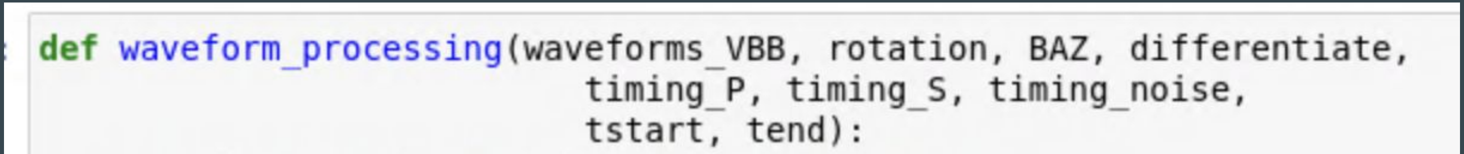
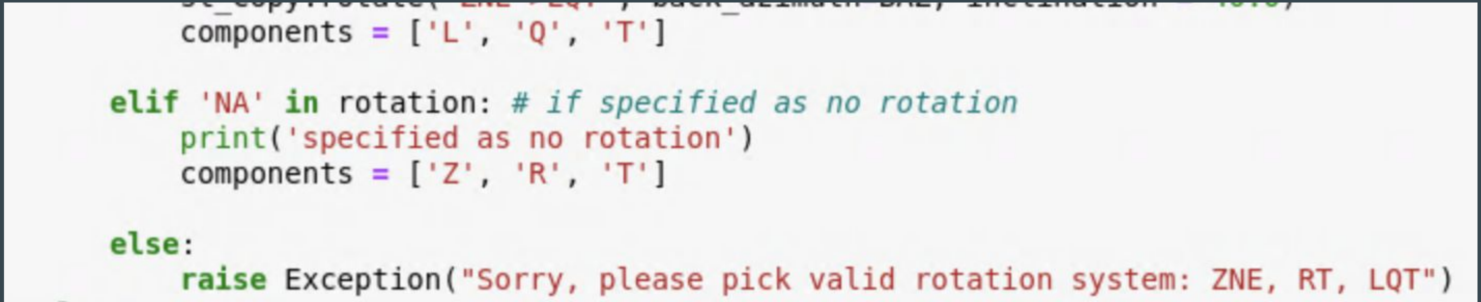
Generalize Polarization Analysis for Saving KDE Plots

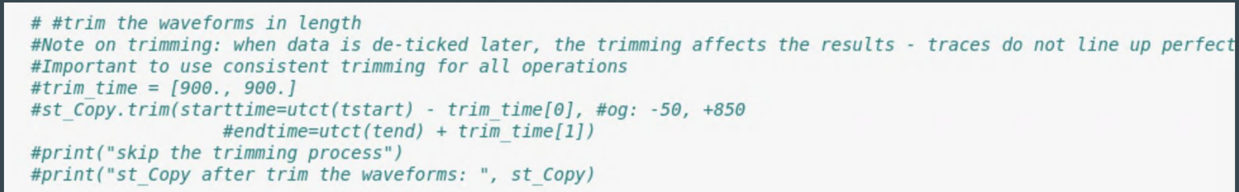
Modified wavefrom_processing function



```
def waveform_processing(waveforms_VBB, rotation, BAZ, differentiate,
                        timing_P, timing_S, timing_noise,
                        tstart, tend):
```



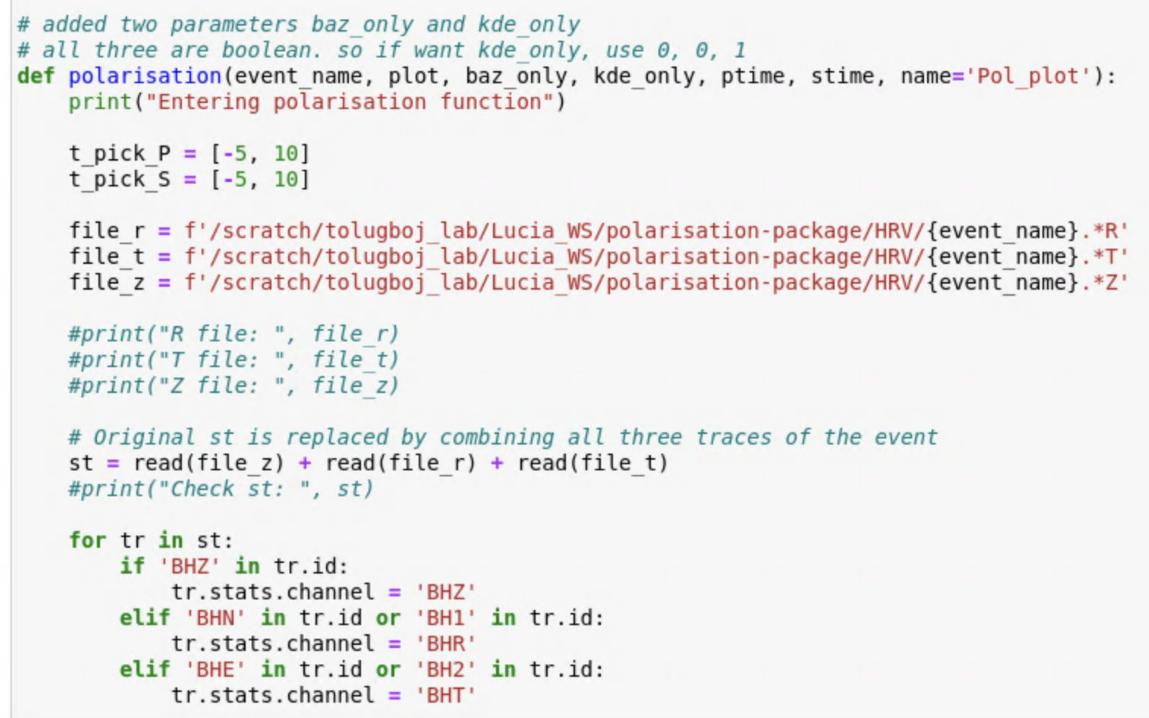
```
        components = ['L', 'Q', 'T']

    elif 'NA' in rotation: # if specified as no rotation
        print('specified as no rotation')
        components = ['Z', 'R', 'T']

    else:
        raise Exception("Sorry, please pick valid rotation system: ZNE, RT, LQT")
```



```
# #trim the waveforms in length
#Note on trimming: when data is de-ticked later, the trimming affects the results - traces do not line up perfect
#Important to use consistent trimming for all operations
#trim_time = [900., 900.]
#st_Copy.trim(starttime=utct(tstart) - trim_time[0], #og: -50, +850
             #endtime=utct(tend) + trim_time[1])
#print("skip the trimming process")
#print("st_Copy after trim the waveforms: ", st_Copy)
```

‘NRT’ has overlap with ‘RT’ so I decided to specify as ‘NA’ for no rotation

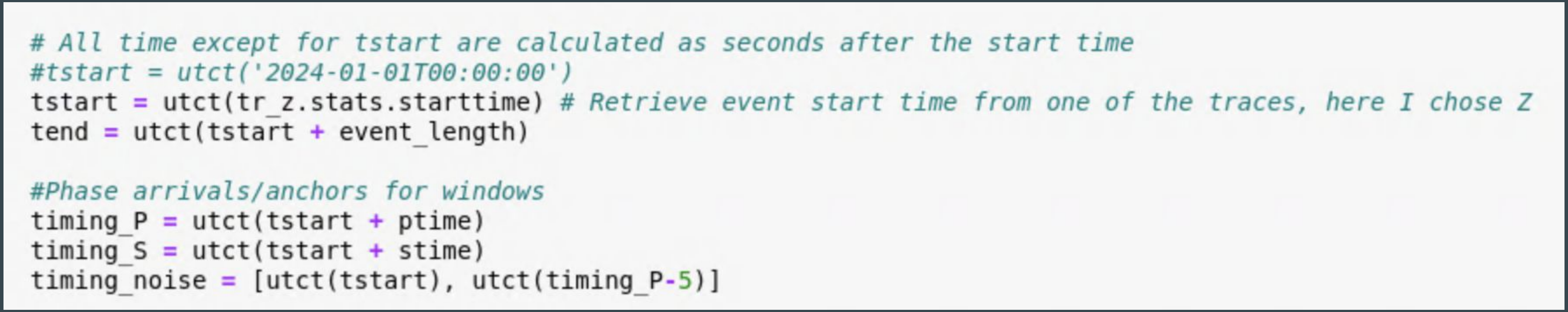
Search for corresponding files in HRV (ending with RTZ)



```python
# added two parameters baz_only and kde_only
# all three are boolean. so if want kde_only, use 0, 0, 1
def polarisation(event_name, plot, baz_only, kde_only, ptime, stime, name='Pol_plot'):
    print("Entering polarisation function")

    t_pick_P = [-5, 10]
    t_pick_S = [-5, 10]

    file_r = f'/scratch/tolugboj_lab/Lucia_WS/polarisation-package/HRV/{event_name}.*R'
    file_t = f'/scratch/tolugboj_lab/Lucia_WS/polarisation-package/HRV/{event_name}.*T'
    file_z = f'/scratch/tolugboj_lab/Lucia_WS/polarisation-package/HRV/{event_name}.*Z'

    #print("R file: ", file_r)
    #print("T file: ", file_t)
    #print("Z file: ", file_z)

    # Original st is replaced by combining all three traces of the event
    st = read(file_z) + read(file_r) + read(file_t)
    #print("Check st: ", st)

    for tr in st:
        if 'BHZ' in tr.id:
            tr.stats.channel = 'BHZ'
        elif 'BHN' in tr.id or 'BH1' in tr.id:
            tr.stats.channel = 'BHR'
        elif 'BHE' in tr.id or 'BH2' in tr.id:
            tr.stats.channel = 'BHT'
```

The files have been rotated to RTZ, while the filenames are correctly in RTZ, the traces info are still in original form. The bottom section of the screenshots bypass this problem by changing them to correct trace channel after rotation.

Set time (Still in polarisation function)

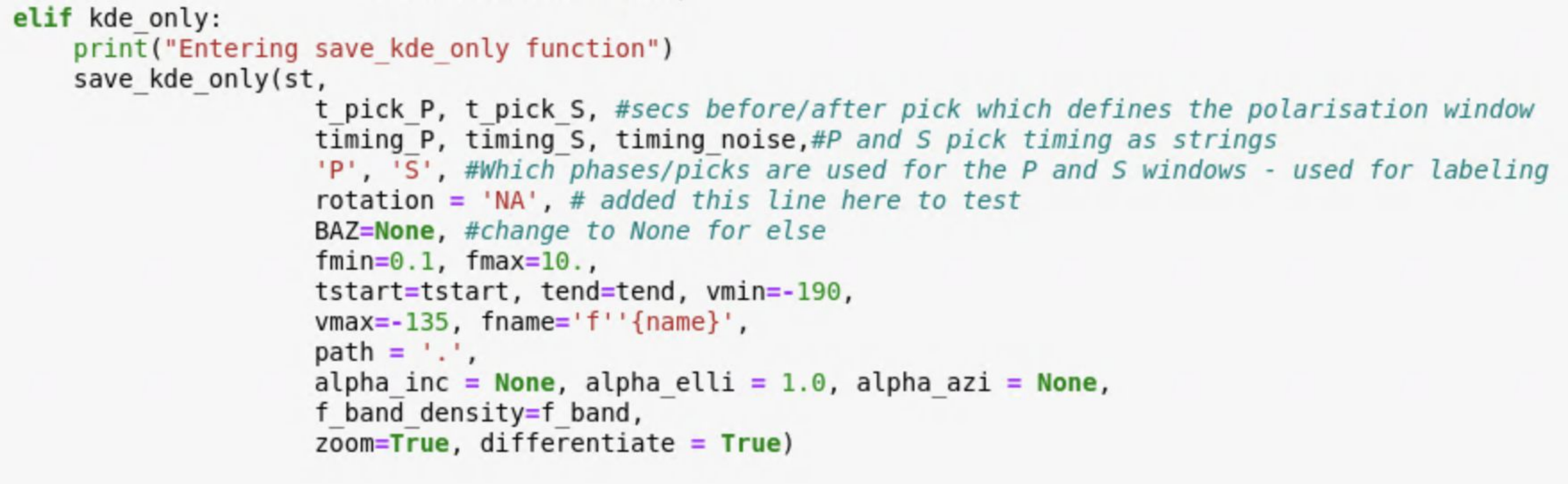


```
# All time except for tstart are calculated as seconds after the start time
#tstart = utct('2024-01-01T00:00:00')
tstart = utct(tr_z.stats.starttime) # Retrieve event start time from one of the traces, here I chose Z
tend = utct(tstart + event_length)

#Phase arrivals/anchors for windows
timing_P = utct(tstart + ptime)
timing_S = utct(tstart + stime)
timing_noise = [utct(tstart), utct(timing_P-5)]
```

Actual event start time is needed for correctly processing data

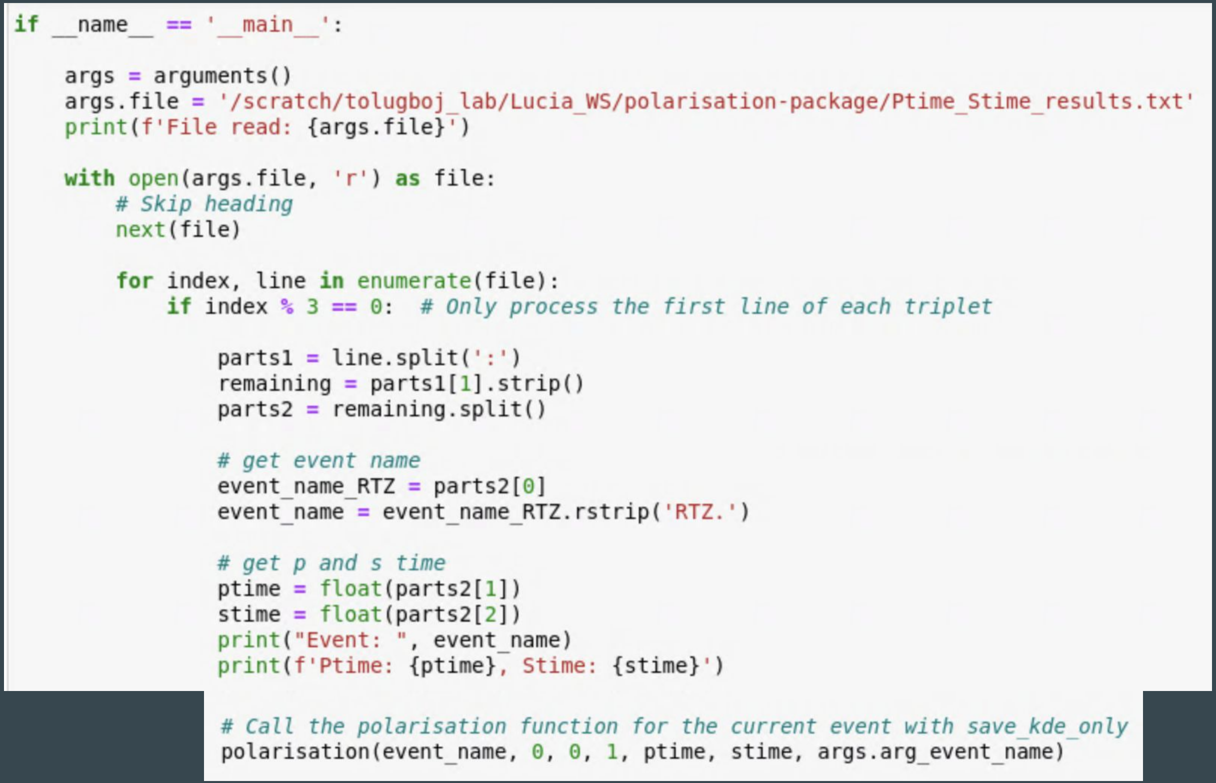
New option in polarisation function

All parameters remain unchanged compared to the plotting option.



```
elif kde_only:
    print("Entering save_kde_only function")
    save_kde_only(st,
                    t_pick_P, t_pick_S, #secs before/after pick which defines the polarisation window
                    timing_P, timing_S, timing_noise,#P and S pick timing as strings
                    'P', 'S', #Which phases/picks are used for the P and S windows - used for labeling
                    rotation = 'NA', # added this line here to test
                    BAZ=None, #change to None for else
                    fmin=0.1, fmax=10.,
                    tstart=tstart, tend=tend, vmin=-190,
                    vmax=-135, fname='f''{name}',
                    path = '.',
                    alpha_inc = None, alpha_elli = 1.0, alpha_azi = None,
                    f_band_density=f_band,
                    zoom=True, differentiate = True)
```

Read file and Call polarisation function



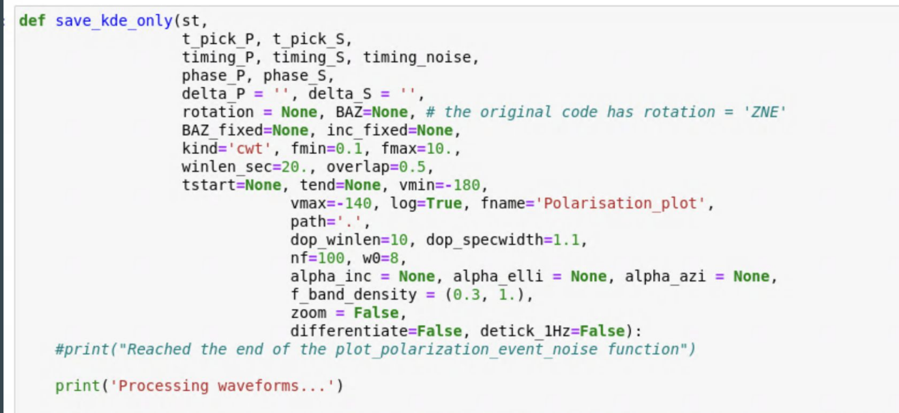
```python
if __name__ == '__main__':

    args = arguments()
    args.file = '/scratch/tolugboj_lab/Lucia_WS/polarisation-package/Ptime_Stime_results.txt'
    print(f'File read: {args.file}')

    with open(args.file, 'r') as file:
        # Skip heading
        next(file)

        for index, line in enumerate(file):
            if index % 3 == 0:  # Only process the first line of each triplet

                parts1 = line.split(':')
                remaining = parts1[1].strip()
                parts2 = remaining.split()

                # get event name
                event_name_RTZ = parts2[0]
                event_name = event_name_RTZ.rstrip('RTZ.')

                # get p and s time
                ptime = float(parts2[1])
                stime = float(parts2[2])
                print("Event: ", event_name)
                print(f'Ptime: {ptime}, Stime: {stime}')

                # Call the polarisation function for the current event with save_kde_only
                polarisation(event_name, 0, 0, 1, ptime, stime, args.arg_event_name)
```

Reads the text file with three parts: event name (with R, T, Z), ptime in seconds, stime in seconds.

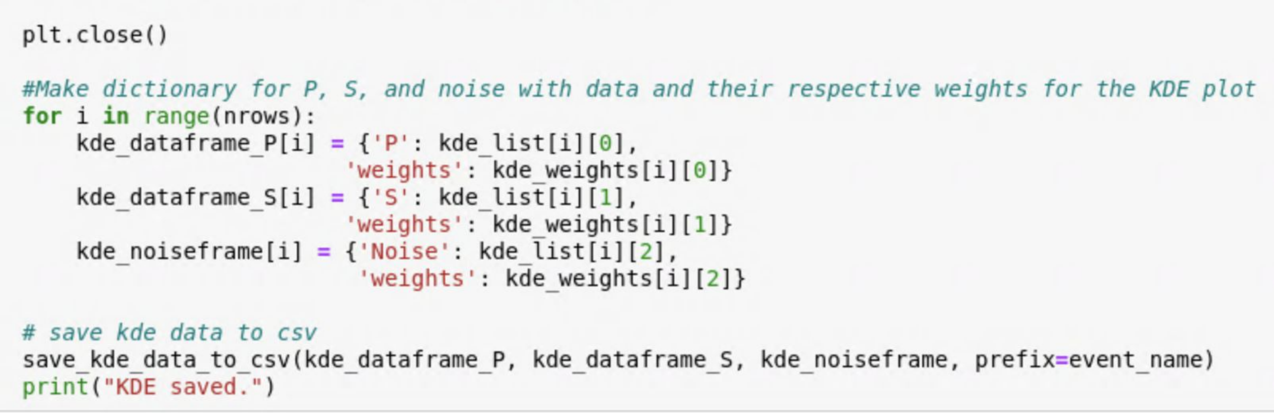
Read one line for every triplets to obtain event name, ptime ,and stime.

Call the polarisation function for current event. (0, 0, 1) indicates save_kde_only.

Save_kde_only function



```
def save_kde_only(st,
                  t_pick_P, t_pick_S,
                  timing_P, timing_S, timing_noise,
                  phase_P, phase_S,
                  delta_P = '', delta_S = '',
                  rotation = None, BAZ=None, # the original code has rotation = 'ZNE'
                  BAZ_fixed=None, inc_fixed=None,
                  kind='cwt', fmin=0.1, fmax=10.,
                  winlen_sec=20., overlap=0.5,
                  tstart=None, tend=None, vmin=-180,
                              vmax=-140, log=True, fname='Polarisation_plot',
                              path='.',
                              dop_winlen=10, dop_specwidth=1.1,
                              nf=100, w0=8,
                              alpha_inc = None, alpha_elli = None, alpha_azi = None,
                              f_band_density = (0.3, 1.),
                              zoom = False,
                              differentiate=False, detick_1Hz=False):
    #print("Reached the end of the plot_polarization_event_noise function")

    print('Processing waveforms...')
```



```
plt.close()

#Make dictionary for P, S, and noise with data and their respective weights for the KDE plot
for i in range(nrows):
    kde_dataframe_P[i] = {'P': kde_list[i][0],
                        'weights': kde_weights[i][0]}
    kde_dataframe_S[i] = {'S': kde_list[i][1],
                        'weights': kde_weights[i][1]}
    kde_noiseframe[i] = {'Noise': kde_list[i][2],
                         'weights': kde_weights[i][2]}

# save kde data to csv
save_kde_data_to_csv(kde_dataframe_P, kde_dataframe_S, kde_noiseframe, prefix=event_name)
print("KDE saved.")
```

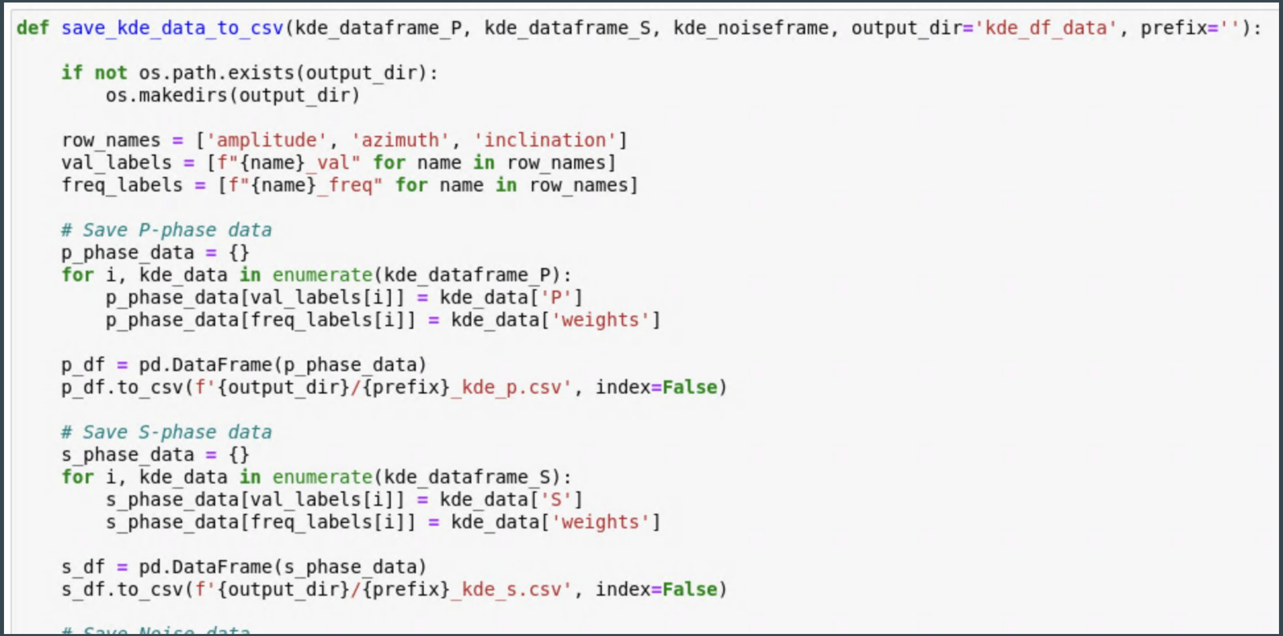
Adjusted based on plot_polarization_event_noise function

Does not generate a plot.

Everything after kde data collection are deleted.

All structures remained the same.

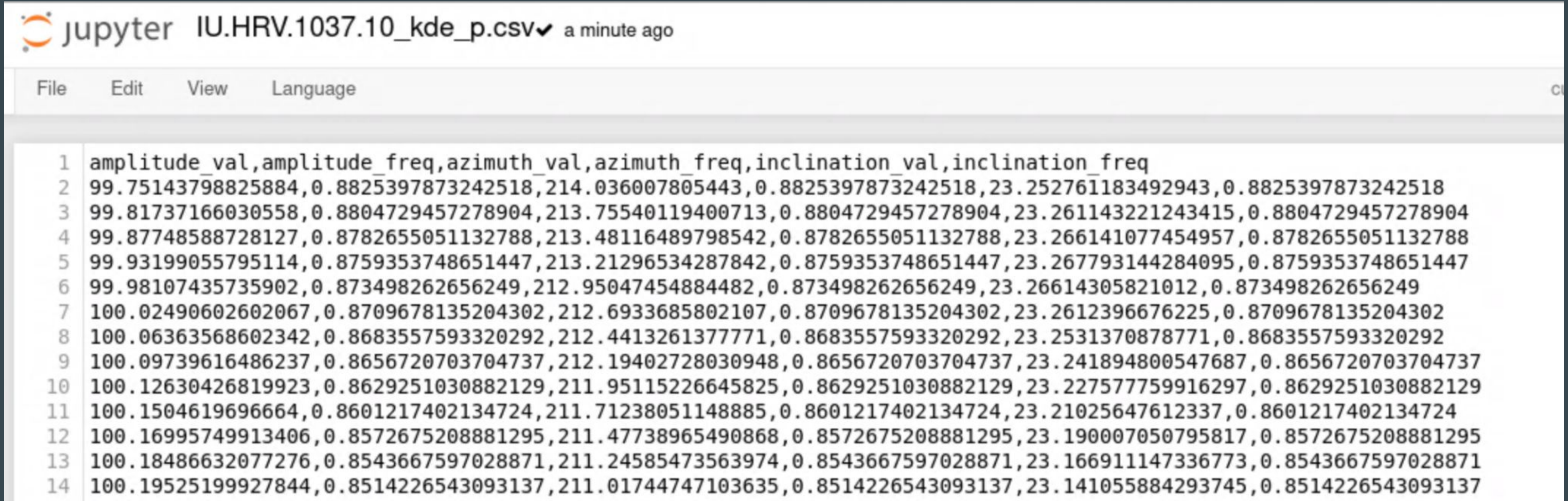
Save_kde_data_to_csv function



```python
def save_kde_data_to_csv(kde_dataframe_P, kde_dataframe_S, kde_noiseframe, output_dir='kde_df_data', prefix=''):

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    row_names = ['amplitude', 'azimuth', 'inclination']
    val_labels = [f"{name}_val" for name in row_names]
    freq_labels = [f"{name}_freq" for name in row_names]

    # Save P-phase data
    p_phase_data = {}
    for i, kde_data in enumerate(kde_dataframe_P):
        p_phase_data[val_labels[i]] = kde_data['P']
        p_phase_data[freq_labels[i]] = kde_data['weights']

    p_df = pd.DataFrame(p_phase_data)
    p_df.to_csv(f'{output_dir}/{prefix}_kde_p.csv', index=False)

    # Save S-phase data
    s_phase_data = {}
    for i, kde_data in enumerate(kde_dataframe_S):
        s_phase_data[val_labels[i]] = kde_data['S']
        s_phase_data[freq_labels[i]] = kde_data['weights']

    s_df = pd.DataFrame(s_phase_data)
    s_df.to_csv(f'{output_dir}/{prefix}_kde_s.csv', index=False)

    # Save Noise data
```

For each event, generates three files corresponding to P, S, and noise.

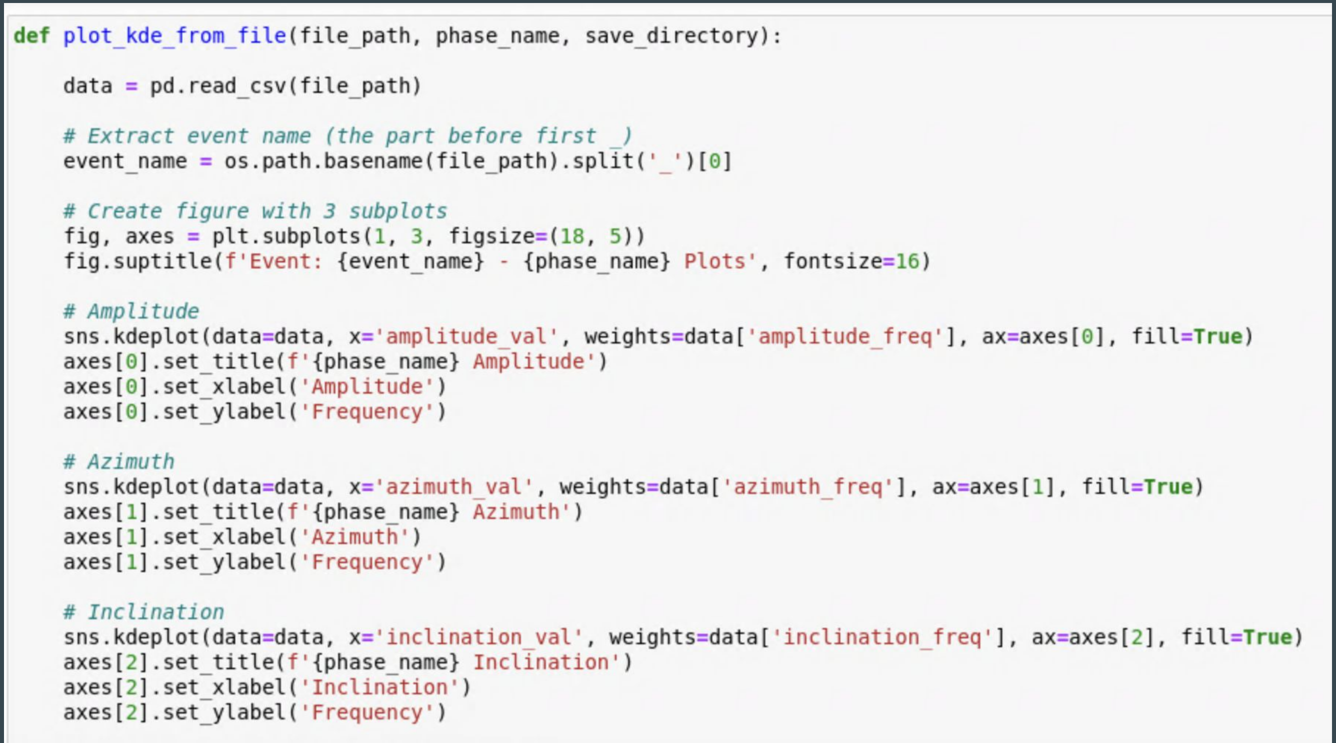
Sample kde data file



```
amplitude_val,amplitude_freq,azimuth_val,azimuth_freq,inclination_val,inclination_freq
99.75143798825884,0.8825397873242518,214.036007805443,0.8825397873242518,23.252761183492943,0.8825397873242518
99.81737166030558,0.8804729457278904,213.75540119400713,0.8804729457278904,23.261143221243415,0.8804729457278904
99.87748588728127,0.8782655051132788,213.48116489798542,0.8782655051132788,23.266141077454957,0.8782655051132788
99.93199055795114,0.8759353748651447,213.21296534287842,0.8759353748651447,23.267793144284095,0.8759353748651447
99.98107435735902,0.873498262656249,212.95047454884482,0.873498262656249,23.26614305821012,0.873498262656249
100.02490602602067,0.8709678135204302,212.6933685802107,0.8709678135204302,23.2612396676225,0.8709678135204302
100.06363568602342,0.8683557593320292,212.4413261377771,0.8683557593320292,23.2531370878771,0.8683557593320292
100.09739616486237,0.8656720703704737,212.19402728030948,0.8656720703704737,23.241894800547687,0.8656720703704737
100.12630426819923,0.8629251030882129,211.95115226645825,0.8629251030882129,23.227577759916297,0.8629251030882129
100.1504619696664,0.8601217402134724,211.71238051148885,0.8601217402134724,23.21025647612337,0.8601217402134724
100.16995749913406,0.8572675208881295,211.47738965490868,0.8572675208881295,23.190007050795817,0.8572675208881295
100.18486632077276,0.8543667597028871,211.24585473563974,0.8543667597028871,23.166911147336773,0.8543667597028871
100.19525199927844,0.8514226543093137,211.01744747103635,0.8514226543093137,23.141055884293745,0.8514226543093137
```

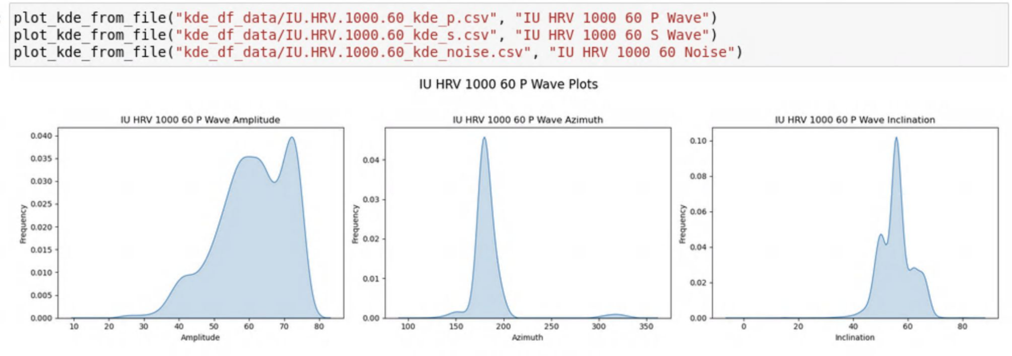
Actually after I saved all the files I figured out he frequencies are the same for each row as it is designed for consistency in each row so the file size can be further reduced if necessary.

Plot from saved kde files

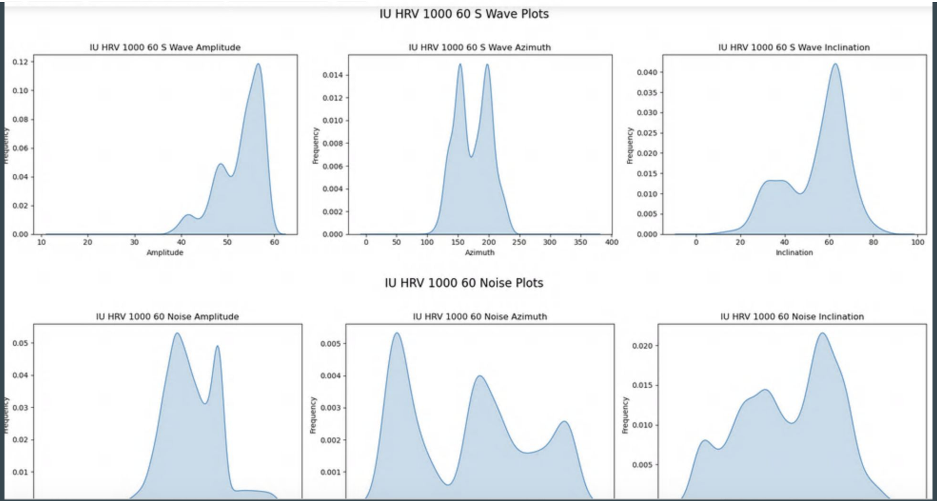


```python
def plot_kde_from_file(file_path, phase_name, save_directory):

    data = pd.read_csv(file_path)

    # Extract event name (the part before first _)
    event_name = os.path.basename(file_path).split('_')[0]

    # Create figure with 3 subplots
    fig, axes = plt.subplots(1, 3, figsize=(18, 5))
    fig.suptitle(f'Event: {event_name} - {phase_name} Plots', fontsize=16)

    # Amplitude
    sns.kdeplot(data=data, x='amplitude_val', weights=data['amplitude_freq'], ax=axes[0], fill=True)
    axes[0].set_title(f'{phase_name} Amplitude')
    axes[0].set_xlabel('Amplitude')
    axes[0].set_ylabel('Frequency')

    # Azimuth
    sns.kdeplot(data=data, x='azimuth_val', weights=data['azimuth_freq'], ax=axes[1], fill=True)
    axes[1].set_title(f'{phase_name} Azimuth')
    axes[1].set_xlabel('Azimuth')
    axes[1].set_ylabel('Frequency')

    # Inclination
    sns.kdeplot(data=data, x='inclination_val', weights=data['inclination_freq'], ax=axes[2], fill=True)
    axes[2].set_title(f'{phase_name} Inclination')
    axes[2].set_xlabel('Inclination')
    axes[2].set_ylabel('Frequency')
```

Sample plot output



```
plot_kde_from_file("kde_df_data/IU.HRV.1000.60_kde_p.csv", "IU HRV 1000 60 P Wave")
plot_kde_from_file("kde_df_data/IU.HRV.1000.60_kde_s.csv", "IU HRV 1000 60 S Wave")
plot_kde_from_file("kde_df_data/IU.HRV.1000.60_kde_noise.csv", "IU HRV 1000 60 Noise")
```



Three plots (each with 3 subplots) are generated and saved for each event.