Downloading O4a Gravitational-Wave Network Dataset with the GW231123 PUT LABELS

We are extracting the data from the GW230529 merger from the O4a network, the first phase of the fourth observing run of the LVK GW network.

In [18]:
```
import requests


def fetch_strain_list(run, detector, gps_start, gps_end):
    "Return the list of strain file info for `run` and `detector`."

    # Get the strain list
    fetch_url = (
        f"https://gwosc.org/archive/links/"
        f"{run}/{detector}/{gps_start}/{gps_end}/json/"
    )
    response = requests.get(fetch_url)
    response.raise_for_status()
    return response.json()["strain"]
```

In [2]:
```
! pip install h5py
```

```
Collecting h5py
  Using cached h5py-3.15.1-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64.w
hl.metadata (3.0 kB)
Requirement already satisfied: numpy>=1.21.2 in /srv/conda/lib/python3.11/site-packa
ges (from h5py) (2.2.6)
Using cached h5py-3.15.1-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl
(4.7 MB)
Installing collected packages: h5py
Successfully installed h5py-3.15.1
```

In [19]:
```
strain_files = fetch_strain_list("O4a_4KHZ_R1", "H1", 1382832018, 1385337618)
print(f"Found {len(strain_files)} files")
print(strain_files[0:5])
```

```
Found 932 files
```


```
[{'GPSstart': 1382891520, 'UTCstart': '2023-11-01T16:31:42', 'detector': 'H1', 'samp
ling_rate': 4096, 'duration': 4096, 'format': 'hdf5', 'url': 'https://gwosc.org/arch
ive/data/O4a_4KHZ_R1/1382023168/H-H1_GWOSC_O4a_4KHZ_R1-1382891520-4096.hdf5', 'min_s
train': -5.372688122354986e-18, 'max_strain': 5.362402307740685e-18, 'mean_strain':
1.9969402872962126e-23, 'stdev_strain': 1.0715299257922202e-18, 'duty_cycle': 46.679
6875, 'BLRMS200': 5.278628681909679e-24, 'BLRMS1000': 6.051544953857376e-21, 'BNS':
141.87615444330953}, {'GPSstart': 1382891520, 'UTCstart': '2023-11-01T16:31:42', 'de
tector': 'H1', 'sampling_rate': 4096, 'duration': 4096, 'format': 'gwf', 'url': 'htt
ps://gwosc.org/archive/data/O4a_4KHZ_R1/1382023168/H-H1_GWOSC_O4a_4KHZ_R1-1382891520
-4096.gwf', 'min_strain': -5.372688122354986e-18, 'max_strain': 5.362402307740685e-1
8, 'mean_strain': 1.9969402872962126e-23, 'stdev_strain': 1.0715299257922202e-18, 'd
uty_cycle': 46.6796875, 'BLRMS200': 5.278628681909679e-24, 'BLRMS1000': 6.0515449538
57376e-21, 'BNS': 141.87615444330953}, {'GPSstart': 1382895616, 'UTCstart': '2023-11
-01T17:39:58', 'detector': 'H1', 'sampling_rate': 4096, 'duration': 4096, 'format':
'hdf5', 'url': 'https://gwosc.org/archive/data/O4a_4KHZ_R1/1382023168/H-H1_GWOSC_O4a
_4KHZ_R1-1382895616-4096.hdf5', 'min_strain': -4.821827522699192e-18, 'max_strain':
4.796736982409019e-18, 'mean_strain': -1.8653898711944875e-22, 'stdev_strain': 1.178
2513127685192e-18, 'duty_cycle': 8.7646484375, 'BLRMS200': 5.004639123993648e-24, 'B
LRMS1000': 4.952106402602299e-21, 'BNS': 142.60280515507844}, {'GPSstart': 138289561
6, 'UTCstart': '2023-11-01T17:39:58', 'detector': 'H1', 'sampling_rate': 4096, 'dura
tion': 4096, 'format': 'gwf', 'url': 'https://gwosc.org/archive/data/O4a_4KHZ_R1/138
2023168/H-H1_GWOSC_O4a_4KHZ_R1-1382895616-4096.gwf', 'min_strain': -4.82182752269919
2e-18, 'max_strain': 4.796736982409019e-18, 'mean_strain': -1.8653898711944875e-22,
'stdev_strain': 1.1782513127685192e-18, 'duty_cycle': 8.7646484375, 'BLRMS200': 5.00
4639123993648e-24, 'BLRMS1000': 4.952106402602299e-21, 'BNS': 142.60280515507844},
{'GPSstart': 1382899712, 'UTCstart': '2023-11-01T18:48:14', 'detector': 'H1', 'sampl
ing_rate': 4096, 'duration': 4096, 'format': 'hdf5', 'url': 'https://gwosc.org/archi
ve/data/O4a_4KHZ_R1/1382023168/H-H1_GWOSC_O4a_4KHZ_R1-1382899712-4096.hdf5', 'min_st
rain': -5.2316219885767225e-18, 'max_strain': 5.5006260950142426e-18, 'mean_strain':
2.262056136806449e-24, 'stdev_strain': 1.0979359938591247e-18, 'duty_cycle': 37.9882
8125, 'BLRMS200': 4.8916326868163955e-24, 'BLRMS1000': 5.570929443447215e-21, 'BNS':
143.65362215817004}]
```


In [20]:

```
def download_strain_file(download_url):
    # saving strain file to disk
    filename = download_url.split("/")[-1]
    with requests.get(download_url, stream=True) as r:
        # navigating file in small incriments (chunks)
        # and nicknaming as r for easier coding
        with open(filename, "wb") as f:
            for chunk in r.iter_content(chunk_size=8192):
                f.write(chunk)
                #iterating over the chunks
        return filename

for file in strain_files:
    if file["GPSstart"] == 1384779776 and file["format"] == 'hdf5':
        print(f"Downloading {file['url']}")
        fname = download_strain_file(file['url'])
```

```
Downloading https://gwosc.org/archive/data/O4a_4KHZ_R1/1384120320/H-H1_GWOSC_O4a_4KH
Z_R1-1384779776-4096.hdf5
```

Accessing the HDF5 File

In [21]:
```
import numpy as np
from matplotlib import mlab
import matplotlib.pyplot as plt
import h5py
```

In [22]:
```
filename = 'H-H1_GWOSC_O4a_4KHZ_R1-1384779776-4096.hdf5'
dataFile = h5py.File(filename, 'r')
```

In [23]:
```
for key in dataFile.keys():
    print(key)
    # accessing data as dictionary
```

```
meta
quality
strain
```

In [24]:
```
strain = dataFile['strain']['Strain']
ts = dataFile['strain']['Strain'].attrs['Xspacing']
print(f"ts = {ts}s, sample rate = {1/ts}Hz")
```

```
ts = 0.000244140625s, sample rate = 4096.0Hz
```

In [25]:
```
# metadata
metaKeys = dataFile['meta'].keys()
meta = dataFile['meta']
for key in metaKeys:
    print(key, meta[key])
```

```
Description <HDF5 dataset "Description": shape (), type "|O">
DescriptionURL <HDF5 dataset "DescriptionURL": shape (), type "|O">
Detector <HDF5 dataset "Detector": shape (), type "|O">
Duration <HDF5 dataset "Duration": shape (), type "<i8">
FrameType <HDF5 dataset "FrameType": shape (), type "|O">
GPSstart <HDF5 dataset "GPSstart": shape (), type "<i8">
Observatory <HDF5 dataset "Observatory": shape (), type "|O">
StrainChannel <HDF5 dataset "StrainChannel": shape (), type "|O">
Type <HDF5 dataset "Type": shape (), type "|O">
UTCstart <HDF5 dataset "UTCstart": shape (), type "|O">
```

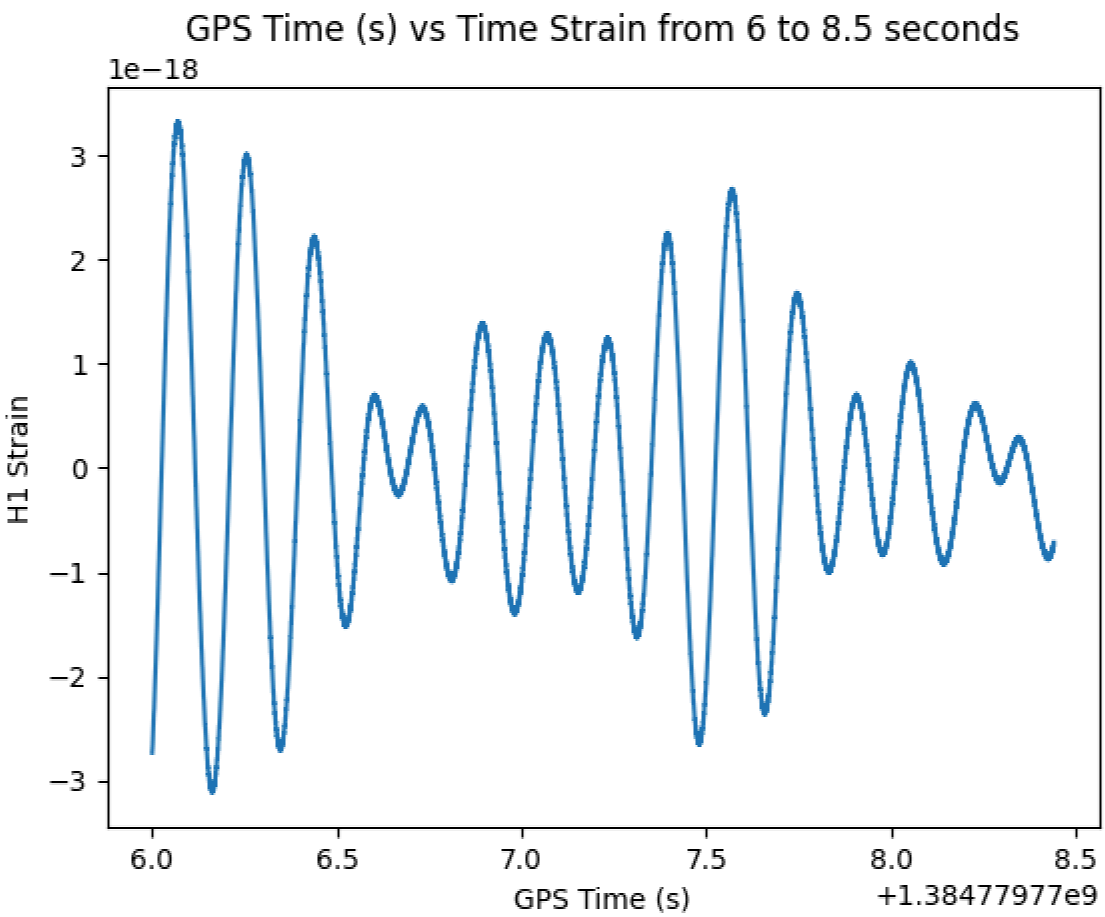
In [26]:
```
# creating time vector
gpsStart = meta['GPSstart'][()]
duration = meta['Duration'][()]
gpsEnd   = gpsStart + duration

time = np.arange(gpsStart, gpsEnd, ts)
```

In [ ]:
```
# plotting the time series acros the 4096 seconds
plt.plot(time, strain[()])
plt.xlabel('GPS Time (s)')
plt.ylabel('H1 Strain')
plt.title("GPS Time vs H1 Strain")
plt.show()
```

In [36]:
```
# zooming in on a specific time series
numsamples = 10000 # 4096 Hz
```

```
startTime  = 1264316116.0
startIndex = np.min(np.nonzero(startTime < time))
time_seg   = time[startIndex:(startIndex+numsamples)]
strain_seg = strain[startIndex:(startIndex+numsamples)]
plt.plot(time_seg, strain_seg)
plt.xlabel('GPS Time (s)')
plt.ylabel('H1 Strain')
plt.title("GPS Time (s) vs Time Strain from 6 to 8.5 seconds")
plt.show()
```



Accessing Data Quality

In [27]:
```
dqInfo = dataFile['quality']['simple']
bitnameList = dqInfo['DQShortnames'][()]
descriptionList = dqInfo['DQDescriptions'][()]
nbits = len(bitnameList)

for bit in range(nbits):
    print(f"Channel #{bit} ({bitnameList[bit].decode()}): {descriptionList[bit].dec
    # 7 data quality categories and 5 injection categories
    # each represented as 1 Hz time series
    # printing dq channel names & descriptions
```

```
Channel #0 (DATA): data present
Channel #1 (CBC_CAT1): passes the cbc CAT1 test
Channel #2 (CBC_CAT2): passes cbc CAT2 test
Channel #3 (CBC_CAT3): passes cbc CAT3 test
Channel #4 (BURST_CAT1): passes burst CAT1 test
Channel #5 (BURST_CAT2): passes burst CAT2 test
Channel #6 (BURST_CAT3): passes burst CAT3 test
Channel #7 (STOCH_CAT1): passes stoch CAT1 test
Channel #8 (CW_CAT1): passes cw CAT1 test
```

In [28]:
```
dqmask = dqInfo['DQmask'][()]
```

In [29]:
```
value = dqmask[2400]
print("Value in decimal: {0}".format(value))
print("Same value in binary (with 7 bits): {0:#09b}".format(value))
# 1 in binary means that it's good data
# 0 means it's bad
```

```
Value in decimal: 511
Same value in binary (with 7 bits): 0b111111111
```

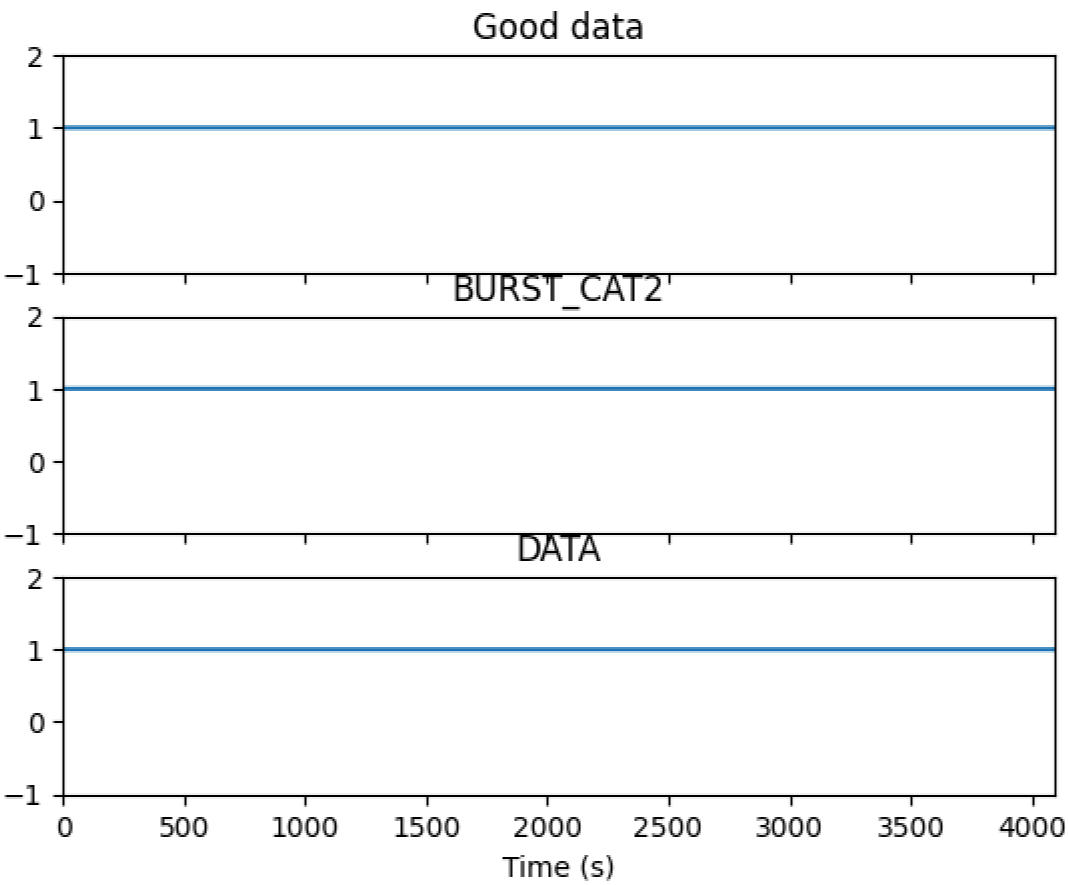
In [42]:
```
data_channel = 0
data_mask = (dqmask >> data_channel) & 1 # 0 everywhere the data fails
                                         # and 1 wherever it pases

burst_cat2_channel = 5
burst_cat2_mask = (dqmask >> burst_cat2_channel) & 1
```

In [30]:
```
goodData_mask_1hz = data_mask & burst_cat2_mask
```

In [49]:
```
fig, (ax0, ax1, ax2) = plt.subplots(3, sharex=True, sharey=True)
ax0.plot(goodData_mask_1hz)
ax0.set_title('Good data')
ax1.plot(burst_cat2_mask)
ax1.set_title('BURST_CAT2')
ax2.plot(data_mask)
ax2.set_title('DATA')
ax2.axis([0, 4096, -1, 2])
ax2.set_xlabel('Time (s)')
# bad data would have spikes to the 0 along the plot
```

Out[49]:
```
Text(0.5, 0, 'Time (s)')
```

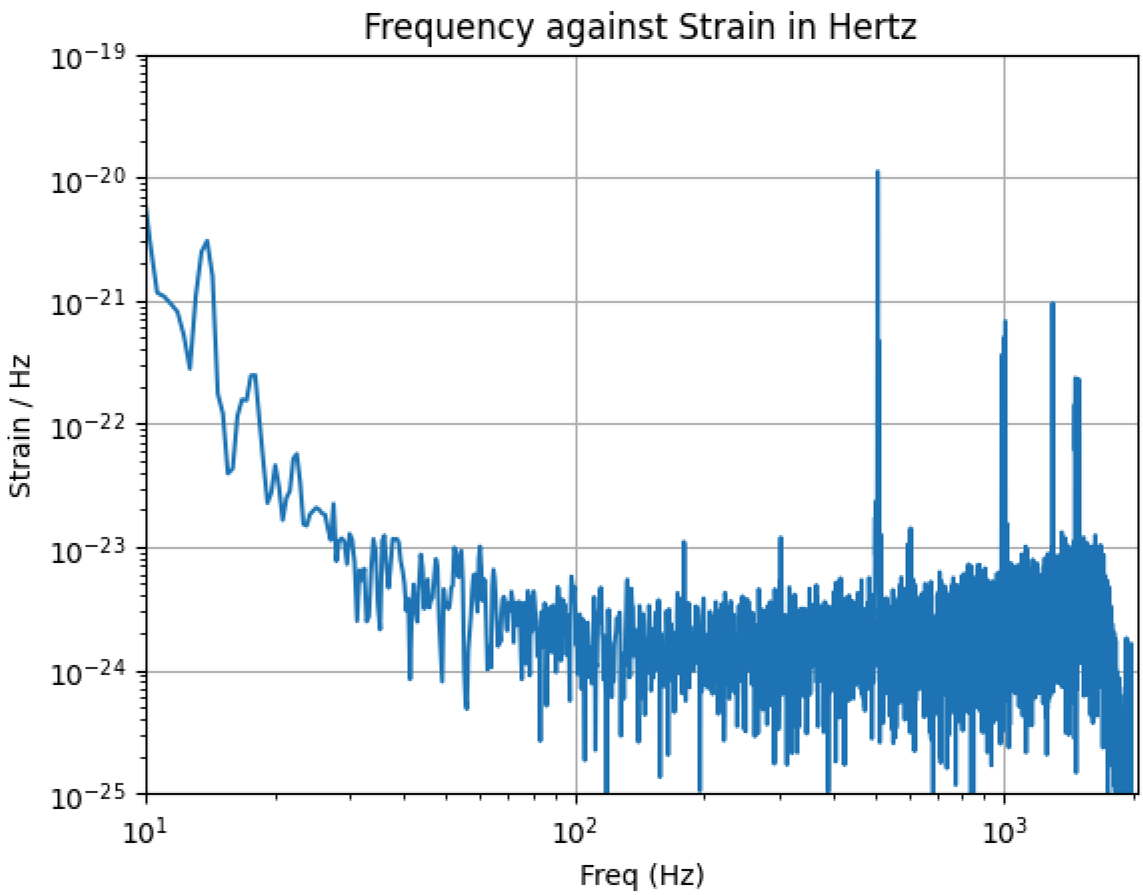


The following tables show that all data is good to use in analysis.

Analyzing frequency

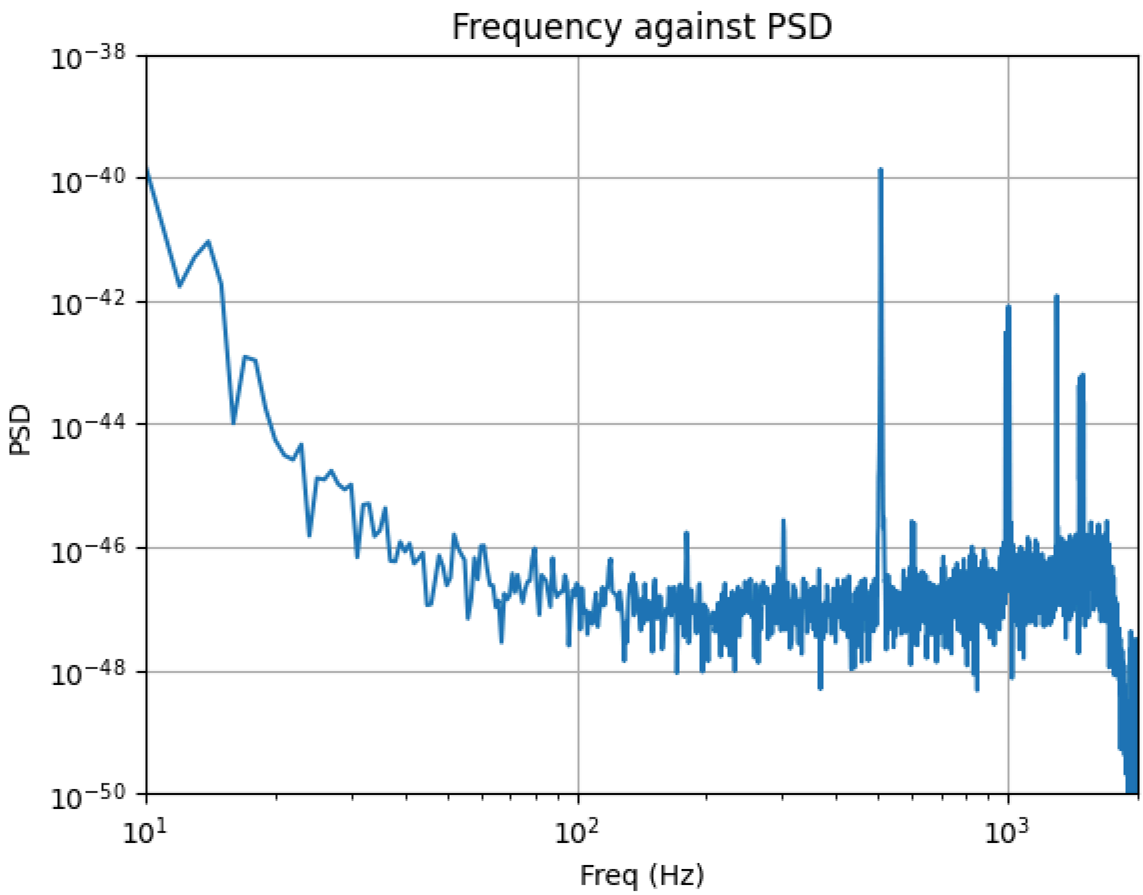
In [31]:
```
# sampling frequency
fs = int(1.0 / ts)
```

In [59]:
```
# using a 'blackman window' to reduce any leakages in data
window          = np.blackman(strain_seg.size)
windowed_strain = strain_seg * window
freq_domain     = np.fft.rfft(windowed_strain) / fs
freq            = np.fft.rfftfreq(len(windowed_strain)) * fs
plt.loglog(freq, abs(freq_domain))
plt.axis([10, fs/2.0, 1e-25, 1e-19])
plt.grid('on')
plt.xlabel('Freq (Hz)')
plt.ylabel('Strain / Hz')
plt.title("Frequency against Strain in Hertz")
plt.show()
```

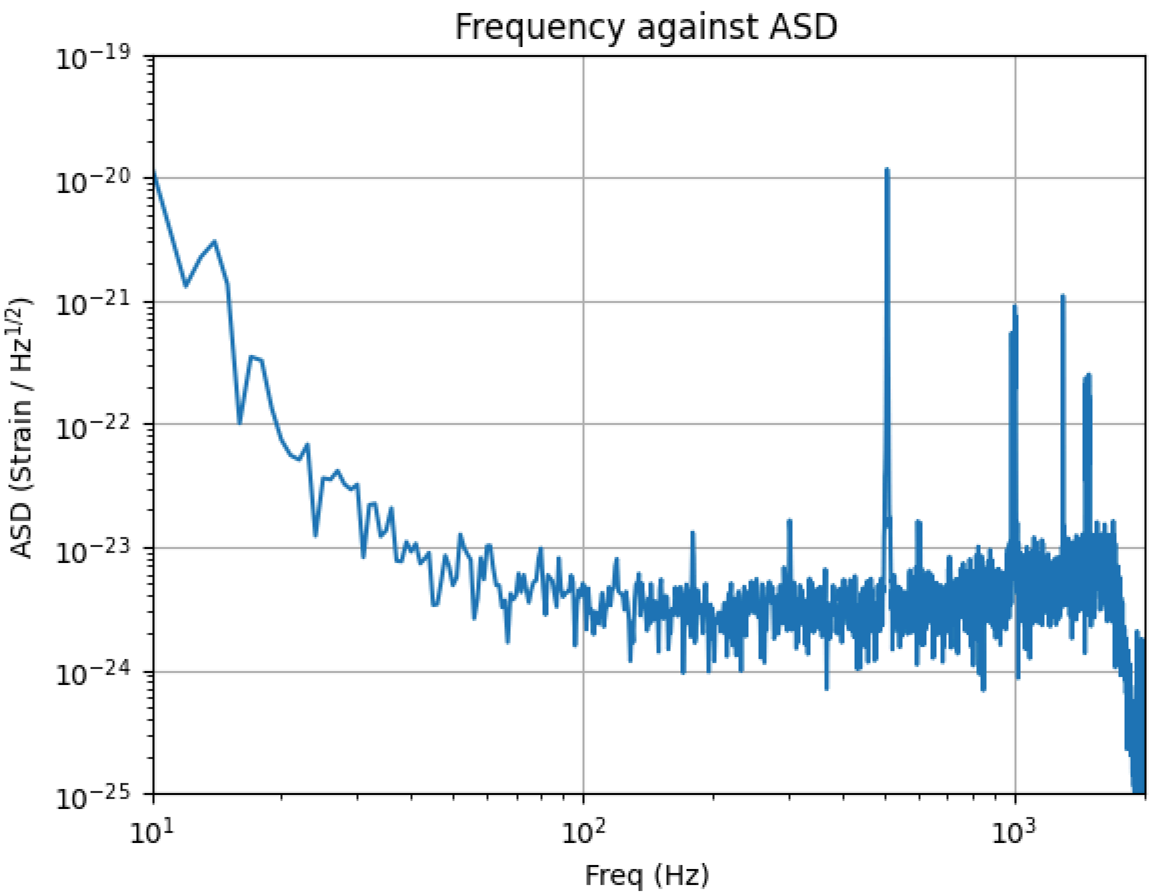


In [60]:
```
# power spectral density to see how power is distributed
# available in matplotlib
Pxx, freqs = mlab.psd(strain_seg, Fs=fs, NFFT=fs)
plt.loglog(freqs, Pxx)
plt.axis([10, 2000, 1e-50, 1e-38])
plt.grid('on')
plt.xlabel('Freq (Hz)')
plt.ylabel('PSD')
plt.title("Frequency against PSD")
```

Out[60]:
```
Text(0.5, 1.0, 'Frequency against PSD')
```

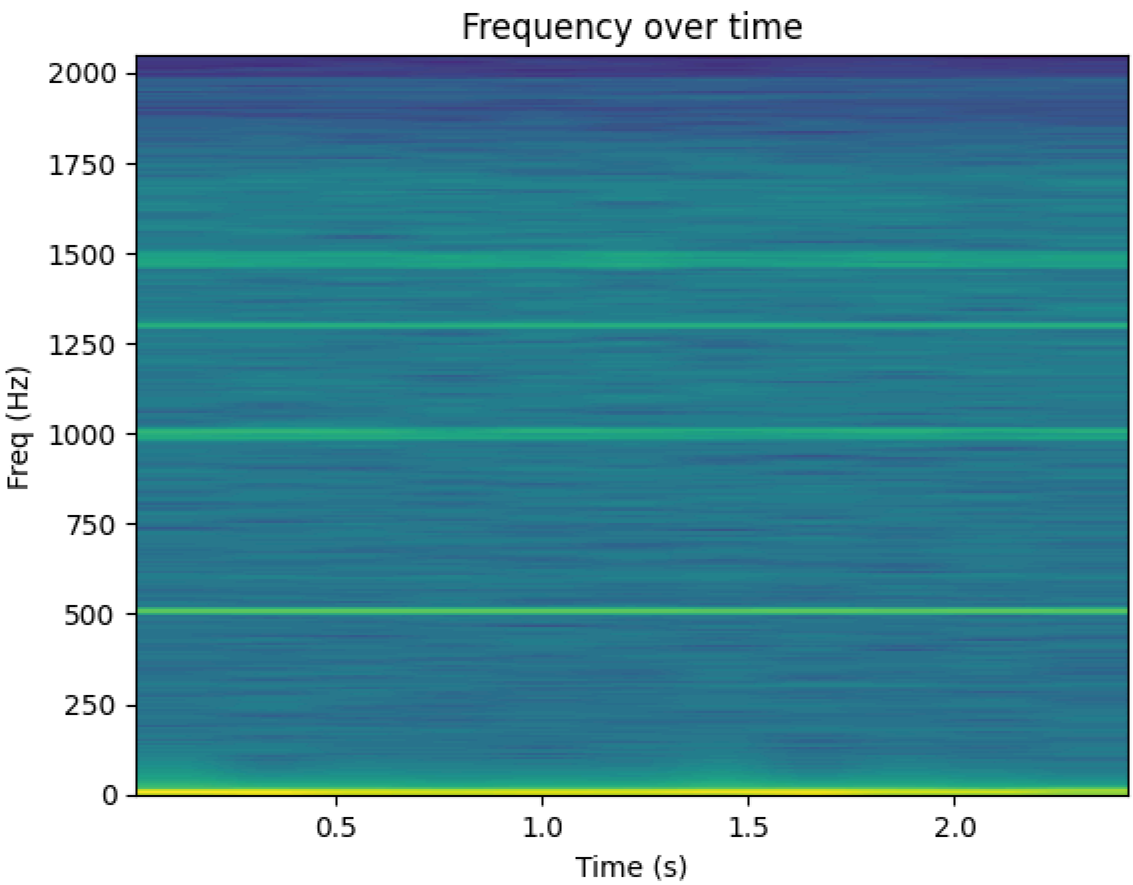


In [62]:
```
plt.loglog(freqs, np.sqrt(Pxx)) # creates a plot with log scale on both axes
plt.axis([10, 2000, 1e-25, 1e-19])
plt.grid('on')
plt.xlabel('Freq (Hz)')
plt.ylabel('ASD (Strain / Hz$^{1/2})$')
plt.title("Frequency against ASD")
plt.show()
```



Creating a spectogram to see how frequency varies over time

In [64]:
```
NFFT = 1024 # fourier transformation
short_window = np.blackman(NFFT)
spec_power, freqs, bins, im = plt.specgram(
    strain_seg, NFFT=NFFT, Fs=fs,
    window=short_window
)
plt.xlabel('Time (s)')
plt.ylabel('Freq (Hz)')
plt.title("Frequency over time")
plt.show()
```


Frequency over time
Freq (Hz)
2000
1750
1500
1250
1000
750
500
250
0
0.5
1.0
1.5
2.0
Time (s)