project_main

December 12, 2024

0.1 2. Data loading and preprocessing

0.1.1 2.1 Let's first import all the necessary tools

[1]:
```
import pandas as pd
from sklearn.neural_network import MLPRegressor
import seaborn as sns
import numpy as np
import warnings
import missingno as msno
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler,␣
 ↪PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.neighbors import KNeighborsRegressor
%matplotlib inline
warnings.filterwarnings('ignore')
```

Read the files There is two datasets to analyse

[2]:
```
df = pd.read_csv('sleepdata.csv', delimiter= ';')
df2 = pd.read_csv('sleepdata_2.csv', delimiter= ';')
```

[3]:
```
df.head()
```

[3]:

	Start	End	Sleep quality	Time in bed	Wake up	Sleep Notes	Heart rate	Activity (steps)
0	2014-12-29 22:57:49	2014-12-30 07:30:13	100%	8:32	:)			
1	2014-12-30 21:17:50	2014-12-30 21:33:54	3%	0:16	:|			
2	2014-12-30 22:42:49	2014-12-31 07:13:31	98%	8:30	:|			
3	2014-12-31 22:31:01	2015-01-01 06:03:01	65%	7:32	NaN			
4	2015-01-01 22:12:10	2015-01-02 04:56:35	72%	6:44	:)			

```
0                     NaN        59.0                 0
1           Stressful day        72.0                 0
2                     NaN        57.0                 0
3                     NaN         NaN                 0
4  Drank coffee:Drank tea        68.0                 0
```

The describet dataset “sleepdata”. Gathered between 2014-2018

[4]:
```
df2.head()
```

[4]:
```
                 Start                   End Sleep Quality Regularity  Mood  \
0  2019-05-12 23:26:13  2019-05-13 06:11:03           60%          0%   NaN
1  2019-05-13 22:10:31  2019-05-14 06:10:42           73%          0%   NaN
2  2019-05-14 21:43:00  2019-05-15 06:10:41           86%         96%   NaN
3  2019-05-15 23:11:51  2019-05-16 06:13:59           77%         92%   NaN
4  2019-05-16 23:12:13  2019-05-17 06:20:32           78%         94%   NaN

   Heart rate (bpm)  Steps Alarm mode  Air Pressure (Pa) City  …  \
0                 0   8350     Normal                NaN  NaN  …
1                 0   4746     Normal                NaN  NaN  …
2                 0   4007     Normal                NaN  NaN  …
3                 0   6578     Normal                NaN  NaN  …
4                 0   4913     Normal                NaN  NaN  …

   Time in bed (seconds)  Time asleep (seconds)  Time before sleep (seconds)  \
0                24289.2                22993.8                        161.9
1                28810.2                25160.9                        192.1
2                30461.5                28430.8                        203.1
3                25327.6                23132.5                        168.9
4                25698.4                22614.6                        171.3

         Window start          Window stop Did snore  Snore time  \
0  2019-05-13 06:00:00  2019-05-13 06:00:00      True        92.0
1  2019-05-14 05:50:00  2019-05-14 05:50:00      True         0.0
2  2019-05-15 05:50:00  2019-05-15 05:50:00      True        74.0
3  2019-05-16 05:50:00  2019-05-16 05:50:00      True         0.0
4  2019-05-17 05:50:00  2019-05-17 05:50:00      True       188.0

   Weather temperature (°C)  Weather type Notes
0                       0.0    No weather   NaN
1                       0.0    No weather   NaN
2                       0.0    No weather   NaN
3                       0.0    No weather   NaN
4                       0.0    No weather   NaN

[5 rows x 21 columns]
```

The second dataset “sleepdata_2”. Gathered between 2018-2022.

It can be seen, that the datasets differ vastly. Although both contain similar variables, dataset_2 has noticeably more variables, which cannot be seen in sleepdata.

0.1.2 2.2 Data cleaning

2.2.1 Column names Let's do a makeover to the column names, so they will be easier and more convenient to work with

[5]:
```
# For 'sleepdata'
df.columns = df.columns.str.lower()
df.columns = df.columns.str.replace(' ', '_')

# For 'sleepdata_2'
df2.columns = df2.columns.str.lower()
df2.columns = df2.columns.str.replace(' ', '_')
```

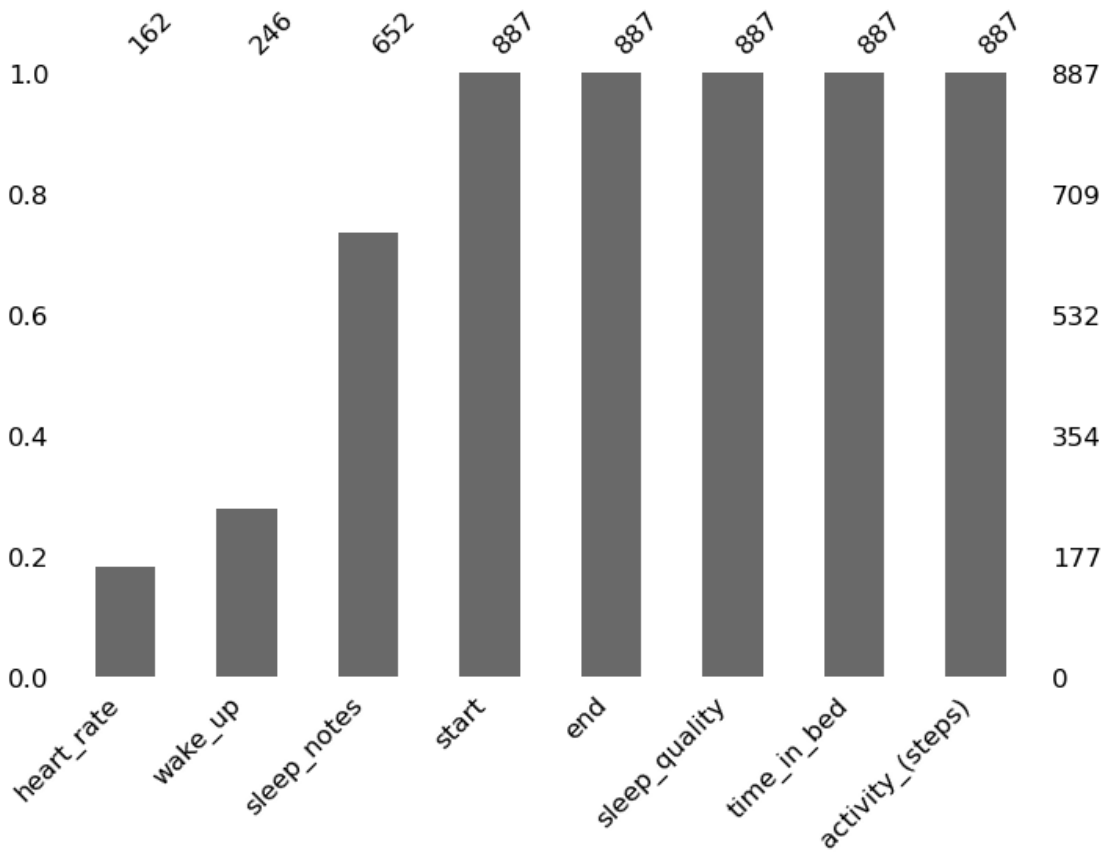
The columns should now be in lower case and whitespaces connected with underscore.

2.2.2 Empty and duplicate values From previous section, it can be seen that the dataset contains empty values. Regardless of that it is a good practice to check them and duplicates.

Missingno package and it's tools are convenient for missing values assessment.

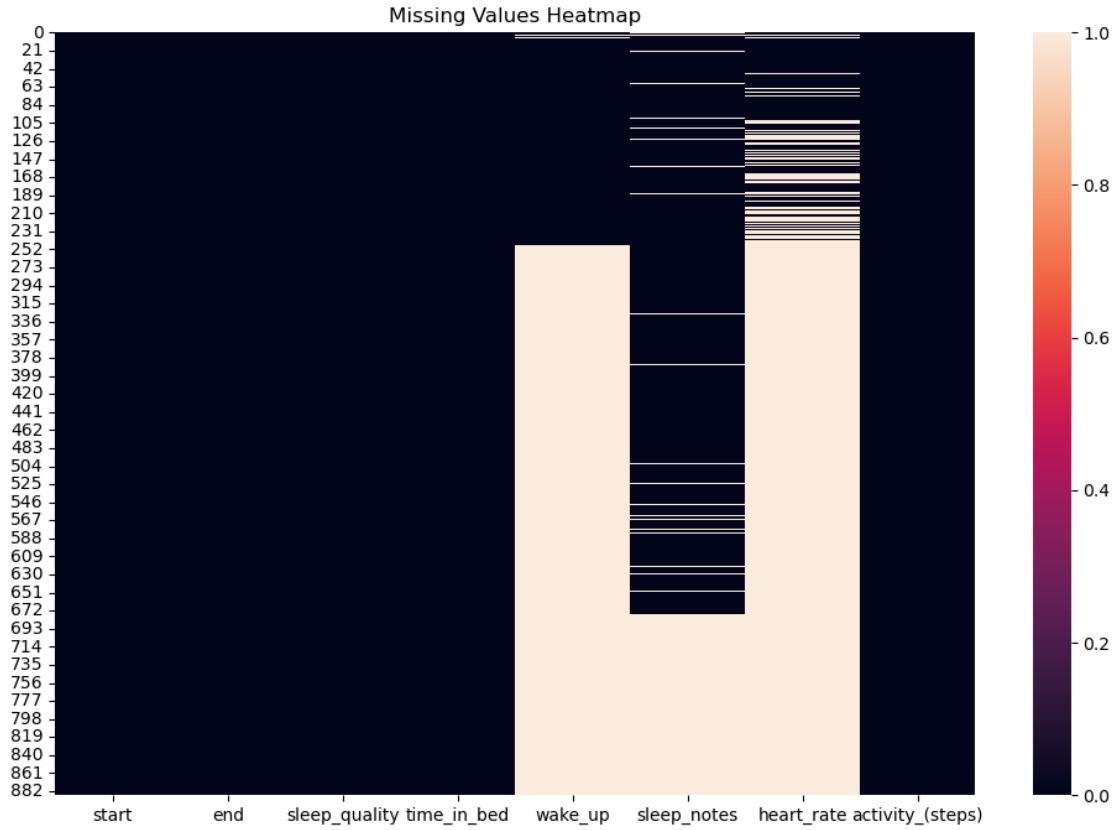
[6]:
```
msno.bar(df, figsize=(8,5), fontsize=12, sort="ascending")
```

[6]:
```
<Axes: >
```



Because of a trouble with missingno's matrix plot, an alternative approach is used here utilizing sns heatmap.

[7]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.isnull())
plt.title("Missing Values Heatmap")
plt.show()
```



It seems that heart rate has a significant amount of empty values, followed by waking up time, and sleep notes. It is quite clear that removing rows containing missing values is not ideal, since removing the rows containing empty value(s) would correspond to a significant data loss (significantly over 5%). Filling missing values in sleep_notes is not an option, since they are personalized notes given by the user. It is also apparent, that these variables did not experince that much missing values at the beginning of the data gathering process, but then as time goes on, less values/notes are registered by the user. Perhaps the motivation for making personal notes decreases with time.

It is preferable to drop these columns containing empty values. Let's save it for later.

Check quickly check for duplicates.

[8]:
```
df.duplicated().sum()
```

[8]: 0

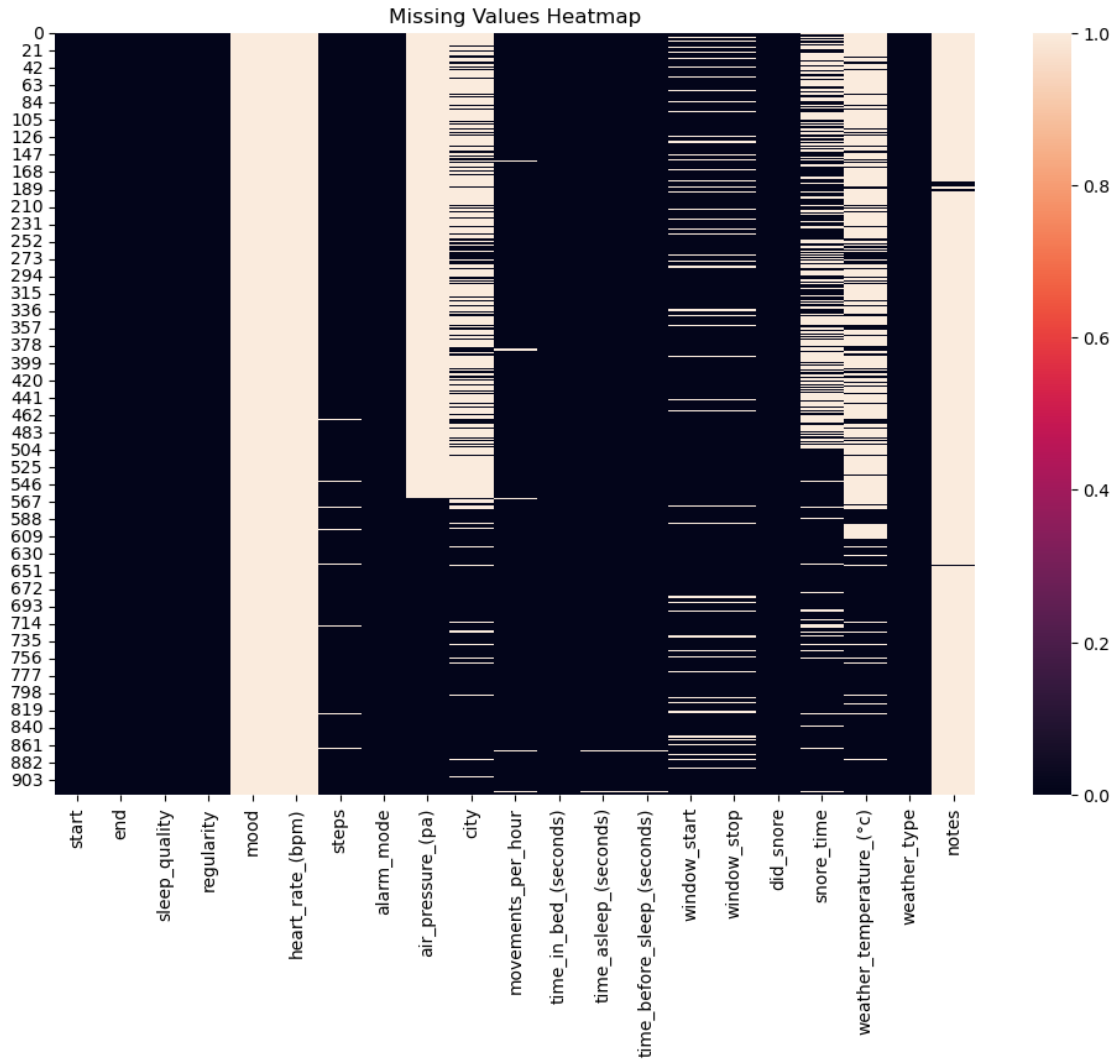
Great! No duplicates. Let's move onto the other dataset.

2.2.3 Matching column names Section 2.1 shows us that 'sleedata_2' contains practically the same variables. However, it can be seen that the names and values differ. Let's first match column names for both datasets. But before that let's do a quick assessment of df2 for future.

Previously we saw few zero values in columns where it does not make sense. For exxample “Heart Rate” and “Weather”. Let’s replace them with empty values.

[9]:
```
df2 = df2.replace({0: np.nan})
```

[10]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df2.isnull())
plt.title("Missing Values Heatmap")
plt.show()
```



It seems that the individual gave up reporting their mood, writing notes and measuring heart rate.

[11]:
```
move = df2['movements_per_hour']
```

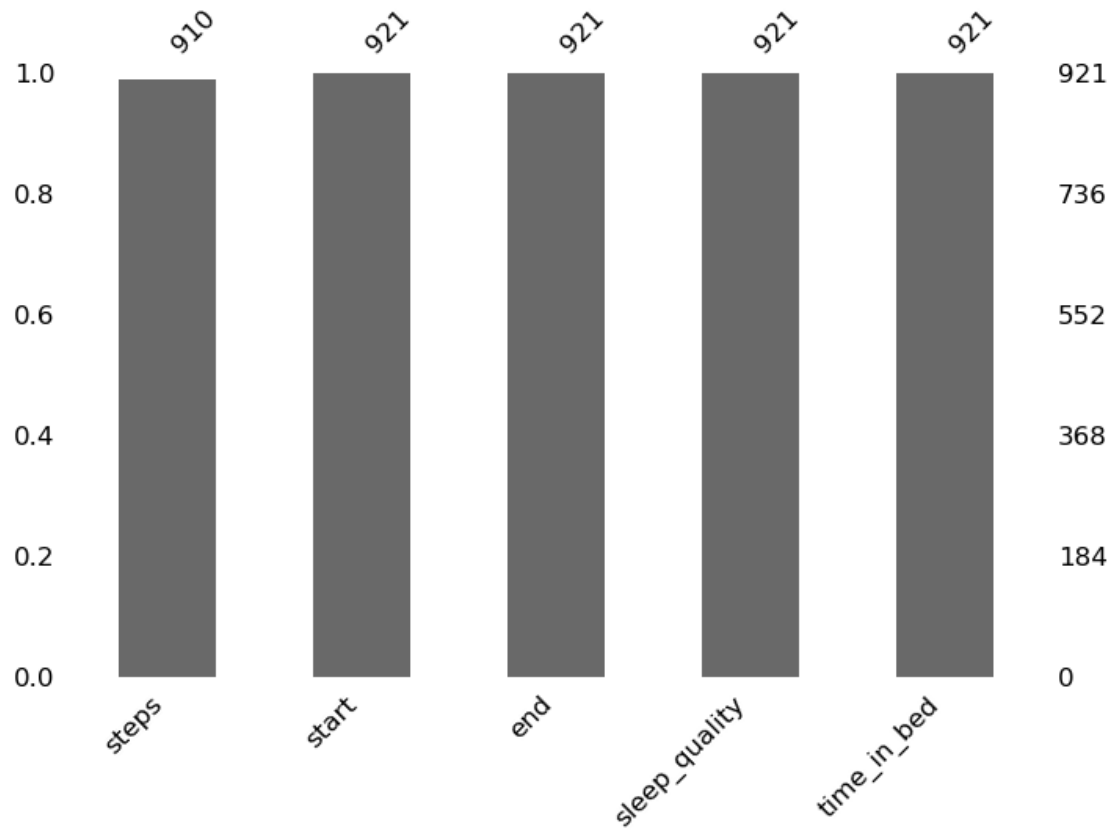
[12]:
```
df2 = df2[['start', 'end', 'sleep_quality','time_in_bed_(seconds)', 'steps']]
df2.columns = df2.columns.str.replace('time_in_bed_(seconds)','time_in_bed')
df2.head()
```

[12]:

	start	end	sleep_quality	time_in_bed	steps
0	2019-05-12 23:26:13	2019-05-13 06:11:03	60%	24289.2	8350.0
1	2019-05-13 22:10:31	2019-05-14 06:10:42	73%	28810.2	4746.0
2	2019-05-14 21:43:00	2019-05-15 06:10:41	86%	30461.5	4007.0
3	2019-05-15 23:11:51	2019-05-16 06:13:59	77%	25327.6	6578.0
4	2019-05-16 23:12:13	2019-05-17 06:20:32	78%	25698.4	4913.0

[13]:
```
msno.bar(df2, figsize=(8,5), fontsize=12, sort="ascending")
print('Quick check for missing values')
```

```
Quick check for missing values
```



[14]:
```
df2.duplicated().sum()
```

[14]:
```
0
```

The datapoints are distinct.

[15]:
```
df.columns = df.columns.str.replace('activity_(steps)','steps')
df.head()
```

[15]:

	start	end	sleep_quality	time_in_bed	wake_up	sleep_notes	heart_rate	steps
0	2014-12-29 22:57:49	2014-12-30 07:30:13	100%	8:32	:)	NaN	59.0	0
1	2014-12-30 21:17:50	2014-12-30 21:33:54	3%	0:16	:|	Stressful day	72.0	0
2	2014-12-30 22:42:49	2014-12-31 07:13:31	98%	8:30	:|	NaN	57.0	0
3	2014-12-31 22:31:01	2015-01-01 06:03:01	65%	7:32	NaN	NaN	NaN	0
4	2015-01-01 22:12:10	2015-01-02 04:56:35	72%	6:44	:)	Drank coffee:Drank tea	68.0	0

0.1.3 2.3 Variable manipulation and data combining

Let's check variable types

[16]:
```
print(f'Data types for each column in df:\n{df.dtypes}\nData types for each column in df2:\n{df2.dtypes}')
```

```
Data types for each column in df:
start             object
end               object
sleep_quality     object
time_in_bed       object
wake_up           object
sleep_notes       object
heart_rate       float64
steps              int64
dtype: object
Data types for each column in df2:
start             object
end               object
sleep_quality     object
time_in_bed      float64
steps            float64
dtype: object
```

It seems that nearly every value is an object type, which is not preferable. **Start** and **end** times of a sleepcycle could be in **datetime** format, **sleep quality** could be converted into an **integer** and **time in the bed** could be in **hours**.

Changing start and end to datetime format

[17]:
```
df['start'] = pd.to_datetime(df['start'])
df['end'] = pd.to_datetime(df['end'])
df2['start'] = pd.to_datetime(df2['start'])
df2['end'] = pd.to_datetime(df2['end'])
```

Changing sleep_quality to integer value

[18]:
```
df['sleep_quality'] = df['sleep_quality'].str.replace('%', '').astype('int')
df2['sleep_quality'] = df2['sleep_quality'].str.replace('%', '').astype('int')
```

Changing time_in_bed to hours float value

[19]:
```
df['time_in_bed'] = df['time_in_bed'].apply(lambda time_str: int(time_str.
  ↪split(':')[0]) + int(time_str.split(':')[1]) / 60)
df2['time_in_bed'] = df2['time_in_bed'].apply(lambda time: time/3600)
```

Let's check the variable types once more

[20]:
```
print(f'Data types for each column in df:\n{df.dtypes}\nData types for each␣
  ↪column in df2:\n{df2.dtypes}')
```

```
Data types for each column in df:
start            datetime64[ns]
end              datetime64[ns]
sleep_quality             int64
time_in_bed             float64
wake_up                  object
sleep_notes              object
heart_rate              float64
steps                     int64
dtype: object
Data types for each column in df2:
start            datetime64[ns]
end              datetime64[ns]
sleep_quality             int64
time_in_bed             float64
steps                   float64
dtype: object
```

Finally let's combine the two datasets to get one big dataset to analyze. However, let's preserve the original datas.

[21]:
```
df_cleaned = df.copy()
df2_cleaned = df2.copy()
df_cleaned = df_cleaned.drop(['wake_up', 'heart_rate', 'sleep_notes'], axis = 1)
sleepdata_big = pd.concat([df_cleaned, df2_cleaned], ignore_index=True)
sleepdata_big.head()
```

[21]:

	start	end	sleep_quality	time_in_bed	steps
0	2014-12-29 22:57:49	2014-12-30 07:30:13	100	8.533333	0.0
1	2014-12-30 21:17:50	2014-12-30 21:33:54	3	0.266667	0.0
2	2014-12-30 22:42:49	2014-12-31 07:13:31	98	8.500000	0.0
3	2014-12-31 22:31:01	2015-01-01 06:03:01	65	7.533333	0.0
4	2015-01-01 22:12:10	2015-01-02 04:56:35	72	6.733333	0.0

[22]:
```
sleepdata_big.shape
```

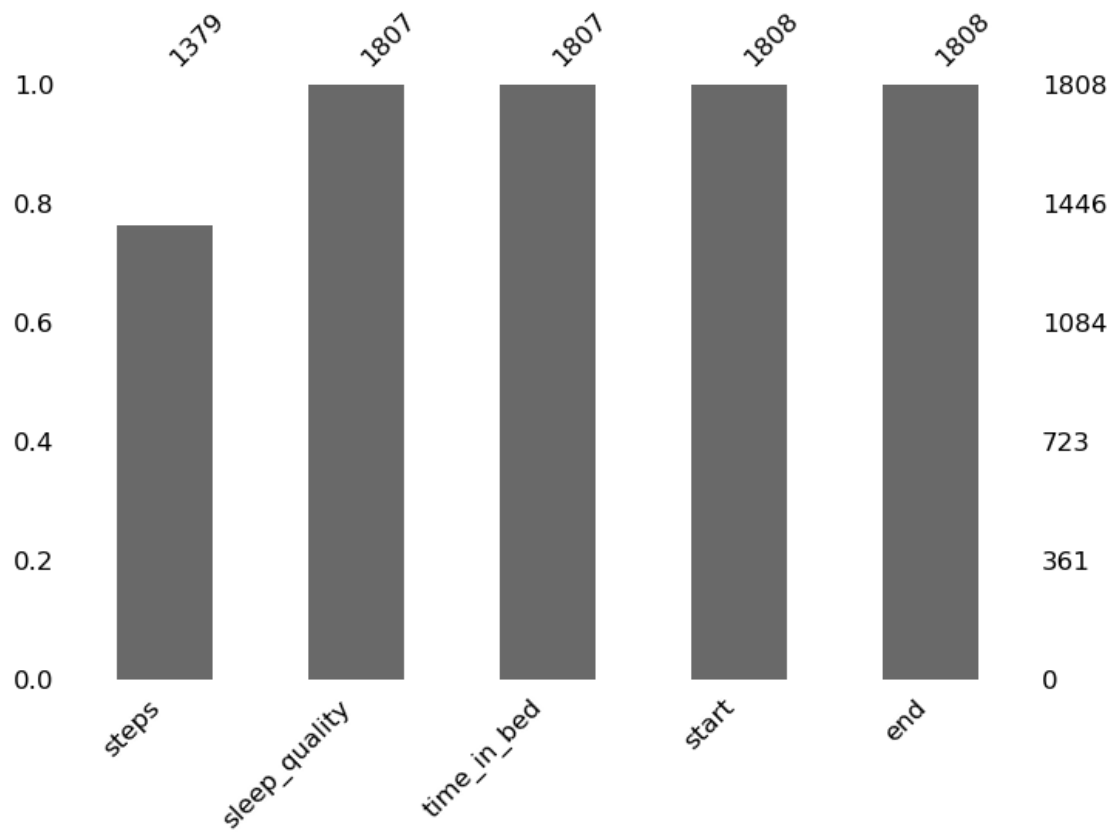
[22]:
```
(1808, 5)
```

Let's do a quick quality check in case of zero values.

[23]:
```
sleepdata_big = sleepdata_big.replace({0: np.nan})
```

[24]:
```
msno.bar(sleepdata_big, figsize=(8,5), fontsize=12, sort="ascending")
print('Quick check for missing values')
```

```
Quick check for missing values
```



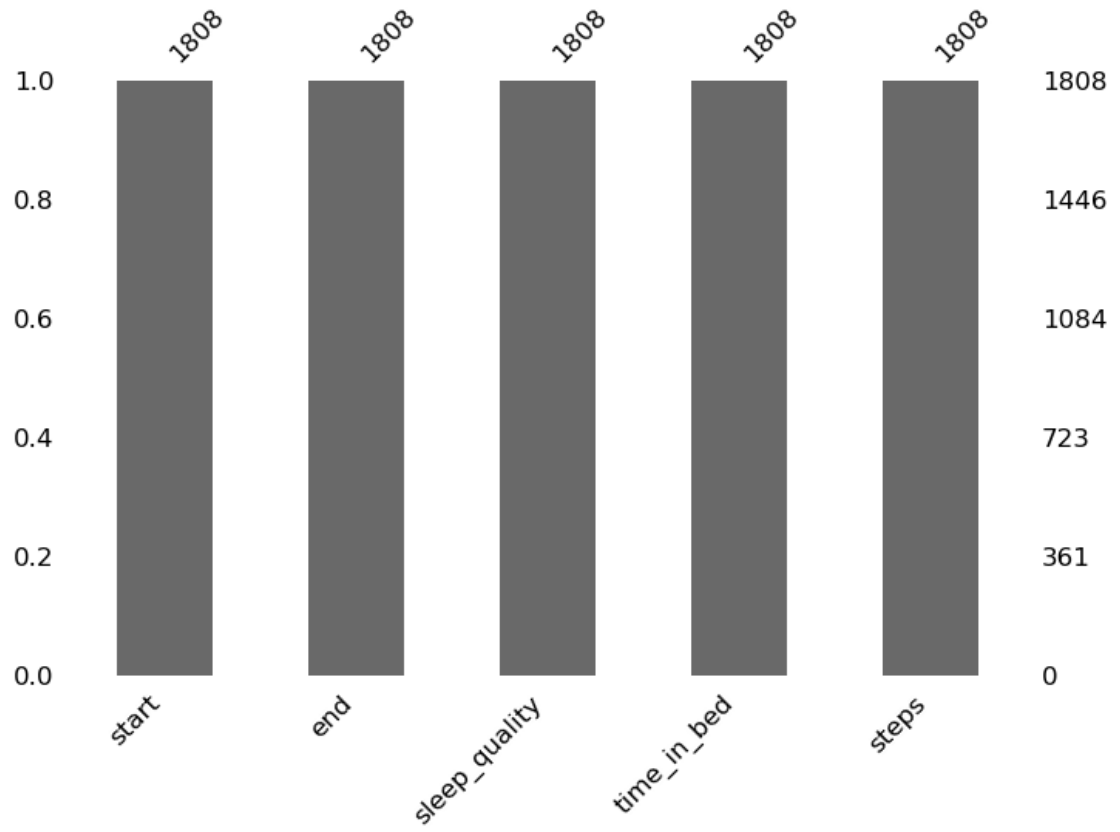
Zero values for steps, sleep quality and time in bed does not make sense. Let's replace them with

column means. Assessing steps with sleep quality seems interesting, so instead of deleting it, let's include it.

[25]:
```
imp_mean = SimpleImputer(missing_values=np.nan, strategy="mean")
sleepdata_big['steps'] = imp_mean.fit_transform(sleepdata_big[['steps']])
sleepdata_big['time_in_bed'] = imp_mean.
 ↪fit_transform(sleepdata_big[['time_in_bed']])
sleepdata_big['sleep_quality'] = imp_mean.
 ↪fit_transform(sleepdata_big[['sleep_quality']])
```

[26]:
```
msno.bar(sleepdata_big, figsize=(8,5), fontsize=12, sort="ascending")
print('Quick check for missing values')
```

```
Quick check for missing values
```



[27]:
```
# One more check for duplicates
if sleepdata_big.duplicated().sum() == 0:
    print("The data is clean and ready for the next step")
else:
    print("Duplicates detected")
```

```
The data is clean and ready for the next step
```

0.2 3. Data visualization

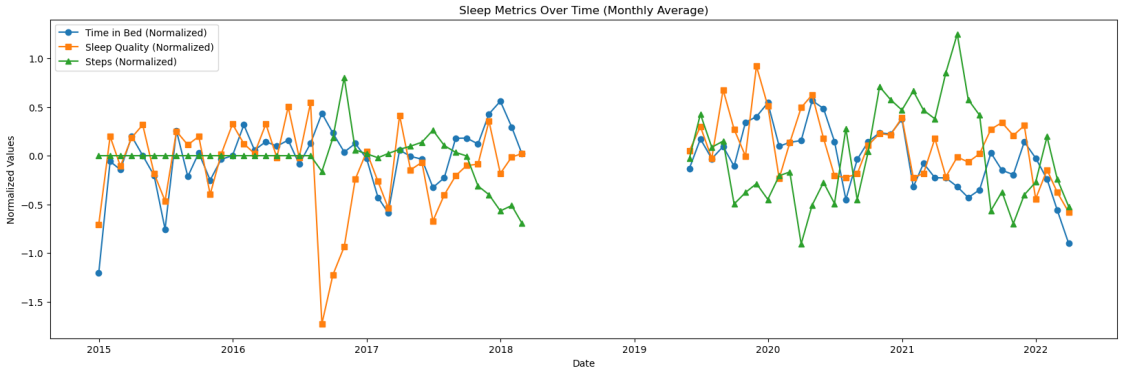
Now that we have a dataset with much more information, we can start to assess patterns and associations in it.

0.2.1 3.1 Time progression

Let's start with analyzing time progression to see if something interesting can be found. Generally, lineplots are ideal for this kinds of cases.

The values in steps-column are significantly larger than the other values (sleep quality, time in bed), so we have to normalize the values.

[28]:
```
sc = StandardScaler()

sleepdata_copy = sleepdata_big.copy()

sleepdata_copy[['time_in_bed_norm', 'sleep_quality_norm', 'steps_norm']] = sc.
 ↪fit_transform(
    sleepdata_copy[['time_in_bed', 'sleep_quality', 'steps']]
)

sleepdata_resampled = sleepdata_copy.resample('M', on='start').mean()

plt.figure(figsize=(20, 6))
plt.plot(sleepdata_resampled.index, sleepdata_resampled['time_in_bed_norm'],␣
 ↪label='Time in Bed (Normalized)', marker='o')
plt.plot(sleepdata_resampled.index, sleepdata_resampled['sleep_quality_norm'],␣
 ↪label='Sleep Quality (Normalized)', marker='s')
plt.plot(sleepdata_resampled.index, sleepdata_resampled['steps_norm'],␣
 ↪label='Steps (Normalized)', marker='^')
plt.xlabel('Date')
plt.ylabel('Normalized Values')
plt.title('Sleep Metrics Over Time (Monthly Average)')
plt.legend()
plt.show()
```



The plot shows a clear trend of sleep quality following the patterns of time in bed, which is not a surprise, since it is quite well enstablished that longer continuous sleep yields a better sleeping quality outcome. The activity (steps) seems to have somewhat negative correlation with sleep quality and/or time in bed. At some points it can be seen, that "excess" activity can lead to a worse sleep quality. This can affect directly (between 2016-2017) and undirectly affecting time in bed, hence sleep quality (between 2021-2022). Perhepas during these times the individual went to a long backpacking trips or something similar, which lead to a excess stress to the body and mind. When activity decreases, it seems that the individual has more time to sleep hence increasing the sleep quality. Other factors may be included, such as personal events in the life, but we cannot conclude anything further. Between 2018 and halfway of 2019 there seems to be a break from sleep data measuring.

0.3 3.2 Sleep conditions and sleep score

Let's assess the sleep conditions and sleep quality. We are interested in the quantity of instances of adequate, inadequate and overslept states.

Additionally according to many sources in internet, sleep quality measured by a sleep track app is generally measured with **Sleep score**. This score categorizes the sleep quality to certain categories.

In the case of our data, "sleep_quality" is the sleep score.

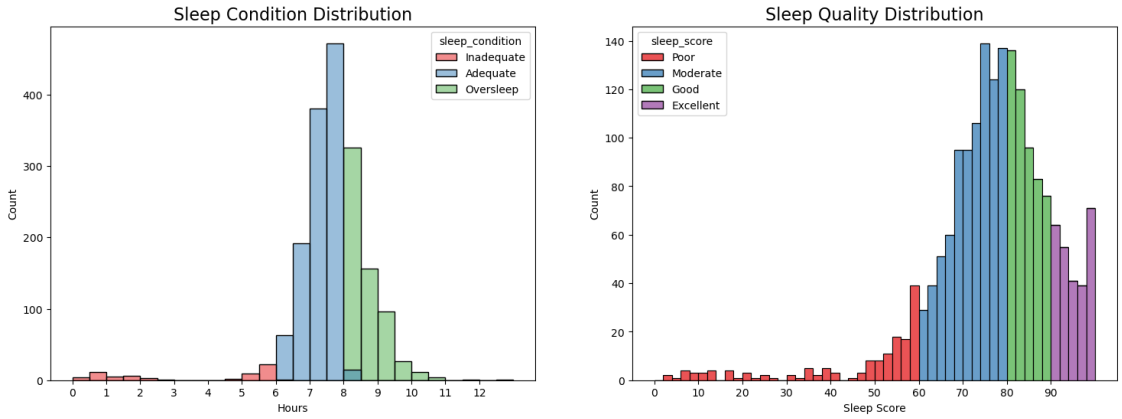
[29]:
```
from tabulate import tabulate
data = [
    ["Oversleep", "Sleep duration > 8 hours"],
    ["Adequate", "6 hours < Sleep duration <= 8 hours"],
    ["Inadequate", "Sleep duration <= 6 hours"],
    [],
    ["Excellent", "Sleep score between 90-100"],
    ["Good", "Sleep score between 80-89"],
    ["Moderate/Fair", "Sleep score between 60-79"],
    ["Poor", "Sleep score < 60"]
]

headers = ["Category", "Criteria"]
```

```
table = tabulate(data, headers=headers, tablefmt="grid")
print(table)
```

```
+---------------+--------------------------------------+
| Category      | Criteria                             |
+===============+======================================+
| Oversleep     | Sleep duration > 8 hours             |
+---------------+--------------------------------------+
| Adequate      | 6 hours < Sleep duration <= 8 hours  |
+---------------+--------------------------------------+
| Inadequate    | Sleep duration <= 6 hours            |
+---------------+--------------------------------------+
|               |                                      |
+---------------+--------------------------------------+
| Excellent     | Sleep score between 90-100           |
+---------------+--------------------------------------+
| Good          | Sleep score between 80-89            |
+---------------+--------------------------------------+
| Moderate/Fair | Sleep score between 60-79            |
+---------------+--------------------------------------+
| Poor          | Sleep score < 60                     |
+---------------+--------------------------------------+
```

[30]:
```
cond_map = pd.cut(x = sleepdata_big['time_in_bed'], bins = [0, 6, 8,␣
 ↪float('inf')], labels = ['Inadequate', 'Adequate', 'Oversleep'])
score_map = pd.cut(x = sleepdata_big['sleep_quality'], bins = [0, 59, 79, 89,␣
 ↪100], labels = ['Poor', 'Moderate', 'Good', 'Excellent'])


sleepdata_big['sleep_condition'] = cond_map
sleepdata_big['sleep_score'] = score_map

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

sns.histplot(ax=ax1,
             data=sleepdata_big,
             x="time_in_bed",
             hue="sleep_condition",
             palette="Set1",
             multiple="layer",
             bins=20,
             binwidth=0.5,
             binrange=(0, 13))
ax1.set_title("Sleep Condition Distribution", fontsize=16)
ax1.set_xlabel("Hours")
ax1.set_xticks(range(0, 13, 1))
```

```
ax1.grid(False)

sns.histplot(ax=ax2,
             data=sleepdata_big,
             x="sleep_quality",
             hue="sleep_score",
             palette="Set1",
             multiple="stack",
             bins=20,
             binwidth=2,
             binrange=(0, 100))
ax2.set_title("Sleep Quality Distribution", fontsize=16)
ax2.set_xlabel("Sleep Score")
ax2.set_xticks(range(0, 100, 10))
ax2.grid(False)

plt.show()
```



The individual experiences adequate sleep most of the time, which carries over to generally good sleep score. In terms of sleep conditions, the peak occurs in roughly 7.5-8h mark. This means that the sleeping time is pretty consistent at a pretty good level. One would think that this would yield better sleeping scores, which do occur as we can see from the second plot. However, the sleep quality is moderate most of the time. Here other factors, such as stress can be a factor.

0.4 3.3 Correlation

Let's assess a simple correlation between the variables using a correlation matrix. However, let's bring few parameters into the consideration.

As we have seen, the plots suggest that the sleeping time has a significant effect on the sleep quality. However, this does not consider the time the individual goes to sleep or wakes up. In the way start and end columns are in the dataset, a clear correlation between values cannot be deemed, since they are in datetime format.

Since we are done with time progression assessment for this dataset, let's change 'start' and 'end' to the hour when the individual went to sleep and woke up.

[31]:
```
hours = sleepdata_big['start'].dt.hour.values
minutes = sleepdata_big['start'].dt.minute.values
seconds = sleepdata_big['start'].dt.second.values

start_time_hour = hours + minutes/60 + seconds/3600

hours = sleepdata_big['end'].dt.hour.values
minutes = sleepdata_big['end'].dt.minute.values
seconds = sleepdata_big['end'].dt.second.values

end_time_hour = hours + minutes/60 + seconds / 3600

sleepdata_big['start_time_hour'] = start_time_hour
sleepdata_big['end_time_hour'] = end_time_hour

sleepdata_big = sleepdata_big.drop(['start', 'end'], axis = 1)
```

[32]:
```
sleepdata_big
```

[32]:

	sleep_quality	time_in_bed	steps	sleep_condition	sleep_score	\
0	100.0	8.533333	5333.307469	Oversleep	Excellent	
1	3.0	0.266667	5333.307469	Inadequate	Poor	
2	98.0	8.500000	5333.307469	Oversleep	Excellent	
3	65.0	7.533333	5333.307469	Adequate	Moderate	
4	72.0	6.733333	5333.307469	Adequate	Moderate	
…	…	…	…	…	…	
1803	71.0	6.798361	3903.000000	Adequate	Moderate	
1804	9.0	0.704500	495.000000	Inadequate	Poor	
1805	49.0	8.943917	13388.000000	Oversleep	Poor	
1806	77.0	5.953778	456.000000	Inadequate	Moderate	
1807	68.0	6.451639	5156.000000	Adequate	Moderate	

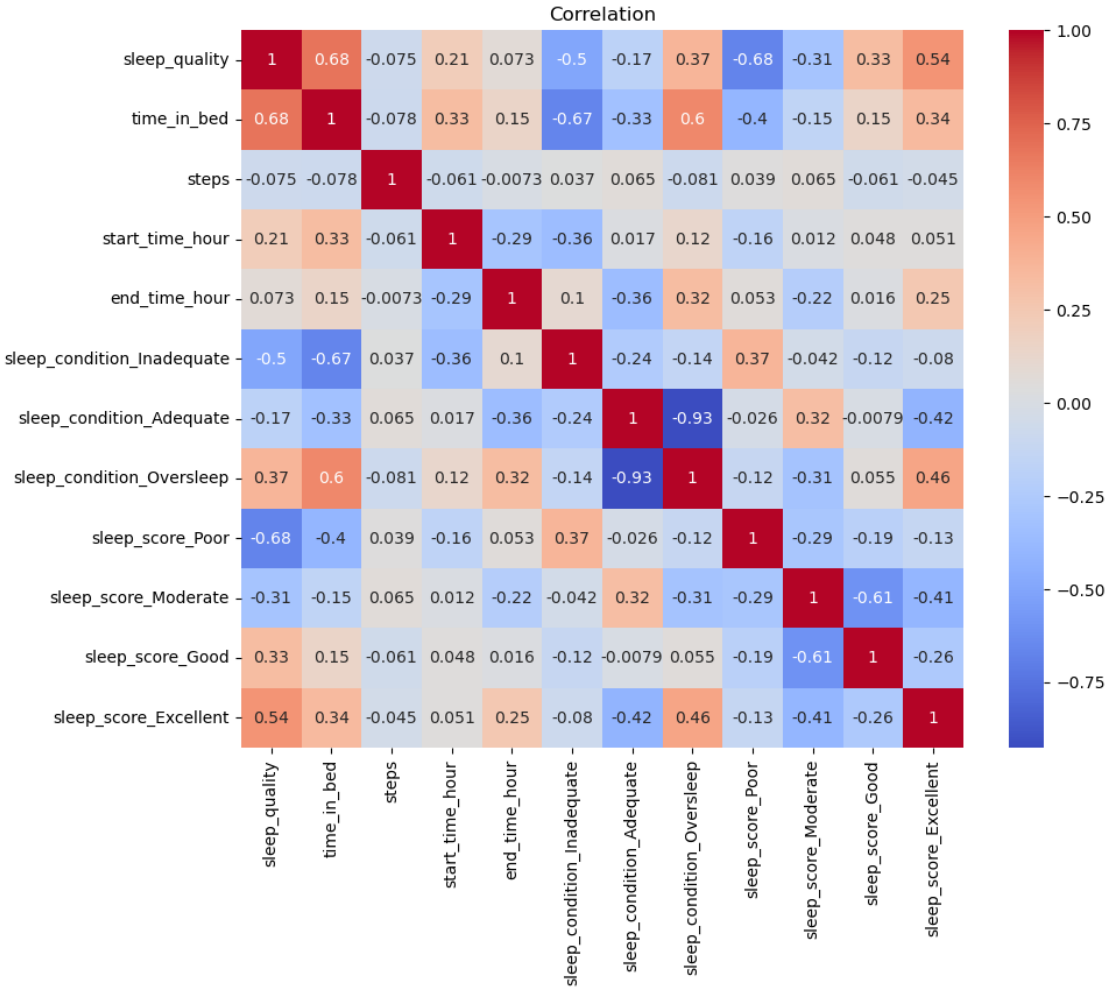
	start_time_hour	end_time_hour
0	22.963611	7.503611
1	21.297222	21.565000
2	22.713611	7.225278
3	22.516944	6.050278
4	22.202778	4.943056
…	…	…
1803	21.562778	4.361111
1804	16.801389	17.506111
1805	21.239722	6.183611
1806	22.889722	4.843333
1807	22.735833	5.187500

```
[1808 rows x 7 columns]
```

[33]:
```
sleepdata_big = pd.get_dummies(sleepdata_big, columns=['sleep_condition',␣
↪'sleep_score'])
```

[34]:
```
fig = plt.figure(figsize = (10,8))
sleep_corr = sns.heatmap(sleepdata_big.corr(),cmap='coolwarm', annot = True)
sleep_corr.set_title('Correlation')
```

[34]:
```
Text(0.5, 1.0, 'Correlation')
```



The correlation matrix suggests that time in bed is clearly the most contributing factor to the sleep quality. It furthermore confirms that activity (steps) does not seem to contribute to any of the variables regardless of strenuous physical activity and fatigue being one of the contributing factors of sleep quality according to NIH. What is surprising is the weak to moderate correlation between good and excellent sleep score, and sleep quality. This means that there must be some

other contributing factors outside of these. Another interesting observation is that the individual seems to have better sleep occasionally, when sleeping over 8 hours.

0.5 3.4 Behavioral analysis

What can be seen in Section 2 is that the datasets contain few variables that are the individual's own input. This includes "mood" and "personal notes". Among all user inputs, these seem the most interesting.

From Section 2.2 it can be seen, that df2 contains hardly any values of mood and notes. Therefore, for this part we will utilize dataframe "df".

0.5.1 3.4.1 Mood

Mood association with the variables can be assessed with pairplot fairly effectively.

[35]:
```
df.dropna(subset=['wake_up'], inplace=True)
df.head()
```

[35]:

	start	end	sleep_quality	time_in_bed	wake_up	\
0	2014-12-29 22:57:49	2014-12-30 07:30:13	100	8.533333	:)	
1	2014-12-30 21:17:50	2014-12-30 21:33:54	3	0.266667	:|	
2	2014-12-30 22:42:49	2014-12-31 07:13:31	98	8.500000	:|	
4	2015-01-01 22:12:10	2015-01-02 04:56:35	72	6.733333	:)	
5	2015-01-03 00:34:57	2015-01-03 07:47:23	83	7.200000	:)	

	sleep_notes	heart_rate	steps
0	NaN	59.0	0
1	Stressful day	72.0	0
2	NaN	57.0	0
4	Drank coffee:Drank tea	68.0	0
5	Drank coffee:Drank tea	60.0	0

Let's implement starting and ending times for sleeping to this dataframe as well, since we are no longer interested in time progression and we cannot do much with dates in this part.

[36]:
```
hours = df['start'].dt.hour.values
minutes = df['start'].dt.minute.values
seconds = df['start'].dt.second.values

start_time_hour = hours + minutes/60 + seconds/3600

hours = df['end'].dt.hour.values
minutes = df['end'].dt.minute.values
seconds = df['end'].dt.second.values

end_time_hour = hours + minutes/60 + seconds / 3600

df['start_h'] = start_time_hour
```

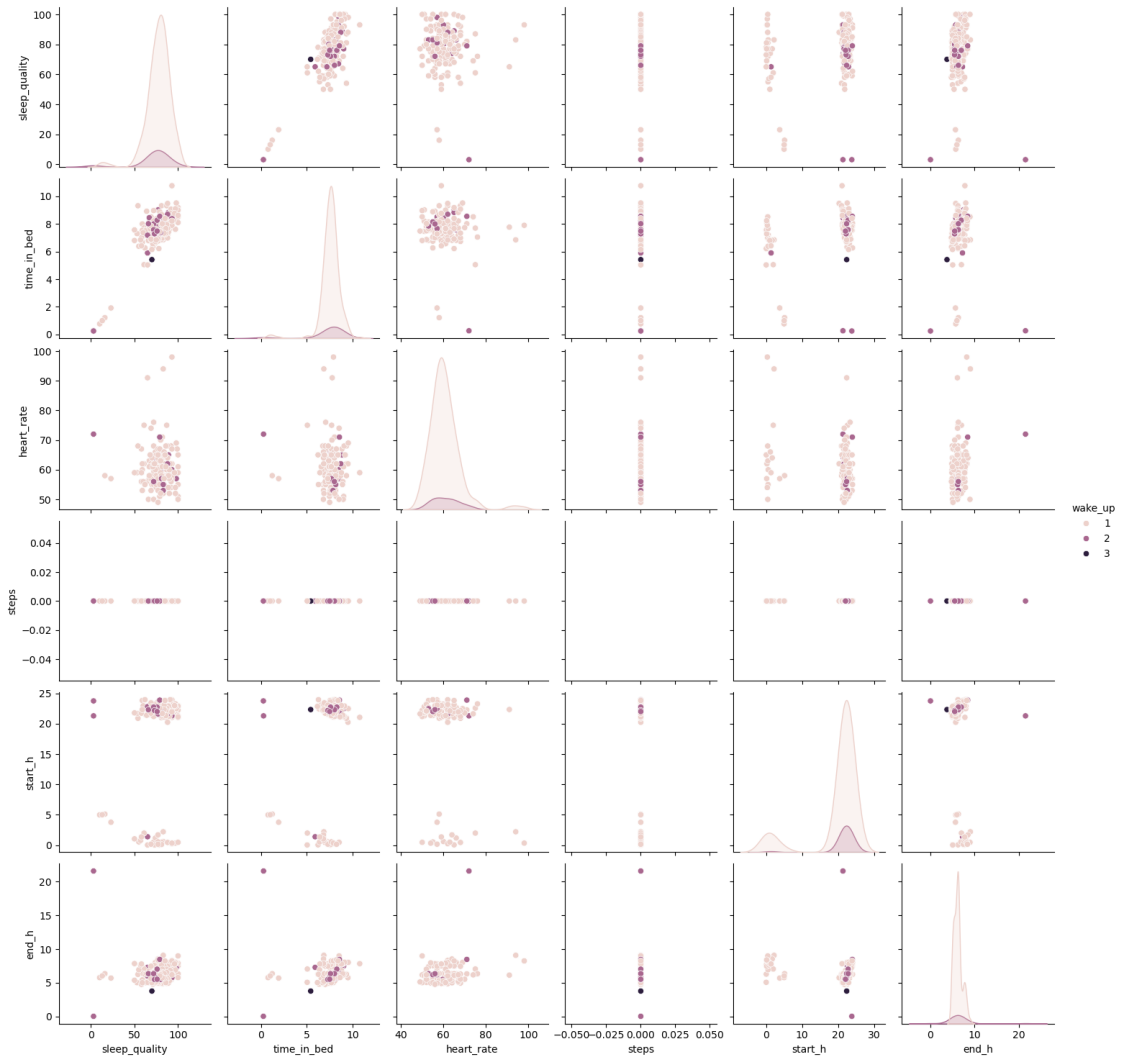
```
df['end_h'] = end_time_hour

df = df.drop(['start', 'end'], axis = 1)
```

Written symbols such as ":)", ":|" and ":(" cannot be used as a hue in pairplot, which means we have to categorize the into values.

[37]:
```
mood_values = {':)': 1, ':|': 2, ':(': 3}
df['wake_up'] = df['wake_up'].map(mood_values)
```

[38]:
```
sns.pairplot(df, hue="wake_up")
```

[38]: <seaborn.axisgrid.PairGrid at 0x7fa0598e79d0>



From the pairplot it can be seen, that the individual is generally in good mood (1). Generally

unhappy instances seem to occur, when little time is spent sleeping. It is well known, that lack of sleep can cause moodiness. Sleep quality is another clear factor affecting to the mood. When sleep quality is close to "Excellent", hardly any other moods than happiness. Interestingly, almost nothing but happiness occurs in situations, where bedtime occurs after midnight. This might be a coincidence or we do not have enough data to assess this.

0.5.2 3.4.2 Consumables and other activity

Let's next analyze, how daily common activities and consumables, such as drinking coffeee or tea, stress and working out affects the sleep quality.

Since these values are written by the user, a language recognition is needed along with categorical values of True or False.

First let's investigate if df has empty values in sleep notes.

[39]:
```
df.isnull().sum(), df.shape
```

[39]:
```
(sleep_quality     0
 time_in_bed       0
 wake_up           0
 sleep_notes       9
 heart_rate       84
 steps             0
 start_h           0
 end_h             0
 dtype: int64,
 (246, 8))
```

Luckily there is very few empty values for sleep notes. Let's remove them.

[40]:
```
df.dropna(subset=['sleep_notes'], inplace=True)
```

[41]:
```
df.isnull().sum(), df.shape
```

[41]:
```
(sleep_quality     0
 time_in_bed       0
 wake_up           0
 sleep_notes       0
 heart_rate       82
 steps             0
 start_h           0
 end_h             0
 dtype: int64,
 (237, 8))
```

[42]:
```
df['sleep_notes'].unique()
```

[42]:
```
array(['Stressful day', 'Drank coffee:Drank tea', 'Ate late:Drank coffee',
       'Drank coffee:Drank tea:Worked out', 'Drank tea:Worked out',
       'Drank coffee:Drank tea:Stressful day', 'Drank tea',
       'Drank coffee', 'Drank coffee:Drank tea:Stressful day:Worked out',
       'Drank coffee:Worked out', 'Ate late:Drank coffee:Drank tea',
       'Ate late:Drank coffee:Drank tea:Worked out',
       'Drank tea:Stressful day', 'Drank tea:Stressful day:Worked out',
       'Drank coffee:Stressful day:Worked out',
       'Drank coffee:Stressful day',
       'Ate late:Drank coffee:Drank tea:Stressful day'], dtype=object)
```

It seems that there are 5 distinct activities: Drank coffee, Drank tea, Ate late, Worked out, Stressful day.

Let's make 5 distinct columns categorizing if the individual did the activity (True), or did not (False). Furthermore, let's generalize coffee and tea consumption into caffeine consuption.

[43]:
```
df['drank_coffee'] = df['sleep_notes'].str.contains('coffee', case=False)
df['drank_tea'] = df['sleep_notes'].str.contains('tea', case=False)
df['stressful_day'] = df['sleep_notes'].str.contains('Stressful', case=False)
df['ate_late'] = df['sleep_notes'].str.contains('Ate late', case=False)
df['worked_out'] = df['sleep_notes'].str.contains('Worked out', case=False)
df['consumed_caffeine'] = df['sleep_notes'].str.contains('coffee|tea',␣
 ↪case=False)
df = df.drop(['sleep_notes', 'steps'], axis = 1)
```

[44]:
```
df.head()
```

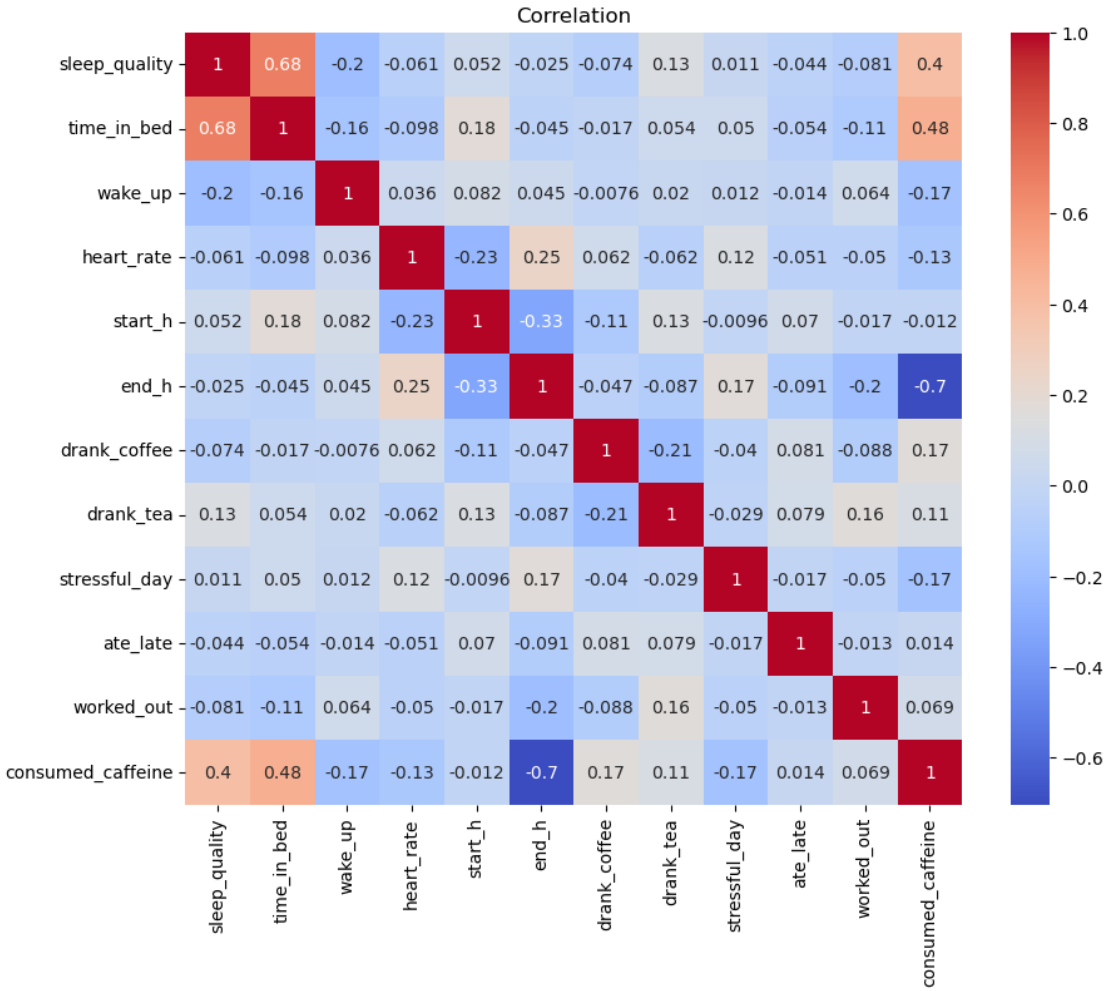
[44]:

	sleep_quality	time_in_bed	wake_up	heart_rate	start_h	end_h	drank_coffee	drank_tea	stressful_day	ate_late	worked_out	consumed_caffeine
1	3	0.266667	2	72.0	21.297222	21.565000	False	False	True	False	False	False
4	72	6.733333	1	68.0	22.202778	4.943056	True	True	False	False	False	True
5	83	7.200000	1	60.0	0.582500	7.789722	True	True	False	False	False	True
7	78	7.300000	1	57.0	21.578889	4.892778	True	False	False	True	False	True
8	69	7.450000	1	56.0	21.540278	5.000833	True	True	False	False	True	True

Let's see if any associations is found.

[45]:
```
fig = plt.figure(figsize = (10,8))
sleep_corr = sns.heatmap(df.corr(),cmap='coolwarm', annot = True)
sleep_corr.set_title('Correlation')
```

[45]:
```
Text(0.5, 1.0, 'Correlation')
```



As we can see from the matrix above, the new categorial variables have little to moderate effect on sleep quality. Out of all these variables, caffeine consumption (coffee | tea) yields the best correaltion with sleep quality. It is a moderate correlation, so it can be deemed that consuming caffeine sometimes increases sleep quality and even time in bed, which is interesting, since on would think that caffeine keeps a person awake longer due to the decresed melatonin. Furthermore, caffeine consumption seems to have a negative correlation with sleep ending time. This means that with increased caffeine consumption, the person tends to wake up earlier.

0.6 Predicting sleep quality with ML methods

The chosen dataset for this part will be "sleepdata_big", since it contains most amount of data and sufficient features.

Let's change the categorical values of True and False to numerical values and create training and test sets.

[46]:
```
categ_columns = ['sleep_condition_Inadequate', 'sleep_condition_Adequate',␣
 ↪'sleep_condition_Oversleep', 'sleep_score_Poor', 'sleep_score_Moderate',␣
 ↪'sleep_score_Good', 'sleep_score_Excellent']
sleepdata_big[categ_columns] = sleepdata_big[categ_columns].astype(int)

X = sleepdata_big.drop(['sleep_quality'], axis=1).values
y = sleepdata_big['sleep_quality'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

0.6.1 Linear regressions

[47]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)

test_accuracy = lr.score(X_test, y_test)
print('Test accuracy (R²):', test_accuracy)

y_pred = lr.predict(X_test)
errors = mean_squared_error(y_test, y_pred, squared=False)
print('Root Mean Squared Error (RMSE):', errors)

scores = cross_val_score(lr, X_train, y_train,␣
 ↪scoring='neg_mean_squared_error', cv=5)
rmse_scores = (-scores) ** 0.5
print('Cross-validated RMSE:', rmse_scores.mean())
```

```
Test accuracy (R²): 0.8785508092941294
Root Mean Squared Error (RMSE): 4.990675615217166
Cross-validated RMSE: 4.9322386831263385
```

0.6.2 Polynomial regression

[48]:
```
degree = 2
pr = make_pipeline(PolynomialFeatures(degree), LinearRegression())
pr.fit(X_train, y_train)

test_accuracy = pr.score(X_test, y_test)
print('Test accuracy (R²):', test_accuracy)

y_pred = pr.predict(X_test)
rmse = mean_squared_error(y_test, y_pred, squared=False)
print('Root Mean Squared Error (RMSE):', rmse)

scores = cross_val_score(pr, X_train, y_train,␣
 ↪scoring='neg_mean_squared_error', cv=5)
rmse_scores = (-scores) ** 0.5
print('Cross-validated RMSE:', rmse_scores.mean())
```

```
Test accuracy (R²): 0.883079109420797
Root Mean Squared Error (RMSE): 4.896751749945378
Cross-validated RMSE: 4.805000727708789
```

0.6.3 K-Nearest Neighbors Regressor

[49]:
```
knn = KNeighborsRegressor(n_neighbors=7)
knn.fit(X_train, y_train)

test_accuracy = knn.score(X_test, y_test)
print('Test accuracy (R²):', test_accuracy)

y_pred = knn.predict(X_test)
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
print('Root Mean Squared Error (RMSE):', rmse)
print('Mean Absolute Error (MAE):', mae)
```

```
Test accuracy (R²): 0.884415081093132
Root Mean Squared Error (RMSE): 4.868695529224414
Mean Absolute Error (MAE): 3.779512629347679
```

0.6.4 Support Vector Regression (SVR)

[60]:
```
svr = SVR(kernel='rbf', C=100, epsilon=0.1)
svr.fit(X_train, y_train)

test_accuracy = svr.score(X_test, y_test)
print('Test accuracy (R²):', test_accuracy)
```

```
y_pred = svr.predict(X_test)
rmse = mean_squared_error(y_test, y_pred, squared=False)
print('Root Mean Squared Error (RMSE):', rmse)

scores = cross_val_score(svr, X_train, y_train,␣
 ↪scoring='neg_mean_squared_error', cv=5)
rmse_scores = (-scores) ** 0.5
print('Cross-validated RMSE:', rmse_scores.mean())
```

```
Test accuracy (R²): 0.8861005984697703
Root Mean Squared Error (RMSE): 4.833066278771449
Cross-validated RMSE: 5.27890134312659
```

0.6.5 Multilayer Perceptron (MLP) Regression

[61]:
```
mlp = MLPRegressor(hidden_layer_sizes=(100, 50), max_iter=500, random_state=42)
mlp.fit(X_train, y_train)

test_accuracy = mlp.score(X_test, y_test)
print('Test accuracy (R²):', test_accuracy)

y_pred = mlp.predict(X_test)
errors = mean_squared_error(y_test, y_pred, squared=False)
print('Root Mean Squared Error (RMSE):', errors)

scores = cross_val_score(mlp, X_train, y_train,␣
 ↪scoring='neg_mean_squared_error', cv=5)
rmse_scores = (-scores) ** 0.5
print('Cross-validated RMSE:', rmse_scores.mean())
```

```
Test accuracy (R²): 0.9038458050822976
Root Mean Squared Error (RMSE): 4.440645942920936
Cross-validated RMSE: 4.655114226743097
```

[ ]: