DBSCAN_Residuals_Gaussians

August 28, 2025

This notebook does DBSCAN clustering on 10s subintegrations of 121 continuous minute observations. Using tempo2 residuals+error, snr and the gaussian fits.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
```

[2]:
```
import os
```

[3]:
```
import pandas as pd
import psrchive
```

[4]:
```
from scipy.optimize import curve_fit
from scipy.stats import skew, kurtosis
from scipy.stats import normaltest
import corner
```

[5]:
```
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource, HoverTool
from bokeh.io.output import output_notebook
import base64
from io import BytesIO
from bokeh.palettes import Viridis256
from matplotlib.colors import Normalize, to_hex
from matplotlib import cm
```

[6]:
```
from sklearn.neighbors import NearestNeighbors
```

[7]:
```
from sklearn.preprocessing import StandardScaler
```

[8]:
```
from bokeh.models import Whisker
```

1 Preparing 2hr data

[9]:
```
cd /idia/projects/pulsar-timing/users/athambiran/MSc_NGC6440A/timing/
↪121mins_10sub/
```

```
/idia/projects/pulsar-timing/users/athambiran/MSc_NGC6440A/timing/121mins_10sub
```

[10]:
```
arch = psrchive.Archive_load("121mins.t10.f4.pazi")
```

[11]:
```
arch.remove_baseline()
arch.dedisperse()
```

[12]:
```
arch.convert_state("Stokes")
arch.get_state()
```

[12]:
```
'Stokes'
```

[13]:
```
arch.fscrunch()
arch.bscrunch(4)
```

[14]:
```
data_for_profiles = arch.get_data().squeeze()
```

[15]:
```
np.shape(data_for_profiles) ## time, Stokes, bins (1024/4)
```

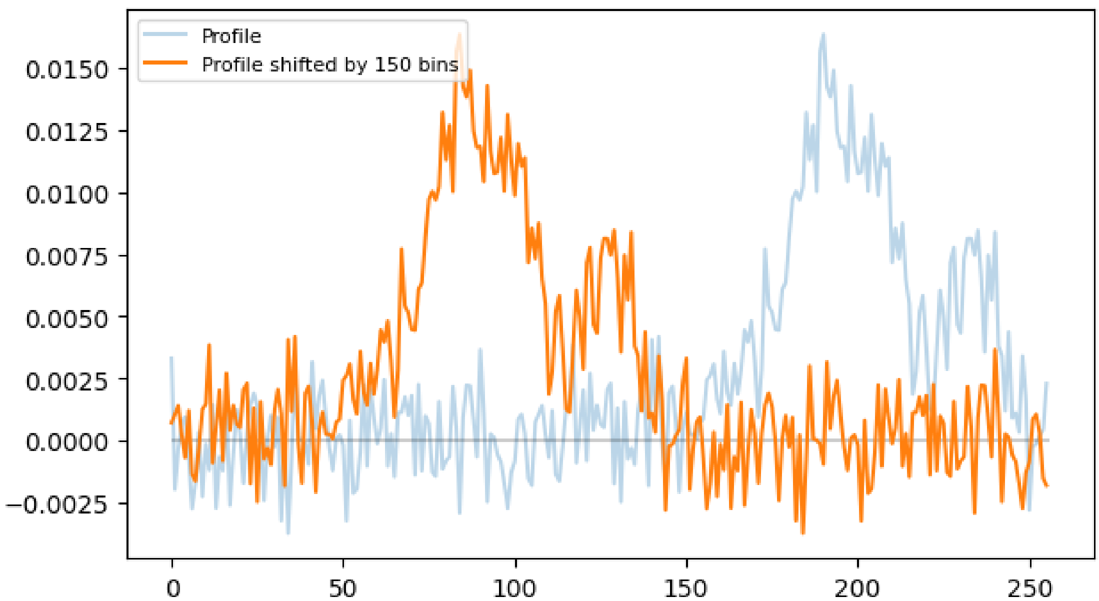
[15]:
```
(912, 4, 256)
```

[16]:
```
obs_duration = arch.integration_length()
num_chan = arch.get_nchan()
num_subint = arch.get_nsubint()
num_bins = arch.get_nbin()
num_pol = arch.get_npol()
pol_state = arch.get_state()

print(f'And has the following characteristics:\n')
print(f'Observation duration of \t\t\t {obs_duration:.2f} seconds')
print(f'Number of frequency channels\t\t\t {num_chan}')
print(f'Number of polarisations \t\t\t {num_pol}')
print(f'In polarisation state \t\t\t\t {pol_state}')
print(f'Number of subintegrations (time blocks)\t\t {num_subint}')
print(f'Each subintegration (time block) is made up of   {num_bins} data bins')
```

```
And has the following characteristics:

Observation duration of                         8826.67 seconds
Number of frequency channels                    1
Number of polarisations                         4
In polarisation state                           Stokes
```

```
Number of subintegrations (time blocks)          912
Each subintegration (time block) is made up of   256 data bins
```

[17]:
```
### Make one plot of subint X to see if we want to center peaks
subint_num = 2

plt.figure(figsize=(7,4))
plt.plot(data_for_profiles[subint_num,0,:], label="Profile", alpha = 0.3)
plt.hlines(0, 0, 256, color = 'black', alpha = 0.2)

roll_with = 150
data_for_profiles_rolled = np.roll(data_for_profiles,roll_with, axis=2) ##note␣
↪axis 2 is the bins axis
plt.plot(data_for_profiles_rolled[subint_num,0,:], label="Profile shifted by %d␣
↪bins" %roll_with)
plt.legend(fontsize=8, loc='best')
plt.show()
```



[18]:
```
cd csvs
```

```
/idia/projects/pulsar-
timing/users/athambiran/MSc_NGC6440A/timing/121mins_10sub/csvs
```

[19]:
```
for subint in range(num_subint):
    df = pd.DataFrame(data_for_profiles_rolled[subint,:,:].
↪T,columns=["I","Q","U","V"]) ##transpose to have I, Q, U, V as columns␣
↪(instead of rows)
```

```
    # df.to_csv("Profile_stokes_%d.csv" %subint, index=False)
```

[20]:
```
np.shape(data_for_profiles_rolled)
```

[20]: (912, 4, 256)

2 Fitting Gaussians

[21]:
```
subints ={}
for i in range(0,num_subint):
    filename = f"Profile_stokes_{i}.csv"
    subints[i] = np.loadtxt(filename, delimiter=",", skiprows=1)
```

[22]:
```
def Gauss(x, a, x0, sigma):
    return a*np.exp(-(x-x0)**2/(2*sigma**2))
```

[23]:
```
x_values = np.arange(1,257)
y_values = {}
popt_dict ={}
pcov_dict = {}
skewness = {}
kurt ={}
fontsize=8.5
sigma ={}
fit_success = {}  # Track which fits succeeded
```

[24]:
```
for i in range(0, num_subint):
    try:
        y_values[i] = subints[i][:, 0]  # Stokes I for subintegration i

        # Optional: skip very noisy or flat signals
        if np.max(y_values[i]) - np.min(y_values[i]) < 1e-3:
            print(f"Skipped {i} due to low signal variation")
            popt_dict[i] = np.zeros(3)  # 3 parameters: amplitude, mean, std
            pcov_dict[i] = np.zeros((3, 3))  # covariance is 3x3 matrix
            fit_success[i] = False
            continue

        # Fit Gaussian
        popt_dict[i], pcov_dict[i] = curve_fit(Gauss, x_values, y_values[i],␣
↪p0=[max(y_values[i]), np.mean(x_values), np.std(x_values)], maxfev = 900)
        skewness[i] = skew(y_values[i])
        kurt[i] = kurtosis(y_values[i])
        fit_success[i] = True
        #print(f"Fit success: {i}")
```

```
    except Exception as e:
        #print(f"Fit failed at {i}: {e}")
        popt_dict[i] = np.zeros(3)
        pcov_dict[i] = np.zeros((3, 3))
        skewness[i] = 0
        kurt[i] = 0
        fit_success[i] = False
```

```
/idia/projects/pulsar-timing/virtual-environment/lib/python3.9/site-
packages/scipy/optimize/_minpack_py.py:833: OptimizeWarning: Covariance of the
parameters could not be estimated
  warnings.warn('Covariance of the parameters could not be estimated',
```

[25]:
```
#first column is subint
#amplitude = popt_dict[i][0]
#std deviation = popt_dict[i][2]
#skewness = skewness[i]
#kurtosis = kurt[i]
#mean = popt_dict[i][1]


Gaussian_data = np.empty((num_subint, 6))

for i in range(0, num_subint):
    Gaussian_data[i][0] = i
    Gaussian_data[i][1]= popt_dict[i][0]
    Gaussian_data[i][2]= popt_dict[i][2]
    Gaussian_data[i][3]= skewness[i]
    Gaussian_data[i][4]= kurt[i]
    Gaussian_data[i][5]= popt_dict[i][1]
```

[26]:
```
amplitudes_G = Gaussian_data[:, 1]
std_devs_G = Gaussian_data[:, 2]
```

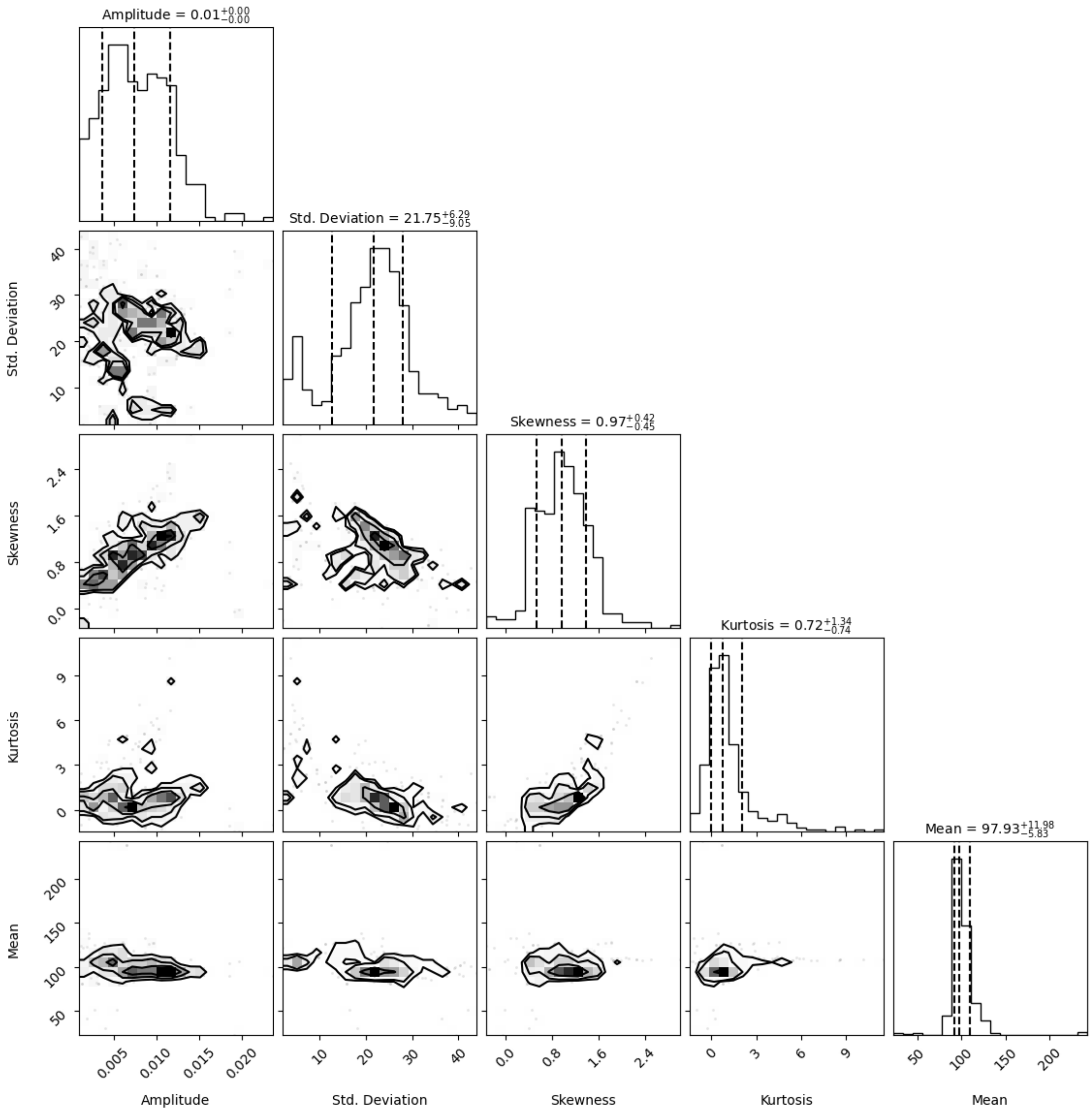
[27]:
```
# Loop through each subint and apply the filters manually
for i in range(0, num_subint):
    amp = Gaussian_data[i][1]
    std = Gaussian_data[i][2]

    # If both amplitude and std dev are 0, reject
    if amp == 0 and std == 0:
        fit_success[i] = False
        continue

    # If amplitude or std dev out of desired range, reject
    if not (0 < amp < 0.025):
        fit_success[i] = False
```

```
        continue

    if not (0 < std < 45):
        fit_success[i] = False
        continue
    #GAUSSIANITY TEST
    data = data_for_profiles_rolled[i, 0, :]
    result=normaltest(data)

    if (result.pvalue >=0.05):
        fit_success[i] = False
        continue


    fit_success[i] = True

#building the filtered_data array using only the successful fits
filtered_data = np.array([Gaussian_data[i] for i in range(num_subint) if␣
 ↪fit_success.get(i, False)])
```

[28]:
```
amplitudes_f = filtered_data[:, 1]
std_devs_f = filtered_data[:, 2]
skewness_f = filtered_data[:,3]
kurtosis_f = filtered_data[:,4]
mean_f = filtered_data[:,5]
```

2.0.1 what is the best combination of 2 parameters?

[29]:
```
# Stack features vertically and transpose to shape (n_samples, n_features)
data = np.vstack([amplitudes_f, std_devs_f, skewness_f, kurtosis_f, mean_f]).T

fig = corner.corner(
    data,
    labels=["Amplitude", "Std. Deviation", "Skewness", "Kurtosis", "Mean"],
    quantiles=[0.16, 0.5, 0.84],
    show_titles=True,
    title_kwargs={"fontsize": 10},
)

#figure.savefig("corner_plot.png", dpi=300, bbox_inches="tight")
```



3 Preparing Residual data

[30]:
```
cd /idia/projects/pulsar-timing/users/athambiran/MSc_NGC6440A/timing/
↪121mins_10sub/
```

/idia/projects/pulsar-timing/users/athambiran/MSc_NGC6440A/timing/121mins_10sub

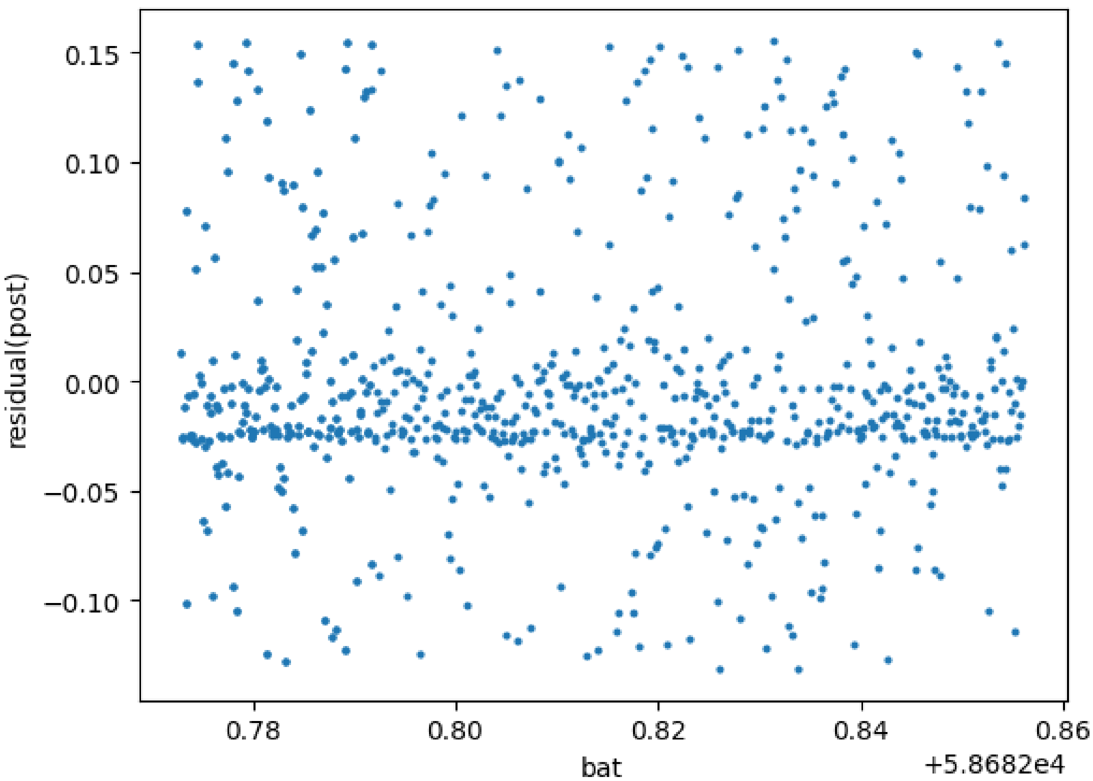
[31]:
```
data = np.loadtxt("final_residuals.txt")
```

[32]:
```
data[0] #sat bat freq residual(post) err(errorbars) snr
```

```
[32]: array([5.86827791e+04, 5.86827729e+04, 1.28307300e+03, 1.29522613e-02,
             1.35958600e+03, 1.58760000e+01])
```

```
[33]: plt.scatter(data[:,1],data[:,3],s=4)
      plt.xlabel('bat')
      plt.ylabel('residual(post)')
```

```
[33]: Text(0, 0.5, 'residual(post)')
```



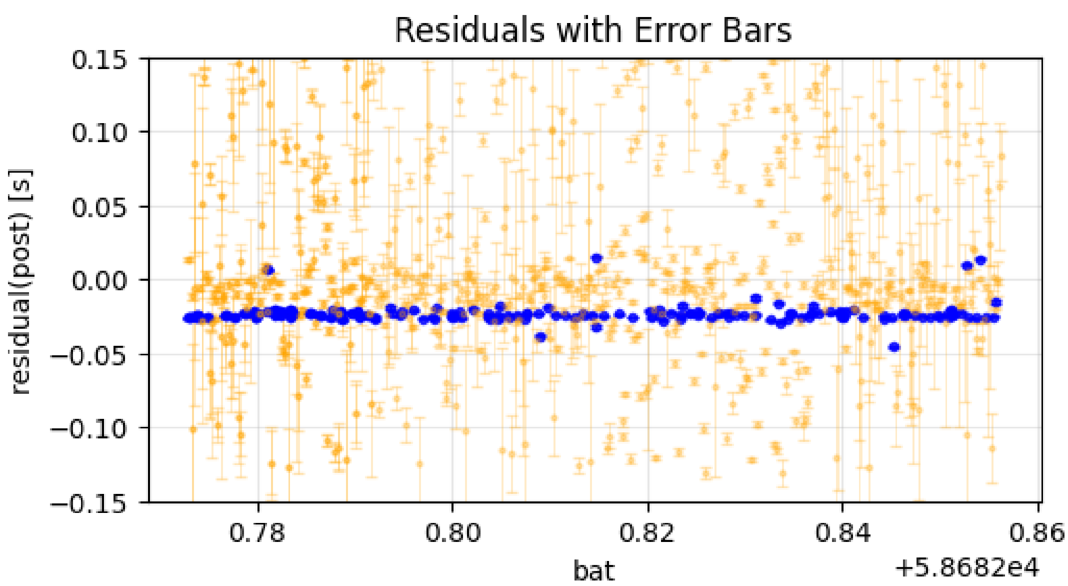

```
[34]: error_bars = data[:,4] * 1e-6
```

```
[35]: np.shape(error_bars)
```

```
[35]: (912,)
```

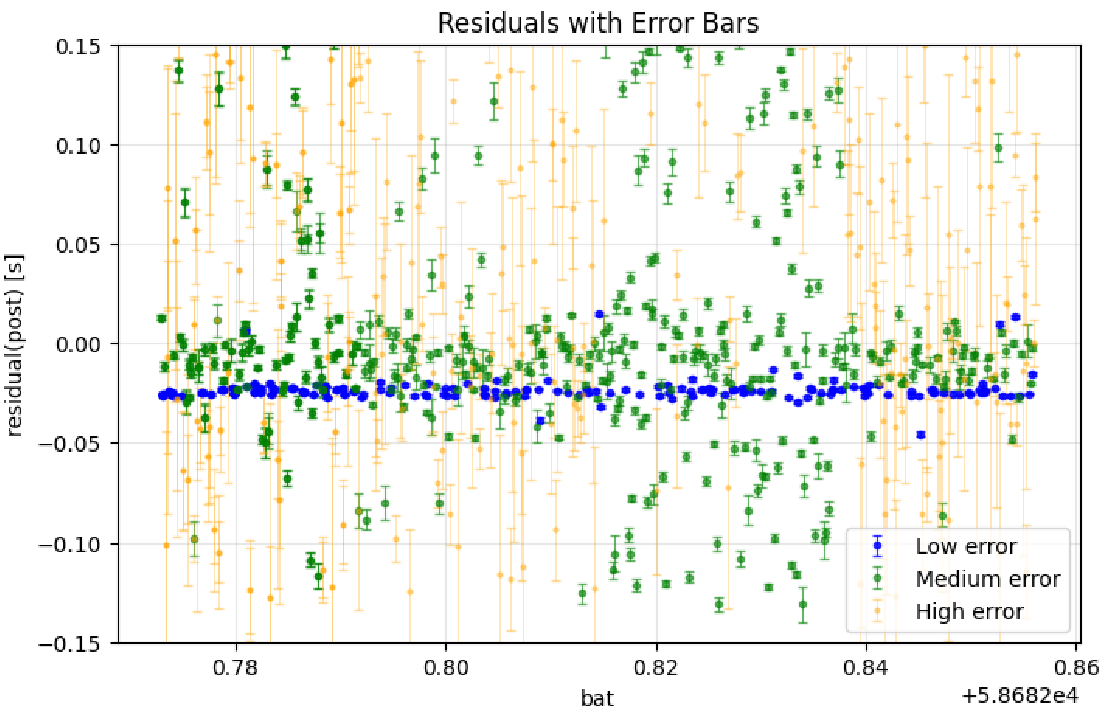
```
[36]: #threshold = np.median(error_bars)
      threshold = 0.001

      small_err_mask = error_bars <= threshold  # 400 μs = 0.0004 s
      large_err_mask = error_bars > threshold

      small_err = data[small_err_mask]
```

```
large_err = data[large_err_mask]
small_err_bars = error_bars[small_err_mask]
large_err_bars = error_bars[large_err_mask]

plt.figure(figsize=(6,3))

# Small errors (blue)
plt.errorbar(small_err[:,1], small_err[:,3],
             yerr=small_err_bars,
             fmt='o', markersize=3, color='blue', alpha=0.8,
             elinewidth=0.8, capsize=2)

# Large errors (orange, faded)
plt.errorbar(large_err[:,1], large_err[:,3],
             yerr=large_err_bars,
             fmt='o', markersize=2, color='orange', alpha=0.3,
             elinewidth=0.5, capsize=2)

plt.xlabel('bat')
plt.ylabel('residual(post) [s]')
plt.title('Residuals with Error Bars')
plt.ylim(-0.15,0.15)
plt.grid(True, alpha=0.3)
plt.show()
```



[37]:
```
# Convert μs to s
error_bars = data[:,4] * 1e-6
```

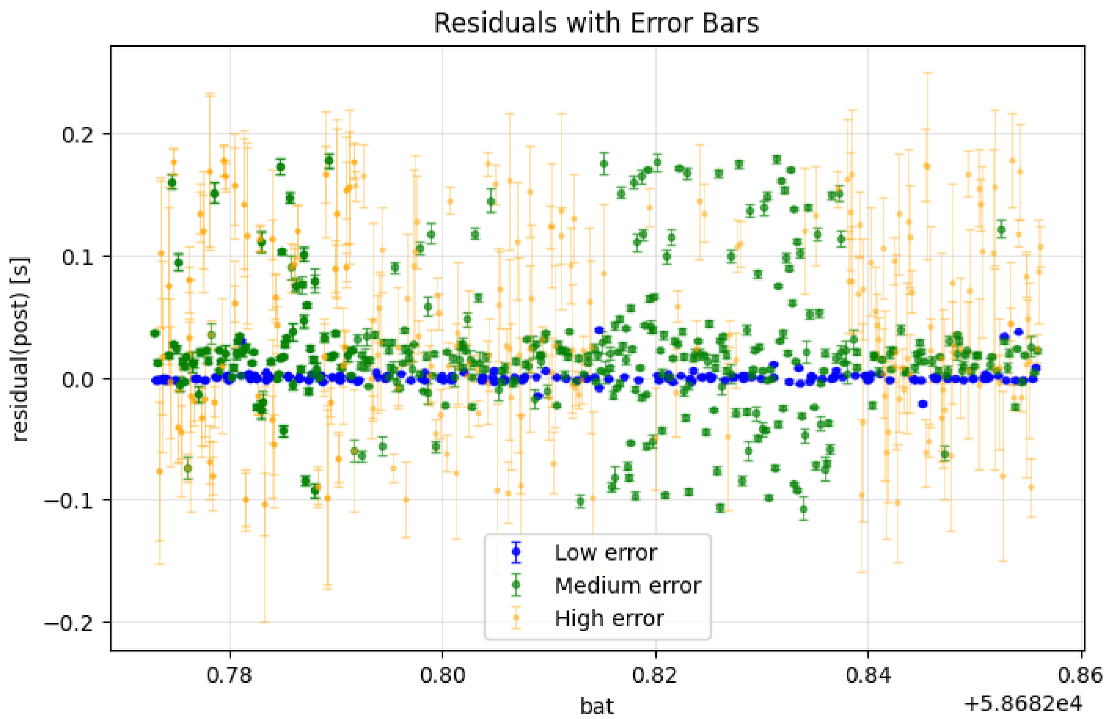
```python
# Thresholds (in seconds)
low_thr = 0.001    # 1 ms
med_thr = 0.01     # 10 ms

# Masks
low_mask = error_bars <= low_thr
med_mask = (error_bars > low_thr) & (error_bars <= med_thr)
high_mask = error_bars > med_thr

# Split data
low_err = data[low_mask]
med_err = data[med_mask]
high_err = data[high_mask]

low_err_bars = error_bars[low_mask]
med_err_bars = error_bars[med_mask]
high_err_bars = error_bars[high_mask]

plt.figure(figsize=(8,5))

# Low errors (blue)
plt.errorbar(low_err[:,1], low_err[:,3],
             yerr=low_err_bars,
             fmt='o', markersize=3, color='blue', alpha=0.8,
             elinewidth=0.8, capsize=2, label="Low error")

# Medium errors (green)
plt.errorbar(med_err[:,1], med_err[:,3],
             yerr=med_err_bars,
             fmt='o', markersize=3, color='green', alpha=0.6,
             elinewidth=0.8, capsize=2, label="Medium error")

# High errors (orange)
plt.errorbar(high_err[:,1], high_err[:,3],
             yerr=high_err_bars,
             fmt='o', markersize=2, color='orange', alpha=0.4,
             elinewidth=0.5, capsize=2, label="High error")

plt.xlabel('bat')
plt.ylabel('residual(post) [s]')
plt.title('Residuals with Error Bars')
plt.ylim(-0.15, 0.15)
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```



[38]:

```
# Convert µs to s
error_bars = data[:,4] * 1e-6

# Thresholds (in seconds)
low_thr = 0.001     # 1 ms
med_thr = 0.01      # 10 ms

# Masks
low_mask = error_bars <= low_thr
med_mask = (error_bars > low_thr) & (error_bars <= med_thr)
high_mask = error_bars > med_thr

# Split data
low_err = data[low_mask]
med_err = data[med_mask]
high_err = data[high_mask]

low_err_bars = error_bars[low_mask]
med_err_bars = error_bars[med_mask]
high_err_bars = error_bars[high_mask]

plt.figure(figsize=(8,5))
```

```
median=np.median(low_err[:,3])

# Low errors (blue)
plt.errorbar(low_err[:,1], low_err[:,3]-median,
             yerr=low_err_bars,
             fmt='o', markersize=3, color='blue', alpha=0.8,
             elinewidth=0.8, capsize=2, label="Low error")

# Medium errors (green)
plt.errorbar(med_err[:,1], med_err[:,3]-median,
             yerr=med_err_bars,
             fmt='o', markersize=3, color='green', alpha=0.6,
             elinewidth=0.8, capsize=2, label="Medium error")

# High errors (orange)
plt.errorbar(high_err[:,1], high_err[:,3]-median,
             yerr=high_err_bars,
             fmt='o', markersize=2, color='orange', alpha=0.4,
             elinewidth=0.5, capsize=2, label="High error")

plt.xlabel('bat')
plt.ylabel('residual(post) [s]')
plt.title('Residuals with Error Bars')
#plt.ylim(-0.15, 0.15)
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```



[39]:
```
median
```

[39]: -0.02414564583413968

3.1 bokeh with just residual data

[42]:
```
images = []
```

[43]:
```
bats_all = data[:,1]
residuals = data[:,3]
residual_errs = error_bars
```

[44]:
```
# Assign colors based on error size
colors = []
for err in error_bars:
    if err <= low_thr:
        colors.append("blue")
    elif err <= med_thr:
        colors.append("green")
    else:
        colors.append("orange")
```

[45]:
```python
for subint in range(num_subint):
    intensities = data_for_profiles_rolled[subint, 0, :]

    #matplotlib figure of the profile
    fig, ax = plt.subplots()
    ax.plot(intensities, color='blue',alpha=0.7)
    ax.set_title(f"Subint {subint}")
    #ax.axis("off")  # Hide axes for cleaner display

    buf = BytesIO()
    plt.savefig(buf, format="png", bbox_inches='tight')
    plt.close(fig)

    encoded = base64.b64encode(buf.getvalue()).decode()
    images.append(f"<img src='data:image/png;base64,{encoded}' width='300'>")

print(np.shape(images))

output_notebook()

#Bokeh data source
source = ColumnDataSource(data=dict(
    bat=bats_all,
    residual=residuals,
    residual_err=residual_errs,
    upper=residuals + residual_errs,
    lower=residuals - residual_errs,
    #subint=subints_list.astype(str),
    image=images,
    color=colors
))

p = figure(title="BAT vs Residual", width=800, height=600,
           x_axis_label="BAT", y_axis_label="Residual",
           tools="pan,wheel_zoom,reset")

p.scatter("bat", "residual", size=8, color="color", alpha=0.5, source=source)

# Add error bars using Whisker
whisker = Whisker(source=source, base="bat", upper="upper", lower="lower")
whisker.upper_head.size = 10  # optional: adjust arrowhead size
whisker.lower_head.size = 10
p.add_layout(whisker)

#image of profiles when hovering
hover = HoverTool(tooltips="""
    <div>
```

```
        <span style="font-size: 12px;">abc
        @image
    </div>
""")

#box = BoxZoomTool()
p.add_tools(hover)

show(p)
```

```
(912,)
```

[46]:
```
images = []
for subint in range(num_subint):
    intensities = data_for_profiles_rolled[subint, 0, :]

    #matplotlib figure of the profile
    fig, ax = plt.subplots()
    ax.plot(intensities, color='blue',alpha=0.7)
    ax.set_title(f"Subint {subint}")

    buf = BytesIO()
    plt.savefig(buf, format="png", bbox_inches='tight')
    plt.close(fig)

    encoded = base64.b64encode(buf.getvalue()).decode()
    images.append(f"<img src='data:image/png;base64,{encoded}' width='300'>")

print(np.shape(images))

output_notebook()

upper = residuals + error_bars
lower = residuals - error_bars

low_mask = error_bars <= low_thr
med_mask = (error_bars > low_thr) & (error_bars <= med_thr)
high_mask = error_bars > med_thr

low_src = ColumnDataSource(data=dict(
    bat=data[low_mask, 1],
    residual=residuals[low_mask],
    upper=upper[low_mask],
    lower=lower[low_mask],
    image=np.array(images)[low_mask]))

med_src = ColumnDataSource(data=dict(
```

```python
    bat=data[med_mask, 1],
    residual=residuals[med_mask],
    upper=upper[med_mask],
    lower=lower[med_mask],
    image=np.array(images)[med_mask]))

high_src = ColumnDataSource(data=dict(
    bat=data[high_mask, 1],
    residual=residuals[high_mask],
    upper=upper[high_mask],
    lower=lower[high_mask],
    image=np.array(images)[high_mask]))


p = figure(title="BAT vs Residual", width=800, height=600,
           x_axis_label="BAT", y_axis_label="Residual",
           tools="pan,wheel_zoom,reset")

p.scatter("bat", "residual", size=8, color="navy", alpha=0.6, source=low_src)
p.scatter("bat", "residual", size=8, color="#8856a7", alpha=0.8, source=med_src)
p.scatter("bat", "residual", size=8, color="lightsteelblue", alpha=0.8,␣
 ↪source=high_src)

low_whisker = Whisker(source=low_src, base="bat", upper="upper",␣
 ↪lower="lower",line_color="navy", line_alpha=0.8)
low_whisker.upper_head.line_color = "navy"
low_whisker.upper_head.line_alpha = 0.8
low_whisker.lower_head.line_color = "navy"
low_whisker.lower_head.line_alpha = 0.8
p.add_layout(low_whisker)

med_whisker = Whisker(source=med_src, base="bat", upper="upper",␣
 ↪lower="lower",line_color="#8856a7", line_alpha=0.8)
med_whisker.upper_head.line_color = "#8856a7"
med_whisker.upper_head.line_alpha = 0.8
med_whisker.lower_head.line_color = "#8856a7"
med_whisker.lower_head.line_alpha = 0.8
p.add_layout(med_whisker)

high_whisker = Whisker(source=high_src, base="bat", upper="upper",␣
 ↪lower="lower",line_color="lightsteelblue", line_alpha=0.8)
high_whisker.upper_head.line_color = "lightsteelblue"
high_whisker.upper_head.line_alpha = 0.8
high_whisker.lower_head.line_color = "lightsteelblue"
high_whisker.lower_head.line_alpha = 0.8
p.add_layout(high_whisker)
```

```
#image of profiles when hovering
hover = HoverTool(tooltips="""
    <div>
        <span style="font-size: 12px;">abc
        @image
    </div>
""")

p.add_tools(hover)

show(p)
```

```
(912,)
```

3.2 dbscan with just residual data

[47]:
```
residual_data = np.column_stack((data[:, 1], data[:, 3], error_bars, data[:
↪,5])) #bat, residual, errors, snr
data_for_dbscan = residual_data[:, 1:] #cluster on everything except bat
```

[48]:
```
residual_data[0]
```

[48]:
```
array([5.86827729e+04, 1.29522613e-02, 1.35958600e-03, 1.58760000e+01])
```

[49]:
```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_for_dbscan)
```

[ ]:
```

```

[ ]:
```

```

4 dbscan with residuals, errors, snr, std deviation, amplitude

[50]:
```
subints_array = np.arange(0, num_subint)
```

[51]:
```
residual_data = np.column_stack((data[:, 1], data[:, 3], data[:,5]))  #bat,␣
↪residual, snr
#gaussian_data = np.column_stack((subints_array, amplitudes_G, std_devs_G))  ␣
↪#subint, amplitudes, std dev
gaussian_data = np.column_stack((amplitudes_G, std_devs_G))   #amplitudes, std␣
↪dev
```

[52]:
```
resid_gauss_data = np.column_stack((residual_data, gaussian_data))
data_for_dbscan = resid_gauss_data[:, :] #cluster on everything except BAT
#dbscan = DBSCAN(eps=0.1, min_samples=5)
```

[53]:
```
np.shape(data_for_dbscan)
```

[53]: (912, 5)

[54]:
```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_for_dbscan)
```

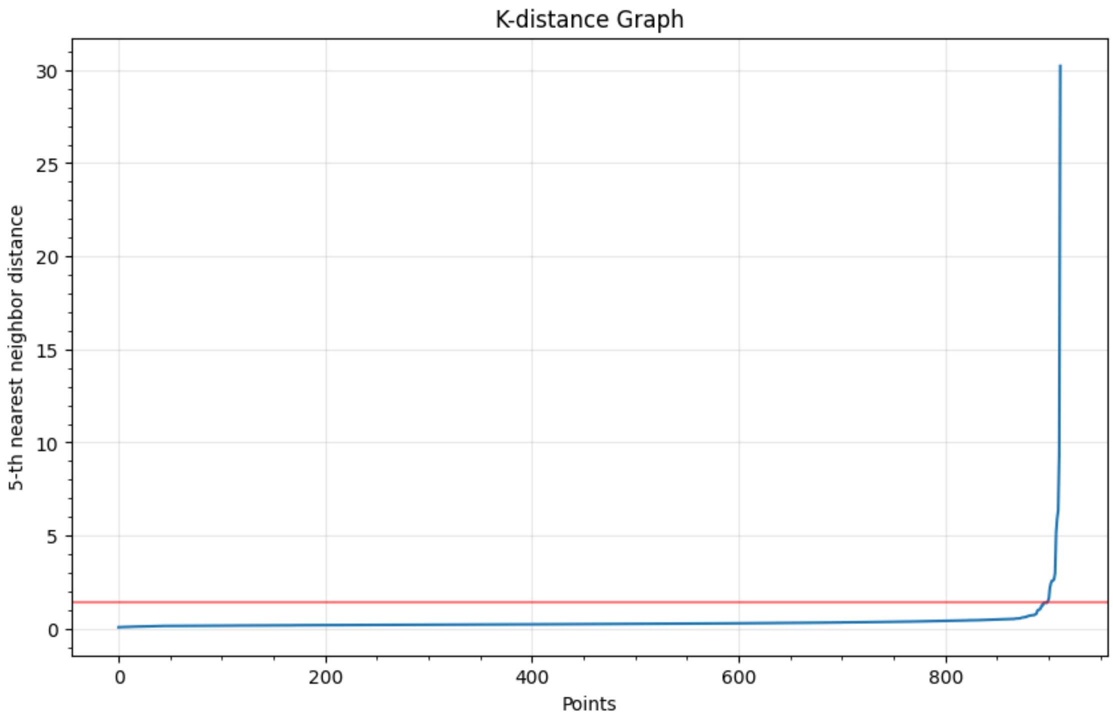
[55]:
```
np.shape(scaled_data)
```

[55]: (912, 5)

[56]:
```
scaled_data[0,:]
```

[56]: array([-1.41974921, 0.2170863 , 0.16221163, -0.03247371, -0.1219917])

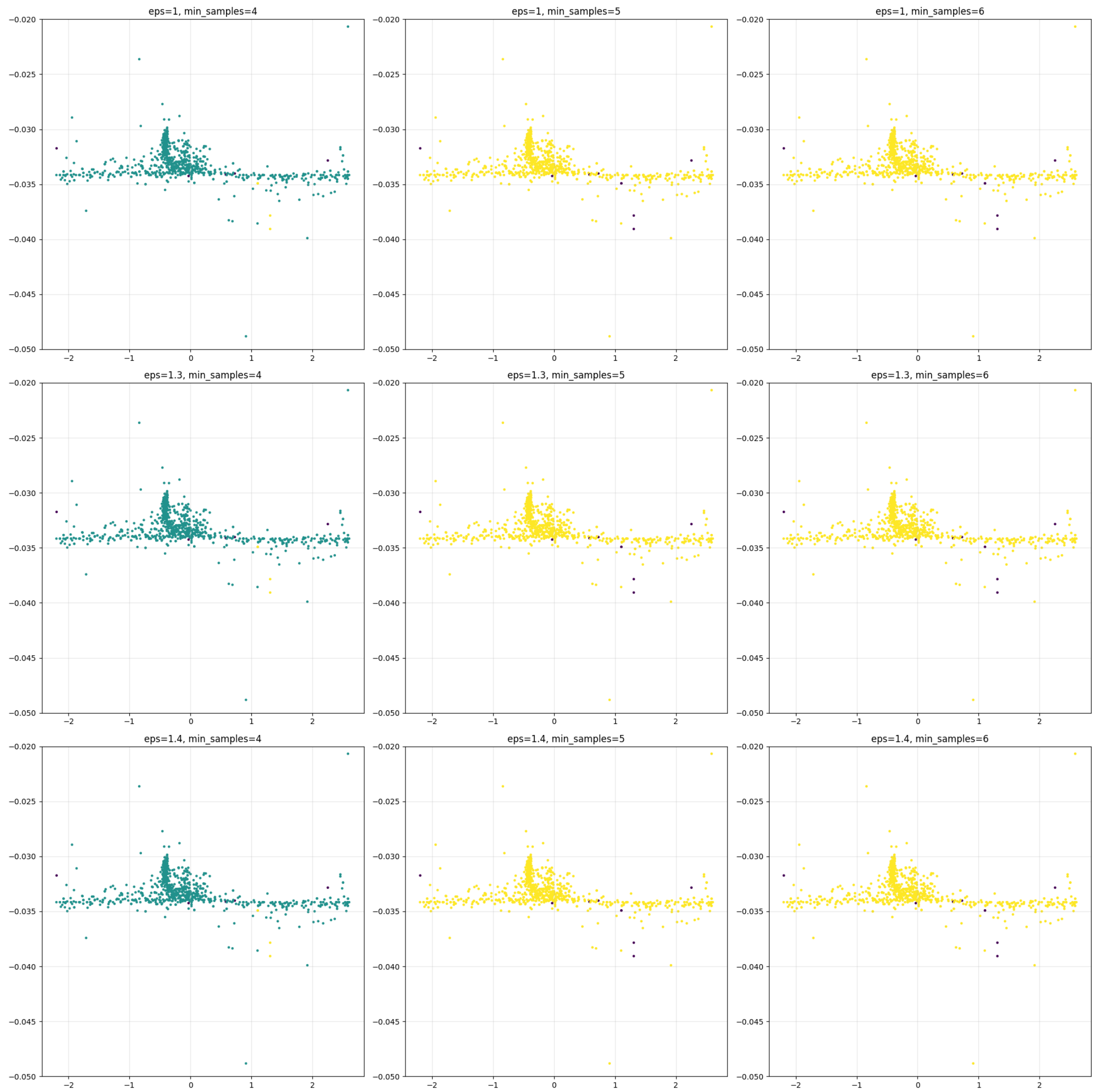
[57]:
```
# Function to plot k-distance graph
y=2.5


def plot_k_distance_graph(X, k):
    neigh = NearestNeighbors(n_neighbors=k)
    neigh.fit(X)
    distances, _ = neigh.kneighbors(X)
    distances = np.sort(distances[:, k-1])
    plt.figure(figsize=(10, 6))
    plt.plot(distances)
    plt.xlabel('Points')
    plt.ylabel(f'{k}-th nearest neighbor distance')
    plt.title('K-distance Graph')
    plt.grid(True, alpha=0.3)
    plt.minorticks_on()
    plt.axhline(1.4, color='r', alpha=0.5)
    plt.show()
# Plot k-distance graph
plot_k_distance_graph(scaled_data, k=5)
```



so eps should be 1.4 and now trying a variety of min_samples

[65]:
```
fig, ax = plt.subplots(3, 3, figsize=(20, 20))

eps_values = [1, 1.3, 1.4]
min_samples_values = [4, 5, 6]

i = 0  #row
j = 0  #col

for eps in eps_values:
    for min_samples in min_samples_values:
        if i >= 3:  #stop if rows exceed subplot rows
            break
        ax[i][j].scatter(scaled_data[:, 1], scaled_data[:, 3], s=5,␣
↪c=DBSCAN(eps=eps, min_samples=min_samples).fit_predict(scaled_data[:, 1:]),␣
↪cmap='viridis')
        ax[i][j].set_title(f"eps={eps}, min_samples={min_samples}")
        ax[i][j].grid(True, alpha=0.3)
        ax[i][j].set_ylim(-0.05,-0.02)

        j += 1
        if j == 3:  # if column index reaches 3, reset and go to next row
            j = 0
```

```
            i += 1

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

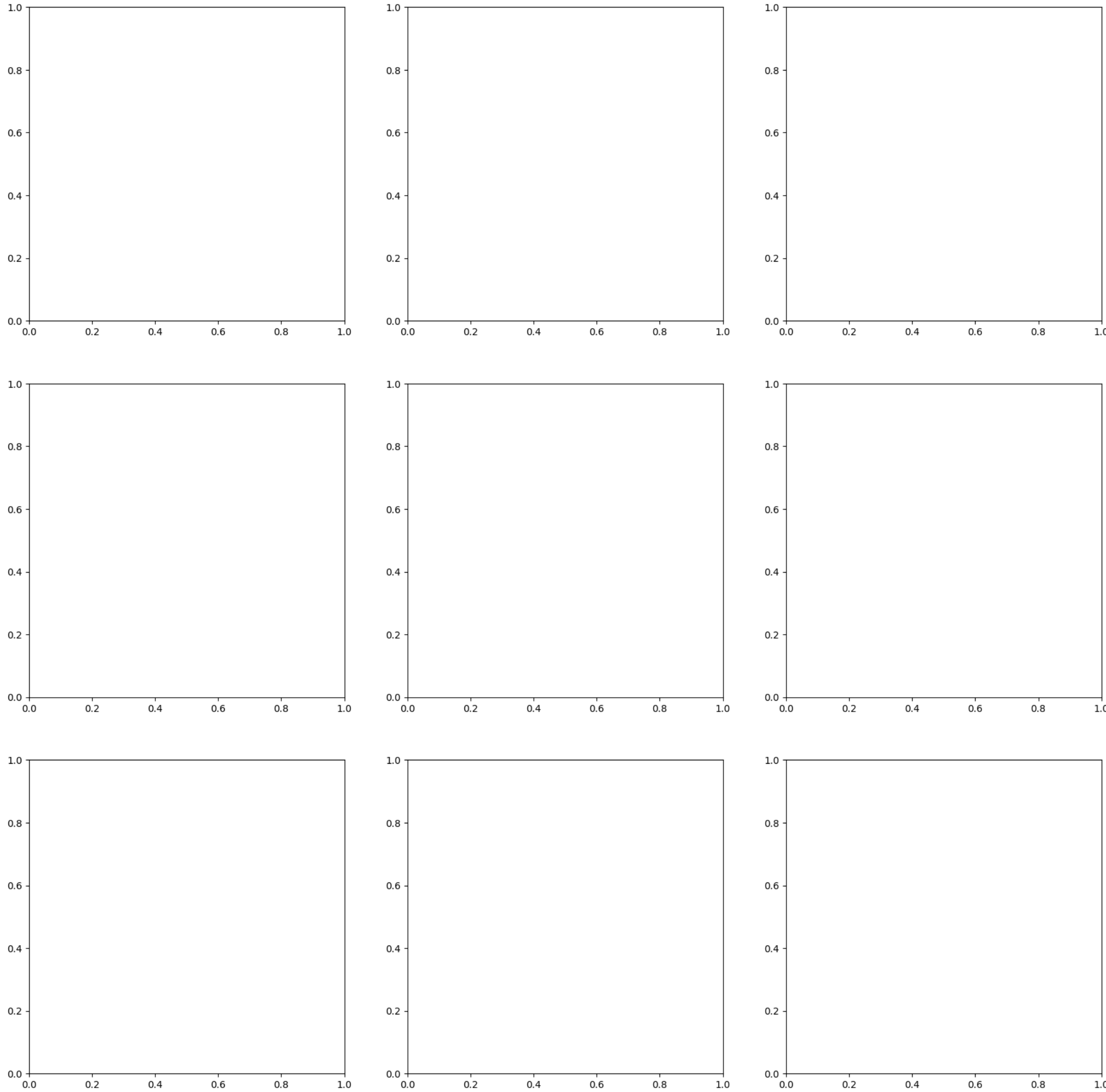


[ ]:

[59]:
```
fig, ax = plt.subplots(3,3, figsize=(20,20))

ax = ax.flatten()
i=0
j=0
```

```python
for eps in [1,2, 1.3, 1.4]:
    for min_samples in [4, 5, 6]:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(scaled_data[:, 1:])
        ax[i][j].scatter(data[:, 1], data[:, 3], s=5, c=labels, cmap='viridis')
        #plt.scatter(data[:, 1], data[:, 3], s=5, c=labels, cmap='viridis')
       # plt.title(f"eps={eps}, min_samples={min_samples}")
        #plt.show()
        i=i+1
        print(i)
    j=j+1
    print(j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[59], line 11
      9   dbscan = DBSCAN(eps=eps, min_samples=min_samples)
     10   labels = dbscan.fit_predict(scaled_data[:, 1:])
---> 11   ax[i][j].scatter(data[:, 1], data[:, 3], s=5, c=labels, cmap='viridis'
     12   #plt.scatter(data[:, 1], data[:, 3], s=5, c=labels, cmap='viridis')
     13 # plt.title(f"eps={eps}, min_samples={min_samples}")
     14   #plt.show()
     15   i=i+1

TypeError: 'AxesSubplot' object is not subscriptable
```



[ ]:

[ ]:

[ ]: