duck_db_exploration

December 27, 2024

1 Duck DB Exploration with FitBit Data

The purpose of this notebook is to explore the versatility of DuckDB while exploring my data from my Fitbit. If you download your data from fitbit you will end up with a zipped file with many directories and a mix of JSON files and CSV files. Building a custom parser to get this data and organize it properly can be time consuming, especially since the JSON files have some deep nesting. Instead what we can do is have DuckDB read in these JSON files, execute SQL against them and then merge data together to explore it. My end goal for this exploration is to create a model to predict my HRV, so what I am going to need is data on my HRV, and I'm also going to bring in my sleep data and my daily heart rate data to see if there are good predictors there. I am fully aware that the HRV is likely not accurate on my Fitbit, but let's have some fun anyway! I will not be building models or building visualizations in this notebook, I will save that for another notebook.

1.1 Getting Started

Before we get going, just a little explanation of the data. Different versions of Fitbits will likely have different sensors and therefore different data. I haven't done any research to know if the directories are the same and I am not really concerned about that. I have a Charge 5 and I have noticed that there is a TON more data here than the application shows you. We have a lots of directories and data, let's take a look at how many files we have and what types. **Note: when I was developing this I used the** *.show()*** function for displaying results directly from DuckDB. This did a really good job displaying nested data in a tabular format, but the display looked terrible on the HTML export of the Jupyter Notebook, so I converted everything to a dataframe and displayed that instead, but the show function works just fine in jupyter, just not the export.**

[1]:
```
import pandas as pd
import os
from collections import defaultdict

def list_directories_and_count_files_by_type_with_summary(root_dir):
    overall_file_type_counts = defaultdict(int)  # To store the overall summary

    for dirpath, dirnames, filenames in os.walk(root_dir):
        file_type_counts = defaultdict(int)

        for filename in filenames:
            file_extension = os.path.splitext(filename)[1].lower()  # Get the␣
↪file extension
```

```
            file_type_counts[file_extension] += 1
            overall_file_type_counts[file_extension] += 1  # Add to overall␣
↪summary

        print(f"Directory: {dirpath}")
        for file_type, count in file_type_counts.items():
            print(f"  {file_type if file_type else '[No Extension]'}: {count}")
        print("-" * 40)

    # Print the overall summary
    print("Summary of all file types across all directories:")
    for file_type, total_count in overall_file_type_counts.items():
        print(f"  {file_type if file_type else '[No Extension]'}:␣
↪{total_count}")
    print("-" * 40)

root_directory = './data/unzipped/Takeout'
list_directories_and_count_files_by_type_with_summary(root_directory)
```

```
Directory: ./data/unzipped/Takeout
  .html: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Account Changes
  .csv: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Active Zone Minutes (AZM)
  .csv: 16
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Activity Goals
  .txt: 1
  .csv: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Atrial Fibrillation ECG
  .csv: 6
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Biometrics
  .txt: 1
  .csv: 210
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Daily Readiness
  .txt: 1
  .csv: 15
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Discover
```

```
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Fitbit Care or Programs
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Fitbit Friends
  .txt: 1
  .csv: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Fitbit Premium
  .txt: 1
  .csv: 2
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Global Export Data
  .json: 1116
  .csv: 473
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Guided Programs
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Health Fitness Data_GoogleData
  .txt: 5
  .csv: 6
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Heart Rate
  .csv: 2
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Heart Rate Variability
  .csv: 1430
  .txt: 4
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Menstrual Health
  .txt: 1
  .csv: 4
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Mindfulness
  .csv: 3
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Oxygen Saturation (SpO2)
  .csv: 478
  .txt: 2
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Paired Devices
  .txt: 1
  .csv: 5
----------------------------------------
```

```
Directory: ./data/unzipped/Takeout\Fitbit\Physical Activity_GoogleData
  .csv: 574
  .txt: 9
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Sleep
  .txt: 1
  .csv: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Sleep Score
  .csv: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Snore and Noise Detect
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Social
  .csv: 1
  .txt: 1
  .png: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Stress Journal
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Stress Score
  .txt: 1
  .csv: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Temperature
  .csv: 490
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Transactions
  .txt: 1
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\User Security Data
  .csv: 2
----------------------------------------
Directory: ./data/unzipped/Takeout\Fitbit\Your Profile
  .jpg: 1
  .csv: 1
  .txt: 1
----------------------------------------
Summary of all file types across all directories:
  .html: 1
  .csv: 3724
  .txt: 40
  .json: 1116
  .png: 1
  .jpg: 1
----------------------------------------
```

That is a lot of files, they're not very big but each style of file likely has it's own way of storing stuff. If it's not a CSV it's going to be annoying to get what we need. In addition to that, if we wanted to write SQL against it, loading data into a database like PostgreSQL would be annoyingly time consuming to setup, especially if our goal is to explore and analyze the data and not necessarily create an application out of it. We just want to organize the data well using simple SQL syntax and then get what we need for our analysis and that's what **DuckDB** is going to help us with. Okay enough chatting let's do something.

1.2 DuckDB - Getting Started

These few lines of code are all we need to load our sleep data. First we import the library, then we make a connection, you don't have to supply a string argument which creates a permanent database. If you don't supply an argument it's like having a temp database, all the rest of the commands in the notebook will work. I did it because later, I will want to store some data in a table, but for now it's not necessary. Finally the 3rd line is where we get a lot of help, we are going to read in all of the JSON files in the *Global Export Data* directory that begin with "sleep-". **DuckDB** will read the json data and store it in the sleep_data relation (what **DuckDB** uses).

[2]:
```
import duckdb
conn = duckdb.connect('./data/fitbit_db.duckdb') # creates a database
sleep_data = duckdb.read_json("./data/unzipped/Takeout/Fitbit/Global Export␣
 ↪Data/sleep-*.json")
print(f'The object type for sleep_data is {type(sleep_data)}')
```

```
The object type for sleep_data is <class 'duckdb.duckdb.DuckDBPyRelation'>
```

This did a pretty good job of reading in the data, it parsed the **dateOfSleep** properly and got the **mainSleep** as a *boolean* instead of a *varchar* which is pretty nice. Some more interesting points is that the **levels** field has a datatype of *struct* and inside we see all the nested JSON data. The first section is called **summary** which holds aggregate information about my different stages of sleep (deep, wake, light, & rem). It also has a bunch of NULL features tacked on to the end. After **summary** we have an array of **data** which shows the chronological sequence of each sleep stage and it's duration in seconds. There's also an array of data called **shortData** which contains small blips of time in chronological order. That makes sense because when I look at the *Sleep Timeline* section on the Fitbit app there are often short small bursts of time in the Awake section. This is likely the data that is displayed there.

Let's reimport the data and convert those timestamps so that we can use this data properly.

[3]:
```
sleep_data = duckdb.read_json("./data/unzipped/Takeout/Fitbit/Global Export␣
 ↪Data/sleep-*.json", timestamp_format="%Y-%m-%dT%H:%M:%S.%g")
display(sleep_data.df().head(5))
```

```
         logId dateOfSleep           startTime             endTime  duration  \
0  42724299942  2023-09-10 2023-09-09 21:34:00 2023-09-10 06:11:00  31020000
1  42712673606  2023-09-09 2023-09-08 22:14:00 2023-09-09 05:37:30  26580000
2  42703109427  2023-09-08 2023-09-07 21:48:00 2023-09-08 05:23:30  27300000
3  42688951099  2023-09-07 2023-09-06 21:58:00 2023-09-07 05:46:30  28080000
4  42676589452  2023-09-06 2023-09-05 22:10:30 2023-09-06 05:37:00  26760000
```

```
   minutesToFallAsleep  minutesAsleep  minutesAwake  minutesAfterWakeup  \
0                    0            457            60                   0
1                    0            400            43                   0
2                    0            398            57                   0
3                    0            413            55                   1
4                    0            376            70                  10

   timeInBed  efficiency    type  infoCode         logType  \
0        517          98  stages         0  auto_detected
1        443         100  stages         0  auto_detected
2        455          99  stages         0  auto_detected
3        468          98  stages         0  auto_detected
4        446          96  stages         0  auto_detected

                                              levels  mainSleep
0  {'summary': {'deep': {'count': 5, 'minutes': 1…       True
1  {'summary': {'deep': {'count': 4, 'minutes': 6…       True
2  {'summary': {'deep': {'count': 6, 'minutes': 7…       True
3  {'summary': {'deep': {'count': 5, 'minutes': 8…       True
4  {'summary': {'deep': {'count': 3, 'minutes': 8…       True
```

That was really easy and fast, what I'd like to do now is explore this dataset a bit. The goal here is to show simple ways to access and point out some interesting features. First you can describe the data, similar to what you are able to do with pandas.

[4]:
```
sleep_data.describe().df()
```

[4]:
```
     aggr         logId dateOfSleep            startTime              endTime  \
0   count  4.810000e+02         481                  481                  481
1    mean  4.506723e+10        None                 None                 None
2  stddev  1.513614e+09        None                 None                 None
3     min  4.240092e+10  2023-08-13  2023-08-12 22:16:00  2023-08-13 05:54:30
4     max  4.759902e+10  2024-11-27  2024-11-26 22:39:00  2024-11-27 04:01:30
5  median  4.511957e+10        None                 None                 None

       duration  minutesToFallAsleep  minutesAsleep  minutesAwake  \
0  4.810000e+02                481.0     481.000000    481.000000
1  2.732532e+07                  0.0     406.800416     48.496881
2  3.387921e+06                  0.0      52.882236     11.543419
3  5.460000e+06                  0.0      82.000000      1.000000
4  3.558000e+07                  0.0     543.000000     90.000000
5  2.748000e+07                  0.0     410.000000     48.000000

   minutesAfterWakeup   timeInBed  efficiency     type   infoCode  \
0          481.000000  481.000000  481.000000      481  481.000000
1            0.557173  455.422037   98.168399     None    0.008316
2            1.348361   56.465353    1.677895     None    0.128831
3            0.000000   91.000000   88.000000  classic    0.000000
```

```
4           10.000000  593.000000  100.000000   stages    2.000000
5            0.000000  458.000000   99.000000     None    0.000000

         logType                                             levels mainSleep
0            481                                                481       481
1           None                                               None      None
2           None                                               None      None
3  auto_detected  {'summary': {'deep': {'count': 1, 'minutes': 4…     false
4  auto_detected  {'summary': {'deep': NULL, 'wake': NULL, 'ligh…      true
5           None                                               None      None
```

I can see that I have 481 entries in this dataset, but from the **dateOfSleep** data I'm not convinced I have 1 record for every night as the date difference between 2023-08-13 and 2024-11-27 is 471 days not 481. I also know that some nights I probably forgot to wear my fitbit so I know the number should be even less than 471. I can also see that some columns aren't going to be very useful to me, like **minutesToFallAsleep** has no mean or stddev so it never changes from 0, similarly **logType** has the same 'auto_detected' value. Let's write some SQL to take a look at the data and see what's up.

[5]:
```
display(duckdb.sql("SELECT dateOfSleep, count(*) as count FROM sleep_data GROUP␣
  ↪BY dateOfSleep Having count(*) > 1").df())
display(duckdb.sql("select count(distinct dateOfSleep) as distinct_dates from␣
  ↪sleep_data").df())
```

	dateOfSleep	count
0	2023-11-09	2
1	2024-07-06	2
2	2023-10-10	2
3	2023-09-10	2
4	2024-03-08	2
5	2024-05-07	2
6	2024-08-05	2
7	2024-06-06	2
8	2024-10-04	2
9	2024-01-08	2
10	2023-12-09	2
11	2024-04-07	2
12	2024-02-07	2
13	2024-09-04	2
14	2023-11-14	2
15	2024-11-03	2

	distinct_dates
0	465

I was right 465 unique dates, but 481 rows so I have some duplicates for some days…something is funky with that data. Before I work on that I just wanted to point out what **DuckDB** was able to do for us, our 'table' is the object that we read in. We can easily write SQL syntax against files or anything we read in just by using it like we would a table in SQL…pretty cool.

Now I want to take a look at the data from those duplicate dates, SQL syntax to the rescue!

[6]:
```
display(duckdb.sql("""SELECT s.* from sleep_data s where dateOfSleep in
           (SELECT dateOfSleep FROM sleep_data GROUP BY dateOfSleep Having␣
 ↪count(*) > 1)
          order by dateOfSleep""").df())
display(duckdb.sql("""SELECT s.logId, unnest(levels.summary.deep) from␣
 ↪sleep_data s where dateOfSleep in
           (SELECT dateOfSleep FROM sleep_data GROUP BY dateOfSleep Having␣
 ↪count(*) > 1)
          order by dateOfSleep""").df())
```

```
          logId dateOfSleep           startTime             endTime  duration  \
0   42724299942  2023-09-10 2023-09-09 21:34:00 2023-09-10 06:11:00  31020000
1   42724299942  2023-09-10 2023-09-09 21:34:00 2023-09-10 06:11:00  31020000
2   43071124476  2023-10-10 2023-10-09 22:29:00 2023-10-10 05:40:30  25860000
3   43071124476  2023-10-10 2023-10-09 22:29:00 2023-10-10 05:40:30  25860000
4   43415888071  2023-11-09 2023-11-08 22:12:00 2023-11-09 05:32:30  26400000
5   43415888071  2023-11-09 2023-11-08 22:12:00 2023-11-09 05:32:30  26400000
6   43483210808  2023-11-14 2023-11-14 22:23:30 2023-11-14 23:55:00   5460000
7   43472303148  2023-11-14 2023-11-13 23:19:30 2023-11-14 06:01:30  24120000
8   43758999257  2023-12-09 2023-12-08 21:45:30 2023-12-09 04:42:00  24960000
9   43758999257  2023-12-09 2023-12-08 21:45:30 2023-12-09 04:42:00  24960000
10  44096210623  2024-01-08 2024-01-07 22:05:00 2024-01-08 05:31:00  26760000
11  44096210623  2024-01-08 2024-01-07 22:05:00 2024-01-08 05:31:00  26760000
12  44452696017  2024-02-07 2024-02-06 21:17:00 2024-02-07 05:30:30  29580000
13  44452696017  2024-02-07 2024-02-06 21:17:00 2024-02-07 05:30:30  29580000
14  44796156145  2024-03-08 2024-03-07 21:00:00 2024-03-08 05:31:30  30660000
15  44796156145  2024-03-08 2024-03-07 21:00:00 2024-03-08 05:31:30  30660000
16  45141553280  2024-04-07 2024-04-06 21:26:30 2024-04-07 06:41:30  33300000
17  45141553280  2024-04-07 2024-04-06 21:26:30 2024-04-07 06:41:30  33300000
18  45472556013  2024-05-07 2024-05-06 22:08:00 2024-05-07 05:21:30  25980000
19  45472556013  2024-05-07 2024-05-06 22:08:00 2024-05-07 05:21:30  25980000
20  45804854753  2024-06-06 2024-06-05 20:59:00 2024-06-06 05:32:30  30780000
21  45804854753  2024-06-06 2024-06-05 20:59:00 2024-06-06 05:32:30  30780000
22  46126383325  2024-07-06 2024-07-05 22:10:00 2024-07-06 06:11:30  28860000
23  46126383325  2024-07-06 2024-07-05 22:10:00 2024-07-06 06:11:30  28860000
24  46439171327  2024-08-05 2024-08-04 22:01:30 2024-08-05 05:31:30  27000000
25  46439171327  2024-08-05 2024-08-04 22:01:30 2024-08-05 05:31:30  27000000
26  46747290318  2024-09-04 2024-09-03 21:45:00 2024-09-04 06:04:30  29940000
27  46747290318  2024-09-04 2024-09-03 21:45:00 2024-09-04 06:04:30  29940000
28  47046675063  2024-10-04 2024-10-03 22:12:30 2024-10-04 05:51:00  27480000
29  47046675063  2024-10-04 2024-10-03 22:12:30 2024-10-04 05:51:00  27480000
30  47353917202  2024-11-03 2024-11-02 21:30:00 2024-11-03 05:48:30  29880000
31  47353917202  2024-11-03 2024-11-02 21:30:00 2024-11-03 05:48:30  29880000

    minutesToFallAsleep  minutesAsleep  minutesAwake  minutesAfterWakeup  \
0                     0            457            60                   0
```

1	0	457	60	0
2	0	384	47	0
3	0	384	47	0
4	0	399	41	1
5	0	399	41	1
6	0	82	9	0
7	0	362	40	7
8	0	373	43	2
9	0	373	43	2
10	0	395	51	1
11	0	395	51	1
12	0	427	66	0
13	0	427	66	0
14	0	465	46	0
15	0	465	46	0
16	0	499	56	0
17	0	499	56	0
18	0	385	48	0
19	0	385	48	0
20	0	448	65	0
21	0	448	65	0
22	0	418	63	0
23	0	418	63	0
24	0	418	32	1
25	0	418	32	1
26	0	445	54	0
27	0	445	54	0
28	0	409	49	0
29	0	409	49	0
30	0	427	71	0
31	0	427	71	0

	timeInBed	efficiency	type	infoCode	logType	\
0	517	98	stages	0	auto_detected	
1	517	98	stages	0	auto_detected	
2	431	99	stages	0	auto_detected	
3	431	99	stages	0	auto_detected	
4	440	98	stages	0	auto_detected	
5	440	98	stages	0	auto_detected	
6	91	90	classic	2	auto_detected	
7	402	98	stages	0	auto_detected	
8	416	97	stages	0	auto_detected	
9	416	97	stages	0	auto_detected	
10	446	98	stages	0	auto_detected	
11	446	98	stages	0	auto_detected	
12	493	99	stages	0	auto_detected	
13	493	99	stages	0	auto_detected	
14	511	100	stages	0	auto_detected	

```
15        511         100   stages         0  auto_detected
16        555          99   stages         0  auto_detected
17        555          99   stages         0  auto_detected
18        433          98   stages         0  auto_detected
19        433          98   stages         0  auto_detected
20        513          99   stages         0  auto_detected
21        513          99   stages         0  auto_detected
22        481          98   stages         0  auto_detected
23        481          98   stages         0  auto_detected
24        450          99   stages         0  auto_detected
25        450          99   stages         0  auto_detected
26        499         100   stages         0  auto_detected
27        499         100   stages         0  auto_detected
28        458          98   stages         0  auto_detected
29        458          98   stages         0  auto_detected
30        498          98   stages         0  auto_detected
31        498          98   stages         0  auto_detected

                                                levels  mainSleep
0   {'summary': {'deep': {'count': 5, 'minutes': 1…       True
1   {'summary': {'deep': {'count': 5, 'minutes': 1…       True
2   {'summary': {'deep': {'count': 5, 'minutes': 8…       True
3   {'summary': {'deep': {'count': 5, 'minutes': 8…       True
4   {'summary': {'deep': {'count': 5, 'minutes': 6…       True
5   {'summary': {'deep': {'count': 5, 'minutes': 6…       True
6   {'summary': {'deep': None, 'wake': None, 'ligh…      False
7   {'summary': {'deep': {'count': 3, 'minutes': 6…       True
8   {'summary': {'deep': {'count': 3, 'minutes': 8…       True
9   {'summary': {'deep': {'count': 3, 'minutes': 8…       True
10  {'summary': {'deep': {'count': 2, 'minutes': 6…       True
11  {'summary': {'deep': {'count': 2, 'minutes': 6…       True
12  {'summary': {'deep': {'count': 4, 'minutes': 7…       True
13  {'summary': {'deep': {'count': 4, 'minutes': 7…       True
14  {'summary': {'deep': {'count': 7, 'minutes': 9…       True
15  {'summary': {'deep': {'count': 7, 'minutes': 9…       True
16  {'summary': {'deep': {'count': 7, 'minutes': 8…       True
17  {'summary': {'deep': {'count': 7, 'minutes': 8…       True
18  {'summary': {'deep': {'count': 3, 'minutes': 7…       True
19  {'summary': {'deep': {'count': 3, 'minutes': 7…       True
20  {'summary': {'deep': {'count': 3, 'minutes': 1…       True
21  {'summary': {'deep': {'count': 3, 'minutes': 1…       True
22  {'summary': {'deep': {'count': 7, 'minutes': 1…       True
23  {'summary': {'deep': {'count': 7, 'minutes': 1…       True
24  {'summary': {'deep': {'count': 6, 'minutes': 9…       True
25  {'summary': {'deep': {'count': 6, 'minutes': 9…       True
26  {'summary': {'deep': {'count': 5, 'minutes': 1…       True
27  {'summary': {'deep': {'count': 5, 'minutes': 1…       True
28  {'summary': {'deep': {'count': 3, 'minutes': 8…       True
```

```
29  {'summary': {'deep': {'count': 3, 'minutes': 8…       True
30  {'summary': {'deep': {'count': 6, 'minutes': 8…       True
31  {'summary': {'deep': {'count': 6, 'minutes': 8…       True

          logId  count  minutes  thirtyDayAvgMinutes
0   42724299942    5.0    104.0                  0.0
1   42724299942    5.0    104.0                 81.0
2   43071124476    5.0     87.0                 80.0
3   43071124476    5.0     87.0                  0.0
4   43415888071    5.0     67.0                 88.0
5   43415888071    5.0     67.0                  0.0
6   43483210808    NaN      NaN                  NaN
7   43472303148    3.0     68.0                 74.0
8   43758999257    3.0     80.0                 77.0
9   43758999257    3.0     80.0                  0.0
10  44096210623    2.0     62.0                  0.0
11  44096210623    2.0     62.0                 91.0
12  44452696017    4.0     79.0                 79.0
13  44452696017    4.0     79.0                  0.0
14  44796156145    7.0     92.0                  0.0
15  44796156145    7.0     92.0                 77.0
16  45141553280    7.0     85.0                 74.0
17  45141553280    7.0     85.0                  0.0
18  45472556013    3.0     75.0                 80.0
19  45472556013    3.0     75.0                  0.0
20  45804854753    3.0    100.0                 73.0
21  45804854753    3.0    100.0                  0.0
22  46126383325    7.0    107.0                 73.0
23  46126383325    7.0    107.0                  0.0
24  46439171327    6.0     90.0                  0.0
25  46439171327    6.0     90.0                 73.0
26  46747290318    5.0    110.0                 81.0
27  46747290318    5.0    110.0                  0.0
28  47046675063    3.0     85.0                  0.0
29  47046675063    3.0     85.0                 79.0
30  47353917202    6.0     84.0                  0.0
31  47353917202    6.0     84.0                 81.0
```

Very interesting, it looks like we have duplicates in the 'levels', specifically in the summary. The **thirtyDayAverageMinutes** seems to be miscomputed for some reason, we can handle this by flattening out the data structure. In addition it looks like we have just 1 record where the **mainSleep** was set to false (2023-11-14), but at least those records have different **logId** fields.

One very helpful function I just ran shows how awesome **DuckDB** is, the *unnest* function can break down json data into it's individual parts. Depending on how the data is structured you can even refer to a lower level of data. In this case I was able to step into the **levels** field of data, into the **summary** subsection of **levels** and then grab one of the types of summaries for a sleep stage which was **deep** sleep and then flatten the data out. This is a very helpful and useful function when working with data and trying to reach important data nested in hierarchical structures.

What I'm going to do next is break up the data into different dataframes using **logId** as the key to link them all together, like setting up database tables with a link between them. I'll start with the outermost level of data, and for what I want to use this for, I don't need to look at sleep data that wasn't part of my main sleep.

```
[7]: sleep_meta_df = duckdb.sql("""Select distinct logId, dateOfSleep, startTime,␣
     ↪endTime, duration, minutesToFallAsleep, minutesAsleep,
                              minutesAwake, minutesAfterWakeup, timeInBed,␣
     ↪efficiency, type, infoCode, logType, mainSleep
                             FROM sleep_data
                             WHERE mainSleep == true""").df()
     display(sleep_meta_df.head(6))
```

```
         logId dateOfSleep           startTime             endTime  duration  \
0  45306958018  2024-04-22 2024-04-21 22:19:30 2024-04-22 05:31:30  25920000
1  46987212286  2024-09-28 2024-09-27 22:12:00 2024-09-28 07:05:30  31980000
2  46876496336  2024-09-17 2024-09-16 23:24:00 2024-09-17 06:12:00  24480000
3  46843361089  2024-09-14 2024-09-13 22:03:30 2024-09-14 04:32:30  23340000
4  46806686902  2024-09-10 2024-09-09 22:13:30 2024-09-10 06:04:30  28260000
5  46795305204  2024-09-09 2024-09-08 21:32:00 2024-09-09 06:09:30  31020000

   minutesToFallAsleep  minutesAsleep  minutesAwake  minutesAfterWakeup  \
0                    0            388            44                   2
1                    0            480            53                   1
2                    0            358            50                   0
3                    0            354            35                   3
4                    0            427            44                   0
5                    0            464            53                   1

   timeInBed  efficiency    type  infoCode        logType  mainSleep
0        432          98  stages         0  auto_detected       True
1        533         100  stages         0  auto_detected       True
2        408          99  stages         0  auto_detected       True
3        389          98  stages         0  auto_detected       True
4        471          99  stages         0  auto_detected       True
5        517          99  stages         0  auto_detected       True
```

By using the *distinct* clause on the outermost level of data I can eliminate all of the duplicates except for the one scenario where I had 2 records for the same night, for that I just filtered the data in the *where* clause for **mainSleep** == true.

Next I want the sleep summary data, the problem is I will need to first unnest the data from the levels.summary, but each stage of sleep (deep, rem, light, & awake) has it's own set of data and the key is the name of the stage. I want the name of the stage as a data value, as well as it's dataset, so what I've done here is build a couple of CTEs. First I unnest the first level, then I break down the different layers and add the key in for each and call that variable "stage". All of the stages have the same data (count, minutes, thirtyDayAverageMinutes) which is why this works so easily. Finally in order to get rid of the data problems we saw earlier where the thirtyDayAverages were zeros I just take the max for those, which avoids the duplicates and gives me my data.

[8]:
```
sleep_summary = duckdb.sql("""with summary_data as (select logId, unnest(levels.
↪summary, max_depth := 1) from sleep_data),
     unioned_data as (
        select logId, 'deep' as stage, unnest(deep) from summary_data
        union all
        select logId, 'wake' as stage, unnest(wake) from summary_data
        union all
        select logId, 'light' as stage, unnest(light) from summary_data
        union all
        select logId, 'rem' as stage, unnest(rem) from summary_data)
     select logId, stage, count, minutes, max(thirtyDayAvgMinutes) as␣
↪thirtyDayAvgMinutes
     from unioned_data
     group by logId, stage, count, minutes
     order by logId, stage""")
sleep_summary_df = sleep_summary.df()
display(sleep_summary_df.head(12))
```

	logId	stage	count	minutes	thirtyDayAvgMinutes
0	42400922920	deep	3.0	66.0	0.0
1	42400922920	light	30.0	272.0	0.0
2	42400922920	rem	7.0	78.0	0.0
3	42400922920	wake	28.0	42.0	0.0
4	42412586924	deep	4.0	86.0	66.0
5	42412586924	light	29.0	218.0	272.0
6	42412586924	rem	7.0	87.0	78.0
7	42412586924	wake	26.0	34.0	42.0
8	42424336860	deep	2.0	58.0	76.0
9	42424336860	light	30.0	260.0	245.0
10	42424336860	rem	7.0	93.0	83.0
11	42424336860	wake	31.0	54.0	38.0

Alright now we're getting somewhere! I could stop here as this is probably the level of detail I will want for my predictions, but let's keep going and get the other data. The next set of data is what makes up the summary, theoretically if we total up the data it should equal what we see above. Let's get the data first and then aggregate it.

[9]:
```
sleep_data_detail = duckdb.sql("""select logId, unnest(levels.data, recursive :
↪= True) from sleep_data order by logId""")
display(sleep_data_detail.df().head(12))
```

	logId	dateTime	level	seconds
0	42400922920	2023-08-12 22:16:00	wake	330
1	42400922920	2023-08-12 22:21:30	light	3960
2	42400922920	2023-08-12 23:27:30	rem	630
3	42400922920	2023-08-12 23:38:00	light	720
4	42400922920	2023-08-12 23:50:00	deep	3180
5	42400922920	2023-08-13 00:43:00	light	540
6	42400922920	2023-08-13 00:52:00	rem	930

```
7   42400922920 2023-08-13 01:07:30  light     3030
8   42400922920 2023-08-13 01:58:00   wake      360
9   42400922920 2023-08-13 02:04:00  light      180
10  42400922920 2023-08-13 02:07:00    rem     1560
11  42400922920 2023-08-13 02:33:00  light     3390
```

[10]:
```
sleep_data_agg = duckdb.sql("""select logId, level, count(*) as␣
↪count_of_stages, sum(seconds)/60 as total_minutes from sleep_data_detail
                            group by logId, level
                            order by logId, level""")
display(sleep_data_agg.df().head(12))
```

```
          logId  level  count_of_stages  total_minutes
0   42400922920   deep                3           66.5
1   42400922920  light               10          293.5
2   42400922920    rem                5           79.5
3   42400922920   wake                3           19.0
4   42412586924   deep                4           86.0
5   42412586924  light               10          240.0
6   42412586924    rem                6           88.0
7   42412586924   wake                2           11.5
8   42424336860   deep                2           59.0
9   42424336860  light               11          279.5
10  42424336860    rem                5           97.5
11  42424336860   wake                5           29.0
```

I definitely did not expect this but it looks like the data in the summary doesn't match the data supplied in the **data** field if you aggregate it. I can understand the time not matching up as the data looks like it only logs 30 second intervals, but the counts are off which is weird. Maybe the **shortData** field is also included and then both the **shortData** and **data** fields are aggregated, let's try that.

[11]:
```
sleep_short_data = duckdb.sql("""select logId, unnest(levels.shortData,␣
↪recursive := True) from sleep_data order by logId""")
display(sleep_short_data.df().head(10))
sleep_data_all_agg = duckdb.sql("""WITH full_sleep_data AS (
                                        select * from sleep_data_detail
                                        union all
                                        select * from sleep_short_data)
                                    select logId, level as stage, count(*) as␣
↪count_of_stages, sum(seconds)/60 as total_minutes from full_sleep_data
                                    group by logId, level
                                    order by logId, level""")
display(sleep_data_all_agg.df().head(12))
```

```
         logId            dateTime level  seconds
0  42400922920 2023-08-12 22:28:00  wake       30
1  42400922920 2023-08-12 23:11:00  wake       30
2  42400922920 2023-08-12 23:23:00  wake       30
```

```
3   42400922920 2023-08-12 23:40:30  wake       90
4   42400922920 2023-08-12 23:44:00  wake       30
5   42400922920 2023-08-13 00:42:00  wake       60
6   42400922920 2023-08-13 00:50:30  wake       30
7   42400922920 2023-08-13 01:04:00  wake       30
8   42400922920 2023-08-13 01:10:00  wake       30
9   42400922920 2023-08-13 01:16:30  wake       60
```

```
          logId  stage  count_of_stages  total_minutes
0   42400922920   deep                3           66.5
1   42400922920  light               10          293.5
2   42400922920    rem                5           79.5
3   42400922920   wake               28           42.5
4   42412586924   deep                4           86.0
5   42412586924  light               10          240.0
6   42412586924    rem                6           88.0
7   42412586924   wake               26           34.5
8   42424336860   deep                2           59.0
9   42424336860  light               11          279.5
10  42424336860    rem                5           97.5
11  42424336860   wake               31           54.0
```

It looks like that fixed the counts with **wake** but not much else, for a decent understanding of how different the data is we can join the 2 sets together and then take the differences. We can plot the data just to get an idea of how different the data is for both count and minutes. Like I mentioned before I am going to stick with the summary data as I have a feeling it's probably more accurate as it is precomputed. I can see the actual data is only accurate to a 30 second interval.

[12]:
```
sleep_data_diff = duckdb.sql("""select s.logId, s.stage, s.count as␣
↪summary_count_of_stages, s.minutes as summary_total_minutes,
                              d.count_of_stages as detail_count_of_stages, d.
↪total_minutes as detail_total_minutes,
                              s.count - d.count_of_stages as␣
↪summary_less_detail_count, s.minutes - d.total_minutes as␣
↪summary_less_detail_minutes
                              from sleep_summary s
                              inner join sleep_data_all_agg d
                              on s.logId = d.logId
                              and s.stage = d.stage
                              order by s.logId, s.stage""").df()
display(sleep_data_diff.head(12))
```

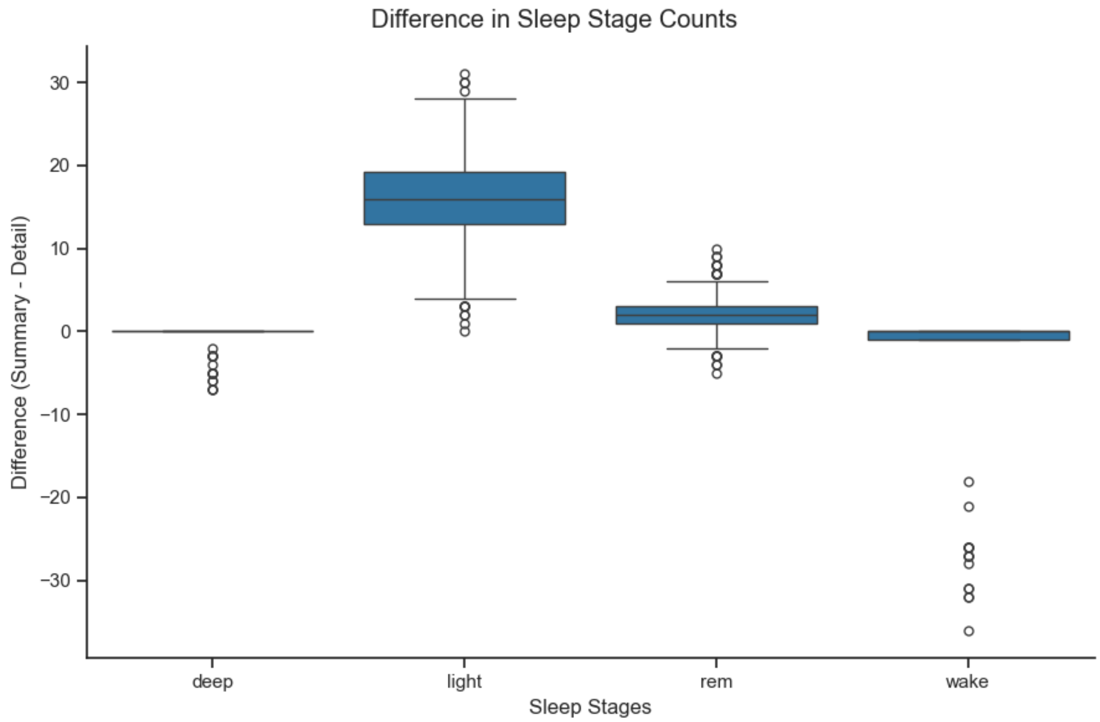
```
          logId  stage  summary_count_of_stages  summary_total_minutes  \
0   42400922920   deep                        3                     66
1   42400922920  light                       30                    272
2   42400922920    rem                        7                     78
3   42400922920   wake                       28                     42
4   42412586924   deep                        4                     86
5   42412586924  light                       29                    218
```

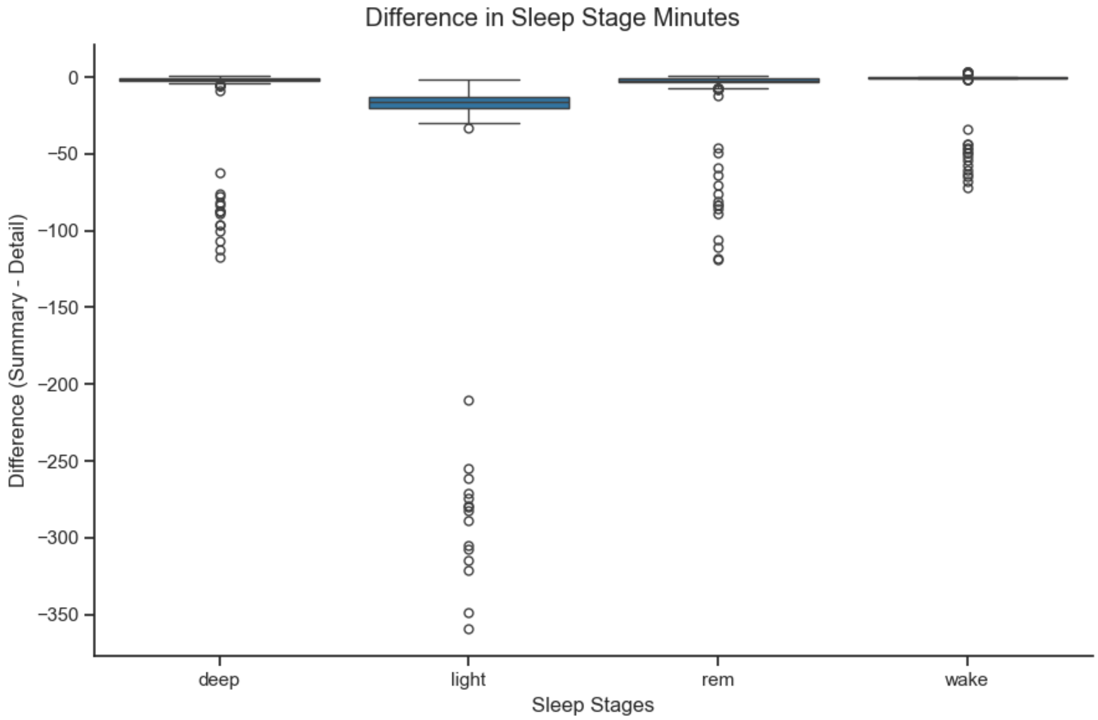
```
6   42412586924    rem                   7                     87
7   42412586924   wake                  26                     34
8   42424336860   deep                   2                     58
9   42424336860  light                  30                    260
10  42424336860    rem                   7                     93
11  42424336860   wake                  31                     54

    detail_count_of_stages  detail_total_minutes  summary_less_detail_count  \
0                        3                  66.5                          0
1                       10                 293.5                         20
2                        5                  79.5                          2
3                       28                  42.5                          0
4                        4                  86.0                          0
5                       10                 240.0                         19
6                        6                  88.0                          1
7                       26                  34.5                          0
8                        2                  59.0                          0
9                       11                 279.5                         19
10                       5                  97.5                          2
11                      31                  54.0                          0

    summary_less_detail_minutes
0                          -0.5
1                         -21.5
2                          -1.5
3                          -0.5
4                           0.0
5                         -22.0
6                          -1.0
7                          -0.5
8                          -1.0
9                         -19.5
10                         -4.5
11                          0.0
```

[13]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("ticks")
sns.set_context("notebook")
g = sns.catplot(data=sleep_data_diff,x="stage", y="summary_less_detail_count",␣
 ↪kind="box", height=6, aspect=1.5)
g.set_axis_labels("Sleep Stages", "Difference (Summary - Detail)")
g.fig.suptitle('Difference in Sleep Stage Counts')
g.fig.subplots_adjust(top=.93)
plt.show()
g = sns.catplot(data=sleep_data_diff,x="stage",␣
 ↪y="summary_less_detail_minutes", kind="box", height=6, aspect=1.5)
```

```
g.set_axis_labels("Sleep Stages", "Difference (Summary - Detail)")
g.fig.suptitle('Difference in Sleep Stage Minutes')
g.fig.subplots_adjust(top=.93)
plt.show()
```





It's clear with these 2 plots that our data doesn't really agree for *light* sleep stages, *deep* & *wake*, seem tolerable and *rem* is okay. That's all we need to know, I figured I'd break up the monotony of text, code, & data with a plot or 2. Now onto other datasets.

1.3 Heart Rate & HRV

We are going to be moving a little faster here so what we want to do is bring in the heart rate data. Both the actual measures of heart rate and then the heart rate variability data. The ***heart rate data*** is in a json format, which has a **bpm** and **confidence** measure for each timestamp. The **confidence** measure goes from 0 (no heart rate detected) to 3 (high confidence). The **dateTime** intervals seem to be irregular measures between 5 and 10 seconds. The ***HRV Data*** is a CSV and is a summary set of data that is gathered once a day, it represents the numbers you see on your sleep metrics for the Heart Rate Variability. There are other more frequent measures of HRV but it's not clear how they get this summary number from those so I'm sticking with the summary. The **rmssd** is the field that we are after for HRV and it's important to note that it takes this measure during sleep and will report it essentially after you wake up, but it gets assigned to that date too. For example if I start sleep on 2024-12-22 at 22:00:00 and wake up on 2024-12-23 at 06:00:00, my HRV score will be measured during sleep and assigned with the 2024-12-23 date.

1.4 Establishing The Right Dataset for Predictions

It is extremely important to understand if my goal is to create a dataset to predict HRV, I want to make this as realistic as possible so using my example date above, I should be taking data from the day of 2024-12-22 to predict the HRV I was assigned on 2024-12-23. Also if I truly wanted to make this a good quality model, I would also want to make sure I do not take any data while I am sleeping, that's kind of like cheating. I want to be able to predict something BEFORE it happens not during the event where it's being measured. In order to make this a strong model I am going to filter my data to 2 hours before I fall asleep, this way someone could actually go get FitBit data from Google, download it and insert it into the model to make a prediction for the HRV score. This also means a lot of data wrangling and filtering which is why I chose DuckDB as I know how to manipulate data a lot easier with SQL than I do with Pandas.

[14]:
```
heart_rate_data = duckdb.read_json('./data/unzipped/Takeout/Fitbit/Global␣
↪Export Data/heart_rate-*.json', timestamp_format="%m/%d/%y %H:%M:%S")
heart_rate_df = duckdb.sql('select dateTime, unnest(value) from␣
↪heart_rate_data').df()
display(heart_rate_df.head(10))

hrv_df = duckdb.read_csv('./data/unzipped/Takeout/Fitbit/Heart Rate Variability/
↪Daily Heart Rate Variability Summary*.csv', timestamp_format="%Y-%m-%dT%H:%M:
↪%S").df()
display(hrv_df.sort_values('timestamp').head(10))
```

```
             dateTime  bpm  confidence
0 2023-08-12 18:36:31   70           0
1 2023-08-12 18:36:46   64           0
2 2023-08-12 18:36:51   63           1
3 2023-08-12 18:36:56   65           1
4 2023-08-12 18:37:01   72           1
5 2023-08-12 18:37:06   76           1
6 2023-08-12 18:37:16   74           2
7 2023-08-12 18:37:21   73           2
8 2023-08-12 18:37:31   71           1
9 2023-08-12 18:37:36   70           1

      timestamp   rmssd  nremhr  entropy
0    2023-08-13  48.934  60.798    2.855
125  2023-08-14  40.485  67.520    2.632
364  2023-08-15  51.453  58.189    2.803
17   2023-08-16  37.462  64.676    2.446
336  2023-08-17  53.514  60.341    2.801
304  2023-08-18  48.468  62.162    2.742
394  2023-08-19  52.700  56.938    2.955
14   2023-08-20  54.221  61.383    3.103
2    2023-08-21  65.983  54.468    2.953
3    2023-08-22  50.733  57.393    2.787
```

In order to prepare the sleep summary data I am going to have to pivot it from a long format (multiple rows per observation) to a wide format (1 row per observation). There are a number of ways to do this in both DuckDB and Pandas. I am going to show you how to produce the same dataset by using *pivot* functions and then time the performance of both. Note that pandas is faster but in my opinion DuckDB is more readable. I only have the data that I have which is ~1800 rows. Perhaps if there was a lot more data DuckDB would be faster…perhaps not, all I can tell you is that not everything will be faster with DuckDB.

[15]:
```
%%timeit
adj_sleep_summary_pandas_df =␣
↪sleep_summary_df[['logId','stage','count','minutes','thirtyDayAvgMinutes']].
↪pivot(index='logId', columns='stage')
adj_sleep_summary_pandas_df.columns = [f"{col[1]}_{col[0]}" for col in␣
↪adj_sleep_summary_pandas_df.columns]

# Reset index for a regular looking DataFrame
adj_sleep_summary_pandas_df.reset_index(inplace=True)
```

```
1.32 ms ± 45  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

[16]:
```
%%timeit
adj_sleep_summary_duck_df = duckdb.sql("""pivot sleep_summary on stage
    using
        first(count) as count,
        first(minutes) as minutes,
        first(thirtyDayAvgMinutes) as thirtyDayAvgMinutes
    order by logId""")
```

```
423 ms ± 22.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[17]:
```
adj_sleep_summary_df = duckdb.sql("""pivot sleep_summary on stage
    using
        first(count) as count,
        first(minutes) as minutes,
        first(thirtyDayAvgMinutes) as thirtyDayAvgMinutes
    order by logId""").df()
display(adj_sleep_summary_df.head(10))
```

```
         logId  deep_count  deep_minutes  deep_thirtyDayAvgMinutes  \
0  42400922920         3.0          66.0                       0.0
1  42412586924         4.0          86.0                      66.0
2  42424336860         2.0          58.0                      76.0
3  42437088181         4.0         128.0                      70.0
4  42449293365         5.0          87.0                      85.0
5  42463261545         5.0          93.0                      85.0
6  42476649087         4.0          47.0                      86.0
7  42487823141         4.0          77.0                      81.0
8  42498649806         4.0          77.0                      80.0
9  42510611886         5.0          87.0                      80.0

   light_count  light_minutes  light_thirtyDayAvgMinutes  rem_count  \
0         30.0          272.0                        0.0        7.0
1         29.0          218.0                      272.0        7.0
2         30.0          260.0                      245.0        7.0
3         28.0          204.0                      250.0        7.0
4         23.0          228.0                      239.0        9.0
5         31.0          265.0                      236.0       12.0
```

```
6          34.0          284.0                          241.0        10.0
7          30.0          276.0                          247.0         7.0
8          27.0          242.0                          251.0         6.0
9          27.0          222.0                          250.0         6.0

   rem_minutes  rem_thirtyDayAvgMinutes  wake_count  wake_minutes  \
0         78.0                      0.0        28.0          42.0
1         87.0                     78.0        26.0          34.0
2         93.0                     83.0        31.0          54.0
3         72.0                     86.0        24.0          51.0
4         70.0                     83.0        27.0          51.0
5         81.0                     80.0        32.0          34.0
6         82.0                     80.0        35.0          42.0
7         75.0                     80.0        29.0          47.0
8        105.0                     80.0        27.0          39.0
9         86.0                     83.0        28.0          42.0

   wake_thirtyDayAvgMinutes
0                       0.0
1                      42.0
2                      38.0
3                      43.0
4                      45.0
5                      46.0
6                      44.0
7                      44.0
8                      44.0
9                      44.0
```

1.5

[18]:
```
print(adj_sleep_summary_df.shape[0])
print(sleep_meta_df.shape[0])
```

```
466
465
```

[19]:
```
complete_sleep_data = sleep_meta_df.merge(adj_sleep_summary_df, on='logId').
 ↪sort_values('dateOfSleep')
display(complete_sleep_data.tail(5))
```

```
           logId dateOfSleep           startTime             endTime  \
71   47560780256  2024-11-23 2024-11-23 00:19:00 2024-11-23 07:32:30
423  47568889247  2024-11-24 2024-11-23 22:22:00 2024-11-24 07:31:30
376  47578085269  2024-11-25 2024-11-24 22:42:30 2024-11-25 05:45:30
375  47589819717  2024-11-26 2024-11-25 22:29:30 2024-11-26 06:44:30
70   47599022157  2024-11-27 2024-11-26 22:39:00 2024-11-27 04:01:30

     duration  minutesToFallAsleep  minutesAsleep  minutesAwake  \
```

```
71   25980000                     0           379            54
423  32940000                     0           488            61
376  25380000                     0           372            51
375  29700000                     0           457            38
70   19320000                     0           290            32

     minutesAfterWakeup  timeInBed  …  deep_thirtyDayAvgMinutes light_count  \
71                    0        433  …                      71.0        22.0
423                   0        549  …                      70.0        38.0
376                   0        423  …                      71.0        19.0
375                   0        495  …                      72.0        21.0
70                    0        322  …                      72.0        14.0

     light_minutes light_thirtyDayAvgMinutes  rem_count  rem_minutes  \
71           279.0                     270.0        7.0         48.0
423          314.0                     270.0        7.0         75.0
376          233.0                     272.0        5.0         64.0
375          262.0                     271.0        7.0        104.0
70           202.0                     270.0        2.0         39.0

     rem_thirtyDayAvgMinutes  wake_count  wake_minutes  \
71                      70.0        23.0          54.0
423                     69.0        35.0          61.0
376                     69.0        20.0          51.0
375                     69.0        23.0          38.0
70                      70.0        13.0          32.0

     wake_thirtyDayAvgMinutes
71                       52.0
423                      52.0
376                      52.0
375                      52.0
70                       51.0

[5 rows x 27 columns]
```

If you're wondering why I didn't just use the SQL syntax below…I was just being lazy, with pandas if you use a merge (effectively an inner join), if you have the same name for the ***on*** column, it won't be repeated in the dataset. I could either write the SQL join and type out all the column names except for the second **logID** column or I could just use pandas, so I used pandas.

[20]:
```
complete_sleep_data = sleep_meta_df.merge(adj_sleep_summary_df, on='logId').
 ↪sort_values('dateOfSleep')
display(complete_sleep_data.tail(5))
```

```
           logId dateOfSleep           startTime             endTime  \
71   47560780256  2024-11-23 2024-11-23 00:19:00 2024-11-23 07:32:30
423  47568889247  2024-11-24 2024-11-23 22:22:00 2024-11-24 07:31:30
```

```
376  47578085269  2024-11-25 2024-11-24 22:42:30 2024-11-25 05:45:30
375  47589819717  2024-11-26 2024-11-25 22:29:30 2024-11-26 06:44:30
70   47599022157  2024-11-27 2024-11-26 22:39:00 2024-11-27 04:01:30

     duration  minutesToFallAsleep  minutesAsleep  minutesAwake  \
71   25980000                    0            379            54
423  32940000                    0            488            61
376  25380000                    0            372            51
375  29700000                    0            457            38
70   19320000                    0            290            32

     minutesAfterWakeup  timeInBed  …  deep_thirtyDayAvgMinutes light_count  \
71                    0        433  …                      71.0        22.0
423                   0        549  …                      70.0        38.0
376                   0        423  …                      71.0        19.0
375                   0        495  …                      72.0        21.0
70                    0        322  …                      72.0        14.0

     light_minutes light_thirtyDayAvgMinutes  rem_count  rem_minutes  \
71           279.0                     270.0        7.0         48.0
423          314.0                     270.0        7.0         75.0
376          233.0                     272.0        5.0         64.0
375          262.0                     271.0        7.0        104.0
70           202.0                     270.0        2.0         39.0

     rem_thirtyDayAvgMinutes  wake_count  wake_minutes  \
71                      70.0        23.0          54.0
423                     69.0        35.0          61.0
376                     69.0        20.0          51.0
375                     69.0        23.0          38.0
70                      70.0        13.0          32.0

     wake_thirtyDayAvgMinutes
71                       52.0
423                      52.0
376                      52.0
375                      52.0
70                       51.0

[5 rows x 27 columns]
```

1.6 Final Aggregation

Here is the final manipulation of the data where I am trying to establish the right dataset for a practical predictive model. I am using CTEs, the first is based on the sleep data which is essentially 1 record per day. I use the sleep data to establish the proper dates and times. I am using the **startTime** less 2 hours to the next row's **startTime** less 2 hours. From a temporal perspective the data starts 2 hours before I sleep, continues through my sleep and ends 2 hours before I fall asleep the next day. The **dateOfSleep** field records the date you woke up, so I want to predict the NEXT days HRV which is measured and computed during that sleeping interval. All of the heart rate data I want to use and aggregate is set by those start times which you can see in the second CTE, which is basically a cross join but where the heart rate is between the 2 start times established by the sleep data. The third and final CTE is used to aggregate that data so we have 1 record per sleep, so what we do is take measures using *max*, *min*, *std dev*, *median*, *10th percentile*, *90th percentile*, and the count of my bpm data, along with all the data from the sleeping records. Also just to see from a temporal perspective how well the previous HRV predicts the next days HRV I include that as a feature in the final query.

[21]:
```
final_df = duckdb.sql("""WITH Complete_Sleep AS (
                select *,
                    (dateOfSleep::Date + 1) as hrv_prediction_date,
                    (startTime - Interval '2 Hours') as␣
 ↪startTime_sleep_less_2_hrs,
                    ((lead(startTime) over (order by logId)) - Interval '2␣
 ↪Hours') as tomorrow_startTime_sleep_less_2_hrs
                from complete_sleep_data),
            HR_with_sleep AS (
                Select hr.*, cs.* from Complete_Sleep cs, heart_rate_df hr
                where hr.dateTime >= cs.startTime_sleep_less_2_hrs
                and hr.dateTime < tomorrow_startTime_sleep_less_2_hrs),
            hr_sleep_agg AS (
                Select dateOfSleep::date as dateOfSleep, hrv_prediction_date,
                max(bpm) as bpm_max, min(bpm) as bpm_min, round(avg(bpm), 2) as␣
 ↪bpm_avg, round(stddev_samp(bpm), 2) as bpm_std,
                round(median(bpm),2) as bpm_median, round(quantile_cont(bpm, 0.
 ↪10),2) as bpm_10th, round(quantile_cont(bpm, 0.90),2) as bpm_90th,
                count(bpm) as bpm_count, minutesAsleep,
                minutesAwake,
                timeInBed,
                deep_count,
                light_count,
                rem_count,
                wake_count,
                deep_minutes,
                light_minutes,
                rem_minutes,
                wake_minutes,
                deep_thirtyDayAvgMinutes,
                light_thirtyDayAvgMinutes,
                rem_thirtyDayAvgMinutes,
                wake_thirtyDayAvgMinutes
            from HR_with_sleep
            Group by dateOfSleep::date, hrv_prediction_date, minutesAsleep,␣
 ↪minutesAwake,
                timeInBed,
```

```
                deep_count,
                light_count,
                rem_count,
                wake_count,
                deep_minutes,
                light_minutes,
                rem_minutes,
                wake_minutes,
                deep_thirtyDayAvgMinutes,
                light_thirtyDayAvgMinutes,
                rem_thirtyDayAvgMinutes,
                wake_thirtyDayAvgMinutes
            )
        select agg.*, lag(rmssd) over (order by dateOfSleep) as prev_hrv, hrv.
↪rmssd as target_hrv from hr_sleep_agg agg inner join hrv_df hrv on agg.
↪hrv_prediction_date = hrv.timestamp Order by dateOfSleep
        """).df()
```

[22]:

```
pd.set_option('display.max_columns', None)
final_df.drop("hrv_prediction_date", axis=1, inplace=True)
display(final_df)
```

```
    dateOfSleep  bpm_max  bpm_min  bpm_avg  bpm_std  bpm_median  bpm_10th  \
0    2023-08-13      141       49    75.77    14.87        75.0      59.0
1    2023-08-14      121       50    80.37    12.20        82.0      64.0
2    2023-08-15      168       49    79.51    21.67        77.0      57.0
3    2023-08-16      140       49    82.52    16.05        84.0      61.0
4    2023-08-17      119       51    73.48    11.42        75.0      58.0
..          …        …        …        …        …           …         …
449  2024-11-21      122       48    72.00    13.56        73.0      55.0
450  2024-11-22      161       53    82.70    21.17        79.0      59.0
451  2024-11-23      105       51    71.38    11.18        71.0      58.0
452  2024-11-24      165       46    78.99    29.02        68.0      52.0
453  2024-11-25      130       47    71.56    14.60        70.0      54.0

     bpm_90th  bpm_count  minutesAsleep  minutesAwake  timeInBed  deep_count  \
0        96.0      11070            416            42        458         3.0
1        95.0      10169            391            34        425         4.0
2       105.0      10797            411            54        465         2.0
3       103.0      10225            404            51        455         4.0
4        87.0      10522            385            51        436         5.0
..          …          …              …             …          …           …
449      88.0      10101            379            50        429         3.0
450     115.0      11472            364            56        420         4.0
451      86.0       9222            379            54        433         3.0
452     129.0      12668            488            61        549         5.0
453      92.0       9869            372            51        423         3.0
```

```
     light_count  rem_count  wake_count  deep_minutes  light_minutes  \
0           30.0        7.0        28.0          66.0          272.0
1           29.0        7.0        26.0          86.0          218.0
2           30.0        7.0        31.0          58.0          260.0
3           28.0        7.0        24.0         128.0          204.0
4           23.0        9.0        27.0          87.0          228.0
..           …          …           …            …             …
449         20.0        5.0        19.0          79.0          248.0
450         23.0        3.0        20.0          62.0          237.0
451         22.0        7.0        23.0          52.0          279.0
452         38.0        7.0        35.0          99.0          314.0
453         19.0        5.0        20.0          75.0          233.0

     rem_minutes  wake_minutes  deep_thirtyDayAvgMinutes  \
0           78.0          42.0                       0.0
1           87.0          34.0                      66.0
2           93.0          54.0                      76.0
3           72.0          51.0                      70.0
4           70.0          51.0                      85.0
..           …             …                        …
449         52.0          50.0                      71.0
450         65.0          56.0                      72.0
451         48.0          54.0                      71.0
452         75.0          61.0                      70.0
453         64.0          51.0                      71.0

     light_thirtyDayAvgMinutes  rem_thirtyDayAvgMinutes  \
0                          0.0                      0.0
1                        272.0                     78.0
2                        245.0                     83.0
3                        250.0                     86.0
4                        239.0                     83.0
..                         …                        …
449                      273.0                     71.0
450                      272.0                     70.0
451                      270.0                     70.0
452                      270.0                     69.0
453                      272.0                     69.0

     wake_thirtyDayAvgMinutes  prev_hrv  target_hrv
0                         0.0       NaN      40.485
1                        42.0    40.485      51.453
2                        38.0    51.453      37.462
3                        43.0    37.462      53.514
4                        45.0    53.514      48.468
..                        …        …          …
449                      51.0    51.494      42.113
450                      51.0    42.113      44.060
```

```
451                         52.0    44.060      58.188
452                         52.0    58.188      60.853
453                         52.0    60.853      49.758

[454 rows x 26 columns]
```

1.7 Bonus: Correlation Filtering

When you're preparing data for a model you generally want to exclude data that is highly correlated. "Highly" correlated is ambiguous so I'm going to use 0.9 as a threshold. You can do this with code automatically but I chose the manual approach to show you. I also show both the correlation matrix and a plot, it's much easier to just see the data and find the correlations than it is to read a square matrix of numbers.

[23]:
```python
final_df.corr()
```

[23]:
```
                           dateOfSleep   bpm_max   bpm_min   bpm_avg  \
dateOfSleep                   1.000000 -0.034703 -0.222612 -0.233377
bpm_max                      -0.034703  1.000000 -0.032256  0.616479
bpm_min                      -0.222612 -0.032256  1.000000  0.435690
bpm_avg                      -0.233377  0.616479  0.435690  1.000000
bpm_std                       0.039252  0.862258 -0.222124  0.601060
bpm_median                   -0.342522  0.174308  0.495046  0.785332
bpm_10th                     -0.300051  0.002621  0.852287  0.551725
bpm_90th                      0.006009  0.770030  0.036941  0.789406
bpm_count                    -0.012505  0.392694 -0.014583  0.358291
minutesAsleep                 0.019802  0.019281 -0.208599 -0.230068
minutesAwake                  0.038499 -0.111518 -0.134464 -0.129557
timeInBed                     0.026381 -0.007233 -0.223246 -0.244961
deep_count                   -0.113175 -0.039884 -0.076466 -0.105497
light_count                  -0.071701  0.037702 -0.167459 -0.130345
rem_count                    -0.182631  0.017710  0.161166  0.057375
wake_count                   -0.136110  0.069070 -0.096298 -0.064391
deep_minutes                 -0.116257  0.040020 -0.019692  0.029081
light_minutes                 0.128219 -0.014968 -0.317837 -0.262923
rem_minutes                  -0.052281  0.066116  0.059835 -0.002994
wake_minutes                  0.044509 -0.101243 -0.149808 -0.101802
deep_thirtyDayAvgMinutes     -0.092159  0.011244  0.110658  0.155230
light_thirtyDayAvgMinutes     0.222209 -0.048390 -0.204543 -0.219024
rem_thirtyDayAvgMinutes      -0.046121 -0.021180  0.031532  0.031240
wake_thirtyDayAvgMinutes      0.186204 -0.026494 -0.006736 -0.053862
prev_hrv                      0.025463  0.027929 -0.729785 -0.402024
target_hrv                    0.025688 -0.294057 -0.225867 -0.318599

                            bpm_std  bpm_median  bpm_10th  bpm_90th  \
dateOfSleep                0.039252   -0.342522 -0.300051  0.006009
bpm_max                    0.862258    0.174308  0.002621  0.770030
bpm_min                   -0.222124    0.495046  0.852287  0.036941
```

bpm_avg	0.601060	0.785332	0.551725	0.789406
bpm_std	1.000000	0.108387	-0.216658	0.911308
bpm_median	0.108387	1.000000	0.603565	0.315115
bpm_10th	-0.216658	0.603565	1.000000	0.077574
bpm_90th	0.911308	0.315115	0.077574	1.000000
bpm_count	0.460249	0.070436	0.025891	0.500103
minutesAsleep	0.081034	-0.362392	-0.215607	-0.010050
minutesAwake	-0.017095	-0.144526	-0.126706	-0.039130
timeInBed	0.072211	-0.374562	-0.229250	-0.017987
deep_count	-0.017979	-0.108674	-0.073619	-0.049728
light_count	0.111178	-0.243340	-0.177767	0.044032
rem_count	-0.025332	0.002315	0.208387	-0.005131
wake_count	0.109687	-0.175163	-0.079483	0.050090
deep_minutes	0.042664	-0.025979	0.065534	0.041224
light_minutes	0.106530	-0.369781	-0.370895	0.007787
rem_minutes	0.034241	-0.082595	0.068026	0.015588
wake_minutes	0.008633	-0.124105	-0.137978	-0.009024
deep_thirtyDayAvgMinutes	-0.009105	0.186094	0.185833	0.036697
light_thirtyDayAvgMinutes	0.015408	-0.243644	-0.296932	-0.057919
rem_thirtyDayAvgMinutes	-0.009556	0.029352	0.056902	-0.002272
wake_thirtyDayAvgMinutes	-0.036509	-0.037234	-0.028147	-0.051236
prev_hrv	0.181155	-0.393489	-0.810916	-0.073119
target_hrv	-0.208096	-0.217225	-0.237110	-0.235557

	bpm_count	minutesAsleep	minutesAwake	timeInBed \
dateOfSleep	-0.012505	0.019802	0.038499	0.026381
bpm_max	0.392694	0.019281	-0.111518	-0.007233
bpm_min	-0.014583	-0.208599	-0.134464	-0.223246
bpm_avg	0.358291	-0.230068	-0.129557	-0.244961
bpm_std	0.460249	0.081034	-0.017095	0.072211
bpm_median	0.070436	-0.362392	-0.144526	-0.374562
bpm_10th	0.025891	-0.215607	-0.126706	-0.229250
bpm_90th	0.500103	-0.010050	-0.039130	-0.017987
bpm_count	1.000000	0.381207	0.169575	0.396812
minutesAsleep	0.381207	1.000000	0.161908	0.976578
minutesAwake	0.169575	0.161908	1.000000	0.364057
timeInBed	0.396812	0.976578	0.364057	1.000000
deep_count	0.104383	0.334886	0.094187	0.333569
light_count	0.233793	0.605575	0.260005	0.629663
rem_count	0.115941	0.320341	-0.132725	0.276157
wake_count	0.215900	0.583821	0.166249	0.589330
deep_minutes	0.237387	0.496881	0.025971	0.477823
light_minutes	0.200950	0.682627	0.315229	0.715528
rem_minutes	0.180984	0.509290	-0.299799	0.419898
wake_minutes	0.135607	0.118121	1.000000	0.328420
deep_thirtyDayAvgMinutes	-0.002358	-0.006983	0.028399	-0.002073
light_thirtyDayAvgMinutes	-0.020538	-0.025883	0.012103	-0.017975

rem_thirtyDayAvgMinutes	-0.073252	0.000271	-0.024911	0.002071
wake_thirtyDayAvgMinutes	0.011239	-0.037440	0.054628	-0.028304
prev_hrv	0.005807	0.116860	0.089138	0.128224
target_hrv	-0.264832	-0.101370	0.052049	-0.083061

	deep_count	light_count	rem_count	wake_count \
dateOfSleep	-0.113175	-0.071701	-0.182631	-0.136110
bpm_max	-0.039884	0.037702	0.017710	0.069070
bpm_min	-0.076466	-0.167459	0.161166	-0.096298
bpm_avg	-0.105497	-0.130345	0.057375	-0.064391
bpm_std	-0.017979	0.111178	-0.025332	0.109687
bpm_median	-0.108674	-0.243340	0.002315	-0.175163
bpm_10th	-0.073619	-0.177767	0.208387	-0.079483
bpm_90th	-0.049728	0.044032	-0.005131	0.050090
bpm_count	0.104383	0.233793	0.115941	0.215900
minutesAsleep	0.334886	0.605575	0.320341	0.583821
minutesAwake	0.094187	0.260005	-0.132725	0.166249
timeInBed	0.333569	0.629663	0.276157	0.589330
deep_count	1.000000	0.342781	0.102288	0.194909
light_count	0.342781	1.000000	0.174682	0.890613
rem_count	0.102288	0.174682	1.000000	0.389770
wake_count	0.194909	0.890613	0.389770	1.000000
deep_minutes	0.359378	0.250720	0.292509	0.294964
light_minutes	0.125919	0.579264	-0.089013	0.486325
rem_minutes	0.169642	0.077876	0.555045	0.147443
wake_minutes	0.094187	0.260005	-0.132725	0.166249
deep_thirtyDayAvgMinutes	0.077491	0.053393	0.057468	0.092272
light_thirtyDayAvgMinutes	-0.052548	-0.003848	-0.099303	-0.025634
rem_thirtyDayAvgMinutes	0.000782	-0.024750	0.035218	0.023740
wake_thirtyDayAvgMinutes	0.040581	-0.056804	-0.101375	-0.070096
prev_hrv	0.128256	0.139161	-0.128832	0.083330
target_hrv	-0.045115	-0.054337	-0.024360	-0.035905

	deep_minutes	light_minutes	rem_minutes \
dateOfSleep	-0.116257	0.128219	-0.052281
bpm_max	0.040020	-0.014968	0.066116
bpm_min	-0.019692	-0.317837	0.059835
bpm_avg	0.029081	-0.262923	-0.002994
bpm_std	0.042664	0.106530	0.034241
bpm_median	-0.025979	-0.369781	-0.082595
bpm_10th	0.065534	-0.370895	0.068026
bpm_90th	0.041224	0.007787	0.015588
bpm_count	0.237387	0.200950	0.180984
minutesAsleep	0.496881	0.682627	0.509290
minutesAwake	0.025971	0.315229	-0.299799
timeInBed	0.477823	0.715528	0.419898
deep_count	0.359378	0.125919	0.169642

light_count	0.250720	0.579264	0.077876
rem_count	0.292509	-0.089013	0.555045
wake_count	0.294964	0.486325	0.147443
deep_minutes	1.000000	-0.109591	0.329167
light_minutes	-0.109591	1.000000	-0.139736
rem_minutes	0.329167	-0.139736	1.000000
wake_minutes	0.025971	0.315229	-0.299799
deep_thirtyDayAvgMinutes	0.013402	-0.019801	0.006406
light_thirtyDayAvgMinutes	-0.113682	0.063728	-0.058717
rem_thirtyDayAvgMinutes	0.031306	-0.022027	0.009317
wake_thirtyDayAvgMinutes	0.005048	-0.026890	-0.037836
prev_hrv	-0.013861	0.212744	-0.065960
target_hrv	-0.078151	0.027930	-0.175154

	wake_minutes	deep_thirtyDayAvgMinutes \
dateOfSleep	0.044509	-0.092159
bpm_max	-0.101243	0.011244
bpm_min	-0.149808	0.110658
bpm_avg	-0.101802	0.155230
bpm_std	0.008633	-0.009105
bpm_median	-0.124105	0.186094
bpm_10th	-0.137978	0.185833
bpm_90th	-0.009024	0.036697
bpm_count	0.135607	-0.002358
minutesAsleep	0.118121	-0.006983
minutesAwake	1.000000	0.028399
timeInBed	0.328420	-0.002073
deep_count	0.094187	0.077491
light_count	0.260005	0.053393
rem_count	-0.132725	0.057468
wake_count	0.166249	0.092272
deep_minutes	0.025971	0.013402
light_minutes	0.315229	-0.019801
rem_minutes	-0.299799	0.006406
wake_minutes	1.000000	0.028399
deep_thirtyDayAvgMinutes	0.028399	1.000000
light_thirtyDayAvgMinutes	0.012103	0.001759
rem_thirtyDayAvgMinutes	-0.024911	0.445296
wake_thirtyDayAvgMinutes	0.054628	0.375852
prev_hrv	0.099092	-0.131735
target_hrv	0.060423	-0.104745

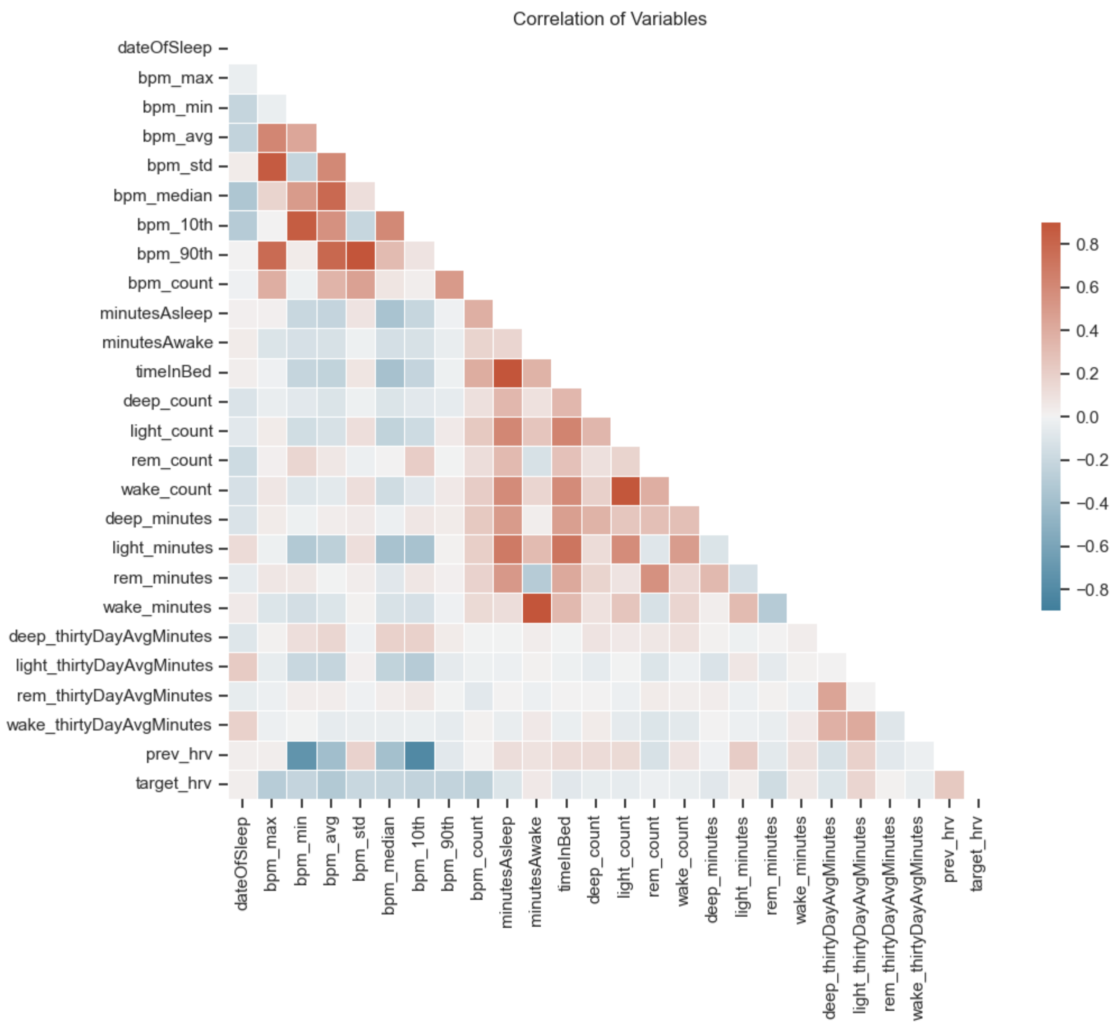
	light_thirtyDayAvgMinutes	rem_thirtyDayAvgMinutes \
dateOfSleep	0.222209	-0.046121
bpm_max	-0.048390	-0.021180
bpm_min	-0.204543	0.031532
bpm_avg	-0.219024	0.031240

bpm_std	0.015408	-0.009556
bpm_median	-0.243644	0.029352
bpm_10th	-0.296932	0.056902
bpm_90th	-0.057919	-0.002272
bpm_count	-0.020538	-0.073252
minutesAsleep	-0.025883	0.000271
minutesAwake	0.012103	-0.024911
timeInBed	-0.017975	0.002071
deep_count	-0.052548	0.000782
light_count	-0.003848	-0.024750
rem_count	-0.099303	0.035218
wake_count	-0.025634	0.023740
deep_minutes	-0.113682	0.031306
light_minutes	0.063728	-0.022027
rem_minutes	-0.058717	0.009317
wake_minutes	0.012103	-0.024911
deep_thirtyDayAvgMinutes	0.001759	0.445296
light_thirtyDayAvgMinutes	1.000000	0.001085
rem_thirtyDayAvgMinutes	0.001085	1.000000
wake_thirtyDayAvgMinutes	0.409954	-0.085060
prev_hrv	0.189455	-0.076416
target_hrv	0.166764	0.011357

	wake_thirtyDayAvgMinutes	prev_hrv	target_hrv
dateOfSleep	0.186204	0.025463	0.025688
bpm_max	-0.026494	0.027929	-0.294057
bpm_min	-0.006736	-0.729785	-0.225867
bpm_avg	-0.053862	-0.402024	-0.318599
bpm_std	-0.036509	0.181155	-0.208096
bpm_median	-0.037234	-0.393489	-0.217225
bpm_10th	-0.028147	-0.810916	-0.237110
bpm_90th	-0.051236	-0.073119	-0.235557
bpm_count	0.011239	0.005807	-0.264832
minutesAsleep	-0.037440	0.116860	-0.101370
minutesAwake	0.054628	0.089138	0.052049
timeInBed	-0.028304	0.128224	-0.083061
deep_count	0.040581	0.128256	-0.045115
light_count	-0.056804	0.139161	-0.054337
rem_count	-0.101375	-0.128832	-0.024360
wake_count	-0.070096	0.083330	-0.035905
deep_minutes	0.005048	-0.013861	-0.078151
light_minutes	-0.026890	0.212744	0.027930
rem_minutes	-0.037836	-0.065960	-0.175154
wake_minutes	0.054628	0.099092	0.060423
deep_thirtyDayAvgMinutes	0.375852	-0.131735	-0.104745
light_thirtyDayAvgMinutes	0.409954	0.189455	0.166764
rem_thirtyDayAvgMinutes	-0.085060	-0.076416	0.011357

```
wake_thirtyDayAvgMinutes                    1.000000 -0.027819   -0.035322
prev_hrv                                   -0.027819  1.000000    0.232980
target_hrv                                 -0.035322  0.232980    1.000000
```

[24]:
```
import numpy as np
corr = final_df.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=0.9, vmin=-0.9, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.title('Correlation of Variables')
plt.show()
```



Based on what I can see we have high (>= 0.90) correlation between a few different variables **wake_minutes** & **minutesAwake**, **timeInBed** & **minutesAsleep**, and **bpm_std** & **bpm_90th**. So I've decided to drop the **bpm_std**, **minutesAwake**, & **timeInBed** so the final filtered dataframe is below.

[25]:
```
final_filt_df = final_df[['dateOfSleep', 'bpm_max', 'bpm_min', 'bpm_avg',
        'bpm_median', 'bpm_10th', 'bpm_90th', 'bpm_count',
        'minutesAsleep', 'deep_count',
        'light_count', 'rem_count', 'wake_count', 'deep_minutes',
        'light_minutes', 'rem_minutes', 'wake_minutes',
        'deep_thirtyDayAvgMinutes', 'light_thirtyDayAvgMinutes',
        'rem_thirtyDayAvgMinutes', 'wake_thirtyDayAvgMinutes', 'prev_hrv',
 'target_hrv']]
```

1.8 Conclusion

So we were able to use DuckDB to read in and manipulate json data, csv data, and dataframes with ease. We took the data we had with a goal in mind of being able to predict heart rate variability so we setup the data to do that and even filtered out highly correlated variables. Our data is now prepared for a machine learning model. Below I have put only the necessary code together to import and organize the data. I could probably make it more efficient but it runs in ~7 seconds to read in the sleep data, heart rate data, and heart rate variability data and manipulate it all to create our final dataset.

[26]:
```
# organize it all
sleep_data = duckdb.read_json("./data/unzipped/Takeout/Fitbit/Global Export␣
↪Data/sleep-*.json", timestamp_format="%Y-%m-%dT%H:%M:%S.%g")
sleep_meta_df = duckdb.sql("""Select distinct logId, dateOfSleep, startTime,␣
↪endTime, duration, minutesToFallAsleep, minutesAsleep,
                            minutesAwake, minutesAfterWakeup, timeInBed,␣
↪efficiency, type, infoCode, logType, mainSleep
                           FROM sleep_data
                           WHERE mainSleep == true""").df()
sleep_summary_df = duckdb.sql("""with summary_data as (select logId,␣
↪unnest(levels.summary, max_depth := 1) from sleep_data),
      unioned_data as (
         select logId, 'deep' as stage, unnest(deep) from summary_data
         union all
         select logId, 'wake' as stage, unnest(wake) from summary_data
         union all
         select logId, 'light' as stage, unnest(light) from summary_data
         union all
         select logId, 'rem' as stage, unnest(rem) from summary_data)
      select logId, stage, count, minutes, max(thirtyDayAvgMinutes) as␣
↪thirtyDayAvgMinutes
      from unioned_data
      group by logId, stage, count, minutes
      order by logId, stage""").df()
sleep_data_detail = duckdb.sql("""select logId, unnest(levels.data, recursive :
↪= True) from sleep_data order by logId""")
sleep_short_data = duckdb.sql("""select logId, unnest(levels.shortData,␣
↪recursive := True) from sleep_data order by logId""")
heart_rate_data = duckdb.read_json('./data/unzipped/Takeout/Fitbit/Global␣
↪Export Data/heart_rate-*.json', timestamp_format="%m/%d/%y %H:%M:%S")
heart_rate_df = duckdb.sql('select dateTime, unnest(value) from␣
↪heart_rate_data').df()
hrv_df = duckdb.read_csv('./data/unzipped/Takeout/Fitbit/Heart Rate Variability/
↪Daily Heart Rate Variability Summary*.csv', timestamp_format="%Y-%m-%dT%H:%M:
↪%S").df()
adj_sleep_summary_df = duckdb.sql("""pivot sleep_summary_df on stage
    using
```

```
        first(count) as count,
        first(minutes) as minutes,
        first(thirtyDayAvgMinutes) as thirtyDayAvgMinutes
    order by logId""").df()
complete_sleep_data = sleep_meta_df.merge(adj_sleep_summary_df, on='logId').
 ↪sort_values('dateOfSleep')
final_df = duckdb.sql("""WITH Complete_Sleep AS (
                select *,
                    (dateOfSleep::Date + 1) as hrv_prediction_date,
                    (startTime - Interval '2 Hours') as␣
 ↪startTime_sleep_less_2_hrs,
                    ((lead(startTime) over (order by logId)) - Interval '2␣
 ↪Hours') as tomorrow_startTime_sleep_less_2_hrs
                from complete_sleep_data),
            HR_with_sleep AS (
                Select hr.*, cs.* from Complete_Sleep cs, heart_rate_df hr
                where hr.dateTime >= cs.startTime_sleep_less_2_hrs
                and hr.dateTime < tomorrow_startTime_sleep_less_2_hrs),
            hr_sleep_agg AS (
                Select dateOfSleep::date as dateOfSleep, hrv_prediction_date,
                max(bpm) as bpm_max, min(bpm) as bpm_min, round(avg(bpm), 2) as␣
 ↪bpm_avg, round(stddev_samp(bpm), 2) as bpm_std,
                round(median(bpm),2) as bpm_median, round(quantile_cont(bpm, 0.
 ↪10),2) as bpm_10th, round(quantile_cont(bpm, 0.90),2) as bpm_90th,
                count(bpm) as bpm_count, minutesAsleep,
                minutesAwake,
                timeInBed,
                deep_count,
                light_count,
                rem_count,
                wake_count,
                deep_minutes,
                light_minutes,
                rem_minutes,
                wake_minutes,
                deep_thirtyDayAvgMinutes,
                light_thirtyDayAvgMinutes,
                rem_thirtyDayAvgMinutes,
                wake_thirtyDayAvgMinutes
            from HR_with_sleep
            Group by dateOfSleep::date, hrv_prediction_date, minutesAsleep,␣
 ↪minutesAwake,
                timeInBed,
                deep_count,
                light_count,
                rem_count,
```

```
                wake_count,
                deep_minutes,
                light_minutes,
                rem_minutes,
                wake_minutes,
                deep_thirtyDayAvgMinutes,
                light_thirtyDayAvgMinutes,
                rem_thirtyDayAvgMinutes,
                wake_thirtyDayAvgMinutes
            )
        select agg.*, lag(rmssd) over (order by dateOfSleep) as prev_hrv, hrv.
 ↪rmssd as target_hrv from hr_sleep_agg agg inner join hrv_df hrv on agg.
 ↪hrv_prediction_date = hrv.timestamp Order by dateOfSleep
           """).df()

final_filt_df = final_df[['dateOfSleep', 'bpm_max', 'bpm_min', 'bpm_avg',
       'bpm_median', 'bpm_10th', 'bpm_90th', 'bpm_count',
       'minutesAsleep', 'deep_count',
       'light_count', 'rem_count', 'wake_count', 'deep_minutes',
       'light_minutes', 'rem_minutes', 'wake_minutes',
       'deep_thirtyDayAvgMinutes', 'light_thirtyDayAvgMinutes',
       'rem_thirtyDayAvgMinutes', 'wake_thirtyDayAvgMinutes', 'prev_hrv',␣
 ↪'target_hrv']]
```