sensor1_analysis

March 5, 2025

[4]:
```python
# imports

import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as mpl

# let's import our file for the analysis:

json_file = './data/sensor1_sleep.json'
with open(json_file, 'r') as infile:
  json_data = json.load(infile)
```

[5]:
```python
# what is this file?
type(json_data)
```

[5]:
```
dict
```

[6]:
```python
# let's see what elements do we have in the dictionary
for element in json_data:
  print(element)
```

```
readiness
restful_periods
sleep
```

[7]:
```python
# we are interested in sleep! let's focus on that element
# find out what do we have inside the 'sleep' element
type(json_data['sleep'])
```

[7]:
```
list
```

[12]:
```python
# let's access the first element of this list

# -- write your code --

json_data["sleep"]
```

[12]:

```
[{'awake': 2760,
  'bedtime_end': '2020-01-14T05:31:00-08:00',
  'bedtime_end_delta': 24480,
  'bedtime_start': '2020-01-13T22:41:00-08:00',
  'bedtime_start_delta': 60,
  'breath_average': 14.125,
  'deep': 3990,
  'duration': 24420,
  'efficiency': 89,
  'hr_5min': [0,
   53,
   54,
   56,
   56,
   55,
   56,
   56,
   56,
   56,
   55,
   55,
   56,
   61,
   62,
   61,
   60,
   60,
   60,
   59,
   59,
   59,
   60,
   60,
   57,
   55,
   55,
   56,
   57,
   56,
   56,
   0,
   57,
   58,
   63,
   0,
   53,
   54,
```

56,
56,
55,
56,
56,
56,
56,
55,
55,
56,
61,
62,
61,
60,
60,
60,
59,
59,
59,
60,
60,
57,
55,
55,
56,
57,
56,
56,
0,
57,
58,
63],
'hr_average': 57.54,
'hr_lowest': 53,
'hypnogram_5min': '222221111144442222234422222222222222112222222433332222233333322
22232222222111113333444',
'is_longest': 1,
'light': 12930,
'midpoint_at_delta': 12310,
'midpoint_time': 12790,
'onset_latency': 30,
'period_id': 0,
'rem': 4740,
'restless': 31,
'rmssd': 57,
'rmssd_5min': [0,
53,

54,
56,
56,
55,
56,
56,
56,
56,
55,
55,
56,
61,
62,
61,
60,
60,
60,
59,
59,
59,
60,
60,
57,
55,
55,
56,
57,
56,
56,
0,
57,
58,
63,
0,
53,
54,
56,
56,
55,
56,
56,
56,
56,
55,
55,
56,
61,

```
  62,
  61,
  60,
  60,
  60,
  59,
  59,
  59,
  60,
  60,
  57,
  55,
  55,
  56,
  57,
  56,
  56,
  0,
  57,
  58,
  63],
 'score': 71,
 'score_alignment': 81,
 'score_deep': 73,
 'score_disturbances': 71,
 'score_efficiency': 80,
 'score_latency': 69,
 'score_rem': 69,
 'score_total': 69,
 'summary_date': '2020-01-13',
 'temperature_delta': -0.03,
 'temperature_deviation': -0.03,
 'temperature_trend_deviation': 0.03,
 'timezone': -480,
 'total': 21560},
{'awake': 5370,
 'bedtime_end': '2020-01-15T07:38:02-08:00',
 'bedtime_end_delta': 27482,
 'bedtime_start': '2020-01-14T23:40:02-08:00',
 'bedtime_start_delta': -1198,
 'breath_average': 13.875,
 'deep': 8280,
 'duration': 28680,
 'efficiency': 81,
 'hr_5min': [0,
  52,
  53,
```

```
53,
55,
54,
51,
50,
51,
51,
51,
51,
52,
52,
51,
52,
54,
55,
58,
56,
55,
56,
57,
57,
58,
58,
58,
58,
57,
56,
57,
57,
55,
60,
67,
62,
55,
54,
53,
55,
56,
56,
56,
56,
55,
54,
54,
55,
55,
56,
```

```
 55,
 55,
 55,
 56,
 56,
 55,
 56,
 59,
 59,
 59,
 59,
 59,
 59,
 57,
 53,
 51,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 56,
 58,
 58,
 59,
 59,
 56,
 54,
 54,
 53,
 54,
 53,
 53,
 54,
 57,
 54,
 56,
 0,
 0,
 52,
 53],
'hr_average': 55.37,
```

'hr_lowest': 50,
'hypnogram_5min': '422244221111111223311111122111133322221111122111313333322
22222224444444444222222222222233443 44',
'is_longest': 1,
'light': 10530,
'midpoint_at_delta': 12122,
'midpoint_time': 13320,
'onset_latency': 330,
'period_id': 0,
'rem': 4500,
'restless': 21,
'rmssd': 65,
'rmssd_5min': [0,
76,
81,
92,
80,
82,
91,
91,
82,
70,
70,
72,
71,
72,
92,
85,
81,
86,
61,
69,
73,
51,
43,
43,
48,
41,
41,
49,
51,
56,
49,
56,
67,
58,
22,

```
30,
71,
86,
84,
62,
58,
68,
60,
64,
76,
75,
81,
63,
57,
69,
66,
68,
73,
69,
64,
77,
66,
39,
43,
47,
37,
39,
43,
90,
74,
79,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
83,
62,
63,
56,
56,
65,
```

```
  73,
  73,
  81,
  84,
  87,
  93,
  90,
  69,
  91,
  84,
  0,
  0,
  88,
  90],
 'score': 77,
 'score_alignment': 88,
 'score_deep': 99,
 'score_disturbances': 90,
 'score_efficiency': 74,
 'score_latency': 75,
 'score_rem': 65,
 'score_total': 68,
 'summary_date': '2020-01-14',
 'temperature_delta': 0.02,
 'temperature_deviation': 0.02,
 'temperature_trend_deviation': 0.08,
 'timezone': -480,
 'total': 23310},
{'awake': 3660,
 'bedtime_end': '2020-01-16T06:09:04-08:00',
 'bedtime_end_delta': 22144,
 'bedtime_start': '2020-01-15T22:54:04-08:00',
 'bedtime_start_delta': -3956,
 'breath_average': 12.625,
 'deep': 4550,
 'duration': 22100,
 'efficiency': 80,
 'hr_5min': [55,
  55,
  55,
  55,
  55,
  56,
  51,
  51,
  51,
  52,
```

```
53,
51,
50,
51,
52,
51,
51,
51,
56,
58,
62,
57,
56,
55,
55,
53,
52,
52,
51,
52,
52,
59,
58,
57,
57,
56,
56,
59,
56,
58,
57,
58,
59,
58,
58,
59,
60,
58,
53,
52,
51,
51,
55,
60,
55,
55,
56,
```

62,
56,
61,
55,
53,
52,
53,
54,
54,
54,
57,
59,
59,
57,
56,
57,
57,
55,
55,
55,
56,
56,
55,
54,
57,
56,
57,
57,
56,
0,
0],
'hr_average': 55.27,
'hr_lowest': 50,
'hypnogram_5min': '44442221111112221122233331111111222142244444222222233333332
2221222222223342333333333334',
'is_longest': 1,
'light': 9340,
'midpoint_at_delta': 9304,
'midpoint_time': 13360,
'onset_latency': 1230,
'period_id': 0,
'rem': 7330,
'restless': 38,
'rmssd': 37,
'rmssd_5min': [63,
69,
94,

```
66,
62,
56,
62,
91,
77,
55,
50,
48,
43,
42,
47,
51,
58,
46,
46,
73,
76,
45,
82,
55,
90,
84,
65,
30,
72,
58,
53,
45,
79,
86,
85,
66,
63,
61,
61,
63,
59,
65,
90,
57,
79,
54,
70,
54,
52,
44,
```

```
 44,
 30,
 41,
 45,
 40,
 56,
 63,
 63,
 52,
 51,
 80,
 90,
 79,
 73,
 76,
 75,
 58,
 37,
 37,
 54,
 53,
 46,
 45,
 39,
 40,
 50,
 48,
 45,
 61,
 59,
 48,
 43,
 67,
 56,
 61,
 58,
 0,
 0],
'score': 68,
'score_alignment': 100,
'score_deep': 95,
'score_disturbances': 27,
'score_efficiency': 63,
'score_latency': 61,
'score_rem': 67,
'score_total': 66,
'summary_date': '2020-01-15',
```

'temperature_delta': -0.21,
'temperature_deviation': -0.21,
'temperature_trend_deviation': 0.02,
'timezone': -480,
'total': 22110},
{'awake': 2760,
'bedtime_end': '2020-01-17T06:48:00-08:00',
'bedtime_end_delta': 24480,
'bedtime_start': '2020-01-17T00:01:00-08:00',
'bedtime_start_delta': 60,
'breath_average': 14.125,
'deep': 3990,
'duration': 24420,
'efficiency': 89,
'hr_5min': [0,
53,
54,
56,
56,
55,
56,
56,
56,
56,
55,
55,
56,
61,
62,
61,
60,
60,
60,
59,
59,
59,
60,
60,
57,
55,
55,
56,
57,
56,
56,
0,
57,

58,
63,
0,
53,
54,
56,
56,
55,
56,
56,
56,
56,
55,
55,
56,
61,
62,
61,
60,
60,
60,
59,
59,
59,
60,
60,
57,
55,
55,
56,
57,
56,
56,
0,
57,
58,
63],
'hr_average': 57.54,
'hr_lowest': 53,
'hypnogram_5min': '222221111144442222234422222222222112222222433332222233333 22
22232222222111113333444',
'is_longest': 1,
'light': 12930,
'midpoint_at_delta': 12310,
'midpoint_time': 12790,
'onset_latency': 30,
'period_id': 0,

'rem': 4740,
'restless': 31,
'rmssd': 57,
'rmssd_5min': [0,
 53,
 54,
 56,
 56,
 55,
 56,
 56,
 56,
 56,
 55,
 55,
 56,
 61,
 62,
 61,
 60,
 60,
 60,
 59,
 59,
 59,
 60,
 60,
 57,
 55,
 55,
 56,
 57,
 56,
 56,
 0,
 57,
 58,
 63,
 0,
 53,
 54,
 56,
 56,
 55,
 56,
 56,
 56,

```
  56,
  55,
  55,
  56,
  61,
  62,
  61,
  60,
  60,
  60,
  59,
  59,
  59,
  60,
  60,
  57,
  55,
  55,
  56,
  57,
  56,
  56,
  0,
  57,
  58,
  63],
 'score': 71,
 'score_alignment': 81,
 'score_deep': 73,
 'score_disturbances': 71,
 'score_efficiency': 80,
 'score_latency': 69,
 'score_rem': 69,
 'score_total': 69,
 'summary_date': '2020-01-16',
 'temperature_delta': -0.03,
 'temperature_deviation': -0.03,
 'temperature_trend_deviation': 0.03,
 'timezone': -480,
 'total': 21560},
{'awake': 2760,
 'bedtime_end': '2020-01-18T06:52:00-08:00',
 'bedtime_end_delta': 24480,
 'bedtime_start': '2020-01-18T00:06:00-08:00',
 'bedtime_start_delta': 60,
 'breath_average': 14.125,
 'deep': 3990,
```

'duration': 24420,
'efficiency': 89,
'hr_5min': [0,
 53,
 54,
 56,
 56,
 55,
 56,
 56,
 56,
 56,
 55,
 55,
 56,
 61,
 62,
 61,
 60,
 60,
 60,
 59,
 59,
 59,
 60,
 60,
 57,
 55,
 55,
 56,
 57,
 56,
 56,
 0,
 57,
 58,
 63,
 0,
 53,
 54,
 56,
 56,
 55,
 56,
 56,
 56,
 56,

55,
55,
56,
61,
62,
61,
60,
60,
60,
59,
59,
59,
60,
60,
57,
55,
55,
56,
57,
56,
56,
0,
57,
58,
63],
'hr_average': 57.54,
'hr_lowest': 53,
'hypnogram_5min': '222221111144442222234422222222222112222222433332222223333322
2223222222211111333344',
'is_longest': 1,
'light': 12930,
'midpoint_at_delta': 12310,
'midpoint_time': 12790,
'onset_latency': 30,
'period_id': 0,
'rem': 4740,
'restless': 31,
'rmssd': 57,
'rmssd_5min': [0,
53,
54,
56,
56,
55,
56,
56,
56,

```
56,
55,
55,
56,
61,
62,
61,
60,
60,
60,
59,
59,
59,
60,
60,
57,
55,
55,
56,
57,
56,
56,
0,
57,
58,
63,
0,
53,
54,
56,
56,
55,
56,
56,
56,
56,
55,
55,
56,
61,
62,
61,
60,
60,
60,
59,
59,
```

```
    59,
    60,
    60,
    57,
    55,
    55,
    56,
    57,
    56,
    56,
    0,
    57,
    58,
    63],
   'score': 71,
   'score_alignment': 81,
   'score_deep': 73,
   'score_disturbances': 71,
   'score_efficiency': 80,
   'score_latency': 69,
   'score_rem': 69,
   'score_total': 69,
   'summary_date': '2020-01-18',
   'temperature_delta': -0.03,
   'temperature_deviation': -0.03,
   'temperature_trend_deviation': 0.03,
   'timezone': -480,
   'total': 21560}]
```


[13]:
```
# let's find the element related to the day of interest! do we have one element␣
↪per day?

# 1. let's find how many days are there

# -- write your code --

len(json_data["sleep"])
```

[13]: 5

[14]:
```
# 2. let's find the index of the day of interest; for each day, let's print␣
↪bedtime_end , so we will know
# which element to use (the night is between 15 and 16 of Jan 2020)

# -- write your code --

json_data['sleep'][1]
```

[14]:

```
{'awake': 5370,
 'bedtime_end': '2020-01-15T07:38:02-08:00',
 'bedtime_end_delta': 27482,
 'bedtime_start': '2020-01-14T23:40:02-08:00',
 'bedtime_start_delta': -1198,
 'breath_average': 13.875,
 'deep': 8280,
 'duration': 28680,
 'efficiency': 81,
 'hr_5min': [0,
  52,
  53,
  53,
  55,
  54,
  51,
  50,
  51,
  51,
  51,
  51,
  52,
  52,
  51,
  52,
  54,
  55,
  58,
  56,
  55,
  56,
  57,
  57,
  58,
  58,
  58,
  58,
  57,
  56,
  57,
  57,
  55,
  60,
  67,
  62,
  55,
  54,
```

```
53,
55,
56,
56,
56,
56,
55,
54,
54,
55,
55,
56,
55,
55,
55,
56,
56,
55,
56,
59,
59,
59,
59,
59,
59,
57,
53,
51,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
56,
58,
58,
59,
59,
56,
54,
54,
53,
```

54,
53,
53,
54,
57,
54,
56,
0,
0,
52,
53],
'hr_average': 55.37,
'hr_lowest': 50,
'hypnogram_5min': '422244221111111122331111111221111333222211111221113133333222
2222224444444444222222222222223344344',
'is_longest': 1,
'light': 10530,
'midpoint_at_delta': 12122,
'midpoint_time': 13320,
'onset_latency': 330,
'period_id': 0,
'rem': 4500,
'restless': 21,
'rmssd': 65,
'rmssd_5min': [0,
76,
81,
92,
80,
82,
91,
91,
82,
70,
70,
72,
71,
72,
92,
85,
81,
86,
61,
69,
73,
51,
43,

```
43,
48,
41,
41,
49,
51,
56,
49,
56,
67,
58,
22,
30,
71,
86,
84,
62,
58,
68,
60,
64,
76,
75,
81,
63,
57,
69,
66,
68,
73,
69,
64,
77,
66,
39,
43,
47,
37,
39,
43,
90,
74,
79,
0,
0,
0,
0,
```

```
  0,
  0,
  0,
  0,
  0,
  0,
  83,
  62,
  63,
  56,
  56,
  65,
  73,
  73,
  81,
  84,
  87,
  93,
  90,
  69,
  91,
  84,
  0,
  0,
  88,
  90],
 'score': 77,
 'score_alignment': 88,
 'score_deep': 99,
 'score_disturbances': 90,
 'score_efficiency': 74,
 'score_latency': 75,
 'score_rem': 65,
 'score_total': 68,
 'summary_date': '2020-01-14',
 'temperature_delta': 0.02,
 'temperature_deviation': 0.02,
 'temperature_trend_deviation': 0.08,
 'timezone': -480,
 'total': 23310}
```


[11]:
```
# 3. let's find the time the participant went to bed, and when s/he woke up
night_index = 2
print('Night starts at: '+json_data['sleep'][night_index]['bedtime_start'])
print('Night end at: '+json_data['sleep'][night_index]['bedtime_end'])
```

```
Night starts at: 2020-01-15T22:54:04-08:00
```

```
Night end at: 2020-01-16T06:09:04-08:00
```

[16]:
```
# 4. total time in bed: put the total time spent in bed by the participant␣
↪(calculate it manually)

# -- write your code --

min_slept = 24-22 + 6 - (.54-.09)
min_slept
```

[16]:
```
7.55
```

[17]:
```
# 5. let's extract the hypnogram !
# the code is written for you, but verify step by step what is happening (add␣
↪comments)!

#'1' = deep (N3) sleep - '2' = light (N1 or N2) sleep - '3' = REM sleep - '4' =␣
↪awake
dic_sleep = {'wake':4 , 'deep':1, 'light':2 , 'rem':3}

hypno_js = json_data['sleep'][night_index]['hypnogram_5min']

hypno = np.array(list(hypno_js))

# we have 1 value every 5 minutes; we need 1 value every 30 seconds (to compare␣
↪it with gold standard)
# let's have each element repeated 10 times (2 per minute)
hypno = np.repeat(hypno,10)
hypno = hypno.astype(int)
```

[18]:
```
# 6. let's verify hypnogram is of the right length
len(hypno)
```

[18]:
```
870
```

[19]:
```
# 7. let's write down the hypnogram to export,
# we need to write the phase of sleep (from 1 to 4) from 11:00pm to 6:00am ,␣
↪with sleep phase for each 30 seconds
total_number_30sec_int = 7*60*2

df = pd.DataFrame(columns = ['IndexTime','SleepPhase'])
df['IndexTime'] = range(total_number_30sec_int)

# let's cut the first and last minutes of hypno (outside the range of interest)
minutes_before_11pm = 6
minutes_after_6am = 9
```

```
# I consider only hypno in the range of interest
df['SleepPhase'] = hypno[minutes_before_11pm*2:-minutes_after_6am*2]
```

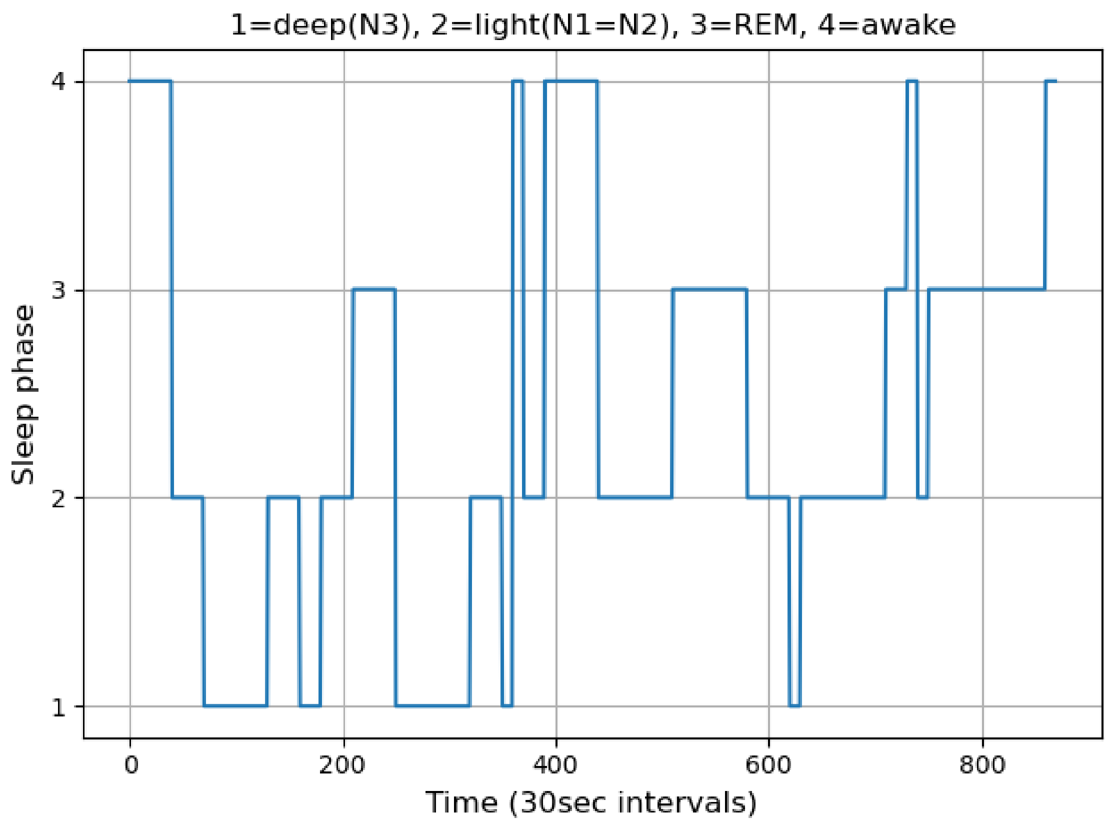
[20]:
```
# 8. let's save the final csv file
df.to_csv('sensor_1_output.csv', index=False)
```

[22]:
```
# 9. let's calculate amount of sleep per phase
# in number of intervals
# in percentage over the 7 hours of analysis

# -- write your code --

sleep_counts = df['SleepPhase'].value_counts(normalize=True).sort_index()
summary_df = (sleep_counts * 100).reset_index()
summary_df.columns = ['SleepPhase', 'Percentage']
summary_df['Intervals'] = (summary_df['Percentage'] / 100) * len(df)
print(summary_df)
```

```
   SleepPhase  Percentage  Intervals
0           1   20.238095      170.0
1           2   40.476190      340.0
2           3   27.619048      232.0
3           4   11.666667       98.0
```

[21]:
```
# 10. let's print the hypnogram!

def plot_hypno(signal): #,color
  x = np.arange(len(signal))
  y = signal

  fontsizeV = 12
  mpl.plot(x,y) #color=color
  mpl.xlabel('Time (30sec intervals)',fontsize=fontsizeV)
  mpl.ylabel('Sleep phase',fontsize=fontsizeV)
  mpl.yticks([1,2,3,4])
  mpl.title('1=deep(N3), 2=light(N1=N2), 3=REM, 4=awake')
  mpl.grid(True)
  mpl.tight_layout()

# -- write your code --

print(plot_hypno(hypno))
```

```
None
```


1=deep(N3), 2=light(N1=N2), 3=REM, 4=awake
Sleep phase
Time (30sec intervals)
4
3
2
1
0
200
400
600
800

```
[ ]:
```

sensor2_analysis

March 5, 2025

[18]:
```python
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as mpl

# let's import our file for the analysis:
json_file = './data/sensor2_sleep.json'

with open(json_file, 'r') as infile:
  json_data = json.load(infile)
```

[19]:
```python
# what is this file?
json_data
```

[19]:

```
{'sleep': [{'dateOfSleep': '2020-01-17',
   'duration': 29460000,
   'efficiency': 96,
   'endTime': '2020-01-17T07:20:30.000',
   'infoCode': 0,
   'isMainSleep': True,
   'levels': {'data': [{'dateTime': '2020-01-16T23:09:30.000',
      'level': 'wake',
      'seconds': 570},
     {'dateTime': '2020-01-16T23:19:00.000',
      'level': 'light',
      'seconds': 1200},
     {'dateTime': '2020-01-16T23:39:00.000', 'level': 'deep', 'seconds': 1650},
     {'dateTime': '2020-01-17T00:06:30.000', 'level': 'light', 'seconds': 360},
     {'dateTime': '2020-01-17T00:12:30.000', 'level': 'rem', 'seconds': 300},
     {'dateTime': '2020-01-17T00:17:30.000',
      'level': 'light',
      'seconds': 1320},
     {'dateTime': '2020-01-17T00:39:30.000', 'level': 'deep', 'seconds': 1770},
     {'dateTime': '2020-01-17T01:09:00.000',
      'level': 'light',
      'seconds': 1920},
     {'dateTime': '2020-01-17T01:41:00.000', 'level': 'rem', 'seconds': 930},
```

{'dateTime': '2020-01-17T01:56:30.000',
 'level': 'light',
 'seconds': 1440},
{'dateTime': '2020-01-17T02:20:30.000', 'level': 'deep', 'seconds': 2190},
{'dateTime': '2020-01-17T02:57:00.000', 'level': 'light', 'seconds': 810},
{'dateTime': '2020-01-17T03:10:30.000', 'level': 'rem', 'seconds': 1320},
{'dateTime': '2020-01-17T03:32:30.000',
 'level': 'light',
 'seconds': 1110},
{'dateTime': '2020-01-17T03:51:00.000', 'level': 'deep', 'seconds': 420},
{'dateTime': '2020-01-17T03:58:00.000',
 'level': 'light',
 'seconds': 1740},
{'dateTime': '2020-01-17T04:27:00.000', 'level': 'rem', 'seconds': 3270},
{'dateTime': '2020-01-17T05:21:30.000', 'level': 'light', 'seconds': 60},
{'dateTime': '2020-01-17T05:22:30.000', 'level': 'wake', 'seconds': 390},
{'dateTime': '2020-01-17T05:29:00.000',
 'level': 'light',
 'seconds': 3720},
{'dateTime': '2020-01-17T06:31:00.000', 'level': 'wake', 'seconds': 360},
{'dateTime': '2020-01-17T06:37:00.000', 'level': 'light', 'seconds': 780},
{'dateTime': '2020-01-17T06:50:00.000',
 'level': 'wake',
 'seconds': 1830}],
'shortData': [{'dateTime': '2020-01-16T23:21:00.000',
 'level': 'wake',
 'seconds': 60},
{'dateTime': '2020-01-16T23:31:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T00:17:30.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T01:08:30.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T01:36:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T01:40:00.000', 'level': 'wake', 'seconds': 60},
{'dateTime': '2020-01-17T01:57:30.000', 'level': 'wake', 'seconds': 60},
{'dateTime': '2020-01-17T02:56:00.000', 'level': 'wake', 'seconds': 60},
{'dateTime': '2020-01-17T03:14:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T03:28:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T03:33:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T03:36:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T04:00:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T04:17:30.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T04:40:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T04:52:30.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T04:58:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T05:13:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T05:45:00.000', 'level': 'wake', 'seconds': 30},
{'dateTime': '2020-01-17T05:55:00.000', 'level': 'wake', 'seconds': 150},
{'dateTime': '2020-01-17T06:11:00.000', 'level': 'wake', 'seconds': 60},

```
    {'dateTime': '2020-01-17T06:25:00.000', 'level': 'wake', 'seconds': 60},
    {'dateTime': '2020-01-17T06:40:00.000', 'level': 'wake', 'seconds': 150},
    {'dateTime': '2020-01-17T06:45:00.000', 'level': 'wake', 'seconds': 60}],
   'summary': {'deep': {'count': 4, 'minutes': 99, 'thirtyDayAvgMinutes': 49},
    'light': {'count': 25, 'minutes': 226, 'thirtyDayAvgMinutes': 280},
    'rem': {'count': 10, 'minutes': 94, 'thirtyDayAvgMinutes': 66},
    'wake': {'count': 28, 'minutes': 72, 'thirtyDayAvgMinutes': 78}}},
  'logId': 25487811377,
  'minutesAfterWakeup': 5,
  'minutesAsleep': 419,
  'minutesAwake': 72,
  'minutesToFallAsleep': 0,
  'startTime': '2020-01-16T23:09:30.000',
  'timeInBed': 491,
  'type': 'stages'},
 {'dateOfSleep': '2020-01-16',
  'duration': 27720000,
  'efficiency': 92,
  'endTime': '2020-01-16T06:16:30.000',
  'infoCode': 0,
  'isMainSleep': True,
  'levels': {'data': [{'dateTime': '2020-01-15T22:34:00.000',
     'level': 'wake',
     'seconds': 600},
    {'dateTime': '2020-01-15T22:44:00.000', 'level': 'deep', 'seconds': 270},
    {'dateTime': '2020-01-15T22:48:30.000', 'level': 'wake', 'seconds': 240},
    {'dateTime': '2020-01-15T22:52:30.000', 'level': 'light', 'seconds': 90},
    {'dateTime': '2020-01-15T22:54:00.000', 'level': 'deep', 'seconds': 420},
    {'dateTime': '2020-01-15T23:01:00.000', 'level': 'wake', 'seconds': 660},
    {'dateTime': '2020-01-15T23:12:00.000', 'level': 'light', 'seconds': 120},
    {'dateTime': '2020-01-15T23:14:00.000', 'level': 'wake', 'seconds': 360},
    {'dateTime': '2020-01-15T23:20:00.000',
     'level': 'light',
     'seconds': 1320},
    {'dateTime': '2020-01-15T23:42:00.000', 'level': 'deep', 'seconds': 390},
    {'dateTime': '2020-01-15T23:48:30.000',
     'level': 'light',
     'seconds': 7380},
    {'dateTime': '2020-01-16T01:51:30.000', 'level': 'wake', 'seconds': 480},
    {'dateTime': '2020-01-16T01:59:30.000', 'level': 'light', 'seconds': 900},
    {'dateTime': '2020-01-16T02:14:30.000', 'level': 'wake', 'seconds': 300},
    {'dateTime': '2020-01-16T02:19:30.000', 'level': 'rem', 'seconds': 330},
    {'dateTime': '2020-01-16T02:25:00.000', 'level': 'wake', 'seconds': 330},
    {'dateTime': '2020-01-16T02:30:30.000',
     'level': 'light',
     'seconds': 2070},
    {'dateTime': '2020-01-16T03:05:00.000', 'level': 'rem', 'seconds': 2640},
```

```
    {'dateTime': '2020-01-16T03:49:00.000', 'level': 'light', 'seconds': 630},
    {'dateTime': '2020-01-16T03:59:30.000', 'level': 'deep', 'seconds': 1560},
    {'dateTime': '2020-01-16T04:25:30.000',
     'level': 'light',
     'seconds': 3330},
    {'dateTime': '2020-01-16T05:21:00.000', 'level': 'rem', 'seconds': 660},
    {'dateTime': '2020-01-16T05:32:00.000', 'level': 'light', 'seconds': 930},
    {'dateTime': '2020-01-16T05:47:30.000', 'level': 'rem', 'seconds': 360},
    {'dateTime': '2020-01-16T05:53:30.000', 'level': 'light', 'seconds': 780},
    {'dateTime': '2020-01-16T06:06:30.000', 'level': 'wake', 'seconds': 600}],
   'shortData': [{'dateTime': '2020-01-15T23:58:30.000',
     'level': 'wake',
     'seconds': 90},
    {'dateTime': '2020-01-16T00:24:30.000', 'level': 'wake', 'seconds': 90},
    {'dateTime': '2020-01-16T01:31:00.000', 'level': 'wake', 'seconds': 120},
    {'dateTime': '2020-01-16T04:29:00.000', 'level': 'wake', 'seconds': 30},
    {'dateTime': '2020-01-16T04:44:00.000', 'level': 'wake', 'seconds': 90},
    {'dateTime': '2020-01-16T04:58:00.000', 'level': 'wake', 'seconds': 120},
    {'dateTime': '2020-01-16T05:03:00.000', 'level': 'wake', 'seconds': 60}],
   'summary': {'deep': {'count': 4, 'minutes': 44, 'thirtyDayAvgMinutes': 54},
    'light': {'count': 17, 'minutes': 283, 'thirtyDayAvgMinutes': 276},
    'rem': {'count': 4, 'minutes': 66, 'thirtyDayAvgMinutes': 65},
    'wake': {'count': 15, 'minutes': 69, 'thirtyDayAvgMinutes': 87}}},
  'logId': 25468095378,
  'minutesAfterWakeup': 0,
  'minutesAsleep': 393,
  'minutesAwake': 69,
  'minutesToFallAsleep': 0,
  'startTime': '2020-01-15T22:34:00.000',
  'timeInBed': 462,
  'type': 'stages'},
 {'dateOfSleep': '2020-01-15',
  'duration': 28920000,
  'efficiency': 88,
  'endTime': '2020-01-15T07:53:30.000',
  'infoCode': 0,
  'isMainSleep': True,
  'levels': {'data': [{'dateTime': '2020-01-14T23:51:00.000',
     'level': 'light',
     'seconds': 330},
    {'dateTime': '2020-01-14T23:56:30.000', 'level': 'wake', 'seconds': 720},
    {'dateTime': '2020-01-15T00:08:30.000',
     'level': 'light',
     'seconds': 1410},
    {'dateTime': '2020-01-15T00:32:00.000', 'level': 'deep', 'seconds': 870},
    {'dateTime': '2020-01-15T00:46:30.000', 'level': 'light', 'seconds': 810},
    {'dateTime': '2020-01-15T01:00:00.000', 'level': 'rem', 'seconds': 1200},
```

```
 {'dateTime': '2020-01-15T01:20:00.000', 'level': 'light', 'seconds': 510},
 {'dateTime': '2020-01-15T01:28:30.000', 'level': 'deep', 'seconds': 840},
 {'dateTime': '2020-01-15T01:42:30.000',
  'level': 'light',
  'seconds': 2520},
 {'dateTime': '2020-01-15T02:24:30.000', 'level': 'rem', 'seconds': 1080},
 {'dateTime': '2020-01-15T02:42:30.000',
  'level': 'light',
  'seconds': 1620},
 {'dateTime': '2020-01-15T03:09:30.000', 'level': 'deep', 'seconds': 930},
 {'dateTime': '2020-01-15T03:25:00.000',
  'level': 'light',
  'seconds': 2400},
 {'dateTime': '2020-01-15T04:05:00.000', 'level': 'rem', 'seconds': 990},
 {'dateTime': '2020-01-15T04:21:30.000',
  'level': 'light',
  'seconds': 1170},
 {'dateTime': '2020-01-15T04:41:00.000', 'level': 'deep', 'seconds': 270},
 {'dateTime': '2020-01-15T04:45:30.000', 'level': 'light', 'seconds': 210},
 {'dateTime': '2020-01-15T04:49:00.000', 'level': 'deep', 'seconds': 330},
 {'dateTime': '2020-01-15T04:54:30.000', 'level': 'light', 'seconds': 990},
 {'dateTime': '2020-01-15T05:11:00.000', 'level': 'wake', 'seconds': 2760},
 {'dateTime': '2020-01-15T05:57:00.000',
  'level': 'light',
  'seconds': 4290},
 {'dateTime': '2020-01-15T07:08:30.000', 'level': 'rem', 'seconds': 780},
 {'dateTime': '2020-01-15T07:21:30.000', 'level': 'light', 'seconds': 870},
 {'dateTime': '2020-01-15T07:36:00.000', 'level': 'wake', 'seconds': 450},
 {'dateTime': '2020-01-15T07:43:30.000',
  'level': 'light',
  'seconds': 600}],
'shortData': [{'dateTime': '2020-01-14T23:53:00.000',
  'level': 'wake',
  'seconds': 120},
 {'dateTime': '2020-01-15T01:05:00.000', 'level': 'wake', 'seconds': 30},
 {'dateTime': '2020-01-15T01:55:30.000', 'level': 'wake', 'seconds': 60},
 {'dateTime': '2020-01-15T02:22:30.000', 'level': 'wake', 'seconds': 30},
 {'dateTime': '2020-01-15T02:42:30.000', 'level': 'wake', 'seconds': 30},
 {'dateTime': '2020-01-15T03:25:00.000', 'level': 'wake', 'seconds': 60},
 {'dateTime': '2020-01-15T04:01:30.000', 'level': 'wake', 'seconds': 30},
 {'dateTime': '2020-01-15T04:22:30.000', 'level': 'wake', 'seconds': 30},
 {'dateTime': '2020-01-15T04:56:00.000', 'level': 'wake', 'seconds': 150},
 {'dateTime': '2020-01-15T06:04:00.000', 'level': 'wake', 'seconds': 30},
 {'dateTime': '2020-01-15T06:06:00.000', 'level': 'wake', 'seconds': 60},
 {'dateTime': '2020-01-15T06:23:00.000', 'level': 'wake', 'seconds': 120},
 {'dateTime': '2020-01-15T06:36:00.000', 'level': 'wake', 'seconds': 60},
 {'dateTime': '2020-01-15T06:39:00.000', 'level': 'wake', 'seconds': 30},
```

```
      {'dateTime': '2020-01-15T06:42:00.000', 'level': 'wake', 'seconds': 30},
      {'dateTime': '2020-01-15T06:46:00.000', 'level': 'wake', 'seconds': 60},
      {'dateTime': '2020-01-15T07:20:00.000', 'level': 'wake', 'seconds': 90},
      {'dateTime': '2020-01-15T07:30:00.000', 'level': 'wake', 'seconds': 180},
      {'dateTime': '2020-01-15T07:48:30.000', 'level': 'wake', 'seconds': 90}],
     'summary': {'deep': {'count': 5, 'minutes': 54, 'thirtyDayAvgMinutes': 0},
      'light': {'count': 28, 'minutes': 276, 'thirtyDayAvgMinutes': 0},
      'rem': {'count': 5, 'minutes': 65, 'thirtyDayAvgMinutes': 0},
      'wake': {'count': 22, 'minutes': 87, 'thirtyDayAvgMinutes': 0}}},
    'logId': 25455855559,
    'minutesAfterWakeup': 0,
    'minutesAsleep': 395,
    'minutesAwake': 87,
    'minutesToFallAsleep': 0,
    'startTime': '2020-01-14T23:51:00.000',
    'timeInBed': 482,
    'type': 'stages'}]}
```


[20]:
```
# let's see what elements do we have in the dictionary
for element in json_data:
  print(element)
```

sleep

[21]:
```
# we are interested in sleep! let's focus on that
type(json_data['sleep'])
```

[21]: list

[22]:
```
# let's find the element related to the day of interest! do we have one element␣
 ↪per day?

# 1. let's find how many days are there

sum([1 for day in json_data['sleep']])
```

[22]: 3

[23]:
```
# 2. let's find the index of the day of interest; for each day, let's print␣
 ↪dateOfSleep , so we will know
# which element to use (the night is between 15 and 16 of Jan 2020)

# Hypnogram of 11pm to 6am

json_data['sleep'][1]
```

[23]:

```
{'dateOfSleep': '2020-01-16',
 'duration': 27720000,
 'efficiency': 92,
 'endTime': '2020-01-16T06:16:30.000',
 'infoCode': 0,
 'isMainSleep': True,
 'levels': {'data': [{'dateTime': '2020-01-15T22:34:00.000',
    'level': 'wake',
    'seconds': 600},
   {'dateTime': '2020-01-15T22:44:00.000', 'level': 'deep', 'seconds': 270},
   {'dateTime': '2020-01-15T22:48:30.000', 'level': 'wake', 'seconds': 240},
   {'dateTime': '2020-01-15T22:52:30.000', 'level': 'light', 'seconds': 90},
   {'dateTime': '2020-01-15T22:54:00.000', 'level': 'deep', 'seconds': 420},
   {'dateTime': '2020-01-15T23:01:00.000', 'level': 'wake', 'seconds': 660},
   {'dateTime': '2020-01-15T23:12:00.000', 'level': 'light', 'seconds': 120},
   {'dateTime': '2020-01-15T23:14:00.000', 'level': 'wake', 'seconds': 360},
   {'dateTime': '2020-01-15T23:20:00.000', 'level': 'light', 'seconds': 1320},
   {'dateTime': '2020-01-15T23:42:00.000', 'level': 'deep', 'seconds': 390},
   {'dateTime': '2020-01-15T23:48:30.000', 'level': 'light', 'seconds': 7380},
   {'dateTime': '2020-01-16T01:51:30.000', 'level': 'wake', 'seconds': 480},
   {'dateTime': '2020-01-16T01:59:30.000', 'level': 'light', 'seconds': 900},
   {'dateTime': '2020-01-16T02:14:30.000', 'level': 'wake', 'seconds': 300},
   {'dateTime': '2020-01-16T02:19:30.000', 'level': 'rem', 'seconds': 330},
   {'dateTime': '2020-01-16T02:25:00.000', 'level': 'wake', 'seconds': 330},
   {'dateTime': '2020-01-16T02:30:30.000', 'level': 'light', 'seconds': 2070},
   {'dateTime': '2020-01-16T03:05:00.000', 'level': 'rem', 'seconds': 2640},
   {'dateTime': '2020-01-16T03:49:00.000', 'level': 'light', 'seconds': 630},
   {'dateTime': '2020-01-16T03:59:30.000', 'level': 'deep', 'seconds': 1560},
   {'dateTime': '2020-01-16T04:25:30.000', 'level': 'light', 'seconds': 3330},
   {'dateTime': '2020-01-16T05:21:00.000', 'level': 'rem', 'seconds': 660},
   {'dateTime': '2020-01-16T05:32:00.000', 'level': 'light', 'seconds': 930},
   {'dateTime': '2020-01-16T05:47:30.000', 'level': 'rem', 'seconds': 360},
   {'dateTime': '2020-01-16T05:53:30.000', 'level': 'light', 'seconds': 780},
   {'dateTime': '2020-01-16T06:06:30.000', 'level': 'wake', 'seconds': 600}],
  'shortData': [{'dateTime': '2020-01-15T23:58:30.000',
    'level': 'wake',
    'seconds': 90},
   {'dateTime': '2020-01-16T00:24:30.000', 'level': 'wake', 'seconds': 90},
   {'dateTime': '2020-01-16T01:31:00.000', 'level': 'wake', 'seconds': 120},
   {'dateTime': '2020-01-16T04:29:00.000', 'level': 'wake', 'seconds': 30},
   {'dateTime': '2020-01-16T04:44:00.000', 'level': 'wake', 'seconds': 90},
   {'dateTime': '2020-01-16T04:58:00.000', 'level': 'wake', 'seconds': 120},
   {'dateTime': '2020-01-16T05:03:00.000', 'level': 'wake', 'seconds': 60}],
  'summary': {'deep': {'count': 4, 'minutes': 44, 'thirtyDayAvgMinutes': 54},
   'light': {'count': 17, 'minutes': 283, 'thirtyDayAvgMinutes': 276},
   'rem': {'count': 4, 'minutes': 66, 'thirtyDayAvgMinutes': 65},
   'wake': {'count': 15, 'minutes': 69, 'thirtyDayAvgMinutes': 87}}},
```

```
 'logId': 25468095378,
 'minutesAfterWakeup': 0,
 'minutesAsleep': 393,
 'minutesAwake': 69,
 'minutesToFallAsleep': 0,
 'startTime': '2020-01-15T22:34:00.000',
 'timeInBed': 462,
 'type': 'stages'}
```

[24]:
```python
# 3. let's find the time the participant went to bed
night_index = 1
json_data['sleep'][night_index]['startTime']
print('Night starts at: '+json_data['sleep'][night_index]['startTime'])
print('Night end at: '+json_data['sleep'][night_index]['endTime'])
```

```
Night starts at: 2020-01-15T22:34:00.000
Night end at: 2020-01-16T06:16:30.000
```

[25]:
```python
# 4. total time in bed: put the total time spent in bed by the participant␣
↪(calculate it manually)

min_slept = 24-22 + 6 - (.34-.165)
min_slept
```

[25]: 7.825

[26]:
```python
# 5. let's extract the hypnogram !
# the code is written for you, but verify step by step what is happening (add␣
↪comments)!

from datetime import datetime
import time

#'1' = deep (N3) sleep - '2' = light (N1 or N2) sleep - '3' = REM sleep - '4' =␣
↪awake
dic_sleep = {'wake':4 , 'deep':1, 'light':2 , 'rem':3}

# Amount of data taken at a specific time period
n_sleep_logs = len(json_data['sleep'][night_index]['levels']['data'])

# total n of intervals (30 min each)
total_n_intervals = 7*60*2 + (26+16)*2 +1

# array of sleep status: from 11pm to 6am (420 minutes)
hypno = np.zeros(total_n_intervals)

# Converts the time he went to bed to a datetime object
```

```
time_start = datetime.strptime('2020-01-15T22:34:00.000', '%Y-%m-%dT%H:%M:%S.
↪%f')

current_sample = 0 #Starting sample
for ind in range(n_sleep_logs): # Loops through the amount of samples
  n_samples_sleep_log =␣
↪int(json_data['sleep'][night_index]['levels']['data'][ind]['seconds']/30) #␣
↪Records the amount of sleep phases per night in an array
  sleep_recorded =␣
↪json_data['sleep'][night_index]['levels']['data'][ind]['level'] # Records␣
↪each level of sleep per night in an array
  hypno[current_sample:current_sample+n_samples_sleep_log] =␣
↪dic_sleep[sleep_recorded] # Switches each indice of hypno with the␣
↪corresponding indice in sleep
  # to be an array of each sleep level as defined in dic_sleep
  current_sample = current_sample+n_samples_sleep_log #Becomes the amount of␣
↪sleep logs
```

[27]:
```
# 6. let's verify hypnogram is the right length
len(hypno)
```

[27]: 925

[28]:
```
# 7. let's write down the hypnogram to export,
# we need to write the phase of sleep (from 1 to 4) from 11:00pm to 6:00am ,␣
↪with sleep phase for each 30 seconds
total_number_30sec_int = 7*60*2

#7 hours of sleep, 60 minutes per hour, 60 seconds per minute, thus 2 sleep␣
↪phases of 30 seconds per minute

df = pd.DataFrame(columns = ['IndexTime','SleepPhase'])
df['IndexTime'] = range(total_number_30sec_int)

# let's cut the first and last 30-sec-intervals of hypno (outside the range of␣
↪interest)
intervals_before_11pm = int((60 - 34) * 2)
# -- calculate it manually (how many 30 sec intervals should we cut?)
intervals_after_6am = int(((16 + 0.5) * 2))
# -- calculate it manually (how many 30 sec intervals should we cut?)


# let's cut the first and last values of hypno (outside the range of interest)
df['SleepPhase'] = hypno[intervals_before_11pm:-(intervals_after_6am)]
df
```

[28]:

```
     IndexTime  SleepPhase
0            0         1.0
1            1         1.0
2            2         4.0
3            3         4.0
4            4         4.0
..         …          …
835        835         2.0
836        836         2.0
837        837         2.0
838        838         2.0
839        839         2.0

[840 rows x 2 columns]
```

[29]:

```
# 8. let's save the final csv file
df.to_csv('sensor_2_output.csv', index=False)
```

[30]:

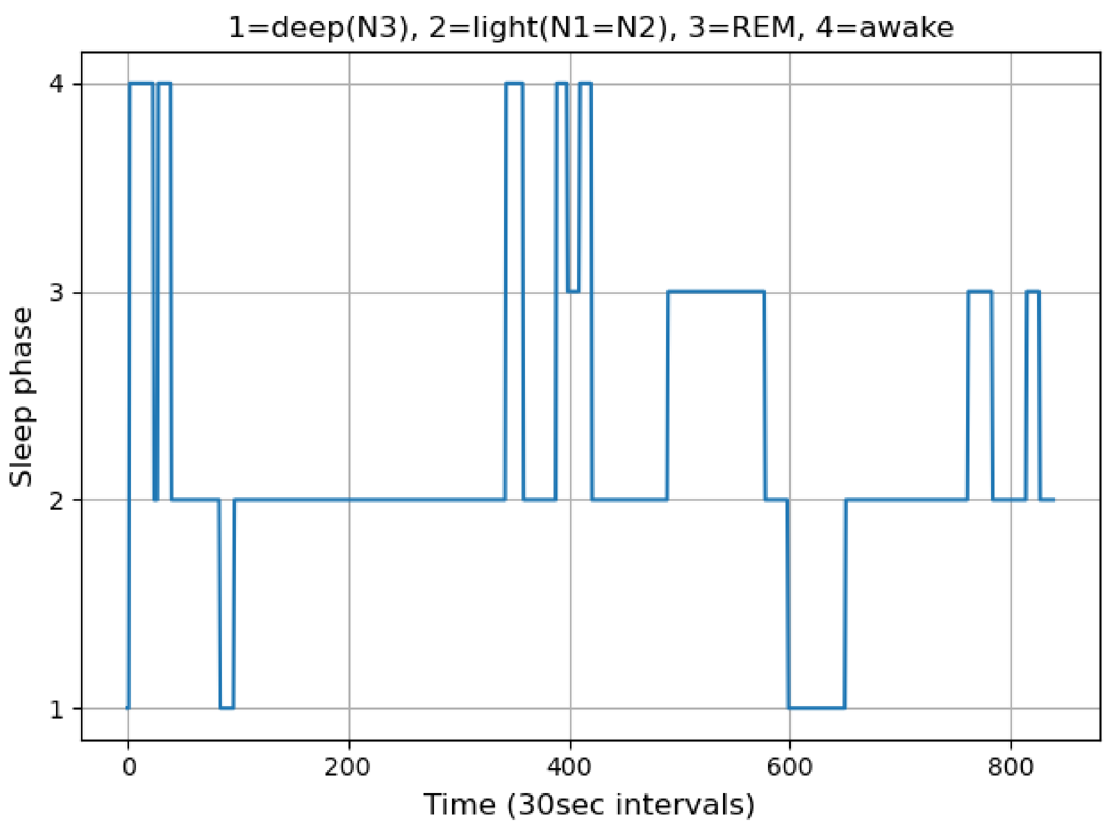
```
# 9. let's calculate amount of sleep per phase
# in number of intervals
# in percentage over the 7 hours of analysis

# -- write your code --
sleep_counts = df['SleepPhase'].value_counts(normalize=True).sort_index()
summary_df = (sleep_counts * 100).reset_index()
summary_df.columns = ['SleepPhase', 'Percentage']
summary_df['Intervals'] = (summary_df['Percentage'] / 100) * len(df)

print(summary_df)
```

```
   SleepPhase  Percentage  Intervals
0         1.0    7.976190       67.0
1         2.0   67.738095      569.0
2         3.0   15.833333      133.0
3         4.0    8.452381       71.0
```

[31]:

```
# 10. let's print the hypnogram!

# df = pd.read_csv('sensor_2_output.csv')

def plot_hypno(signal): #,color
  x = np.arange(len(signal))
  y = signal

  fontsizeV = 12
  mpl.plot(x,y) #color=color
  mpl.xlabel('Time (30sec intervals)',fontsize=fontsizeV)
```

```
  mpl.ylabel('Sleep phase',fontsize=fontsizeV)
  mpl.yticks([1,2,3,4])
  mpl.title('1=deep(N3), 2=light(N1=N2), 3=REM, 4=awake')
  mpl.grid(True)
  mpl.tight_layout()

# -- write your code --
# summary_df.sort_values(by='').plot(kind='line', x='Age', y='Weight')

plot_hypno(df.get(["SleepPhase"]))
```



[ ]:

```

```

[ ]:

```

```

psg

March 5, 2025

[167]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as mpl
```

[168]:
```
# let's look at the GOLD STANDARD, the polysomnography!
# you can find it in report_simplified.html

# 0. open it in your browser (click on it) and see how it looks like, you will␣
↪need it!
```

[169]:
```
#let's access this report; we will import all as pandas dataframes (this may␣
↪take up to 1 minute)
dfs = pd.read_html('report_simplified.html',header=0)

# NOTE: dfs is a list of pandas dataframes
type(dfs)
```

[169]:
```
list
```

[170]:
```
# 1. how many dataframes were collected?
n_dfs = len(dfs)
n_dfs
```

[170]:
```
24
```

[171]:
```
# 2. get a quick look at dataframes 0-3: do we need them?

# YOUR CODE HERE .

for i in range(4):
    print(f"DataFrame {i}:")
    print(dfs[i].head())
    print("\n")
```

```
DataFrame 0:
  General Information  General Information.1 General Information.2  \
0               Name:                    NaN                 PSXXX
1                 ID:                    NaN                   DDD
```

```
2              Sex:                    NaN                  XXX
3              Age:                    NaN        XX [1/1/19XX ]

  General Information.3
0                   NaN
1                   NaN
2                   NaN
3         Scorer: zzzzz


DataFrame 1:
  Unnamed: 0  Epochs
0      Awake     102
1    Stage 1      86
2    Stage 2     340
3    Stage 3     153
4    Stage 4       0


DataFrame 2:
  Lights Out:  1/15/2020 22:41:22
0  Lights On:  1/16/2020 06:12:50


DataFrame 3:
Empty DataFrame
Columns: [Unnamed: 0, 16 lights-on  epochs.  0 no-data  epochs scored as wake.]
Index: []
```

[172]:

```python
# 3. we have identified the dataframes we need; they are all of the same type,␣
 ↪let's create one dataframe
# with all the data we need!

# here I create a df with dfs[4] and dfs[5]
df_new = pd.concat([dfs[4], dfs[5]], ignore_index=True)

# let's create one datadrame with all the data we need!
# Epoch Number: from 1 to 919

# YOUR CODE HERE ...

df_new = pd.concat(dfs[4:], ignore_index=True)
df_new
```

[172]:
```
      # Epoch Number Stg  MicAr  {#}Any  Apnea  {#}Apnea  Obstructive  \
0                1   L      0             0                       0
1                2   L      0             0                       0
2                3   L      0             0                       0
3                4   L      0             0                       0
4                5   L      0             0                       0
..             …  ..     …          …                   …
914            915  N2      1             0                       0
915            916  N1      0             0                       0
916            917  N2      1             0                       0
917            918  N2      1             0                       0
918            919   L      0             0                       0

      {#} Apnea  Mixed  {#}Apnea  Central  {#}Hypopnea Total  \
0                  0                  0                  0
1                  0                  0                  0
2                  0                  0                  0
3                  0                  0                  0
4                  0                  0                  0
..               …                …                …
914                0                  0                  0
915                0                  0                  0
916                0                  0                  0
917                0                  0                  0
918                0                  0                  0

      SUM(#DESAT_100_REM, {#})  Desaturation  {#}Leg Movement    #
0                                       0                 0    1
1                                       0                 0    2
2                                       0                 0    3
3                                       0                 0    4
4                                       0                 0    5
..                                    …               …   …
914                                     0                 0  915
915                                     0                 0  916
916                                     0                 0  917
917                                     0                 0  918
918                                     0                 0  919

[919 rows x 11 columns]
```

[173]:
```
# 4. Create a new column in the dataframe with the same notation for the
 ↪hypnogram
# we defined: #'1' = deep (N3) sleep - '2' = light (N1 or N2) sleep - '3' = REM
 ↪sleep - '4' = awake

# create a new column and initialize it to value 0
```

```
df_new['hypno'] = 0

# what are the different values of sleep here?
print(set(np.array(df_new['Stg'])))
# ask if you do not know how to interpret one value!

# define a dictionary as we did last week

dic_sleep = {'L': 2, 'N1': 2, 'N2': 2, 'N3': 1, 'W': 4, 'R': 3}

# write the corresponding value in df_new['hypno']
# for j in range(len(df_new['hypno'])): df_new['hypno'][j] =␣
↪dic_sleep[df_new['Stg'][j]] #This way led to many errors, so I used .loc

for j in range(len(df_new['hypno'])):
    df_new.loc[j, 'hypno'] = dic_sleep[df_new['Stg'][j]]
```

```
{'L', 'N1', 'N2', 'N3', 'W', 'R'}
```

[174]:
```
# 5. PROBLEM: we have 919 30-sec intervals, we need 840 (from 11pm to 6am)

# from the report, they turned off the lights (and turned on again) at:
# Lights Out: 1/15/2020 22:41:22
# Lights On: 1/16/2020 06:12:50

# first, we need to remove intervals when the lights were still on:
print('Initial length hypno = '+str(len(df_new['hypno'])))

# first, we have data with lights on... we need to remove them (15 at the␣
↪beginning, 1 at the end)
hypno = np.array(df_new['hypno'])[15:-1]

print('Length excluding light on = '+str(len(hypno)))
```

```
Initial length hypno = 919
Length excluding light on = 903
```

[175]:
```
# 6. let's write down the hypnogram to export,
# we need to write the phase of sleep (from 1 to 4) from 11:00pm to 6:00am ,␣
↪with sleep phase for each 30 seconds
total_number_30sec_int = 7*60*2

df = pd.DataFrame(columns = ['IndexTime','SleepPhase'])
df['IndexTime'] = range(total_number_30sec_int)

# let's cut the first and last 30-sec-intervals of hypno (outside the range of␣
↪interest)
```

```
# from the report, they turned off the lights (and turned on again) at:
# Lights Out: 1/15/2020 22:41:22
# Lights On: 1/16/2020 06:12:50
intervals_before_11pm =  int(((18 * 60 + 38) / 30))
# -- calculate it manually (how many 30 sec intervals should we cut?)
intervals_after_6am = int(((12 * 61 + 51) / 30))
# -- calculate it manually (how many 30 sec intervals should we cut?)

# let's cut the first and last values of hypno (outside the range of interest)
df['SleepPhase'] = hypno[intervals_before_11pm:-(intervals_after_6am)]
```

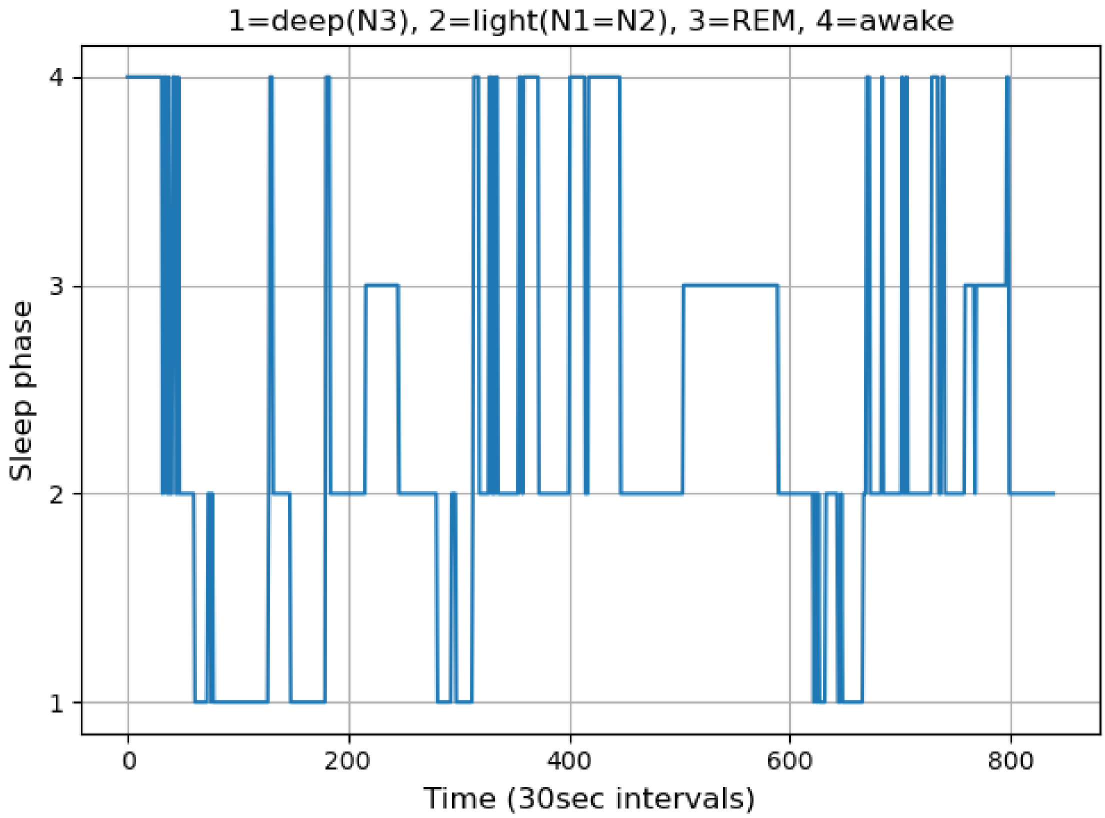
[176]:
```
# 7. let's save the final csv file as sensor_GS_output.csv

df.to_csv('sensor_GS_output.csv', index=False)
```

[177]:
```
# 8. let's calculate amount of sleep per phase
# in number of intervals
# in percentage over the 7 hours of analysis

# -- write your code --

phase_counts = df['SleepPhase'].value_counts().sort_index()
total_intervals = 840  # 7 hours

for phase in range(1, 5):
    count = phase_counts.get(phase, 0)
    percentage = (count / total_intervals) * 100
    print(f"Phase {phase}: {count} intervals, {percentage:.2f}%")
```

```
Phase 1: 153 intervals, 18.21%
Phase 2: 400 intervals, 47.62%
Phase 3: 153 intervals, 18.21%
Phase 4: 134 intervals, 15.95%
```

[178]:
```
# 9. let's print the hypnogram!

def plot_hypno(signal): #,color
  x = np.arange(len(signal))
  y = signal

  fontsizeV = 12
  mpl.plot(x,y) #color=color
  mpl.xlabel('Time (30sec intervals)',fontsize=fontsizeV)
  mpl.ylabel('Sleep phase',fontsize=fontsizeV)
  mpl.yticks([1,2,3,4])
  mpl.title('1=deep(N3), 2=light(N1=N2), 3=REM, 4=awake')
  mpl.grid(True)
```

```
  mpl.tight_layout()

# -- write your code --

plot_hypno(df['SleepPhase'])
mpl.show()
```



[ ]:

```

```

A04_Health_teams

March 5, 2025

1 Assignment 04

Submission by: see Canvas.

[5]:
```
# import packages
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as mpl
```

2 STEP 0

0: Write the scope of the project, what are you trying to achieve (1 point) we are trying to achieve a comparison between two sensors based on hypnograms plots and the varying accuracy of sensor data based on the type and measurement kind of the sensors. we will do this through putting the sleep data in a dataframe and then manipulating it to be put into a hynogram for both sensors adn then compare it to the gold standard corresponding and analyze these results on assignment 4

2.1 Assignment 04: how to

Your Group: Noah Baesa (Canvas Master), Shaun Israni (Inter-Group Facilitator), Tyler Valdez (Screen Master)

Your sensor (2)

1) put all the code you developed to extract data from json
2) follow point by point that .ipynb
3) add proper comments through your work; share your result with your complementary team (e.g., Team B:sensor 1 and Team B:sensor 2 should share their results)

Comparison

4) show statistical comparison:

- n of points that are different between Sensor 1 and 2, and between your sensor and the gold standard

5) plot hypnogram from each sensor and from the gold standard

6) plot interesting comparisons between the datasets (try it out)

- considering only deep sleep, which sensor is more accurete? and if we consider only REM?

7) prepare a pdf with your results (including the work to obtain your sensor data, and to compare it with the gold standard)

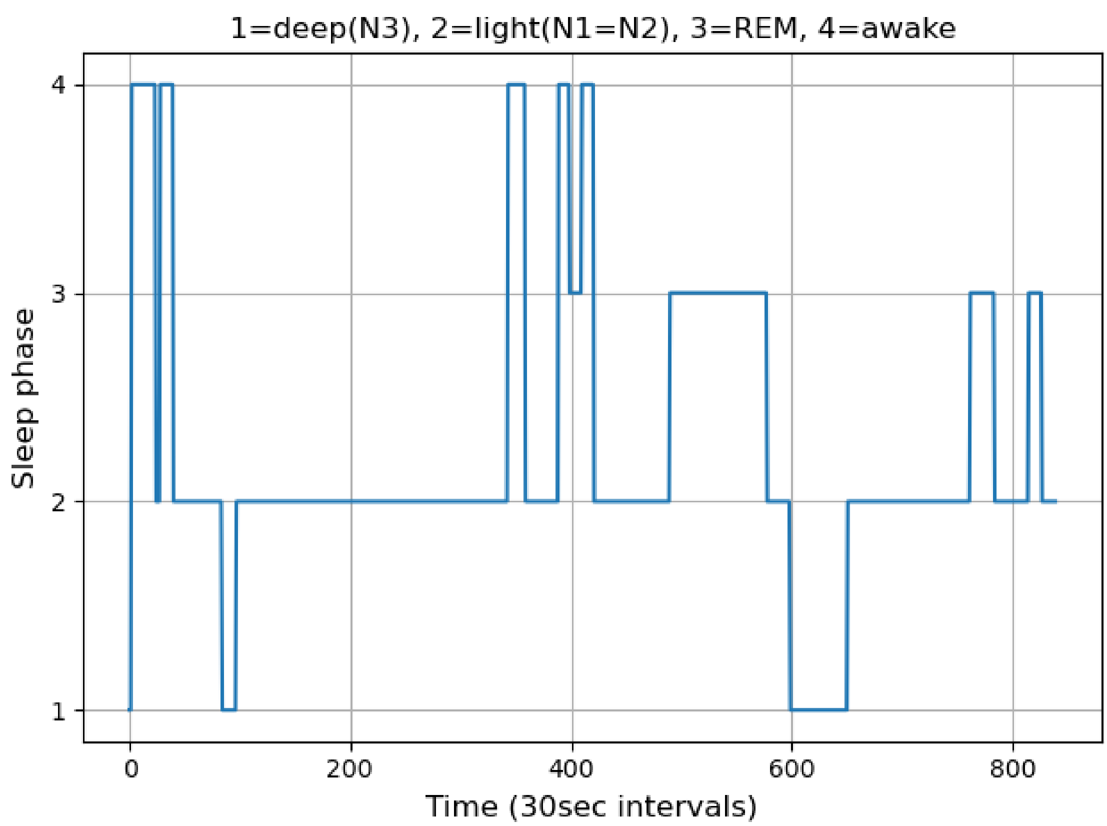
[6]:
```
# your code

import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as mpl

# let's import our file for the analysis:
json_file = 'sensor2/data/sensor2_sleep.json'

with open(json_file, 'r') as infile:
  json_data = json.load(infile)

# let's find the element related to the day of interest! do we have one element␣
↪per day?

# 1. let's find how many days are there

sum([1 for day in json_data['sleep']])

# 2. let's find the index of the day of interest; for each day, let's print␣
↪dateOfSleep , so we will know
# which element to use (the night is between 15 and 16 of Jan 2020)

# Hypnogram of 11pm to 6am

json_data['sleep'][1]

# 3. let's find the time the participant went to bed
night_index = 1
json_data['sleep'][night_index]['startTime']
print('Night starts at: '+json_data['sleep'][night_index]['startTime'])
print('Night end at: '+json_data['sleep'][night_index]['endTime'])

# 4. total time in bed: put the total time spent in bed by the participant␣
↪(calculate it manually)

min_slept = 24-22 + 6 - (.34-.165)
min_slept
```

```
# 5. let's extract the hypnogram !
# the code is written for you, but verify step by step what is happening (add␣
↪comments)!

from datetime import datetime
import time

#'1' = deep (N3) sleep - '2' = light (N1 or N2) sleep - '3' = REM sleep - '4' =␣
↪awake
dic_sleep = {'wake':4 , 'deep':1, 'light':2 , 'rem':3}

# Amount of data taken at a specific time period
n_sleep_logs = len(json_data['sleep'][night_index]['levels']['data'])

# total n of intervals (30 min each)
total_n_intervals = 7*60*2 + (26+16)*2 +1

# array of sleep status: from 11pm to 6am (420 minutes)
hypno = np.zeros(total_n_intervals)

# Converts the time he went to bed to a datetime object
time_start = datetime.strptime('2020-01-15T22:34:00.000', '%Y-%m-%dT%H:%M:%S.
↪%f')

current_sample = 0 #Starting sample
for ind in range(n_sleep_logs): # Loops through the amount of samples
  n_samples_sleep_log =␣
↪int(json_data['sleep'][night_index]['levels']['data'][ind]['seconds']/30) #␣
↪Records the amount of sleep phases per night in an array
  sleep_recorded =␣
↪json_data['sleep'][night_index]['levels']['data'][ind]['level'] # Records␣
↪each level of sleep per night in an array
  hypno[current_sample:current_sample+n_samples_sleep_log] =␣
↪dic_sleep[sleep_recorded] # Switches each indice of hypno with the␣
↪corresponding indice in sleep
  # to be an array of each sleep level as defined in dic_sleep
  current_sample = current_sample+n_samples_sleep_log #Becomes the amount of␣
↪sleep logs

# 6. let's verify hypnogram is the right length
len(hypno)


# 7. let's write down the hypnogram to export,
# we need to write the phase of sleep (from 1 to 4) from 11:00pm to 6:00am ,␣
↪with sleep phase for each 30 seconds
```

```python
total_number_30sec_int = 7*60*2

#7 hours of sleep, 60 minutes per hour, 60 seconds per minute, thus 2 sleep␣
↪phases of 30 seconds per minute

df = pd.DataFrame(columns = ['IndexTime','SleepPhase'])
df['IndexTime'] = range(total_number_30sec_int)

# let's cut the first and last 30-sec-intervals of hypno (outside the range of␣
↪interest)
intervals_before_11pm = int((60 - 34) * 2)
# -- calculate it manually (how many 30 sec intervals should we cut?)
intervals_after_6am = int(((16 + 0.5) * 2))
# -- calculate it manually (how many 30 sec intervals should we cut?)


# let's cut the first and last values of hypno (outside the range of interest)
df['SleepPhase'] = hypno[intervals_before_11pm:-(intervals_after_6am)]
df

# 8. let's save the final csv file
df.to_csv('sensor_2_output.csv', index=False)

# 9. let's calculate amount of sleep per phase
# in number of intervals
# in percentage over the 7 hours of analysis

# -- write your code --
sleep_counts = df['SleepPhase'].value_counts(normalize=True).sort_index()
summary_df = (sleep_counts * 100).reset_index()
summary_df.columns = ['SleepPhase', 'Percentage']
summary_df['Intervals'] = (summary_df['Percentage'] / 100) * len(df)

print(summary_df)

# 10. let's print the hypnogram!

# df = pd.read_csv('sensor_2_output.csv')

def plot_hypno(signal): #,color
  x = np.arange(len(signal))
  y = signal

  fontsizeV = 12
  mpl.plot(x,y) #color=color
  mpl.xlabel('Time (30sec intervals)',fontsize=fontsizeV)
  mpl.ylabel('Sleep phase',fontsize=fontsizeV)
```

```
  mpl.yticks([1,2,3,4])
  mpl.title('1=deep(N3), 2=light(N1=N2), 3=REM, 4=awake')
  mpl.grid(True)
  mpl.tight_layout()

# -- write your code --
# summary_df.sort_values(by='').plot(kind='line', x='Age', y='Weight')

plot_hypno(df.get(["SleepPhase"]))
```

```
Night starts at: 2020-01-15T22:34:00.000
Night end at: 2020-01-16T06:16:30.000
   SleepPhase  Percentage  Intervals
0         1.0    7.976190       67.0
1         2.0   67.738095      569.0
2         3.0   15.833333      133.0
3         4.0    8.452381       71.0
```

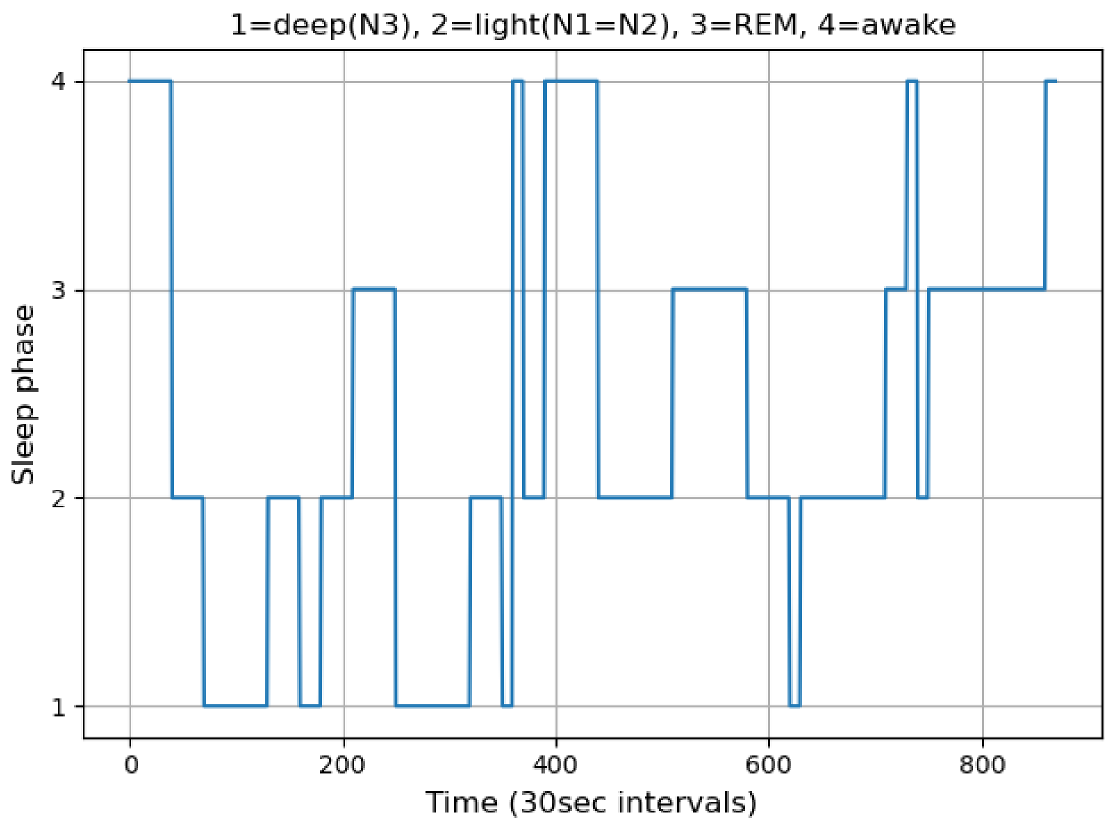


[8]:
```
# 1. import all hypnograms saved: hypno_sensor1 hypno_sensor2 hypno_GS

# my suggestion:
```

```python
# - read the csv file
# - transform the SleepPhase column into numpy

# you can also create some random hypnogram (made of random number 1 2 3 4),
↪they will be useful to check the differences: hypno_RAND hypno_RAND2

# YOUR CODE HERE ...

# imports

import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as mpl

# let's import our file for the analysis:

json_file = 'sensor1/data/sensor1_sleep.json'

with open(json_file, 'r') as infile:
  json_data = json.load(infile)

# index of interest
night_index = 2

# let's extract the hypnogram !

#'1' = deep (N3) sleep - '2' = light (N1 or N2) sleep - '3' = REM sleep - '4' =
↪awake
dic_sleep = {'wake':4 , 'deep':1, 'light':2 , 'rem':3}

# accesses values from json_data dictionary's key 'sleep'
# accesses values from night_index dictionary's key 'hypnogram_5min'
hypno_js = json_data['sleep'][night_index]['hypnogram_5min']

# the data in array form for plotting
hypno = np.array(list(hypno_js))

# we have 1 value every 5 minutes; we need 1 value every 30 seconds (to compare
↪it with gold standard)
# let's have each element repeated 10 times (2 per minute)
hypno = np.repeat(hypno,10)
hypno = hypno.astype(int)

# let's write down the hypnogram to export,
# we need to write the phase of sleep (from 1 to 4) from 11:00pm to 6:00am ,
↪with sleep phase for each 30 seconds
```

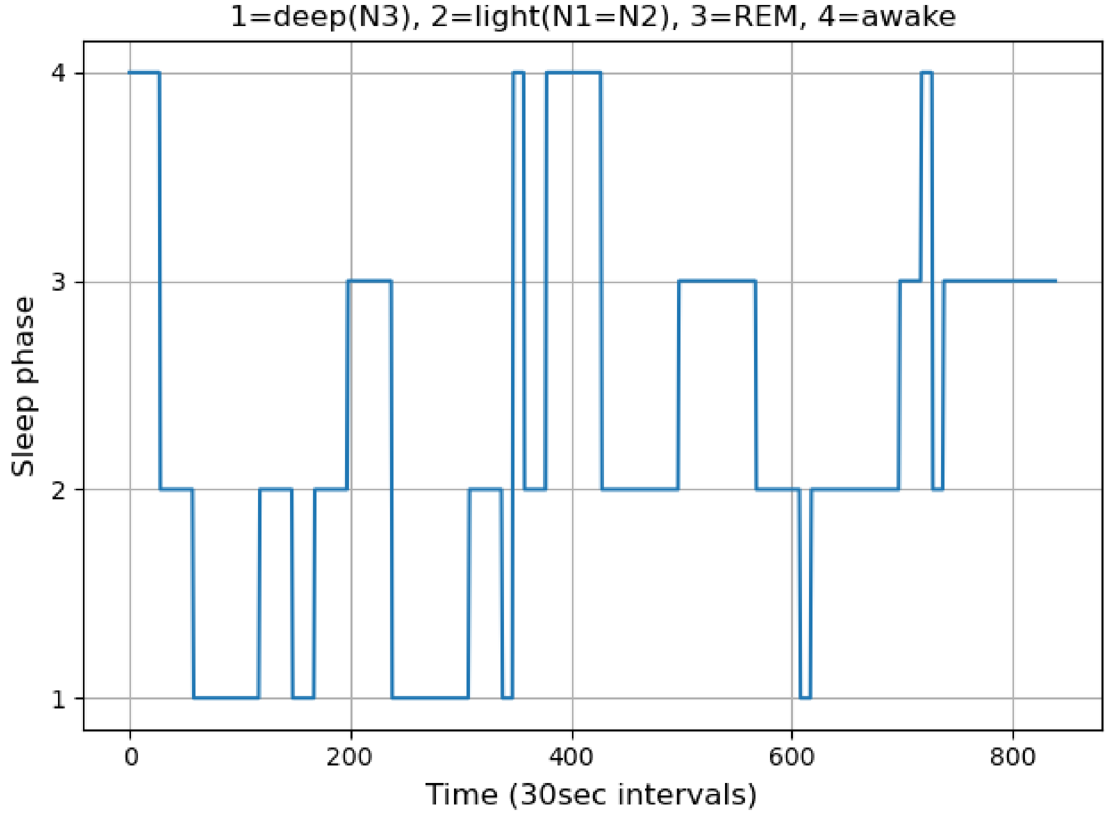
```
total_number_30sec_int = 7*60*2

# building the dataframe
df = pd.DataFrame(columns = ['IndexTime','SleepPhase'])
df['IndexTime'] = range(total_number_30sec_int)

# let's cut the first and last minutes of hypno (outside the range of interest)
minutes_before_11pm = 6
minutes_after_6am = 9

# consider only hypno in the range of interest
df['SleepPhase'] = hypno[minutes_before_11pm*2:-minutes_after_6am*2]

# let's save the final csv file
df.to_csv('sensor_1_output.csv', index=False)

# let's print the hypnogram!

def plot_hypno(signal): #,color
  x = np.arange(len(signal))
  y = signal

  fontsizeV = 12
  mpl.plot(x,y) #color=color
  mpl.xlabel('Time (30sec intervals)',fontsize=fontsizeV)
  mpl.ylabel('Sleep phase',fontsize=fontsizeV)
  mpl.yticks([1,2,3,4])
  mpl.title('1=deep(N3), 2=light(N1=N2), 3=REM, 4=awake')
  mpl.grid(True)
  mpl.tight_layout()

print(plot_hypno(hypno))

# let's import our file for t
```

```
None
```



[18]:
```
# 7. let's save the final csv file as sensor_GS_output.csv

df.to_csv('sensor_GS_output.csv', index=False)

# 9. let's print the hypnogram!

def plot_hypno(signal): #,color
  x = np.arange(len(signal))
  y = signal

  fontsizeV = 12
  mpl.plot(x,y) #color=color
  mpl.xlabel('Time (30sec intervals)',fontsize=fontsizeV)
  mpl.ylabel('Sleep phase',fontsize=fontsizeV)
  mpl.yticks([1,2,3,4])
  mpl.title('1=deep(N3), 2=light(N1=N2), 3=REM, 4=awake')
  mpl.grid(True)
  mpl.tight_layout()

# -- write your code --
```

```
plot_hypno(df['SleepPhase'])
mpl.show()
```



[20]:
```
sensor_1_csv = 'sensor_1_output.csv'
sensor_1_read = pd.read_csv(sensor_1_csv)
hypno_sensor1 = sensor_1_read.get('SleepPhase')

sensor_2_csv = 'sensor_2_output.csv'
sensor_2_read = pd.read_csv(sensor_2_csv)
hypno_sensor2 = sensor_2_read.get('SleepPhase')

psg_csv = 'gold_standard/sensor_GS_output.csv'
psg_read = pd.read_csv(psg_csv)
hypno_psg = psg_read.get('SleepPhase')

random_hypno = np.random.randint(1, 5, size=840)
```

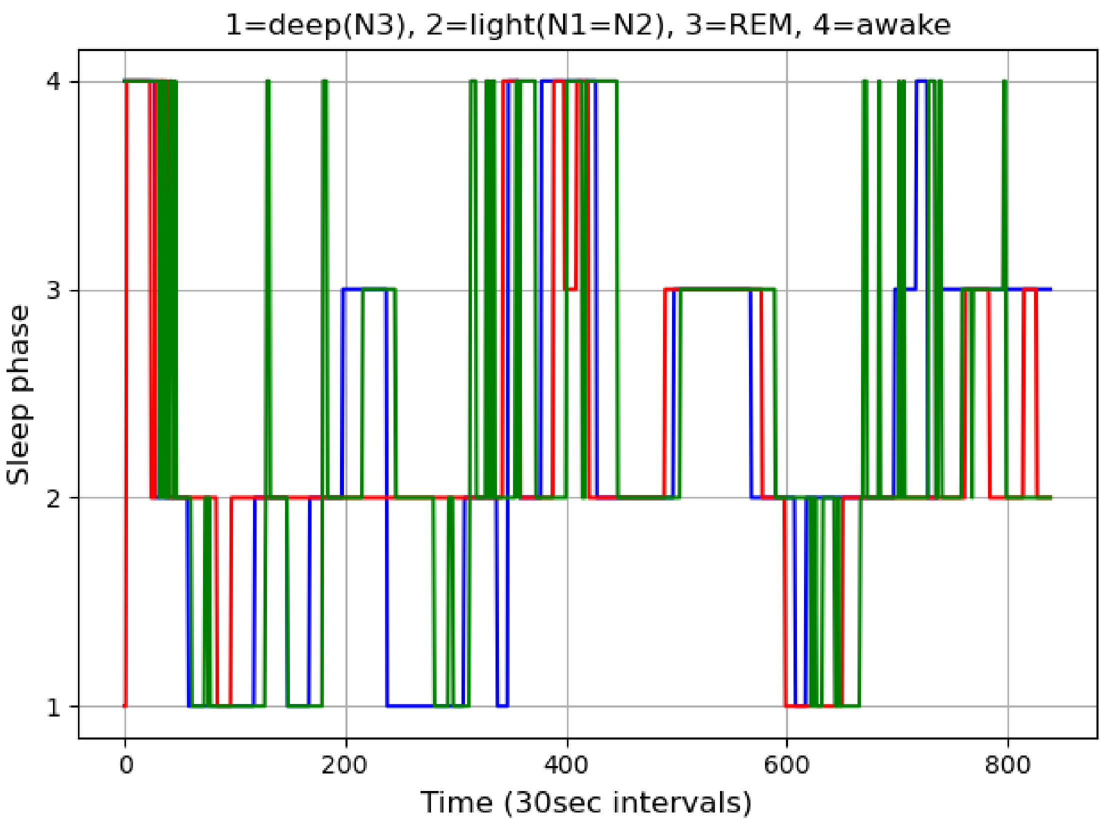
[22]:
```
# 2. let's calculate the norm 0 distance between these hypnograms
# if you want to learn more about norms: https://en.wikipedia.org/wiki/
↪Norm_(mathematics)
```

```
# example of norm 0 distance between two hypnograms (fraction of intervals that␣
↪are the same)
distance_1_2 = np.linalg.norm(hypno_sensor1 - hypno_sensor2,0)/
↪len(hypno_sensor1)

# # YOUR CODE HERE ... for all other differences!
distance_1_psg = np.linalg.norm(hypno_sensor1 - hypno_psg,0)/len(hypno_sensor1)
distance_2_psg = np.linalg.norm(hypno_sensor2 - hypno_psg,0)/len(hypno_sensor2)
distance_1_random = np.linalg.norm(hypno_sensor1 - random_hypno,0)/
↪len(hypno_sensor1)
distance_2_random = np.linalg.norm(hypno_sensor2 - random_hypno,0)/
↪len(hypno_sensor2)
distance_psg_random = np.linalg.norm(hypno_psg - random_hypno,0)/
↪len(random_hypno)

# are the two sensors different?
print(f"Difference between Sensor 1 and Sensor 2 is {distance_1_2}")

# which one is more accurate? is it significantly more accurate?
print(f"Difference between Sensor 1 and the psg is {distance_1_psg}")
print(f"Difference between Sensor 2 and the psg is {distance_2_psg}")

# what is the distance to a random signal? do we expect it to be this value?
print(f"Difference between Sensor 1 and a random is {distance_1_random}")
```

```
Difference between Sensor 1 and Sensor 2 is 0.46785714285714286
Difference between Sensor 1 and the psg is 0.4488095238095238
Difference between Sensor 2 and the psg is 0.4452380952380952
Difference between Sensor 1 and a random is 0.7273809523809524
```

[27]:
```
# 3. let's also print the signals
# print the three signals,
# and try to print two signals in the same figure so we can check the␣
↪differences!

def plot_hypno(signal,color):
  # signal is a numpy array
  # color is e.g., 'b', check here for your color: https://matplotlib.org/3.1.0/
↪gallery/color/named_colors.html
  x = np.arange(len(signal))
  y = signal

  fontsizeV = 12
  mpl.plot(x,y,color=color)
  mpl.xlabel('Time (30sec intervals)',fontsize=fontsizeV)
  mpl.ylabel('Sleep phase',fontsize=fontsizeV)
  mpl.yticks([1,2,3,4])
```

```
    mpl.title('1=deep(N3), 2=light(N1=N2), 3=REM, 4=awake')
    mpl.grid(True)
    mpl.tight_layout()

# YOUR CODE HERE ... (print all 3 hypnograms!!!)

plot_hypno(hypno_sensor1, "b")
plot_hypno(hypno_sensor2, "r")
plot_hypno(hypno_psg, "g")
```



[ ]:
```
# Conclusion

#clarify which sensor is better (1 point)
# the sensor 2 is better because we see less red than blue on the plotted␣
↪hynograms on the plot for the three hynograms. SInce the psg (gold standard)␣
↪hynogram is plotted last, that means that the green overtraces the red and␣
↪blue and with this we cadn see the time period in which each was accurate␣
↪and we see more blue than red, and beacuse the red is sensor 2 color of hte␣
↪data and blue is sensor 1 that means that sensor 2 is more accurate than␣
↪sensor 1
# remember to comment your results, and
```

```
# write a conclusion of your work with what you discovered (1 point)
# with this, we discovered that sensor 2 is better and more accurate than␣
↪sensor 1 and that the type of sensor2 is more effective than that of sensor␣
↪1 because it matches the gold standard at a higher percentage
```

```
[ ]: # Bonus point:
# 4. Can you design a confusion matrix to better highlight the differences␣
↪between the sensors?
```