charge'] > charge_threshold]

    # Check and count clusters based on spatial proximity
    cluster_count = 0
    for _, dom in significant_doms.iterrows():
        neighboring_doms = significant_doms[
            significant_doms.apply(lambda x: are_doms_spatially_adjacent(dom,␣
↪x), axis=1)
        ]
        if len(neighboring_doms) >= min_dom_count:
            cluster_count += 1
    clusters_per_time_bin.append((time_bin, cluster_count))

# Prepare data for the step function
time_bins = [time_bin for time_bin, count in clusters_per_time_bin]
cluster_counts = [count for time_bin, count in clusters_per_time_bin]

# Plot the step function
plt.figure(figsize=(12, 6))
plt.step(range(1, len(cluster_counts) + 1), cluster_counts, where='mid',␣
↪label="Cluster Counts")
plt.xlabel("Time Bin (1 to 80)")
plt.ylabel("Cluster Counts")
plt.title("Cluster Counts per Time Bin (Filtered to Time Bin <= 80)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
plt.xlim(1,81)
plt.show()
```

```
/tmp/ipykernel_41155/3506441835.py:8: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future version of
pandas. Pass observed=False to retain current behavior or observed=True to adopt
the future default and silence this warning.
  for time_bin, group in filtered_time_p_stats.groupby('time_bin'):
```


Cluster Counts per Time Bin (Filtered to Time Bin <= 80)
Cluster Counts
Cluster Counts
Time Bin (1 to 80)
0
20
40
60
80
100
120
10
20
30
40
50
60
70
80

```
[ ]:
```