1 Project Header

In this project, we are looking at actigraphy data sourced from the Depresjon (https://datasets.simula.no/depresjon/) dataset with the goal of developing a classification algorithm to aid in non-invasive recognition of those with a Major Depressive Disorder. The study included 32 healthy and 23 afflicted participants and collected activity data through participant's continual wearing of a watch containing an accelerometer. A record exists for each minute over 13 or more days where the activity associated with each minute is the sum of the number of movements with an acceleration greater than .5g. Related literature includes:

- Enrique Garcia-Ceja, Michael Riegler, Petter Jakobsen, Jim Tørresen, Tine Nordgreen, Ketil J. Oedegaard, and Ole Bernt Fasmer. 2018. Depresjon: a motor activity database of depression episodes in unipolar and bipolar patients. In Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18). Association for Computing Machinery, New York, NY, USA, 472–477. DOI:https://doi.org/10.1145/3204949.3208125 (https://doi.org/10.1145/3204949.3208125)
- Pacheco-Gonzalez, S.L., Zanella-Calzada, L.A., Galvan-Tejada, C.E., Chavez-Lamas, N.M., Rivera-Gomez, J.F., abnd Galvan-Tejada, J.I.. 2019. Evaluation of five classifiers for depression episodes detection, *Res. Comput. Sci.*, vol. 148, pp. 129-138.
- Rodríguez-Ruiz, J.G., Galván-Tejada, C.E., Vázquez-Reyes, S. et al. Classification of Depressive Episodes Using Nighttime Data; a Multivariate and Univariate Analysis. Program Comput Soft 46, 689–698 (2020). https://doi.org/10.1134/S0361768820080198 (https://doi.org/10.1134/S0361768820080198)
- Zanella-Calzada LA, Galván-Tejada CE, Chávez-Lamas NM, Gracia-Cortés MDC, Magallanes-Quintanar R, Celaya-Padilla JM, Galván-Tejada JI, Gamboa-Rosales H. Feature Extraction in Motor Activity Signal: Towards a Depression Episodes Detection in Unipolar and Bipolar Patients. Diagnostics (Basel). 2019 Jan 10;9(1):8. doi: https://10.3390/diagnostics9010008 (https://10.3390/diagnostics9010008). PMID: 30634621; PMCID: PMC6468429.

2 This Notebook

In this notebook we will be doing our initial exploration and pre-processing of the data. We hope to increase the ability of classification algorithms to make predictions through addition of features that quantify and qualify data as it is divided into time sections for patients.

2.1 Importing Data and Libraries

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from glob import glob
```

executed in 800ms, finished 23:02:07 2021-07-07

In [3]:
```
condition_list = glob("./data/condition/*.csv")
control_list = glob("./data/control/*.csv")
cleaned_conditions = [filename.split('\\')[1].replace('.csv', '')
                      for filename in condition_list]
cleaned_controls = [filename.split('\\')[1].replace('.csv', '')
                      for filename in control_list]
```

executed in 14ms, finished 23:02:07 2021-07-07

In [4]:
```
condition_dfs = {}
control_dfs = {}
for idx,file in enumerate(condition_list):
    condition_dfs.update({cleaned_conditions[idx]: pd.read_csv(file)})
for idx,file in enumerate(control_list):
    control_dfs.update({cleaned_controls[idx]: pd.read_csv(file)})
```

executed in 889ms, finished 23:02:08 2021-07-07

2.2 Exploring Dataset

In [5]:
```
condition_dfs['condition_1'].head()
```

executed in 15ms, finished 23:02:08 2021-07-07

Out[5]:

	timestamp	date	activity
0	2003-05-07 12:00:00	2003-05-07	0
1	2003-05-07 12:01:00	2003-05-07	143
2	2003-05-07 12:02:00	2003-05-07	0
3	2003-05-07 12:03:00	2003-05-07	20
4	2003-05-07 12:04:00	2003-05-07	166

In [6]:
```
condition_dfs['condition_1'].info()
```

executed in 15ms, finished 23:02:08 2021-07-07

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23244 entries, 0 to 23243
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   timestamp  23244 non-null  object
 1   date       23244 non-null  object
 2   activity   23244 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 544.9+ KB
```

In [7]:
```
type(condition_dfs['condition_5'].iloc[0].timestamp)
test_stamp = pd.to_datetime(condition_dfs['condition_5'].iloc[0].timestamp)
print(test_stamp)
print(test_stamp.hour)
```

executed in 15ms, finished 23:02:08 2021-07-07

```
2003-06-12 10:30:00
10
```

Okay, so we can fix up these dataframes a little to start to make them easier to work with. Since we can cast the timestamp column into a datetime obj, we can get rid of the date column as it becomes redundant. So, let's recast timestamp and drop date for each dataframe.

In [8]:
```
for df in condition_dfs:
    condition_dfs[df] = condition_dfs[df].drop(columns = 'date')
    condition_dfs[df].timestamp = pd.to_datetime(condition_dfs[df].timestamp)
```

executed in 313ms, finished 23:02:08 2021-07-07

In [9]:
```
for df in control_dfs:
    control_dfs[df] = control_dfs[df].drop(columns = 'date')
    control_dfs[df].timestamp = pd.to_datetime(control_dfs[df].timestamp)
```

executed in 509ms, finished 23:02:09 2021-07-07

Let's see how many days each subject wore the watch for.

In [10]:
```
for patient in condition_dfs:
    print(condition_dfs[patient].shape[0]/60/24)
```

executed in 14ms, finished 23:02:09 2021-07-07

```
16.141666666666666
14.96875
15.965277777777779
15.379861111111111
17.993055555555554
15.031944444444443
15.159027777777778
29.06041666666667
14.952083333333334
14.824305555555556
14.74375
27.031944444444445
17.949305555555558
14.227083333333333
15.119444444444445
21.864583333333332
15.033333333333333
14.969444444444443
14.925694444444444
14.884027777777776
15.399305555555555
13.402083333333332
14.109722222222222
```

In [11]:
```
for patient in control_dfs:
    print(control_dfs[patient].shape[0]/60/24)
```

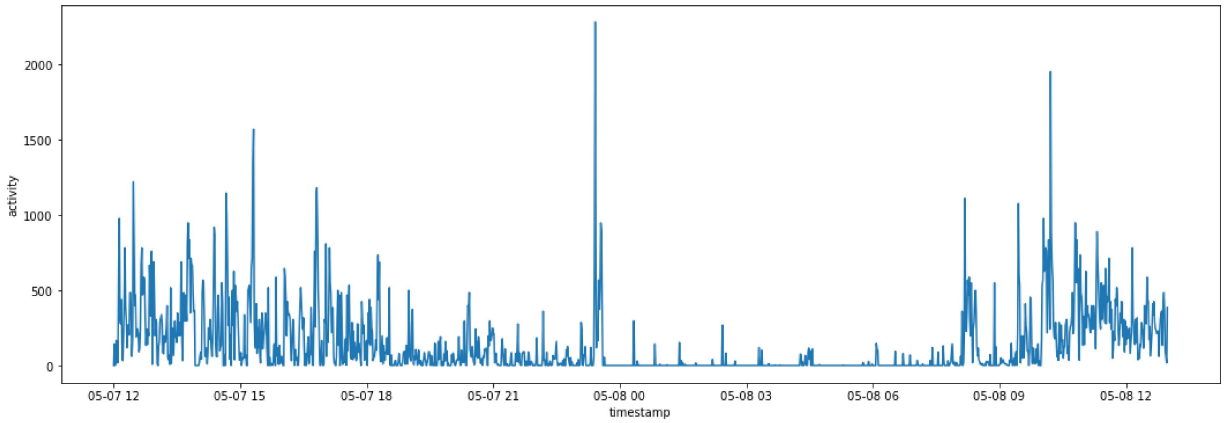
executed in 14ms, finished 23:02:09 2021-07-07

```
35.84097222222222
15.038194444444445
17.20277777777778
24.053472222222222
15.456249999999999
15.457638888888889
15.460416666666667
17.050694444444442
20.072916666666668
17.093055555555555
19.172916666666666
21.85625
22.146527777777777
22.177777777777777
15.06111111111111
15.065277777777778
15.155555555555557
15.151388888888889
23.17013888888889
14.229166666666666
17.038888888888888
16.96736111111111
45.421527777777776
15.047916666666666
35.68680555555556
35.84652777777778
21.84375
32.731249999999996
35.825694444444444
35.75694444444444
20.081944444444442
20.165277777777778
```

Hmm.. so the subjects did not wear the watches for a standard number of days. In fact, there is a pretty wide distribution in the amount of time the watch was worn, from a minimum near 13 days all the way up to a maximum in the control group for 45 days. I think the first thing I would like to do is examine the Fourier Transform results, then determine how to efficiently establish sleeping vs not sleeping for each patient. I think in the future I will have to group the data into hour or half-hour lapses, based on the literature, so I will have to decide between making partial-hour cutoffs at hard sleep vs not sleep divisions, or decide on some metric in which the subject is primarily awake or asleep during pre-defined hour divisions. Let's see what our actigraphy data actually looks like.

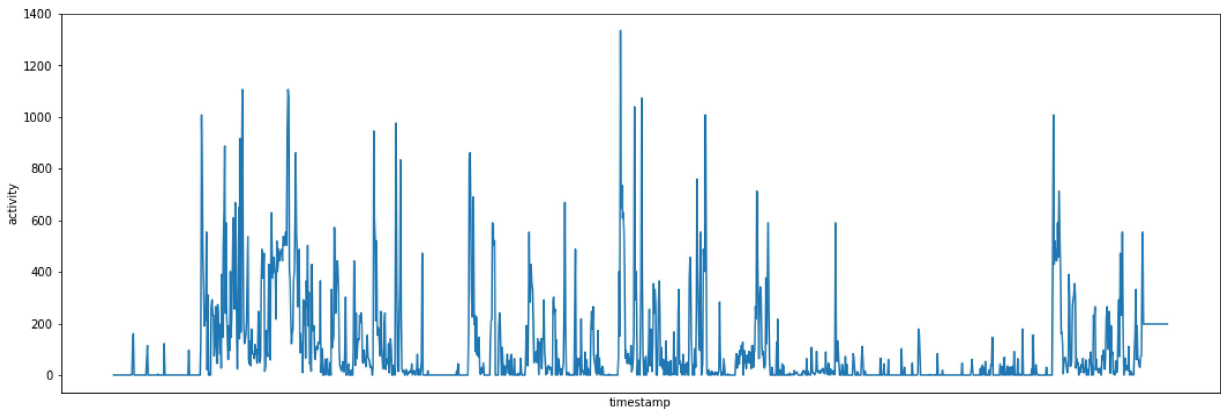
In [12]:
```
fig1, ax1 = plt.subplots(figsize = (18,6));
sns.lineplot(data = condition_dfs['condition_1'].iloc[:1500], x = 'timestamp', y
```

executed in 252ms, finished 23:02:09 2021-07-07



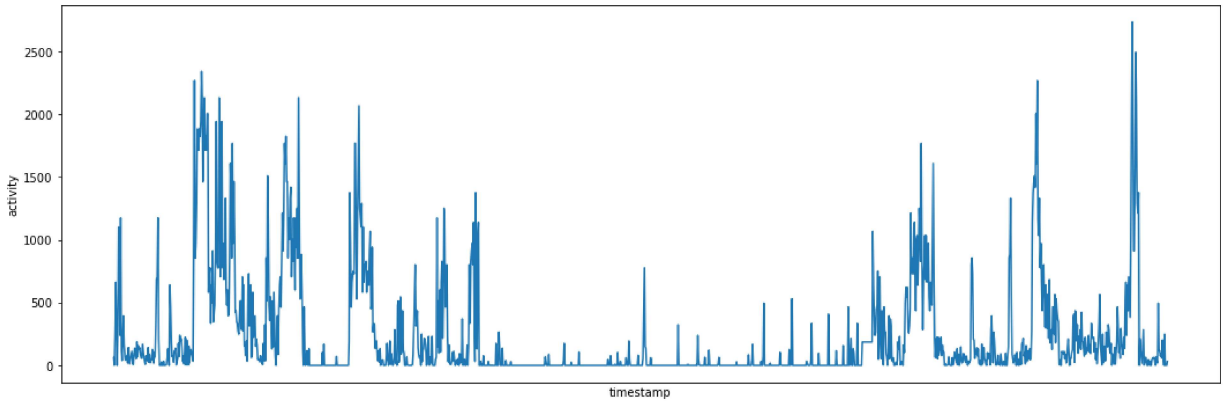
In [13]:
```
fig3, ax3 = plt.subplots(figsize = (18,6));
sns.lineplot(data = condition_dfs['condition_5'].iloc[:1500], x = 'timestamp', y
ax3.axes.xaxis.set_ticks([]);
```

executed in 171ms, finished 23:02:09 2021-07-07



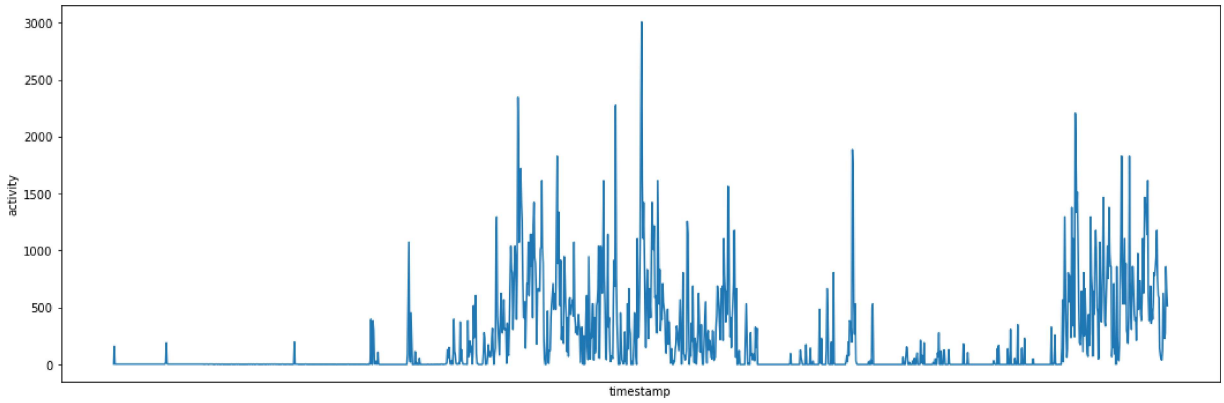
In [14]:
```
fig2, ax2 = plt.subplots(figsize = (18,6));
sns.lineplot(data = control_dfs['control_1'].iloc[:1500], x = 'timestamp', y = 'a
ax2.axes.xaxis.set_ticks([]);
```

executed in 171ms, finished 23:02:09 2021-07-07



In [15]:
```
fig4, ax4 = plt.subplots(figsize = (18,6));
sns.lineplot(data = control_dfs['control_8'].iloc[:1500], x = 'timestamp', y = 'a
ax4.axes.xaxis.set_ticks([]);
```

executed in 170ms, finished 23:02:10 2021-07-07



In [16]:
```
condition_dfs['condition_1'].iloc[0]
```

executed in 15ms, finished 23:02:10 2021-07-07

Out[16]:
```
timestamp    2003-05-07 12:00:00
activity                       0
Name: 0, dtype: object
```

In [17]:
```
control_dfs['control_1'].iloc[0]
```

executed in 15ms, finished 23:02:10 2021-07-07

Out[17]:
```
timestamp    2003-03-18 15:00:00
activity                      60
Name: 0, dtype: object
```

So they started and stopped wearing the devices at different times in addition to different lengths.

2.3 Exploring Frequency Domain

In the most recent paper related to this dataset, they were able to radically improve classification metrics by taking an FFT and including some features generated from the FFT distributions. Now, based on my understanding of FFTs and their output, the ones they included don't really make sense. Generally, the output from purely real (no imaginary components) data as in this case is symmetric around the zero frequency, so you would only calculate and look at the positive frequencies. This would result in the highest number of points at 0 and it tailing off extremely hard as you get into higher frequencies. Even if you plotted both sides to get a exceptionally narrow gaussian like distribution, it would still be an exceptionally sharp central peak with, depending on the data, a few other noticeable peaks some distance away from the center on either side. With that in mind, seeing that their model found that the frequency dependent metrics that were useful to the model were `[mean, variance, coefficient of variance]` makes me feel like the frequency part of their analysis was useless. However, with domain knowledge suggesting that MDDs affect circadian rhythms, their idea to look into the frequency space for useful features was pretty good. So, we will explore frequency related metrics as well.

In [18]:
```
peaks_arr = abs(np.fft.rfft(control_dfs['control_1'][:60].activity))
```
executed in 15ms, finished 23:02:10 2021-07-07

In [19]:
```
peaks_arr
```
executed in 15ms, finished 23:02:10 2021-07-07

Out[19]:
```
array([9389.        , 4365.60610338, 4584.7427142 , 2874.08137289,
       2260.48822866, 1394.34130491, 1036.73907594, 1164.53276011,
       1195.02218732, 1983.31198078, 1743.41647348, 1964.22467956,
        912.74717497,  902.06917342, 1377.10909598, 1677.90136778,
       2419.40392411, 1997.70896159, 1741.47066826, 1592.62712753,
       1265.78473683, 1252.0688921 , 1212.09788082, 1279.70051232,
       1136.98926758, 1272.04886912, 1560.61446541, 1120.86811817,
       1190.35859804, 1376.75261087,  603.        ])
```

In [20]:
```
n = control_dfs['control_1'][:60].activity.size
freq_arr = np.fft.rfftfreq(n, d = 60)
```
executed in 14ms, finished 23:02:10 2021-07-07

In [21]:
```
print(len(freq_arr))
print(len(peaks_arr))
print(n)
```
executed in 15ms, finished 23:02:10 2021-07-07

```
31
31
60
```

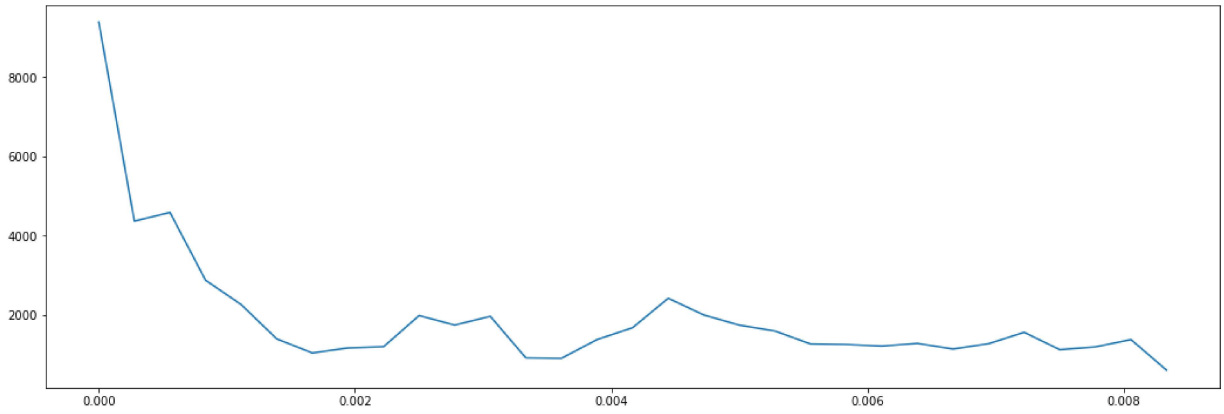
In [22]:
```
freq_arr
```
executed in 15ms, finished 23:02:10 2021-07-07

Out[22]:
```
array([0.        , 0.00027778, 0.00055556, 0.00083333, 0.00111111,
       0.00138889, 0.00166667, 0.00194444, 0.00222222, 0.0025    ,
       0.00277778, 0.00305556, 0.00333333, 0.00361111, 0.00388889,
       0.00416667, 0.00444444, 0.00472222, 0.005     , 0.00527778,
       0.00555556, 0.00583333, 0.00611111, 0.00638889, 0.00666667,
       0.00694444, 0.00722222, 0.0075    , 0.00777778, 0.00805556,
       0.00833333])
```

As a self-reminder for these plots, as you increase in frequency, you decrease in period. So, as you go to the right on the x-axis, the time scale on which the cyclic behavior is happening is decreasing.

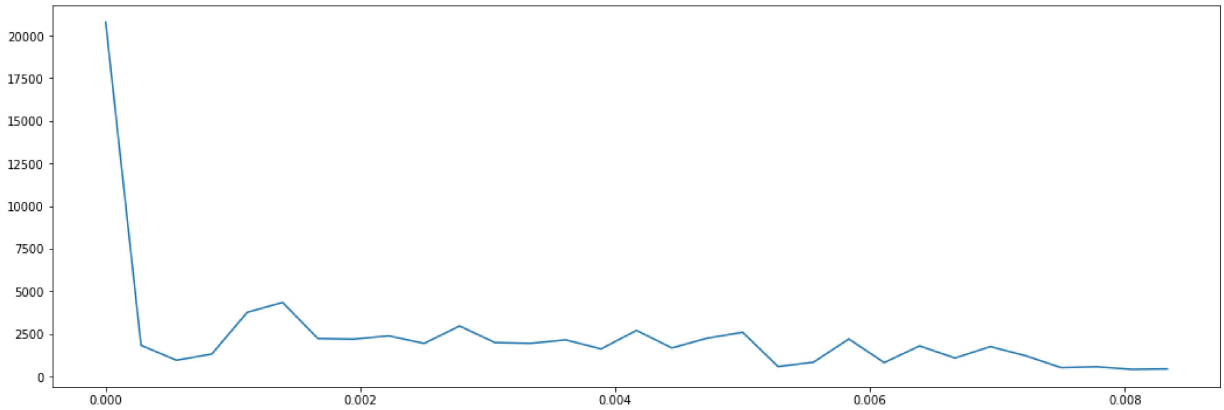
In [23]:
```
fig5, ax5 = plt.subplots(figsize = (18,6))
sns.lineplot(y=peaks_arr,x=freq_arr)
```
executed in 107ms, finished 23:02:10 2021-07-07

Out[23]:
```
<AxesSubplot:>
```



In [24]:
```
peaks_arr = abs(np.fft.rfft(condition_dfs['condition_1'][:60].activity))
fig6, ax6 = plt.subplots(figsize = (18,6))
sns.lineplot(y=peaks_arr,x=freq_arr)
```
executed in 136ms, finished 23:02:10 2021-07-07

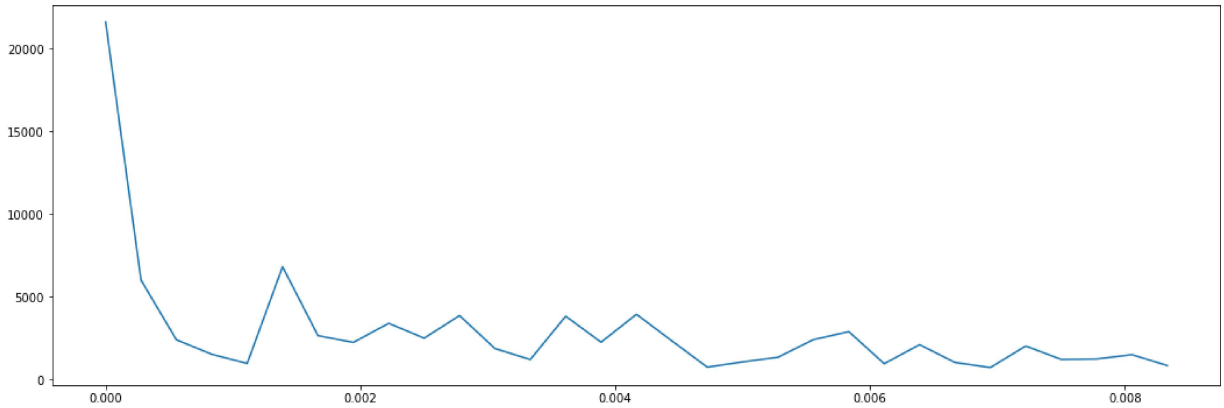
Out[24]:
```
<AxesSubplot:>
```



In [25]:
```
peaks_arr = abs(np.fft.rfft(condition_dfs['condition_2'][:60].activity))
fig7, ax7 = plt.subplots(figsize = (18,6))
sns.lineplot(y=peaks_arr,x=freq_arr)
```

executed in 108ms, finished 23:02:10 2021-07-07

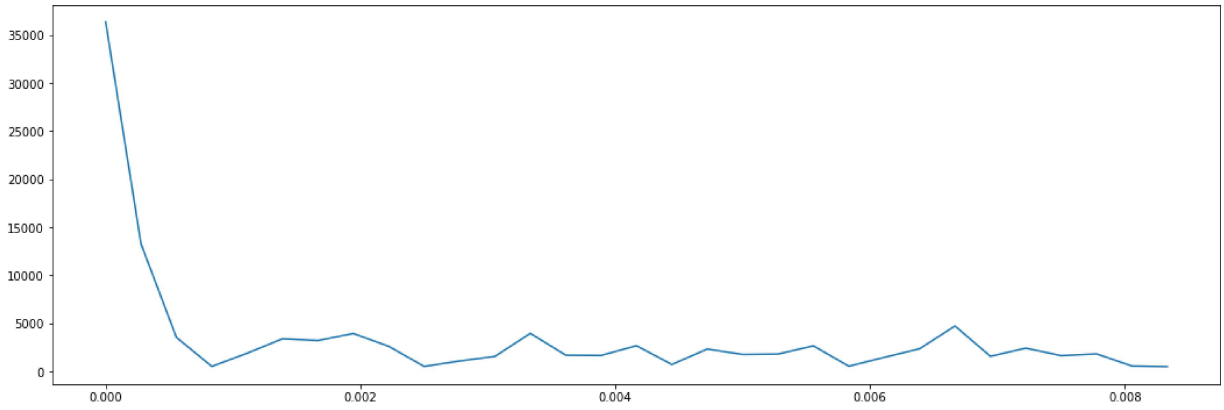
Out[25]: <AxesSubplot:>



In [26]:
```
peaks_arr = abs(np.fft.rfft(control_dfs['control_2'][:60].activity))
fig8, ax8 = plt.subplots(figsize = (18,6))
sns.lineplot(y=peaks_arr,x=freq_arr)
```

executed in 123ms, finished 23:02:10 2021-07-07

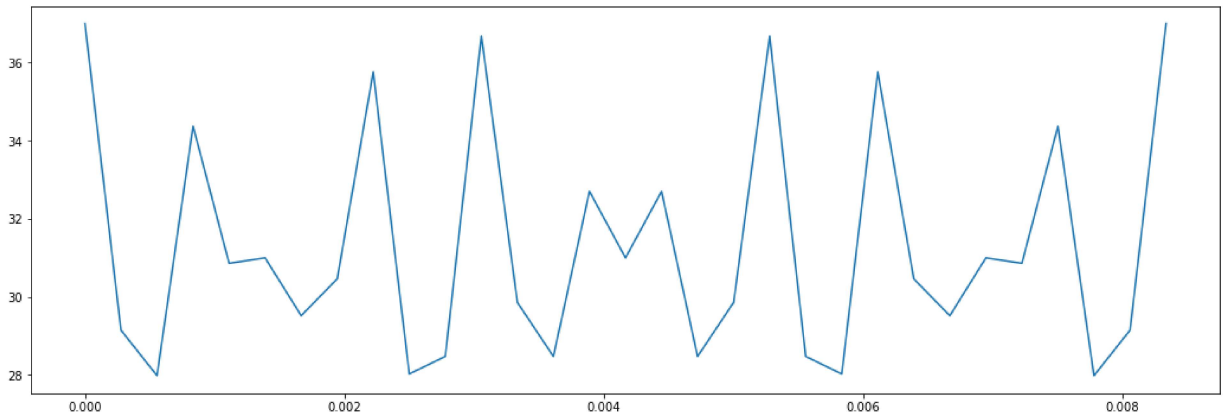
Out[26]: <AxesSubplot:>



In [27]:
```
peaks_arr = abs(np.fft.rfft(control_dfs['control_3'][:60].activity))
fig9, ax9 = plt.subplots(figsize = (18,6))
sns.lineplot(y=peaks_arr,x=freq_arr)
```

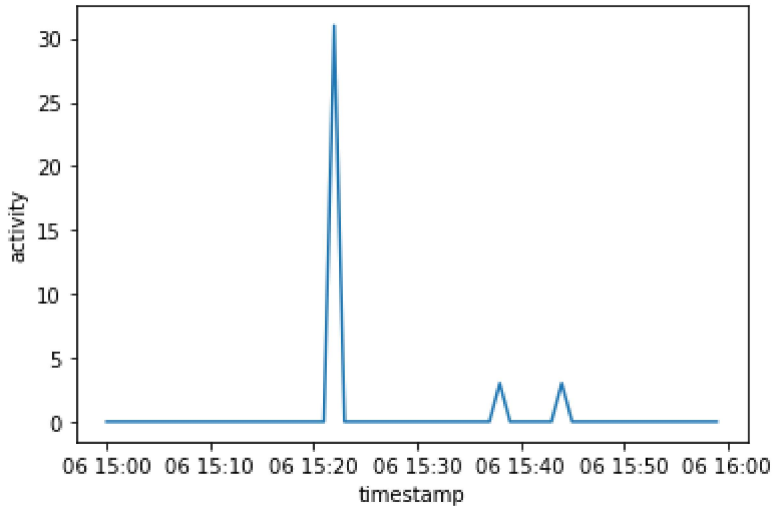
executed in 107ms, finished 23:02:10 2021-07-07

Out[27]: <AxesSubplot:>



In [28]:
```
sns.lineplot(data = control_dfs['control_3'].iloc[:60], x = 'timestamp', y = 'act
```

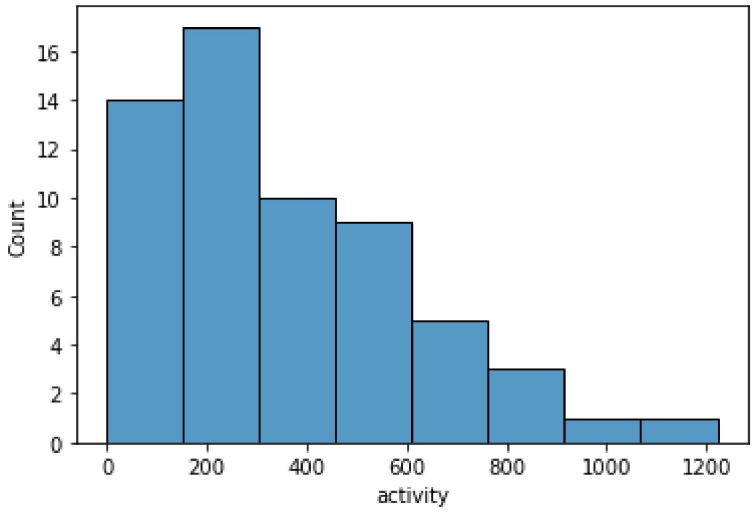
executed in 173ms, finished 23:02:10 2021-07-07



So, as we're seeing, taking FFTs of hour long lapses results in pretty meaningless information. This is especially true for the most recent two plots for `control_3` . We can see that, for primarly nighttime data as they used, if the subject is asleep such that there are very few spikes, there is literally nothing to be had in the output of an FFT.

In [29]:
```
sns.histplot(data = condition_dfs['condition_1'].iloc[:60], x = 'activity');
```

executed in 138ms, finished 23:02:11 2021-07-07

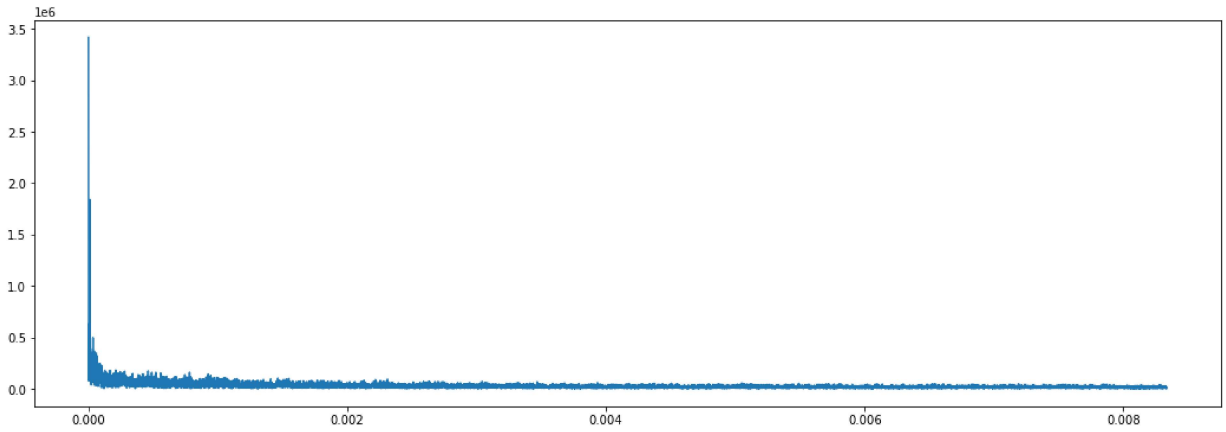


Just quickly checking, since they also used distribution statistics in the time-domain, if the time histograms look amenable to such treatment. Seems much more reasonable. Okay, so now another idea: while the FFT of a half hour cycle should be enough to pick up on longer frequency components, the signal would have to be really strong. And as we saw, FFTs on the primarily sleeping sections are basically meaningless. So, what may be more elucidative to look at is a FFT of the patients entire actigraphy time series. Let's look into that.

In [30]:
```
peaks_arr = abs(np.fft.rfft(condition_dfs['condition_1'].activity))
n = len(condition_dfs['condition_1'].activity)
fs = 1/60
freq_arr = np.fft.rfftfreq(n,d=1/fs)
fig6, ax6 = plt.subplots(figsize = (18,6))
sns.lineplot(y=peaks_arr,x=freq_arr)
```

executed in 528ms, finished 23:02:11 2021-07-07

Out[30]: <AxesSubplot:>

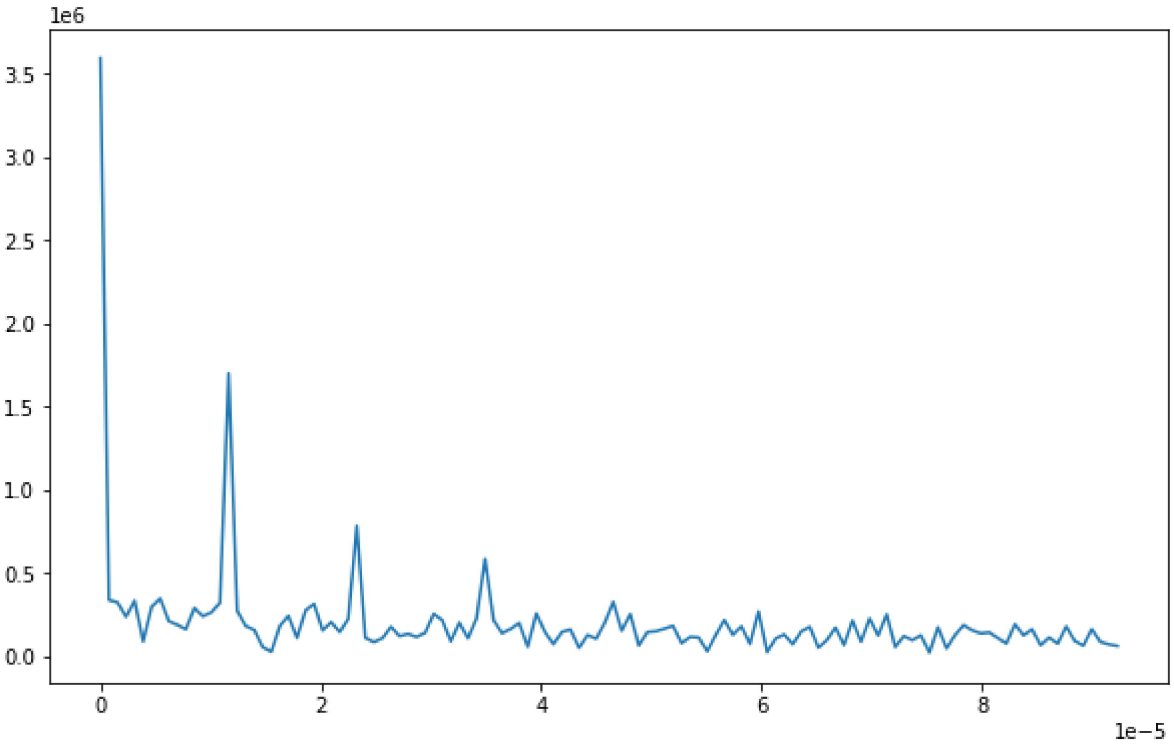


As we can see, it is sensitive to frequencies within the same range as an FFT on an hour lapse, but there is much higher resolution since we have much more data.

In [141]:
```
N = len(condition_dfs['condition_5'].activity)
T = N * 60
fs = 1/60
peaks_arr = abs(np.fft.rfft(condition_dfs['condition_5'].activity))
freq_arr = np.fft.rfftfreq(N, d = 1/fs)
fig20, ax20 = plt.subplots(figsize = (10,6))
sns.lineplot(y=peaks_arr[:120],x=freq_arr[:120])
```

executed in 137ms, finished 00:05:03 2021-07-08

Out[141]: <AxesSubplot:>



In [139]:
```
period_arr[0:5]
```

executed in 16ms, finished 00:01:10 2021-07-08

Out[139]:
```
array([358.21666667, 179.10833333, 119.40555556,  89.55416667,
        71.64333333])
```

Looking through different subjects at random, we find this subject is a really good example demonstrating the FFTs ability to find patterns in time series. The peak at zero is the DC component in frequency (no modulation), but moving to the right we see peaks which occur at frequencies corresponding to roughly 24, 12, and 8 hours, suggesting that this subject has daily oscillations in activity on those time scales.

In [32]:
```
sorted(zip(peaks_arr[freq_arr >= 1e-5],freq_arr[freq_arr >= 1e-5]),reverse = True
```

executed in 14ms, finished 23:02:11 2021-07-07

Out[32]:
```
[(1701369.04892939, 1.1631694039919974e-05),
 (784291.7343873837, 2.326338807983995e-05),
 (584490.3283857745, 3.489508211975992e-05),
 (325972.03110379097, 4.65267761596799e-05),
 (321886.9971762392, 1.0856247770591976e-05),
 (314704.12906671496, 1.9386156733199956e-05),
 (278023.33151189657, 0.00011864327920718374),
 (275914.3356154133, 1.8610710463871958e-05),
 (273559.5644537683, 1.2407140309247973e-05),
 (269634.76481982926, 5.970936273825587e-05)]
```

In [33]:
```
freq_tups = sorted(zip(peaks_arr[freq_arr >= 1e-5],freq_arr[freq_arr >= 1e-5]),re
for tup in freq_tups:
    print(1/tup[1]/60/60)
```

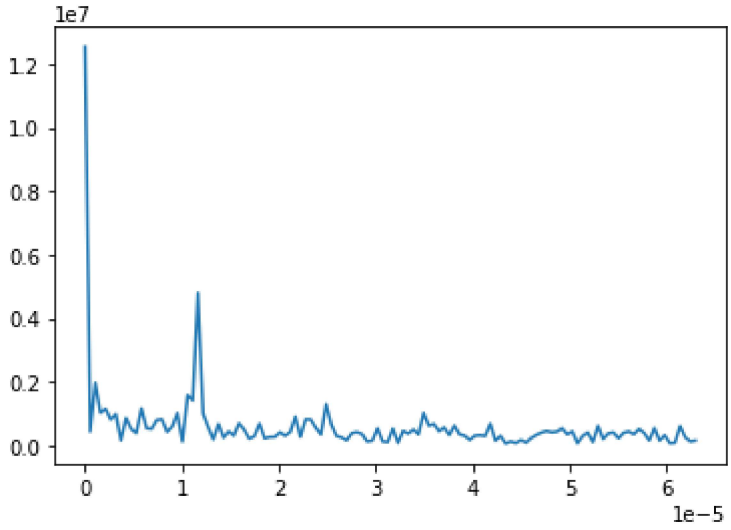
executed in 14ms, finished 23:02:11 2021-07-07

```
23.88111111111111
11.940555555555555
7.96037037037037
5.970277777777778
25.586904761904762
14.328666666666667
2.341285403050109
14.925694444444444
22.388541666666665
4.652164502164502
```

In [34]:
```
N = len(control_dfs['control_2'].activity)
T = N * 60
fs = 1/60
peaks_arr = abs(np.fft.rfft(control_dfs['control_2'].activity))
freq_arr = np.fft.rfftfreq(N, d = 1/fs)
sns.lineplot(y=peaks_arr[:120],x=freq_arr[:120])
```

executed in 124ms, finished 23:02:11 2021-07-07

Out[34]: <AxesSubplot:>



In [35]:
```
sorted(zip(peaks_arr[freq_arr >= 1e-5],freq_arr[freq_arr >= 1e-5]),reverse = True
```

executed in 30ms, finished 23:02:11 2021-07-07

Out[35]:
```
[(4808292.7867452, 1.1650197523803473e-05),
 (1582671.7747243114, 1.0591088658003157e-05),
 (1415594.0608849267, 1.1120643090903316e-05),
 (1298575.6201410312, 2.488905834630742e-05),
 (1019157.7823709416, 3.495059257141042e-05),
 (1004696.3527185008, 1.217975195670363e-05),
 (936112.5978759036, 6.937163070992068e-05),
 (908365.3667217366, 2.1711731748906473e-05),
 (818843.5814752674, 2.2770840614706787e-05),
 (816490.4573502366, 2.3300395047606946e-05)]
```

In [36]:
```
freq_tups = sorted(zip(peaks_arr[freq_arr >= 1e-5],freq_arr[freq_arr >= 1e-5]),re
for tup in freq_tups:
    print(tup)
    print(1/tup[1]/60/60)
```

executed in 30ms, finished 23:02:11 2021-07-07

```
(4808292.7867452, 1.1650197523803473e-05)
23.843181818181815
(1582671.7747243114, 1.0591088658003157e-05)
26.2275
(1415594.0608849267, 1.1120643090903316e-05)
24.978571428571424
(1298575.6201410312, 2.488905834630742e-05)
11.160638297872339
(1019157.7823709416, 3.495059257141042e-05)
7.947727272727271
(1004696.3527185008, 1.217975195670363e-05)
22.80652173913044
(936112.5978759036, 6.937163070992068e-05)
4.004198473282442
(908365.3667217366, 2.1711731748906473e-05)
12.793902439024388
(818843.5814752674, 2.2770840614706787e-05)
12.198837209302324
(816490.4573502366, 2.3300395047606946e-05)
11.921590909090908
```

Looking through subjects at random, we see that most have at least a definied peak near 24h, if not others. I think the magnitude and frequencies at which these peaks occur could be useful features for our model. If the peak magnitude is higher, it indicates a stronger cyclic rhythm. The condition and control patients could have peaks at different frequencies as well. I noticed that sometimes, when listing the 10 highest peaks as above, that some of the listed peaks are simply points on the way up to a peak. Let us thus make sure that we look for the three highest peaks and magnitudes by preventing inclusion of adjacent points.

In [37]:
```
np.where(freq_arr == freq_tups[1][1])[0][0]
```

executed in 15ms, finished 23:02:11 2021-07-07

Out[37]: 20

In [38]:
```
freq_arr[30]
```

executed in 15ms, finished 23:02:11 2021-07-07

Out[38]: 1.5886632987004734e-05

In [39]:
```
def find_peaks(dataframe):
    # conduct FFT
    peaks_arr = abs(np.fft.rfft(dataframe.activity))
    N = len(dataframe.activity)
    fs = 1/60
    freq_arr = np.fft.rfftfreq(N, d = 1/fs)
    freq_tups = sorted(zip(peaks_arr[freq_arr >= 1e-5],freq_arr[freq_arr >= 1e-5]
    # choose 3 highest peak magnitudes as guess for three highest peaks
    peaks_guess = freq_tups[:3]
    # check that none of the bottom two are adjacent to the top one, and that the
    # if an adjaceny is detected, pop that tuple and create a new guess for the t
    checking = 1
    while checking:
        if abs(np.where(peaks_arr == peaks_guess[0][0])[0][0] - np.where(peaks_ar
            freq_tups.pop(1)
            peaks_guess = freq_tups[:3]
        elif abs(np.where(peaks_arr == peaks_guess[0][0])[0][0] - np.where(peaks_
            freq_tups.pop(2)
            peaks_guess = freq_tups[:3]
        elif abs(np.where(peaks_arr == peaks_guess[1][0])[0][0] - np.where(peaks_
            freq_tups.pop(2)
            peaks_guess = freq_tups[:3]
        else:
            checking = 0
    peaks = [0, 0, 0]
    # convert peak frequency to a period in hours for easier interprebility
    for idx, peak in enumerate(peaks_guess):
        peaks[idx] = (peak[0],1/peak[1]/60/60)
    return peaks
```

executed in 14ms, finished 23:02:11 2021-07-07

In [40]:
```
peaks = find_peaks(control_dfs['control_2'])
```

executed in 30ms, finished 23:02:11 2021-07-07

In [41]:
```
peaks
```

executed in 15ms, finished 23:02:11 2021-07-07

Out[41]:
```
[(4808292.7867452, 23.843181818181815),
 (1582671.7747243114, 26.2275),
 (1298575.6201410312, 11.160638297872339)]
```

In [42]:
```
for tup in peaks:
    print(tup)
    print(1/tup[1]/60/60)
```

executed in 15ms, finished 23:02:11 2021-07-07

```
(4808292.7867452, 23.843181818181815)
1.1650197523803473e-05
(1582671.7747243114, 26.2275)
1.0591088658003155e-05
(1298575.6201410312, 11.160638297872339)
2.4889058346307422e-05
```

Okay, that works, so let's do it for every subject and store the peak information in a dataframe.

In [43]:
```
freq_features = ['peak1_mag','peak1_period','peak2_mag','peak2_period','peak3_mag
condis_freq_df = pd.DataFrame(np.zeros((len(condition_dfs),6)),index = cleaned_co
conts_freq_df = pd.DataFrame(np.zeros((len(control_dfs),6)),index = cleaned_contr
```

executed in 15ms, finished 23:02:12 2021-07-07

In [44]:
```
for subject in cleaned_conditions:
    peaks = find_peaks(condition_dfs[subject])
    condis_freq_df.loc[subject] = peaks[0] + peaks[1] + peaks[2]
for subject in cleaned_controls:
    peaks = find_peaks(control_dfs[subject])
    conts_freq_df.loc[subject] = peaks[0] + peaks[1] + peaks[2]
```

executed in 724ms, finished 23:02:12 2021-07-07

In [45]:
```
condis_freq_df
```

executed in 14ms, finished 23:02:12 2021-07-07

Out[45]:

	peak1_mag	peak1_period	peak2_mag	peak2_period	peak3_mag	peak3_perio
condition_1	1.837927e+06	24.212500	4.972789e+05	8.070833	391560.280231	19.37000
condition_10	2.877936e+06	23.950000	7.069326e+05	11.975000	558695.926577	14.96875
condition_11	1.312164e+06	23.947917	7.010827e+05	27.369048	309570.204507	11.97395
condition_12	1.408309e+06	24.607778	3.539134e+05	21.712745	324595.264331	8.02427
condition_13	1.887328e+06	23.990741	9.201943e+05	11.995370	464656.928984	12.70098
condition_14	6.847130e+05	24.051111	2.867833e+05	8.017037	208336.265427	18.98771
condition_15	7.264984e+05	24.254444	3.325649e+05	6.166384	317300.390744	7.57951
condition_16	3.382546e+06	24.050000	9.836883e+05	25.831481	836363.824633	12.02500
condition_17	6.708631e+05	23.923333	2.979942e+05	11.961667	198994.406618	5.43712
condition_18	2.391903e+05	23.718889	1.617192e+05	12.268391	157439.307185	9.61576
condition_19	1.311736e+06	23.590000	4.141134e+05	12.201724	351940.959736	20.81470
condition_2	3.399209e+06	24.028395	9.845114e+05	12.014198	776491.856101	27.03194
condition_20	5.324534e+05	23.932407	2.191471e+05	11.966204	204445.875997	21.53916
condition_21	7.365036e+05	24.389286	3.013924e+05	7.940698	252657.711212	17.97105
condition_22	1.300660e+06	24.191111	4.848342e+05	12.095556	330222.511162	20.15925
condition_23	3.111263e+06	23.852273	6.593330e+05	20.990000	644775.668120	27.61842
condition_3	2.769540e+06	24.053333	8.802276e+05	21.223529	672962.559507	12.02666
condition_4	2.148989e+06	23.951111	8.904265e+05	27.635897	736197.430306	19.95925
condition_5	1.701369e+06	23.881111	7.842917e+05	11.940556	584490.328386	7.96037
condition_6	1.825856e+06	23.814444	4.166744e+05	14.884028	399471.953849	11.90722
condition_7	2.241750e+06	23.098958	1.062790e+06	26.398810	924839.894433	12.74425
condition_8	1.501526e+06	24.742308	4.999245e+05	21.443333	418262.093012	7.31022
condition_9	1.146971e+06	24.188095	6.898314e+05	12.094048	474976.786539	6.91088

In [46]:
```
conts_freq_df
```
executed in 15ms, finished 23:02:12 2021-07-07

Out[46]:

	peak1_mag	peak1_period	peak2_mag	peak2_period	peak3_mag	peak3_period
control_1	1.998631e+06	23.893981	7.872443e+05	22.055983	5.481958e+05	20.48055
control_10	2.334687e+06	24.061111	1.001673e+06	12.030556	8.036311e+05	15.03819
control_11	1.398948e+06	24.286275	4.019575e+05	10.586325	3.515984e+05	3.30293
control_12	1.456694e+06	24.053472	5.309473e+05	12.026736	4.264636e+05	26.24015
control_13	1.238365e+06	24.730000	3.929636e+05	19.523684	3.384860e+05	9.76184
control_14	2.719233e+06	24.732222	1.263626e+06	19.525439	8.414356e+05	10.91127
control_15	1.817826e+06	23.190625	5.748603e+05	7.894681	5.591609e+05	11.96935
control_16	1.876426e+06	24.071569	6.431068e+05	21.537719	4.756707e+05	13.20053
control_17	2.054016e+06	24.087500	6.076937e+05	21.897727	5.380113e+05	26.76388
control_18	2.349425e+06	24.131373	9.078386e+05	12.065686	5.040123e+05	21.59122
control_19	1.374677e+06	24.218421	7.169782e+05	21.911905	6.565635e+05	11.22317
control_2	4.808293e+06	23.843182	1.582672e+06	26.227500	1.298576e+06	11.16063
control_20	3.343185e+06	24.159848	9.821463e+05	12.079924	6.796813e+05	3.99636
control_21	8.647370e+05	24.193939	8.579852e+05	11.088889	8.287454e+05	21.29066
control_22	1.928332e+06	24.097778	6.307342e+05	12.048889	4.240023e+05	10.63137
control_23	1.325033e+06	24.104444	5.022454e+05	12.052222	3.682765e+05	4.01740
control_24	3.297069e+06	24.248889	7.929269e+05	9.326496	7.697170e+05	11.02222
control_25	2.154310e+06	24.242222	1.006125e+06	12.121111	7.428164e+05	21.39019
control_26	3.002526e+06	24.177536	1.078798e+06	18.536111	9.337098e+05	17.37760
control_27	1.874762e+06	24.392857	8.435669e+05	7.941860	8.222942e+05	4.01764
control_28	2.146665e+06	24.054902	6.543511e+05	8.018301	5.769839e+05	12.02745
control_29	2.127040e+06	23.953922	6.981119e+05	7.984641	5.442434e+05	21.43245
control_3	2.015438e+06	23.698188	1.404924e+06	24.775379	7.620980e+05	21.37483
control_30	1.897726e+06	24.076667	4.922189e+05	7.684043	4.688549e+05	11.28593
control_31	3.232657e+06	23.791204	1.121022e+06	25.190686	1.062689e+06	12.06314
control_32	1.032482e+06	23.897685	6.404556e+05	20.483730	3.993965e+05	10.36526
control_4	2.001303e+06	23.829545	7.019674e+05	12.191860	6.899039e+05	21.84375
control_5	2.379547e+06	23.804545	1.033339e+06	12.085385	9.520002e+05	25.34032
control_6	2.577040e+06	23.883796	1.009636e+06	25.288725	9.882853e+05	14.09535
control_7	2.542464e+06	23.837963	1.200311e+06	20.432540	1.139881e+06	12.08685
control_8	2.305062e+06	24.098333	1.460628e+06	12.049167	1.024122e+06	21.90757
control_9	1.312497e+06	24.198333	4.007175e+05	19.358667	3.264383e+05	5.37740

Let's add our target row for classification, `'condition'` and stack them into one big dataframe.

In [47]:
```
condis_freq_df['condition'] = 1
conts_freq_df['condition'] = 0
all_freqs = pd.concat([condis_freq_df,conts_freq_df], axis = 0)
```

executed in 15ms, finished 23:02:12 2021-07-07

In [48]:
```
all_freqs
```

executed in 30ms, finished 23:02:12 2021-07-07

Out[48]:

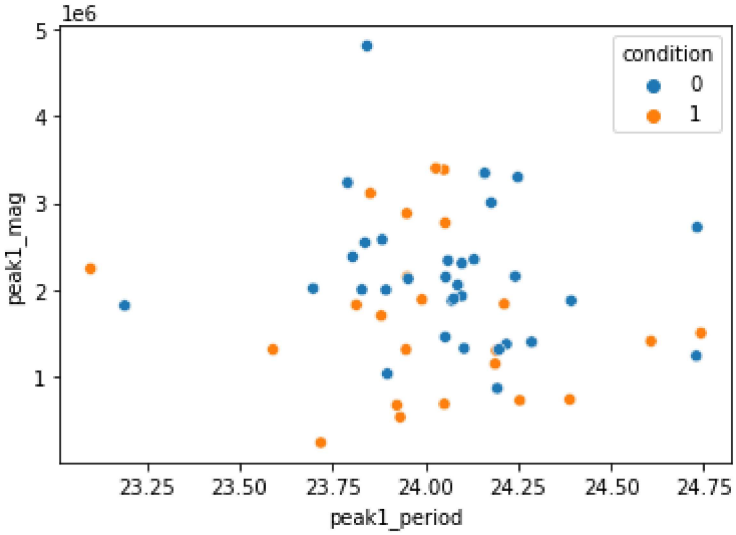
	peak1_mag	peak1_period	peak2_mag	peak2_period	peak3_mag	peak3_per
condition_1	1.837927e+06	24.212500	4.972789e+05	8.070833	3.915603e+05	19.370
condition_10	2.877936e+06	23.950000	7.069326e+05	11.975000	5.586959e+05	14.968
condition_11	1.312164e+06	23.947917	7.010827e+05	27.369048	3.095702e+05	11.973
condition_12	1.408309e+06	24.607778	3.539134e+05	21.712745	3.245953e+05	8.024
condition_13	1.887328e+06	23.990741	9.201943e+05	11.995370	4.646569e+05	12.700
condition_14	6.847130e+05	24.051111	2.867833e+05	8.017037	2.083363e+05	18.987
condition_15	7.264984e+05	24.254444	3.325649e+05	6.166384	3.173004e+05	7.579
condition_16	3.382546e+06	24.050000	9.836883e+05	25.831481	8.363638e+05	12.025
condition_17	6.708631e+05	23.923333	2.979942e+05	11.961667	1.989944e+05	5.437
condition_18	2.391903e+05	23.718889	1.617192e+05	12.268391	1.574393e+05	9.615
condition_19	1.311736e+06	23.590000	4.141134e+05	12.201724	3.519410e+05	20.814
condition_2	3.399209e+06	24.028395	9.845114e+05	12.014198	7.764919e+05	27.031
condition_20	5.324534e+05	23.932407	2.191471e+05	11.966204	2.044459e+05	21.539
condition_21	7.365036e+05	24.389286	3.013924e+05	7.940698	2.526577e+05	17.971
condition_22	1.300660e+06	24.191111	4.848342e+05	12.095556	3.302225e+05	20.159
condition_23	3.111263e+06	23.852273	6.593330e+05	20.990000	6.447757e+05	27.618
condition_3	2.769540e+06	24.053333	8.802276e+05	21.223529	6.729626e+05	12.026
condition_4	2.148989e+06	23.951111	8.904265e+05	27.635897	7.361974e+05	19.959
condition_5	1.701369e+06	23.881111	7.842917e+05	11.940556	5.844903e+05	7.960
condition_6	1.825856e+06	23.814444	4.166744e+05	14.884028	3.994720e+05	11.907
condition_7	2.241750e+06	23.098958	1.062790e+06	26.398810	9.248399e+05	12.744
condition_8	1.501526e+06	24.742308	4.999245e+05	21.443333	4.182621e+05	7.310
condition_9	1.146971e+06	24.188095	6.898314e+05	12.094048	4.749768e+05	6.910
control_1	1.998631e+06	23.893981	7.872443e+05	22.055983	5.481958e+05	20.480
control_10	2.334687e+06	24.061111	1.001673e+06	12.030556	8.036311e+05	15.038
control_11	1.398948e+06	24.286275	4.019575e+05	10.586325	3.515984e+05	3.302
control_12	1.456694e+06	24.053472	5.309473e+05	12.026736	4.264636e+05	26.240
control_13	1.238365e+06	24.730000	3.929636e+05	19.523684	3.384860e+05	9.761
control_14	2.719233e+06	24.732222	1.263626e+06	19.525439	8.414356e+05	10.911
control_15	1.817826e+06	23.190625	5.748603e+05	7.894681	5.591609e+05	11.969
control_16	1.876426e+06	24.071569	6.431068e+05	21.537719	4.756707e+05	13.200
control_17	2.054016e+06	24.087500	6.076937e+05	21.897727	5.380113e+05	26.763
control_18	2.349425e+06	24.131373	9.078386e+05	12.065686	5.040123e+05	21.591

	peak1_mag	peak1_period	peak2_mag	peak2_period	peak3_mag	peak3_per
control_19	1.374677e+06	24.218421	7.169782e+05	21.911905	6.565635e+05	11.223
control_2	4.808293e+06	23.843182	1.582672e+06	26.227500	1.298576e+06	11.160
control_20	3.343185e+06	24.159848	9.821463e+05	12.079924	6.796813e+05	3.996
control_21	8.647370e+05	24.193939	8.579852e+05	11.088889	8.287454e+05	21.290
control_22	1.928332e+06	24.097778	6.307342e+05	12.048889	4.240023e+05	10.631
control_23	1.325033e+06	24.104444	5.022454e+05	12.052222	3.682765e+05	4.017
control_24	3.297069e+06	24.248889	7.929269e+05	9.326496	7.697170e+05	11.022
control_25	2.154310e+06	24.242222	1.006125e+06	12.121111	7.428164e+05	21.390
control_26	3.002526e+06	24.177536	1.078798e+06	18.536111	9.337098e+05	17.377
control_27	1.874762e+06	24.392857	8.435669e+05	7.941860	8.222942e+05	4.017
control_28	2.146665e+06	24.054902	6.543511e+05	8.018301	5.769839e+05	12.027
control_29	2.127040e+06	23.953922	6.981119e+05	7.984641	5.442434e+05	21.432
control_3	2.015438e+06	23.698188	1.404924e+06	24.775379	7.620980e+05	21.374
control_30	1.897726e+06	24.076667	4.922189e+05	7.684043	4.688549e+05	11.285
control_31	3.232657e+06	23.791204	1.121022e+06	25.190686	1.062689e+06	12.063
control_32	1.032482e+06	23.897685	6.404556e+05	20.483730	3.993965e+05	10.365
control_4	2.001303e+06	23.829545	7.019674e+05	12.191860	6.899039e+05	21.843
control_5	2.379547e+06	23.804545	1.033339e+06	12.085385	9.520002e+05	25.340
control_6	2.577040e+06	23.883796	1.009636e+06	25.288725	9.882853e+05	14.095
control_7	2.542464e+06	23.837963	1.200311e+06	20.432540	1.139881e+06	12.086
control_8	2.305062e+06	24.098333	1.460628e+06	12.049167	1.024122e+06	21.907
control_9	1.312497e+06	24.198333	4.007175e+05	19.358667	3.264383e+05	5.377

Sweet. Now we can do some quick scatter plots to see if, based on condition, it looks there there are groupings based on these features.

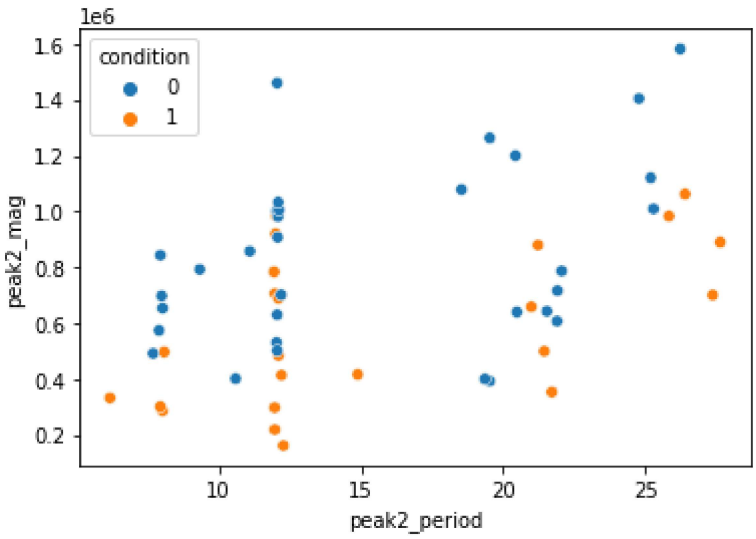
In [49]:
```
sns.scatterplot(data = all_freqs, x = 'peak1_period', y = 'peak1_mag', hue = 'cor
```
executed in 172ms, finished 23:02:12 2021-07-07

Out[49]:
```
<AxesSubplot:xlabel='peak1_period', ylabel='peak1_mag'>
```



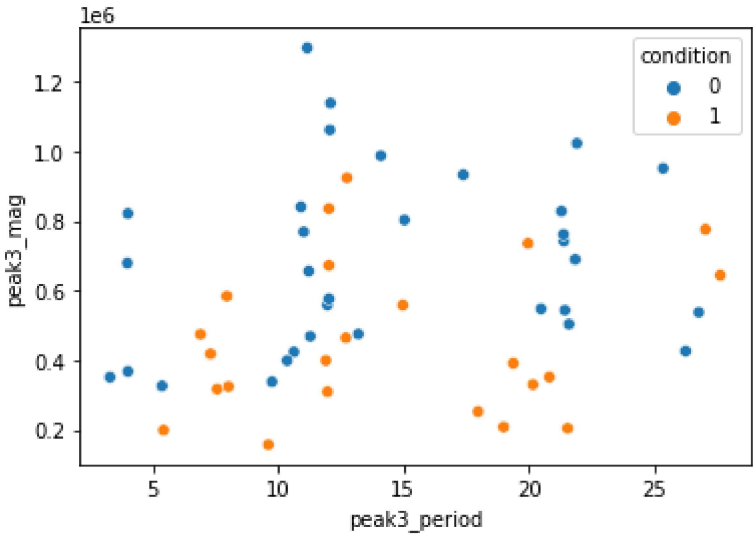
In [50]:
```
sns.scatterplot(data = all_freqs, x = 'peak2_period', y = 'peak2_mag', hue = 'cor
```
executed in 171ms, finished 23:02:13 2021-07-07

Out[50]:
```
<AxesSubplot:xlabel='peak2_period', ylabel='peak2_mag'>
```



In [51]:
```
sns.scatterplot(data = all_freqs, x = 'peak3_period', y = 'peak3_mag', hue = 'cor
```
executed in 158ms, finished 23:02:13 2021-07-07

Out[51]: <AxesSubplot:xlabel='peak3_period', ylabel='peak3_mag'>



While there is a lot of overlap, the afflicted subjects definitely have peaks with lower magnitudes at the same period.

2.4 Determining Sleeping Condition

One other thing the paper does is set nighttime hours as from 9pm to 7am. While we may do this to compare subject hours in which they are resting to when is traditionally considered nighttime, I would like to see if I can figure out a way to classify periods as either wakeful or sleeping. I am going to look at basing it on either a threshold of average hourly activity or hourly median activity. Below I will be calculating each for hourly lapses and comparing the classification performance of the two metrics while tweaking the thresholds.

In [52]:
```
by_hour_conds = condition_dfs['condition_1'].groupby(by = [condition_dfs['conditi
                                                          condition_dfs['condit
                                                          condition_dfs['condit
                axis = 0).agg(['sum','median','mean'],axis = 0)
```
executed in 31ms, finished 23:02:13 2021-07-07

In [53]:
```
by_hour_conds
```
executed in 14ms, finished 23:02:13 2021-07-07

Out[53]:

			activity		
			sum	median	mean
timestamp	timestamp	timestamp			
5	7	12	20793	272.5	346.550000
		13	17074	250.0	284.566667
		14	16751	188.0	279.183333
		15	13127	116.0	218.783333
		16	14313	163.0	238.550000
	...	...	...	...	...
	23	11	0	0.0	0.000000
		12	0	0.0	0.000000
		13	0	0.0	0.000000
		14	684	0.0	11.400000
		15	1908	0.0	79.500000

388 rows × 3 columns

In [54]:
```
by_hour_conds.loc[(5,7,12),('activity','sum')]
```
executed in 14ms, finished 23:02:13 2021-07-07

Out[54]:
```
20793.0
```

In [55]:
```
condi_1 = condition_dfs['condition_1'].copy()
```
executed in 14ms, finished 23:02:13 2021-07-07

In [56]:
```
condi_1['hourly_median'] = condi_1.timestamp.apply(lambda x: by_hour_conds.loc[(x
condi_1['hourly_average'] = condi_1.timestamp.apply(lambda x: by_hour_conds.loc[(
```
executed in 9.04s, finished 23:02:22 2021-07-07

In [57]:
```
condi_1.head(70)
```
executed in 14ms, finished 23:02:22 2021-07-07

Out[57]:

	timestamp	activity	hourly_median	hourly_average
0	2003-05-07 12:00:00	0	272.5	346.550000
1	2003-05-07 12:01:00	143	272.5	346.550000
2	2003-05-07 12:02:00	0	272.5	346.550000
3	2003-05-07 12:03:00	20	272.5	346.550000
4	2003-05-07 12:04:00	166	272.5	346.550000
...	...	...	...	...
65	2003-05-07 13:05:00	160	250.0	284.566667
66	2003-05-07 13:06:00	296	250.0	284.566667
67	2003-05-07 13:07:00	317	250.0	284.566667
68	2003-05-07 13:08:00	338	250.0	284.566667
69	2003-05-07 13:09:00	277	250.0	284.566667

70 rows × 4 columns

In [58]:
```
condi_1['sleeping'] = condi_1['hourly_median'].apply(lambda x: 1 if x == 0 else 0
```
executed in 14ms, finished 23:02:22 2021-07-07

In [59]:
```
condi_1.head(70)
```
executed in 15ms, finished 23:02:22 2021-07-07

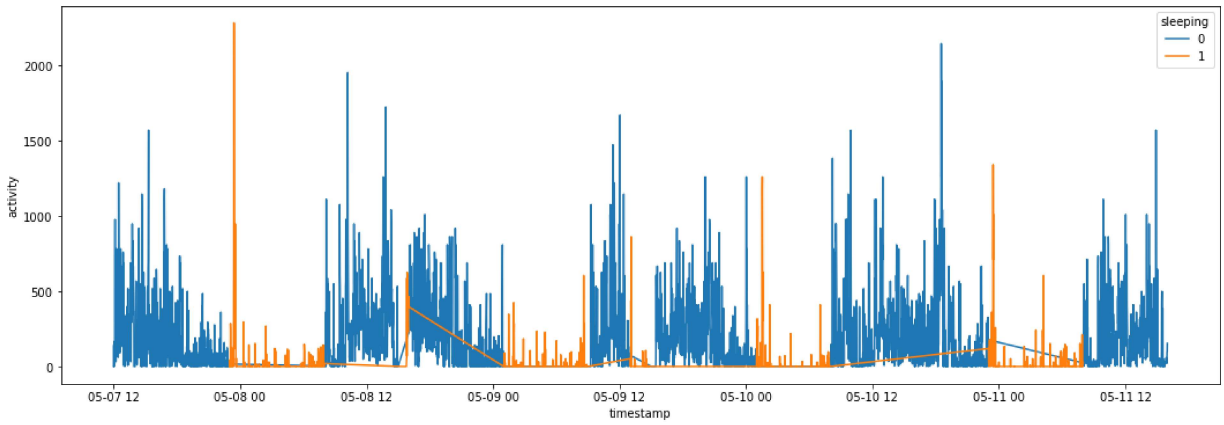
Out[59]:

	timestamp	activity	hourly_median	hourly_average	sleeping
0	2003-05-07 12:00:00	0	272.5	346.550000	0
1	2003-05-07 12:01:00	143	272.5	346.550000	0
2	2003-05-07 12:02:00	0	272.5	346.550000	0
3	2003-05-07 12:03:00	20	272.5	346.550000	0
4	2003-05-07 12:04:00	166	272.5	346.550000	0
...	...	...	...	...	...
65	2003-05-07 13:05:00	160	250.0	284.566667	0
66	2003-05-07 13:06:00	296	250.0	284.566667	0
67	2003-05-07 13:07:00	317	250.0	284.566667	0
68	2003-05-07 13:08:00	338	250.0	284.566667	0
69	2003-05-07 13:09:00	277	250.0	284.566667	0

70 rows × 5 columns

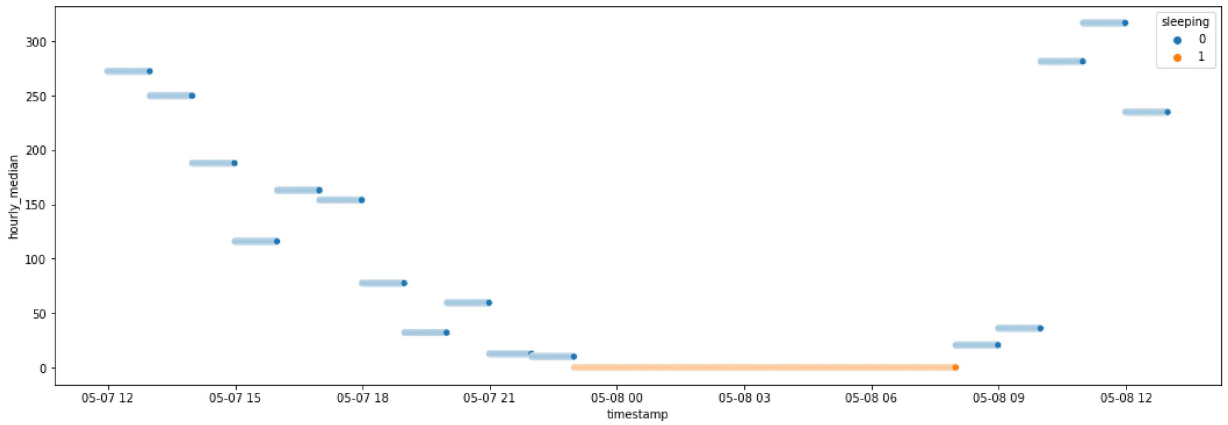
In [60]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = condi_1.iloc[:6000], x = 'timestamp', y = 'activity', hue =
```

executed in 420ms, finished 23:02:22 2021-07-07



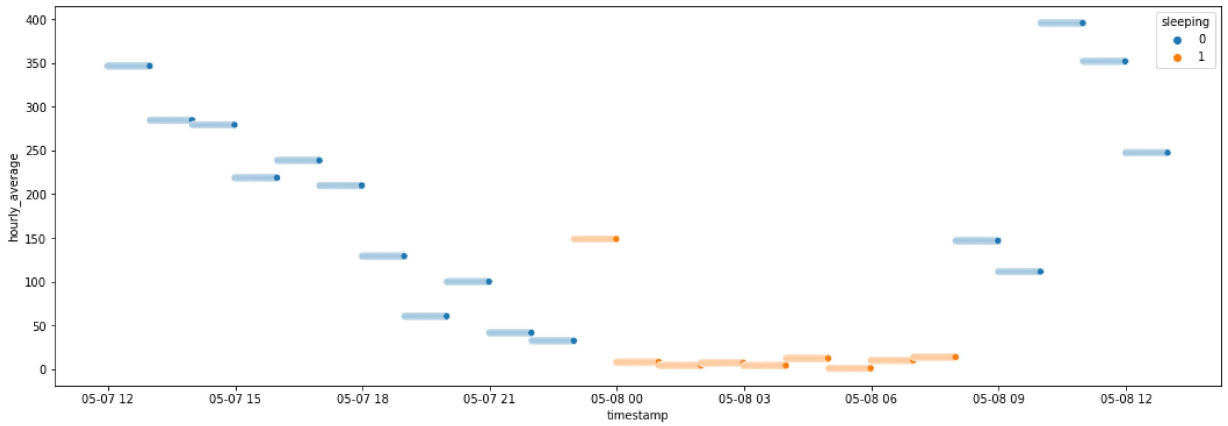
In [61]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.scatterplot(data = condi_1.iloc[:1500], x = 'timestamp', y = 'hourly_median',
```

executed in 234ms, finished 23:02:23 2021-07-07



In [62]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.scatterplot(data = condi_1.iloc[:1500], x = 'timestamp', y = 'hourly_average'
```

executed in 250ms, finished 23:02:23 2021-07-07

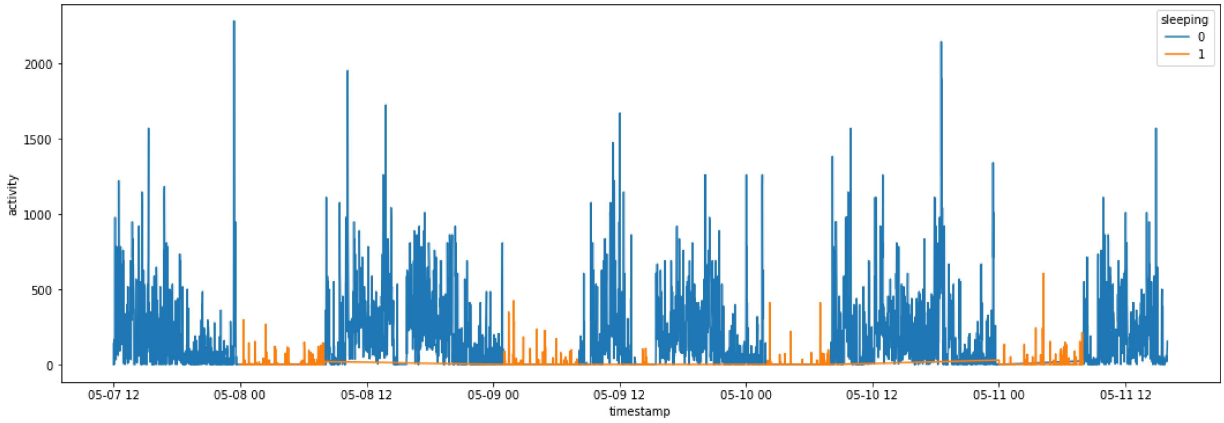


In [63]:
```
by_hour_conds.loc[(5,7,23),('activity','median')]
```
executed in 15ms, finished 23:02:23 2021-07-07

Out[63]:
```
0.0
```

In [64]:
```
condi_1['sleeping'] = condi_1['hourly_average'].apply(lambda x: 1 if x <= 30 else
```
executed in 15ms, finished 23:02:23 2021-07-07

In [65]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = condi_1.iloc[:6000], x = 'timestamp', y = 'activity', hue = '
```
executed in 424ms, finished 23:02:23 2021-07-07



In [66]:
```
cont_1 = control_dfs['control_1'].copy()
by_hour_conts = cont_1.groupby(by = [cont_1.timestamp.dt.month,
                                     cont_1.timestamp.dt.day,
                                     cont_1.timestamp.dt.hour],
                axis = 0).agg(['sum','median','mean'],axis = 0)
```
executed in 30ms, finished 23:02:23 2021-07-07

In [67]:
```
cont_1['hourly_median'] = cont_1.timestamp.apply(lambda x: by_hour_conts.loc[(x.m
cont_1['hourly_average'] = cont_1.timestamp.apply(lambda x: by_hour_conts.loc[(x.
```
executed in 20.2s, finished 23:02:44 2021-07-07

In [68]:
```
cont_1['sleeping'] = cont_1['hourly_average'].apply(lambda x: 1 if x <= 30 else 0
```
executed in 30ms, finished 23:02:44 2021-07-07

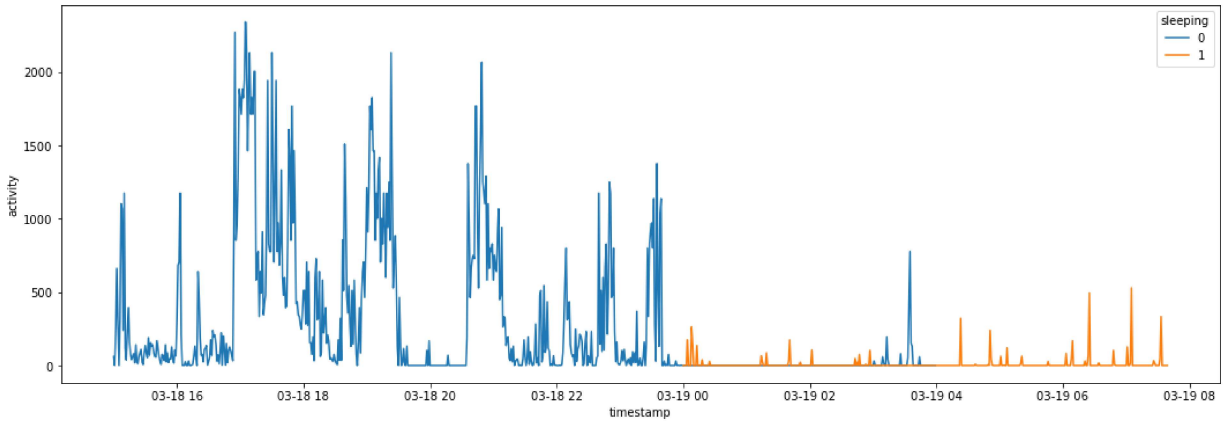
In [69]:
```
cont_1.head(70)
```
executed in 25ms, finished 23:02:44 2021-07-07

Out[69]:

	timestamp	activity	hourly_median	hourly_average	sleeping
0	2003-03-18 15:00:00	60	78.0	156.483333	0
1	2003-03-18 15:01:00	0	78.0	156.483333	0
2	2003-03-18 15:02:00	264	78.0	156.483333	0
3	2003-03-18 15:03:00	662	78.0	156.483333	0
4	2003-03-18 15:04:00	293	78.0	156.483333	0
...	...	...	...	...	...
65	2003-03-18 16:05:00	0	98.0	264.550000	0
66	2003-03-18 16:06:00	3	98.0	264.550000	0
67	2003-03-18 16:07:00	0	98.0	264.550000	0
68	2003-03-18 16:08:00	25	98.0	264.550000	0
69	2003-03-18 16:09:00	0	98.0	264.550000	0

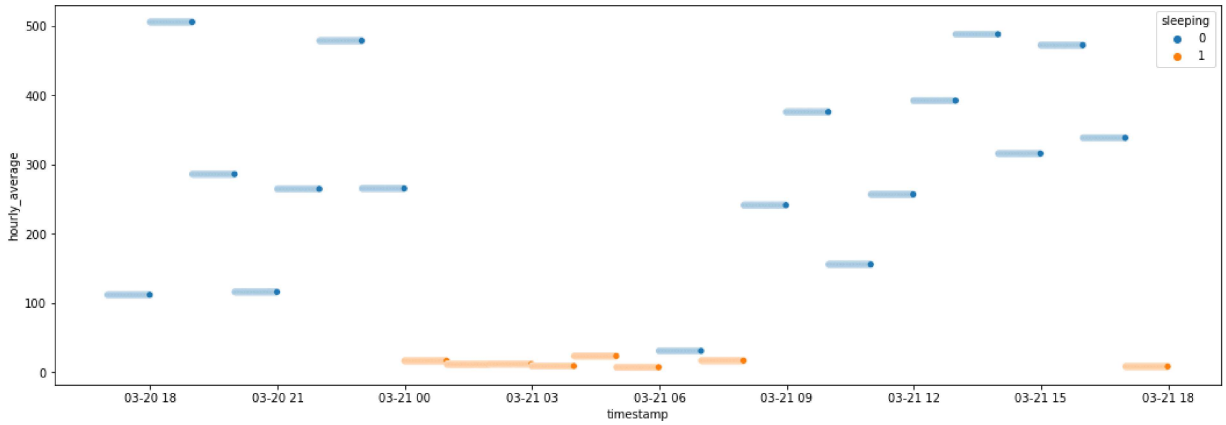
70 rows × 5 columns

In [70]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = cont_1.iloc[:1000], x = 'timestamp', y = 'activity', hue = 's
```
executed in 267ms, finished 23:02:44 2021-07-07



In [71]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.scatterplot(data = cont_1.iloc[3000:4500], x = 'timestamp', y = 'hourly_avera
```

executed in 231ms, finished 23:02:44 2021-07-07

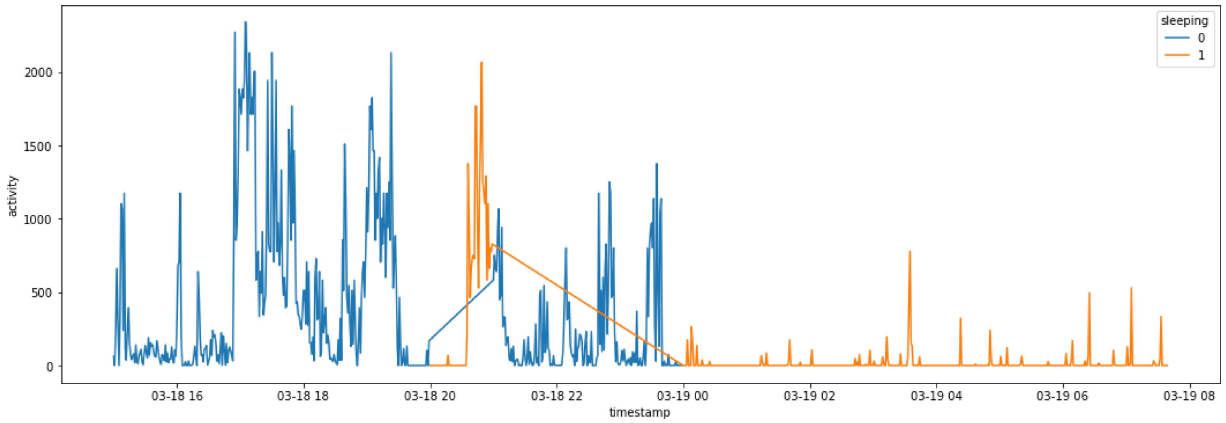


In [72]:
```
cont_1['sleeping'] = cont_1['hourly_median'].apply(lambda x: 1 if x == 0 else 0)
```

executed in 30ms, finished 23:02:44 2021-07-07

In [73]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = cont_1.iloc[:1000], x = 'timestamp', y = 'activity', hue = 's
```

executed in 222ms, finished 23:02:44 2021-07-07



In [74]:
```
cont_1.timestamp.dt.minute
```
executed in 15ms, finished 23:02:44 2021-07-07

Out[74]:
```
0         0
1         1
2         2
3         3
4         4
         ..
51606     6
51607     7
51608     8
51609     9
51610    10
Name: timestamp, Length: 51611, dtype: int64
```

The previous study uses hourly lapses; I can't go too short in time, since the number of samples will be too drastically reduced. It seems worth it to try half an hour lapses, though, to see if we can improve classification near the times when the subject is waking up or falling asleep.

In [75]:
```
cont_1_h = control_dfs['control_1'].copy()
cont_1_h['hour_half'] = cont_1_h.timestamp.apply(lambda x: 1 if x.minute < 30 els
by_half_hour_conts = cont_1_h.groupby(by = [cont_1_h.timestamp.dt.month,
                                            cont_1_h.timestamp.dt.day,
                                            cont_1_h.timestamp.dt.hour,
                                            cont_1_h.hour_half],
                  axis = 0).agg(['sum','median','mean'],axis = 0)
```
executed in 172ms, finished 23:02:45 2021-07-07

In [76]:
```
by_half_hour_conts.head()
```
executed in 27ms, finished 23:02:45 2021-07-07

Out[76]:

				activity		
				sum	median	mean
timestamp	timestamp	timestamp	hour_half			
3	18	15	0	2431	68.0	81.033333
			1	6958	88.0	231.933333
		16	0	10638	118.5	354.600000
			1	5235	57.0	174.500000
		17	0	26947	802.0	898.233333

In [77]:
```
cont_1_h
```
executed in 14ms, finished 23:02:45 2021-07-07

Out[77]:

	timestamp	activity	hour_half
0	2003-03-18 15:00:00	60	1
1	2003-03-18 15:01:00	0	1
2	2003-03-18 15:02:00	264	1
3	2003-03-18 15:03:00	662	1
4	2003-03-18 15:04:00	293	1
...	...	...	...
51606	2003-04-23 12:06:00	3	1
51607	2003-04-23 12:07:00	3	1
51608	2003-04-23 12:08:00	3	1
51609	2003-04-23 12:09:00	3	1
51610	2003-04-23 12:10:00	0	1

51611 rows × 3 columns

In [78]:
```
cont_1_h['hourly_median'] = cont_1_h.apply(lambda x: by_half_hour_conts.loc[(x[0]
cont_1_h['hourly_average'] = cont_1_h.apply(lambda x: by_half_hour_conts.loc[(x[0
```
executed in 22.0s, finished 23:03:07 2021-07-07

In [79]:
```
cont_1_h.head(61)
```
executed in 14ms, finished 23:03:07 2021-07-07

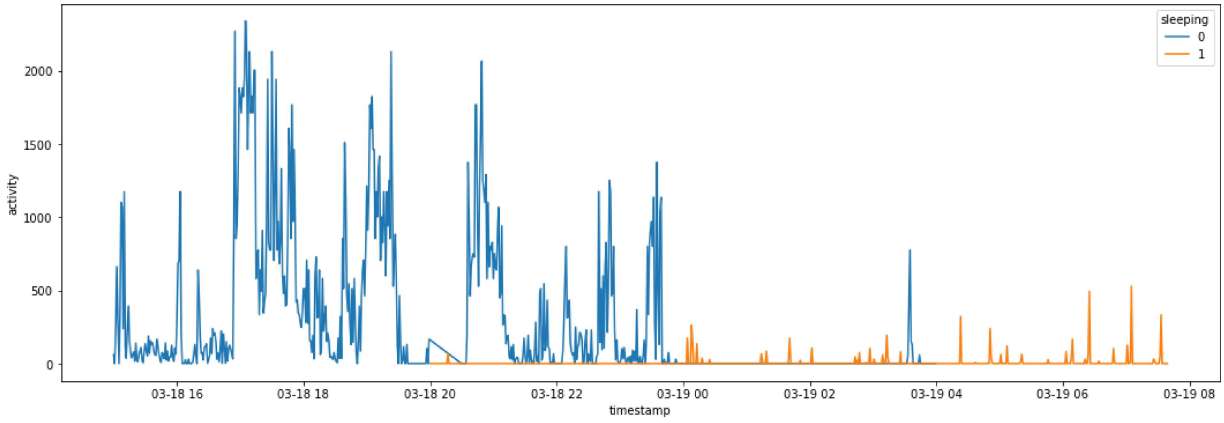
Out[79]:

	timestamp	activity	hour_half	hourly_median	hourly_average
0	2003-03-18 15:00:00	60	1	88.0	231.933333
1	2003-03-18 15:01:00	0	1	88.0	231.933333
2	2003-03-18 15:02:00	264	1	88.0	231.933333
3	2003-03-18 15:03:00	662	1	88.0	231.933333
4	2003-03-18 15:04:00	293	1	88.0	231.933333
...	...	...	...	...	...
56	2003-03-18 15:56:00	42	0	68.0	81.033333
57	2003-03-18 15:57:00	17	0	68.0	81.033333
58	2003-03-18 15:58:00	106	0	68.0	81.033333
59	2003-03-18 15:59:00	66	0	68.0	81.033333
60	2003-03-18 16:00:00	322	1	57.0	174.500000

61 rows × 5 columns

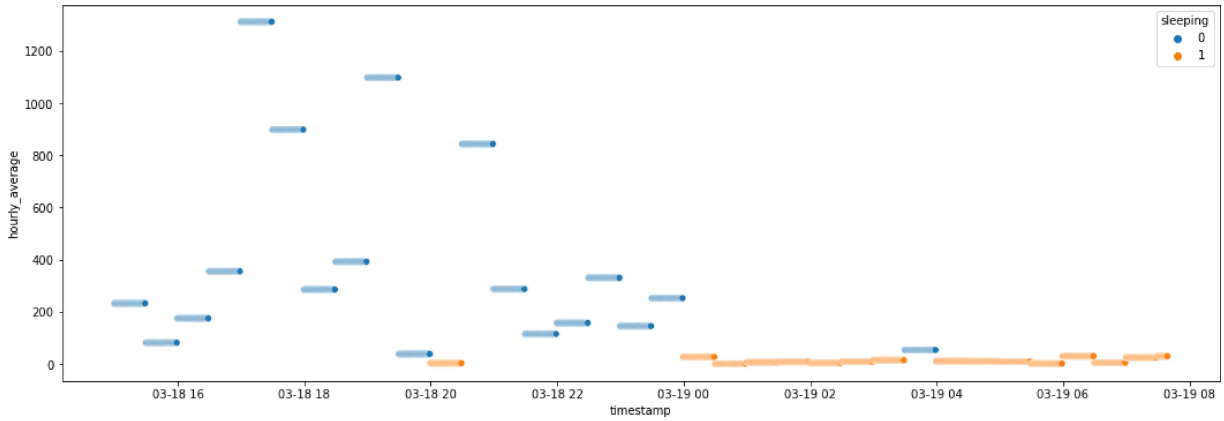
In [80]:
```
cont_1_h['sleeping'] = cont_1_h['hourly_average'].apply(lambda x: 1 if x <= 30 el
```
executed in 31ms, finished 23:03:07 2021-07-07

In [81]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = cont_1_h.iloc[:1000], x = 'timestamp', y = 'activity', hue =
```
executed in 238ms, finished 23:03:07 2021-07-07



In [82]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.scatterplot(data = cont_1_h.iloc[:1000], x = 'timestamp', y = 'hourly_average
```

executed in 234ms, finished 23:03:07 2021-07-07

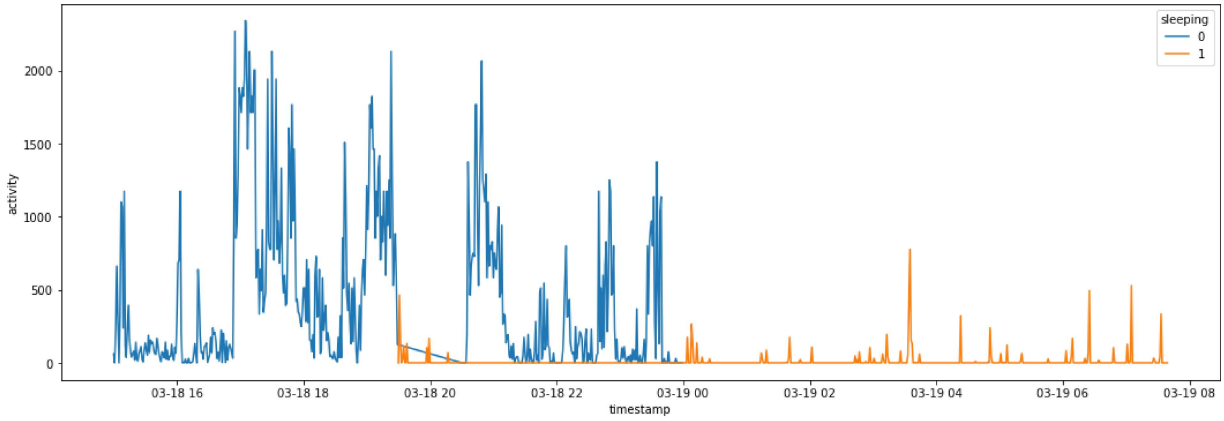


In [83]:
```
cont_1_h['sleeping'] = cont_1_h['hourly_median'].apply(lambda x: 1 if x == 0 else
```

executed in 30ms, finished 23:03:07 2021-07-07

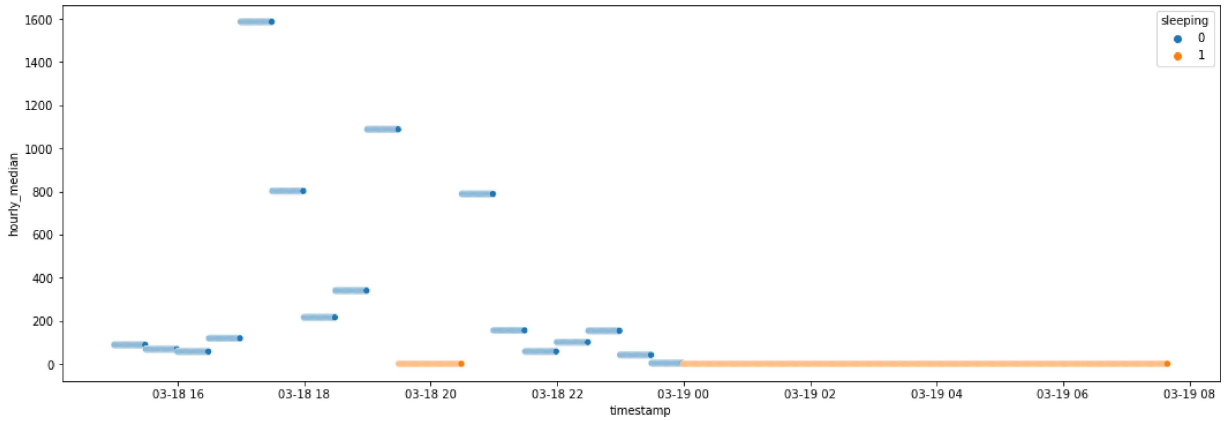
In [84]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = cont_1_h.iloc[:1000], x = 'timestamp', y = 'activity', hue =
```

executed in 217ms, finished 23:03:07 2021-07-07



In [85]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.scatterplot(data = cont_1_h.iloc[:1000], x = 'timestamp', y = 'hourly_median'
```

executed in 232ms, finished 23:03:08 2021-07-07



In [86]:
```
cond_1_h = condition_dfs['condition_1'].copy()
cond_1_h['hour_half'] = cond_1_h.timestamp.apply(lambda x: 1 if x.minute < 30 els
by_half_hour_conds = cond_1_h.groupby(by = [cond_1_h.timestamp.dt.month,
                                            cond_1_h.timestamp.dt.day,
                                            cond_1_h.timestamp.dt.hour,
                                            cond_1_h.hour_half],
                                      axis = 0).agg(['sum','median','mean','max', 'std', 'var', lambda
cond_1_h['hourly_median'] = cond_1_h.apply(lambda x: by_half_hour_conds.loc[(x[0]
cond_1_h['hourly_average'] = cond_1_h.apply(lambda x: by_half_hour_conds.loc[(x[0
```

executed in 10.4s, finished 23:03:18 2021-07-07

In [87]:
```
by_half_hour_conds.loc[(5,8)]
```

executed in 31ms, finished 23:03:18 2021-07-07

Out[87]:

		activity						
		sum	median	mean	max	std	var	<lambda_0>
timestamp	hour_half							
0	0	151	0.0	5.033333	143	26.098597	681.136782	2
	1	329	0.0	10.966667	296	54.112580	2928.171264	3
1	0	30	0.0	1.000000	17	3.464102	12.000000	3
	1	215	0.0	7.166667	154	28.687476	822.971264	4
2	0	112	0.0	3.733333	82	15.763190	248.478161	2
	1	312	0.0	10.400000	268	49.224748	2423.075862	3
3	0	17	0.0	0.566667	11	2.112089	4.460920	3
	1	225	0.0	7.500000	120	28.238821	797.431034	3
4	0	261	0.0	8.700000	111	27.258849	743.044828	5
	1	472	0.0	15.733333	116	32.629512	1064.685057	9
5	0	55	0.0	1.833333	26	6.034860	36.419540	3
	1	3	0.0	0.100000	3	0.547723	0.300000	1
6	0	243	0.0	8.100000	94	24.919664	620.989655	3
	1	357	0.0	11.900000	148	36.943900	1364.851724	3
7	0	614	0.0	20.466667	143	38.864693	1510.464368	12
	1	208	0.0	6.933333	120	23.170929	536.891954	5
8	0	1062	3.0	35.400000	550	102.359273	10477.420690	15
	1	7742	234.5	258.066667	1111	254.405180	64721.995402	25
9	0	3544	96.0	118.133333	454	130.373134	16997.154023	26
	1	3143	20.5	104.766667	1076	233.592289	54565.357471	24
10	0	9969	268.0	332.300000	948	242.295936	58707.320690	30
	1	13770	286.0	459.000000	1954	397.834136	158272.000000	30
11	0	9706	291.0	323.533333	712	157.627613	24846.464368	30
	1	11411	333.0	380.366667	890	169.430178	28706.585057	30
12	0	8168	250.5	272.266667	587	142.912545	20423.995402	30
	1	6672	215.5	222.400000	783	141.269515	19957.075862	30
13	0	13797	361.0	459.900000	1725	406.193291	164992.989655	30
	1	7085	212.0	236.166667	550	139.358897	19420.902299	30
14	0	2435	0.0	81.166667	533	152.919651	23384.419540	13
	1	14358	461.5	478.600000	1042	236.745986	56048.662069	30
15	0	4555	0.0	151.833333	626	222.211313	49377.867816	13

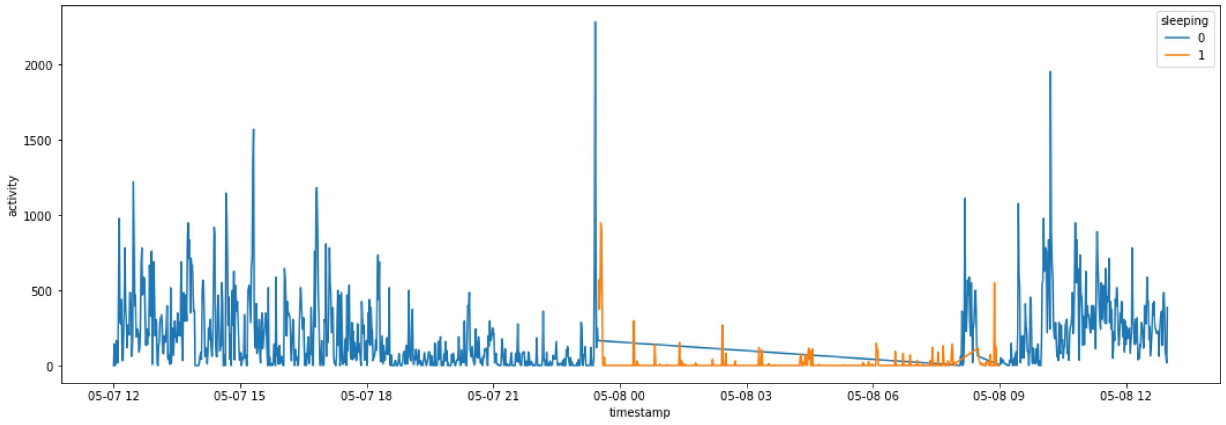
		activity						
		sum	median	mean	max	std	var	<lambda_0>
timestamp	hour_half							
	1	0	0.0	0.000000	0	0.000000	0.000000	0
16	0	11077	361.0	369.233333	890	309.290588	95660.667816	29
	1	11013	306.0	367.100000	809	172.779978	29852.920690	30
17	0	11376	296.0	379.200000	1010	232.129779	53884.234483	30
	1	12302	317.0	410.066667	919	258.925090	67042.202299	30
18	0	8124	204.0	270.800000	712	197.886314	39158.993103	30
	1	10344	317.0	344.800000	759	179.158069	32097.613793	30
19	0	9037	272.5	301.233333	862	228.724301	52314.805747	28
	1	6777	212.0	225.900000	689	193.816541	37564.851724	27
20	0	2209	17.5	73.633333	606	136.027503	18503.481609	22
	1	7650	96.5	255.000000	919	308.986106	95472.413793	23
21	0	5107	151.5	170.233333	689	159.218497	25350.529885	24
	1	2096	27.0	69.866667	568	119.768090	14344.395402	19
22	0	1775	15.5	59.166667	411	96.276571	9269.178161	19
	1	1016	5.5	33.866667	143	49.180690	2418.740230	16
23	0	1799	16.0	59.966667	485	102.686830	10544.585057	23
	1	2273	10.0	75.766667	485	134.852329	18185.150575	21

In [88]:
```
cond_1_h_med = cond_1_h.copy()
cond_1_h_med['sleeping'] = cond_1_h_med['hourly_median'].apply(lambda x: 1 if x
```

executed in 15ms, finished 23:03:18 2021-07-07

In [89]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = cond_1_h_med.iloc[:1500], x = 'timestamp', y = 'activity', hu
```

executed in 252ms, finished 23:03:18 2021-07-07

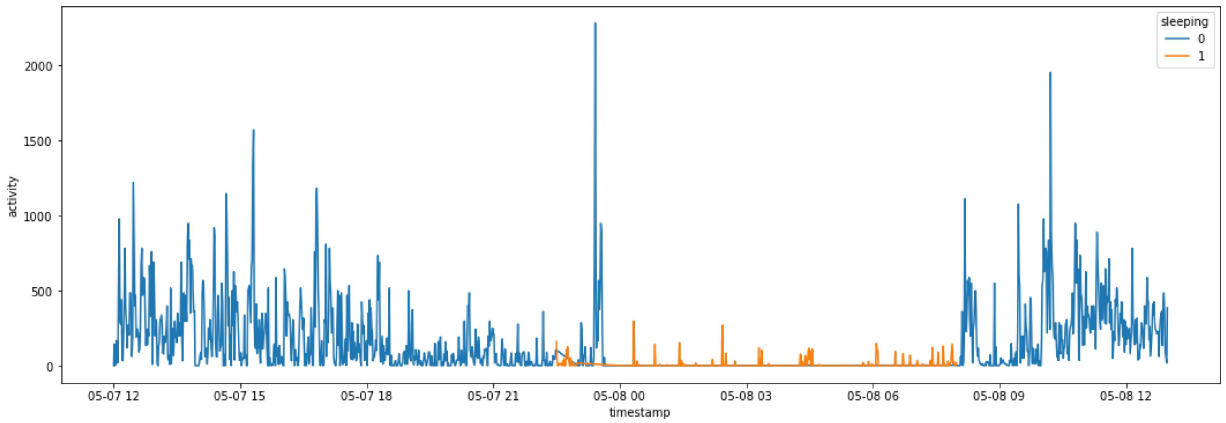


In [90]:
```
cond_1_h_avg = cond_1_h.copy()
cond_1_h_avg['sleeping'] = cond_1_h_avg['hourly_average'].apply(lambda x: 1 if x
```

executed in 15ms, finished 23:03:18 2021-07-07

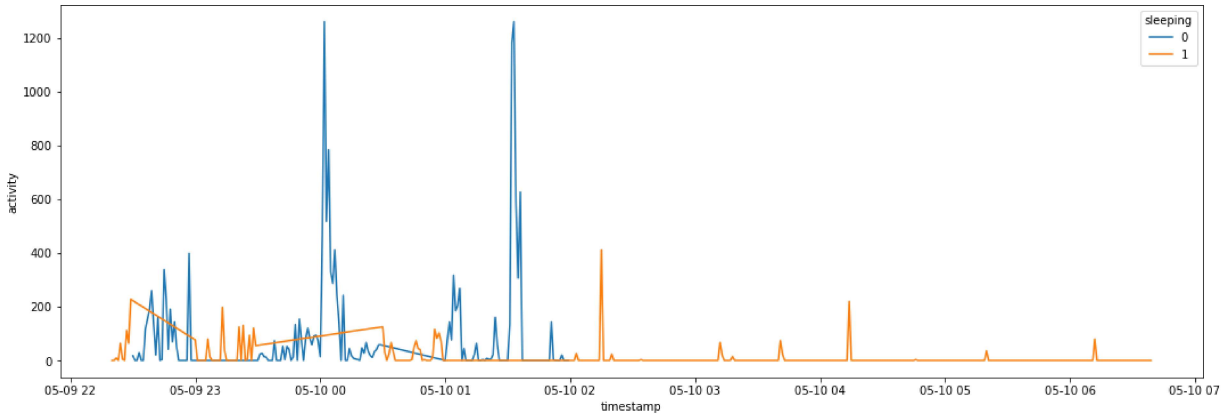
In [91]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = cond_1_h_avg.iloc[:1500], x = 'timestamp', y = 'activity', hu
```

executed in 251ms, finished 23:03:19 2021-07-07



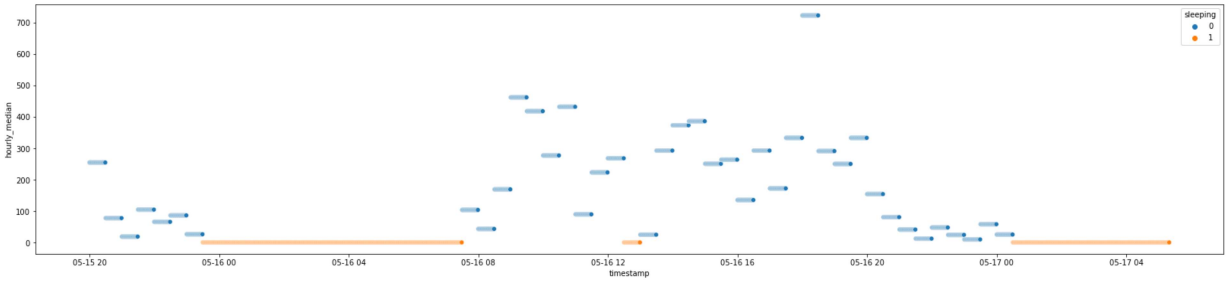
In [92]:
```
fig9, ax9 = plt.subplots(figsize = (18,6));
sns.lineplot(data = cond_1_h_avg.iloc[3500:4000], x = 'timestamp', y = 'activity'
```

executed in 232ms, finished 23:03:19 2021-07-07



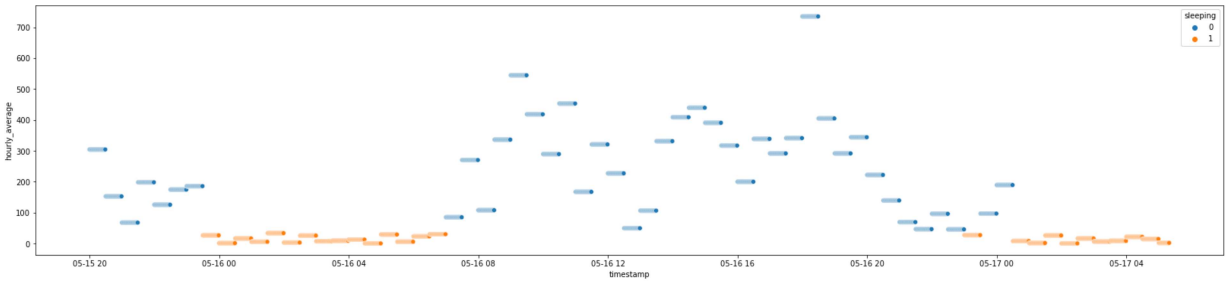
In [93]:
```
fig9, ax9 = plt.subplots(figsize = (28,6));
sns.scatterplot(data = cond_1_h_med.iloc[12000:14000], x = 'timestamp', y = 'hour
```

executed in 248ms, finished 23:03:19 2021-07-07



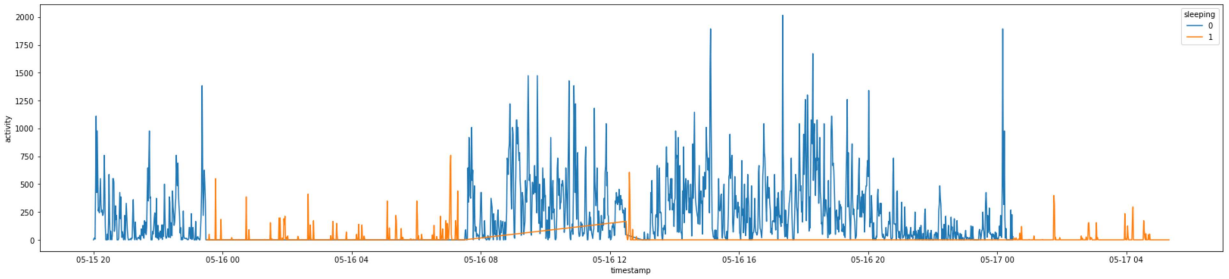
In [94]:
```
fig9, ax9 = plt.subplots(figsize = (28,6));
sns.scatterplot(data = cond_1_h_avg.iloc[12000:14000], x = 'timestamp', y = 'hour
```

executed in 294ms, finished 23:03:19 2021-07-07



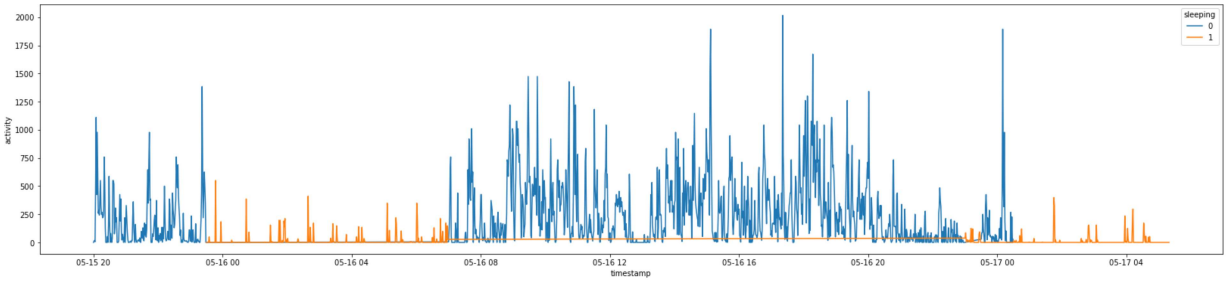
In [95]:
```
fig9, ax9 = plt.subplots(figsize = (28,6));
sns.lineplot(data = cond_1_h_med.iloc[12000:14000], x = 'timestamp', y = 'activit
```

executed in 299ms, finished 23:03:20 2021-07-07



In [96]:
```
fig9, ax9 = plt.subplots(figsize = (28,6));
sns.lineplot(data = cond_1_h_avg.iloc[12000:14000], x = 'timestamp', y = 'activit
```

executed in 293ms, finished 23:03:20 2021-07-07



Using the above four plots, I meticulously combed through the actigraphy series for `'condition_1'` while looking at both the median and average thresholds. I have determined that, while they are individually often wrong in a way that is challenging for using a singular threshold, they are actually wrong nearly all the time in different half hour segments while when they both overlap in classifying the person as asleep it is a solid looking classification. Thus, moving forward, we will use the intersection between the two `True` groups resulting from the application of each threshold.

In [97]:
```
by_half_hour_conds.loc[(5,21,19),('activity','mean')]
```

executed in 15ms, finished 23:03:20 2021-07-07

Out[97]:
```
hour_half
0    153.333333
1     83.433333
Name: (activity, mean), dtype: float64
```

In [98]:
```
cond_2_h = condition_dfs['condition_2'].copy()
cond_2_h['hour_half'] = cond_2_h.timestamp.apply(lambda x: 1 if x.minute < 30 els
by_half_hour_cond_2 = cond_2_h.groupby(by = [cond_2_h.timestamp.dt.month,
                                     cond_2_h.timestamp.dt.day,
                                     cond_2_h.timestamp.dt.hour,
                                     cond_2_h.hour_half],
                  axis = 0).agg(['sum','median','mean','max', 'std', 'var', ('restf
by_half_hour_cond_2['zero_med_bool'] = by_half_hour_cond_2.loc[:,('activity','med
by_half_hour_cond_2['avg_thresh_bool'] = by_half_hour_cond_2.loc[:,('activity','m
by_half_hour_cond_2['resting'] = by_half_hour_cond_2.apply(lambda x: 1 if ((x[8]
by_half_hour_cond_2['night'] = by_half_hour_cond_2.apply(lambda x: 1 if ((x.name[
cond_2_h['zero_med_bool'] = cond_2_h.apply(lambda x: by_half_hour_cond_2.loc[(x[0
cond_2_h['avg_thresh_bool'] = cond_2_h.apply(lambda x: by_half_hour_cond_2.loc[(x
cond_2_h['resting'] = cond_2_h.apply(lambda x: by_half_hour_cond_2.loc[(x[0].mont
```

executed in 26.4s, finished 23:03:46 2021-07-07

In [99]:
```
by_half_hour_cond_2.iloc[0].name[2]
```

executed in 15ms, finished 23:03:46 2021-07-07

Out[99]: 15

In [100]:
```
by_half_hour_cond_2.loc[(5,8),:]
```
executed in 38ms, finished 23:03:46 2021-07-07

Out[100]:

		activity						
		sum	median	mean	max	std	var	restful_min
timestamp	hour_half							
0	0	919	24.0	30.633333	316	56.052613	3141.895402	1
	1	613	4.0	20.433333	250	45.825519	2099.978161	1
1	0	790	4.0	26.333333	349	74.094224	5489.954023	2
	1	745	3.5	24.833333	337	69.188938	4787.109195	2
2	0	441	3.0	14.700000	268	49.404837	2440.837931	2
	1	105	3.0	3.500000	9	1.137147	1.293103	2
3	0	127	4.0	4.233333	15	2.062528	4.254023	2
	1	143	4.0	4.766667	35	5.727630	32.805747	2
4	0	4494	21.0	149.800000	945	212.003318	44945.406897	1
	1	1210	4.0	40.333333	316	77.221729	5963.195402	1
5	0	1457	6.5	48.566667	549	118.032924	13931.771264	2
	1	7990	201.0	266.333333	1515	267.289632	71443.747126	
6	0	11713	372.0	390.433333	1006	288.095013	82998.736782	
	1	16897	483.0	563.233333	1886	379.167208	143767.771264	
7	0	4885	101.0	162.833333	515	139.181123	19371.385057	
	1	14742	349.0	491.400000	1665	415.264071	172444.248276	
8	0	9844	185.5	328.133333	1295	318.977000	101746.326437	
	1	14624	379.5	487.466667	1255	416.530392	173497.567816	
9	0	14741	460.5	491.366667	2008	457.982719	209748.171264	
	1	15288	468.0	509.600000	1178	361.990150	131036.868966	
10	0	33094	873.0	1103.133333	3736	831.792196	691878.257471	
	1	9991	343.0	333.033333	1073	259.312019	67242.722989	
11	0	6722	103.0	224.066667	733	229.282379	52570.409195	
	1	13233	354.5	441.100000	1515	345.542387	119399.541379	
12	0	12092	360.0	403.066667	1295	323.385105	104577.926437	
	1	6956	126.0	231.866667	781	238.051970	56668.740230	
13	0	13089	390.5	436.300000	1295	294.007759	86440.562069	
	1	17756	360.5	591.866667	1829	491.058865	241138.809195	
14	0	15158	316.5	505.266667	2008	466.404289	217532.960920	
	1	8658	208.5	288.600000	915	242.086135	58605.696552	
15	0	17981	445.5	599.366667	2205	498.449837	248452.240230	

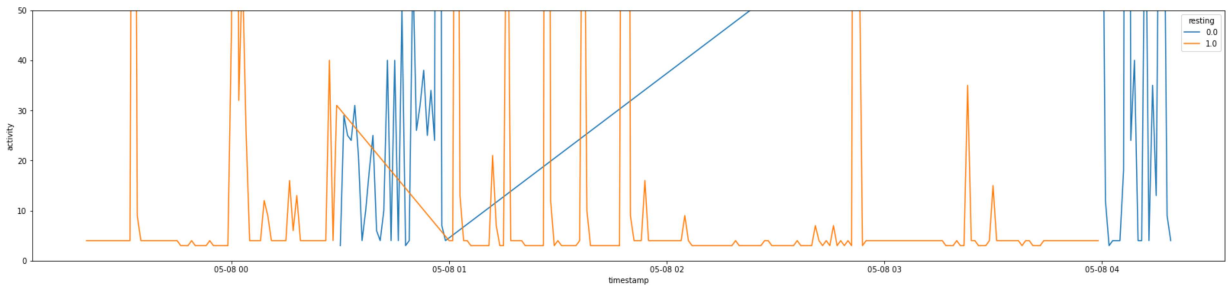
		activity						
		sum	median	mean	max	std	var	restful_min
timestamp	hour_half							
	1	11576	285.0	385.866667	1178	349.623910	122236.878161	
16	0	6848	187.0	228.266667	859	203.811937	41539.305747	
	1	16356	605.0	545.200000	1039	286.887747	82304.579310	
17	0	7610	141.5	253.666667	915	247.427918	61220.574713	
	1	8370	181.5	279.000000	945	258.507120	66825.931034	
18	0	3033	40.5	101.100000	360	115.221631	13276.024138	
	1	6859	70.5	228.633333	1295	313.227524	98111.481609	
19	0	7023	56.0	234.100000	1295	383.385315	146984.300000	
	1	1842	14.0	61.400000	384	91.236940	8324.179310	
20	0	986	11.0	32.866667	277	60.491512	3659.222989	1
	1	1293	16.0	43.100000	178	50.106335	2510.644828	
21	0	3717	4.0	123.900000	1039	270.540882	73192.368966	1
	1	20435	636.5	681.166667	1515	431.637147	186310.626437	
22	0	853	4.0	28.433333	384	79.190161	6271.081609	2
	1	307	4.0	10.233333	74	18.191857	330.943678	2
23	0	90	3.0	3.000000	3	0.000000	0.000000	3
	1	193	4.0	6.433333	79	13.753202	189.150575	2

Seems to be working. Now let's look through a different subject and see how we feel about the performance of the classification. I have renamed the 'sleeping' feature to 'resting' since we can't technically be sure they are asleep, but only that their activity is such that they are minimally active.

In [101]:
```
fig9, ax9 = plt.subplots(figsize = (28,6));
sns.lineplot(data = cond_2_h.iloc[500:800], x = 'timestamp', y = 'activity', hue
ax9.set_ylim((0,50))
```

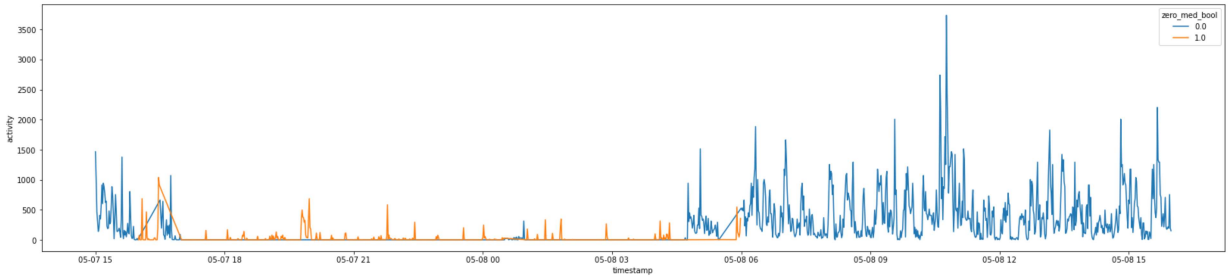
executed in 195ms, finished 23:03:47 2021-07-07

Out[101]: (0.0, 50.0)



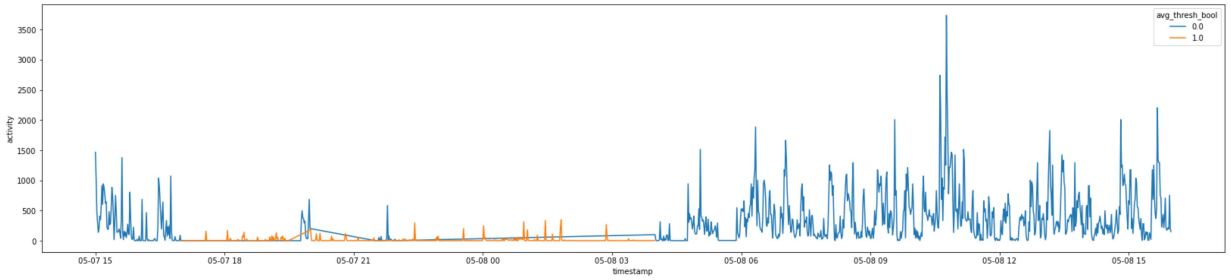
In [102]:
```
fig9, ax9 = plt.subplots(figsize = (28,6));
sns.lineplot(data = cond_2_h.iloc[:1500], x = 'timestamp', y = 'activity', hue =
```

executed in 281ms, finished 23:03:47 2021-07-07



In [103]:
```
fig9, ax9 = plt.subplots(figsize = (28,6));
sns.lineplot(data = cond_2_h.iloc[:1500], x = 'timestamp', y = 'activity', hue =
```

executed in 266ms, finished 23:03:47 2021-07-07



I am noticing here that for `'condition_2'` the flat parts don't have zero activity as they did for condition one. Let's look at how many of the subjects this might be true for.

In [104]:
```
cond_2_h[cond_2_h.activity <= 10].activity.value_counts()
```

executed in 14ms, finished 23:03:47 2021-07-07

Out[104]:
```
4     8484
9     6450
7     4487
6     1799
10    1404
3     1322
0        2
Name: activity, dtype: int64
```

In [105]:
```
for subject in cleaned_conditions:
    print("---------------------------------")
    print(f"Subject: {subject}")
    df = condition_dfs[subject]
    print(df[df.activity <=10].activity.value_counts())
    print("---------------------------------")
for subject in cleaned_controls:
    print("---------------------------------")
    print(f"Subject: {subject}")
    df = control_dfs[subject]
    print(df[df.activity <=10].activity.value_counts())
    print("---------------------------------")
```

executed in 123ms, finished 23:03:47 2021-07-07

```
---------------------------------
Subject: condition_1
0    10902
3      309
8      160
5      146
9      140
6       77
Name: activity, dtype: int64
---------------------------------
---------------------------------
Subject: condition_10
0     7806
3      145
4       76
7       64
9       60
10      51
6       35
```

Hmmm.. so it is hard to say, really. It could be device error, or it could be that the person truly never stops moving during that hour, even when resting. However, that the flat areas are so flat, it makes me think that there is an error in measurement due to device inaccuracy. Thus, instead of using median activity = 0 for resting, based on the above, I feel okay setting the median requirement for resting at less than or equal to 7 activity. It still seems to perform as well as the zero median case for subjects with zero activity values during resting but drastically improves performance for subjects whose activity has less or no zero values.

2.5 Constructing Features in Time Domain

Now it is time to construct our dfs with aggregate features by half hourly lapse. We'll want to include all the aggregate features the paper found to be important, plus a few of our own that I expect to have some use based on my limited but non-zero domain knowledge. So, besides the statistical measures, we will include columns for restful minutes within the half hour, zero activity minutes within the half hour, a boolean resting column, and a boolean column based on whether

the half hour falls into the traditional nighttime period as defined in the most recent paper. I also feel as if there is enough data to eliminate half hours at the beginning or end of a dataset which may not be fully thirty minutes. The function below will construct the desired dataframe.

In [106]:
```
def perform_agg(df_orig):
    df = df_orig.copy()
    # 1 and 2 as in first and second half of the hour
    df['hour_half'] = df.timestamp.apply(lambda x: 1 if x.minute < 30 else 2 )
    grouped_df = df.groupby(by = [df.timestamp.dt.month,
                                  df.timestamp.dt.day,
                                  df.timestamp.dt.hour,
                                  df.hour_half],
                axis = 0).agg(['sum','median','mean','max', 'std', 'var', 'count'
    grouped_df = grouped_df.drop(grouped_df[grouped_df[('activity','count')] < 30
    grouped_df['zero_med_bool'] = grouped_df.loc[:,('activity','median')].apply(l
    grouped_df['avg_thresh_bool'] = grouped_df.loc[:,('activity','mean')].apply(l
    grouped_df['resting'] = grouped_df.apply(lambda x: 1 if ((x[9] == 1) & (x[10]
    grouped_df = grouped_df.drop(columns = ['zero_med_bool','avg_thresh_bool',('a
    grouped_df['night'] = grouped_df.apply(lambda x: 1 if ((x.name[2] >= 21) | (x
    return grouped_df
```

executed in 14ms, finished 23:03:47 2021-07-07

In [107]:
```
perform_agg(condition_dfs['condition_2']).head(60)
```

executed in 531ms, finished 23:03:48 2021-07-07

Out[107]:

				activity					
				sum	median	mean	max	std	var
timestamp	timestamp	timestamp	hour_half						
5	7	15	1	16197	483.0	539.900000	1468	320.033764	1024
			2	5396	120.5	179.866667	1379	278.224510	774
		16	1	3608	4.0	120.266667	1039	277.117295	767
			2	5206	75.0	173.533333	1073	251.264489	631
		17	1	105	3.0	3.500000	7	0.820008	
			2	252	3.0	8.400000	160	28.642385	8
		18	1	643	4.0	21.433333	172	41.471788	17
			2	219	4.0	7.300000	62	12.020528	1
		19	1	804	6.0	26.800000	134	33.434857	11
			2	3743	4.0	124.766667	687	189.416469	358
		20	1	636	4.0	21.200000	184	44.241812	19
			2	382	4.0	12.733333	116	28.106857	7
		21	1	244	4.0	8.133333	53	12.243882	1
			2	1085	6.0	36.166667	586	106.165155	112
		22	1	543	4.0	18.100000	296	53.239440	28
			2	277	3.0	9.233333	79	18.501973	3
		23	1	117	4.0	3.900000	4	0.305129	
			2	314	4.0	10.466667	205	36.758047	13
	8	0	1	613	4.0	20.433333	250	45.825519	20
			2	919	24.0	30.633333	316	56.052613	31
		1	1	745	3.5	24.833333	337	69.188938	47
			2	790	4.0	26.333333	349	74.094224	54
		2	1	105	3.0	3.500000	9	1.137147	
			2	441	3.0	14.700000	268	49.404837	24
		3	1	143	4.0	4.766667	35	5.727630	
			2	127	4.0	4.233333	15	2.062528	
		4	1	1210	4.0	40.333333	316	77.221729	59
			2	4494	21.0	149.800000	945	212.003318	449
		5	1	7990	201.0	266.333333	1515	267.289632	714
			2	1457	6.5	48.566667	549	118.032924	139
		6	1	16897	483.0	563.233333	1886	379.167208	1437

				activity					
				sum	median	mean	max	std	var
timestamp	timestamp	timestamp	hour_half						
			2	11713	372.0	390.433333	1006	288.095013	829
		7	1	14742	349.0	491.400000	1665	415.264071	1724
			2	4885	101.0	162.833333	515	139.181123	193
		8	1	14624	379.5	487.466667	1255	416.530392	1734
			2	9844	185.5	328.133333	1295	318.977000	1017
		9	1	15288	468.0	509.600000	1178	361.990150	1310
			2	14741	460.5	491.366667	2008	457.982719	2097
		10	1	9991	343.0	333.033333	1073	259.312019	672
			2	33094	873.0	1103.133333	3736	831.792196	6918
		11	1	13233	354.5	441.100000	1515	345.542387	1193
			2	6722	103.0	224.066667	733	229.282379	525
		12	1	6956	126.0	231.866667	781	238.051970	566
			2	12092	360.0	403.066667	1295	323.385105	1045
		13	1	17756	360.5	591.866667	1829	491.058865	2411
			2	13089	390.5	436.300000	1295	294.007759	864
		14	1	8658	208.5	288.600000	915	242.086135	586
			2	15158	316.5	505.266667	2008	466.404289	2175
		15	1	11576	285.0	385.866667	1178	349.623910	1222
			2	17981	445.5	599.366667	2205	498.449837	2484
		16	1	16356	605.0	545.200000	1039	286.887747	823
			2	6848	187.0	228.266667	859	203.811937	415
		17	1	8370	181.5	279.000000	945	258.507120	668
			2	7610	141.5	253.666667	915	247.427918	612
		18	1	6859	70.5	228.633333	1295	313.227524	981
			2	3033	40.5	101.100000	360	115.221631	132
		19	1	1842	14.0	61.400000	384	91.236940	83
			2	7023	56.0	234.100000	1295	383.385315	1469
		20	1	1293	16.0	43.100000	178	50.106335	25
			2	986	11.0	32.866667	277	60.491512	36

Seems to work, now let's do it for all patients.

In [108]:
```
grouped_condition_dfs = {}
for subject in cleaned_conditions:
    grouped_df = perform_agg(condition_dfs[subject])
    grouped_condition_dfs.update({subject: grouped_df})
grouped_control_dfs = {}
for subject in cleaned_controls:
    grouped_df = perform_agg(control_dfs[subject])
    grouped_control_dfs.update({subject: grouped_df})
```

executed in 20.3s, finished 23:04:08 2021-07-07

In [109]:
```
grouped_condition_dfs['condition_10'].head()
```

executed in 14ms, finished 23:04:08 2021-07-07

Out[109]:

				activity					
				sum	median	mean	max	std	var
timestamp	timestamp	timestamp	hour_half						
8	31	9	1	175	0.0	5.833333	91	17.855905	318.83
			2	32	0.0	1.066667	29	5.304086	28.13
		10	1	107	0.0	3.566667	41	11.028750	121.63
			2	188	0.0	6.266667	50	14.112683	199.16
		11	1	1632	0.0	54.400000	1178	217.904851	47482.52

In [110]:
```
grouped_control_dfs['control_10'].head()
```

executed in 15ms, finished 23:04:08 2021-07-07

Out[110]:

				activity					
				sum	median	mean	max	std	var
timestamp	timestamp	timestamp	hour_half						
11	18	9	1	311	3.0	10.366667	130	26.076126	679.9
			2	61	3.0	2.033333	8	2.189053	4.7
		10	1	90	3.0	3.000000	3	0.000000	0.0
			2	104	3.0	3.466667	17	2.556039	6.5
		11	1	4837	3.0	161.233333	759	263.161244	69253.8

Sick!

2.6 Combining time and frequency features into Dataframe for Model

So, now we need the model to be able to look at both the time and frequency information. However, because of how the features have been constructed, they are built on different time

scales. So, based on a suggestion during office hours, I will simply repeat the frequency features, which only occur once per subject, in every row of the time-feature-based dataframes belonging to that subject.

Based on some warnings I got and internet searching, I need to first make the frequency dataframes multiindex as are the time dataframes, then merge the two on the participant index.

In [111]:
```
all_freqs.loc['condition_1']
```
executed in 15ms, finished 23:04:08 2021-07-07

Out[111]:
```
peak1_mag       1.837927e+06
peak1_period    2.421250e+01
peak2_mag       4.972789e+05
peak2_period    8.070833e+00
peak3_mag       3.915603e+05
peak3_period    1.937000e+01
condition       1.000000e+00
Name: condition_1, dtype: float64
```

In [112]:
```
list(all_freqs.columns)
```
executed in 15ms, finished 23:04:08 2021-07-07

Out[112]:
```
['peak1_mag',
 'peak1_period',
 'peak2_mag',
 'peak2_period',
 'peak3_mag',
 'peak3_period',
 'condition']
```

In [113]:
```
condition_df = grouped_condition_dfs['condition_1']
condition_df['subject'] = 'condition_1'
all_freqs.columns = pd.MultiIndex.from_product([list(all_freqs.columns),['']])
```
executed in 15ms, finished 23:04:08 2021-07-07

In [114]:
```
list(all_freqs.columns)
```
executed in 14ms, finished 23:04:08 2021-07-07

Out[114]:
```
[('peak1_mag', ''),
 ('peak1_period', ''),
 ('peak2_mag', ''),
 ('peak2_period', ''),
 ('peak3_mag', ''),
 ('peak3_period', ''),
 ('condition', '')]
```

I added a subject column to the time dataframe, then had it merge on equivalency of that column and the index of the frequency dataframe which is the subject label.

In [115]:
```
condition_df = condition_df.merge(all_freqs,how='left',left_on='subject',right_ir
```
executed in 15ms, finished 23:04:08 2021-07-07

In [116]:
```
condition_df.drop(columns = ('subject',''))
```
executed in 30ms, finished 23:04:08 2021-07-07

Out[116]:

				activity					
				sum	median	mean	max	std	var
timestamp	timestamp	timestamp	hour_half						
5	7	12	1	9822	268.0	327.400000	1221	300.803934	90483.
			2	10971	306.0	365.700000	783	214.970912	46212.
		13	1	5514	181.0	183.800000	517	128.602676	16538.
			2	11560	355.0	385.333333	948	271.096920	73493.
		14	1	7206	129.5	240.200000	919	250.644328	62822.
	...	...	...	...	...	...	...	...	
	23	12	2	0	0.0	0.000000	0	0.000000	0.
		13	1	0	0.0	0.000000	0	0.000000	0.
			2	0	0.0	0.000000	0	0.000000	0.
		14	1	194	0.0	6.466667	160	29.209981	853.
			2	490	0.0	16.333333	306	62.017424	3846.

774 rows × 17 columns

Sick! Works, and also since the frequency dfs had a condition column, we also have our target column in this final dataframe. Now let's do it for all the subjects.

In [117]:
```
grouped_condition_all = pd.DataFrame()
grouped_control_all = pd.DataFrame()
for subject in cleaned_conditions:
    condition_df = grouped_condition_dfs[subject]
    condition_df['subject'] = subject
    condition_df = condition_df.merge(all_freqs,how='left',left_on='subject',righ
    condition_df = condition_df.drop(columns = ('subject',''))
    grouped_condition_all = pd.concat([grouped_condition_all,condition_df], axis
for subject in cleaned_controls:
    control_df = grouped_control_dfs[subject]
    control_df['subject'] = subject
    control_df = control_df.merge(all_freqs,how='left',left_on='subject',right_ir
    control_df = control_df.drop(columns = ('subject',''))
    grouped_control_all = pd.concat([grouped_control_all,control_df], axis = 0)
```
executed in 529ms, finished 23:04:09 2021-07-07

Okay, and finally, stack them into one big dataframe!!

In [118]:
```
all_grouped_dfs = pd.concat([grouped_condition_all,grouped_control_all],axis = 0)
```
executed in 15ms, finished 23:04:09 2021-07-07

In [119]:
```
all_grouped_dfs.head()
```
executed in 15ms, finished 23:04:09 2021-07-07

Out[119]:

				activity					
				sum	median	mean	max	std	var
timestamp	timestamp	timestamp	hour_half						
5	7	12	1	9822	268.0	327.400000	1221	300.803934	90483.
			2	10971	306.0	365.700000	783	214.970912	46212.
		13	1	5514	181.0	183.800000	517	128.602676	16538.
			2	11560	355.0	385.333333	948	271.096920	73493.
		14	1	7206	129.5	240.200000	919	250.644328	62822.

In [120]:
```
all_grouped_dfs.info()
```
executed in 31ms, finished 23:04:09 2021-07-07

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 52361 entries, (5, 7, 12, 1) to (12, 1, 12, 1)
Data columns (total 17 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   (activity, sum)            52361 non-null  int64
 1   (activity, median)         52361 non-null  float64
 2   (activity, mean)           52361 non-null  float64
 3   (activity, max)            52361 non-null  int64
 4   (activity, std)            52361 non-null  float64
 5   (activity, var)            52361 non-null  float64
 6   (activity, restful_mins)   52361 non-null  int64
 7   (activity, zero_act_mins)  52361 non-null  int64
 8   (resting, )                52361 non-null  int64
 9   (night, )                  52361 non-null  int64
 10  (peak1_mag, )              52361 non-null  float64
 11  (peak1_period, )           52361 non-null  float64
 12  (peak2_mag, )              52361 non-null  float64
 13  (peak2_period, )           52361 non-null  float64
 14  (peak3_mag, )              52361 non-null  float64
 15  (peak3_period, )           52361 non-null  float64
 16  (condition, )              52361 non-null  int64
dtypes: float64(10), int64(7)
memory usage: 7.0 MB
```

Huh.. two rows don't have std or var?

In [121]:
```
all_grouped_dfs.loc[pd.isna(all_grouped_dfs[('activity','std')])]
```
executed in 14ms, finished 23:04:09 2021-07-07

Out[121]:

				activity							
				sum	median	mean	max	std	var	restful_mins	ze
timestamp	timestamp	timestamp	hour_half								

Okay, so it would seem that those weren't calculable since these half hour sections only had one minute in them. Maybe I should go back up and figure out how to include only total half hours? (I went back up and made this change, that is why the above is empty now). Finally, it is worth noting that there are non-unique indexes in the final dataframe.

Finally, thinking about what a model looking at these individual rows can see, it has access to things like activity while resting, whether resting occurs at night or during the day, amongst many other things. One things that may be useful that, since the model does not consider the time order of things, is to include a boolean column for whether a period of wakefulness is interrupting rest; in other words, a True case is if an active period is preceded and anteceded by a restful period. So, let's see about implementing that.

In [122]:
```
# Did I awake in this current half hour? This hour resting = 0, previous hour res
# Thus subtracting 0 - 1 gives us a desired -1
awoke_inds = all_grouped_dfs[all_grouped_dfs.resting.diff(periods= 1).eq(-1)].inc
waking_inds
```
executed in 124ms, finished 23:04:09 2021-07-07

```
NameError: name 'waking_inds' is not defined
```

In [ ]:
```
# Do I fall asleep after this current half hour? This hour resting = 0, next hour
# Thus again subtracting 0 - 1 gives us a desired -1
falling_inds = all_grouped_dfs[all_grouped_dfs.resting.diff(periods = -1).eq(-1)]
falling_inds
```
executed in 2m 3s, finished 23:04:09 2021-07-07

In [ ]:
```
all_grouped_dfs[((all_grouped_dfs.resting.diff(periods= 1).eq(-1)) & (all_groupec
```
executed in 2m 3s, finished 23:04:09 2021-07-07

In [ ]:
```
all_grouped_dfs.loc[(5,11,7),:]
```
executed in 2m 3s, finished 23:04:09 2021-07-07

Okay so if this is working than out of roughly 2600 waking and falling cycles (it is good to see that there is bsaically the same number of wakings and fallings detected), 860 of the waking and falling happened as an interruption to normal sleep. Now i just need to take the indexes of those, and create a boolean column `'interrupting_rest'` .

In [ ]:
```
interrupting_inds = all_grouped_dfs[((all_grouped_dfs.resting.diff(periods= 1).ec
interrupting_inds
```
executed in 2m 3s, finished 23:04:09 2021-07-07

In [ ]:
```
all_grouped_dfs.loc[interrupting_inds,('rest_interrupted','')] = 1
```
executed in 2m 3s, finished 23:04:09 2021-07-07

In [ ]:
```
all_grouped_dfs.info()
```
executed in 2m 3s, finished 23:04:09 2021-07-07

Okay, as expected, just need to replace all the newly created NaNs with zeros.

In [ ]:
```
all_grouped_dfs = all_grouped_dfs.fillna(0)
```
executed in 2m 3s, finished 23:04:09 2021-07-07

In [ ]:
```
all_grouped_dfs.info()
```
executed in 2m 3s, finished 23:04:09 2021-07-07

Okay! For now, feature engineering is done. Let's save this dataframe as a csv we can load into another notebook.

In [ ]:
```
all_grouped_dfs.to_csv('./data/analysis_frame_all_subjects.csv', index = False)
```
executed in 2m 3s, finished 23:04:09 2021-07-07