Sleep health and Lifestyle datasets (kaggle) - PROJECT

Link to dataset: https://www.kaggle.com/datasets/uom190346a/sleep-health-and-lifestyle-dataset The Sleep Health and Lifestyle dataset consists of 400 rows and 13 columns, covering a wide range of variables related to sleep and daily habits. It includes details such as gender, age, occupation, sleep duration, sleep quality, physical activity level, stress level, BMI category, blood pressure, heart rate, daily steps and the presence or absence of sleep disorders.

Columns: `Person ID`: An identifier for each individual. `Gender`: The gender of the person (Male/Female). `Age`: The age of the person in years. `Occupation`: The occupation or profession of the person. `Sleep Duration (hours)`: The number of hours the person sleeps per day. `Quality of Sleep (scale: 1-10)`: A subjective rating of the quality of sleep, ranging from 1 to 10. `Physical Activity Level (minutes/day)`: The number of minutes the person engages in physical activity daily. `Stress Level (scale: 1-10)`: A subjective rating of the stress level experienced by the person, ranging from 1 to 10. `BMI Category`: The BMI category of the person (e.g., Underweight, Normal, Overweight). `Blood Pressure (systolic/diastolic)`: The blood pressure measurement of the person, indicated as systolic pressure over diastolic pressure. `Heart Rate (bpm)`: The resting heart rate of the person in beats per minute. `Daily Steps`: The number of steps the person takes per day. `Sleep Disorder`: The presence or absence of a sleep disorder in the person (None, Insomnia, Sleep Apnea).

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
data = pd.read_csv('Sleep_health_and_lifestyle_dataset_1.csv')
```

Exploratory Data Analytics

```
data.head(30)
```

```
    Person ID  Gender  Age            Occupation  Sleep Duration  \
0           1    Male   27     Software Engineer             6.1   
1           2    Male   28                Doctor             6.2   
2           3    Male   28                Doctor             6.2   
3           4    Male   28  Sales Representative             5.9   
4           5    Male   28  Sales Representative             5.9   
5           6    Male   28     Software Engineer             5.9   
6           7    Male   29               Teacher             6.3   
7           8    Male   29                Doctor             7.8   
8           9    Male   29                Doctor             7.8   
9          10    Male   29                Doctor             7.8   
10         11    Male   29                Doctor             6.1   
```

```
11          12    Male   29                Doctor             7.8
12          13    Male   29                Doctor             6.1
13          14    Male   29                Doctor             6.0
14          15    Male   29                Doctor             6.0
15          16    Male   29                Doctor             6.0
16          17  Female   29                 Nurse             6.5
17          18    Male   29                Doctor             6.0
18          19  Female   29                 Nurse             6.5
19          20    Male   30                Doctor             7.6
20          21    Male   30                Doctor             7.7
21          22    Male   30                Doctor             7.7
22          23    Male   30                Doctor             7.7
23          24    Male   30                Doctor             7.7
24          25    Male   30                Doctor             7.8
25          26    Male   30                Doctor             7.9
26          27    Male   30                Doctor             7.8
27          28    Male   30                Doctor             7.9
28          29    Male   30                Doctor             7.9
29          30    Male   30                Doctor             7.9

    Quality of Sleep  Physical Activity Level  Stress Level   BMI
Category  \
0                  6                       42             6
Overweight
1                  6                       60             8
Normal
2                  6                       60             8
Normal
3                  4                       30             8
Obese
4                  4                       30             8
Obese
5                  4                       30             8
Obese
6                  6                       40             7
Obese
7                  7                       75             6
Normal
8                  7                       75             6
Normal
9                  7                       75             6
Normal
10                 6                       30             8
Normal
11                 7                       75             6
Normal
12                 6                       30             8
Normal
13                 6                       30             8
Normal
```

14	6	30	8	Normal
15	6	30	8	Normal
16	5	40	7	Normal Weight
17	6	30	8	Normal
18	5	40	7	Normal Weight
19	7	75	6	Normal
20	7	75	6	Normal
21	7	75	6	Normal
22	7	75	6	Normal
23	7	75	6	Normal
24	7	75	6	Normal
25	7	75	6	Normal
26	7	75	6	Normal
27	7	75	6	Normal
28	7	75	6	Normal
29	7	75	6	Normal

	Blood Pressure	Heart Rate	Daily Steps	Sleep Disorder
0	126/83	77	4200	None
1	125/80	75	10000	None
2	125/80	75	10000	None
3	140/90	85	3000	Sleep Apnea
4	140/90	85	3000	Sleep Apnea
5	140/90	85	3000	Insomnia
6	140/90	82	3500	Insomnia
7	120/80	70	8000	None
8	120/80	70	8000	None
9	120/80	70	8000	None
10	120/80	70	8000	None
11	120/80	70	8000	None
12	120/80	70	8000	None
13	120/80	70	8000	None
14	120/80	70	8000	None
15	120/80	70	8000	None

```
16         132/87          80         4000    Sleep Apnea
17         120/80          70         8000    Sleep Apnea
18         132/87          80         4000       Insomnia
19         120/80          70         8000           None
20         120/80          70         8000           None
21         120/80          70         8000           None
22         120/80          70         8000           None
23         120/80          70         8000           None
24         120/80          70         8000           None
25         120/80          70         8000           None
26         120/80          70         8000           None
27         120/80          70         8000           None
28         120/80          70         8000           None
29         120/80          70         8000           None
```

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Person ID                374 non-null    int64  
 1   Gender                   374 non-null    object 
 2   Age                      374 non-null    int64  
 3   Occupation               374 non-null    object 
 4   Sleep Duration           374 non-null    float64
 5   Quality of Sleep         374 non-null    int64  
 6   Physical Activity Level  374 non-null    int64  
 7   Stress Level             374 non-null    int64  
 8   BMI Category             374 non-null    object 
 9   Blood Pressure           374 non-null    object 
 10  Heart Rate               374 non-null    int64  
 11  Daily Steps              374 non-null    int64  
 12  Sleep Disorder           374 non-null    object 
dtypes: float64(1), int64(7), object(5)
memory usage: 38.1+ KB
```

```python
data['Sleep Disorder'].unique()
```

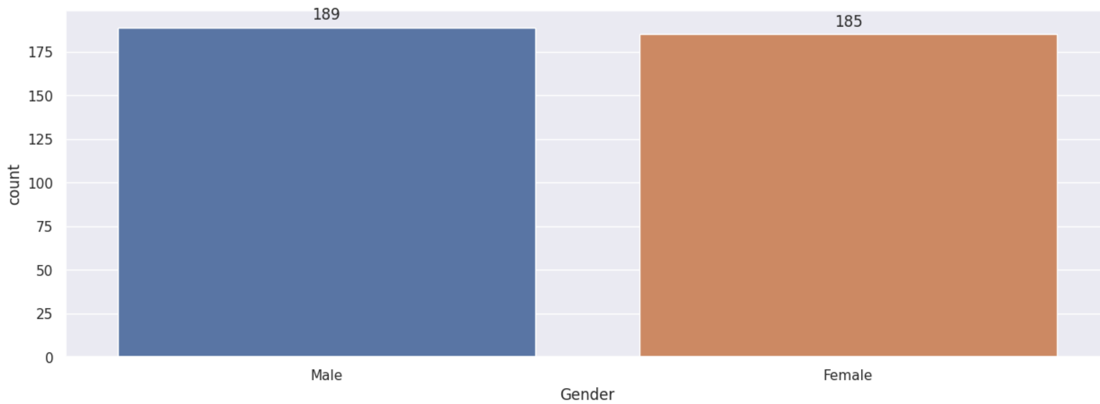
```
array(['None', 'Sleep Apnea', 'Insomnia'], dtype=object)
```

```python
data['Gender'].unique()
```

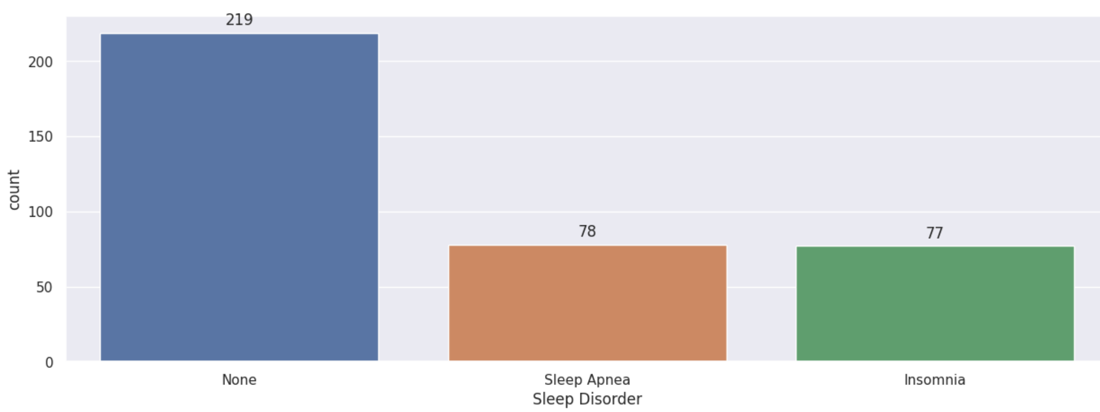
```
array(['Male', 'Female'], dtype=object)
```

```python
ax = sns.countplot(x = 'Gender', data = data)

for p in ax.patches:
    ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + 
p.get_width() / 2., p.get_height()), ha = 'center', va = 'center', 
xytext = (0, 10), textcoords = 'offset points')
```

```
plt.show()
```

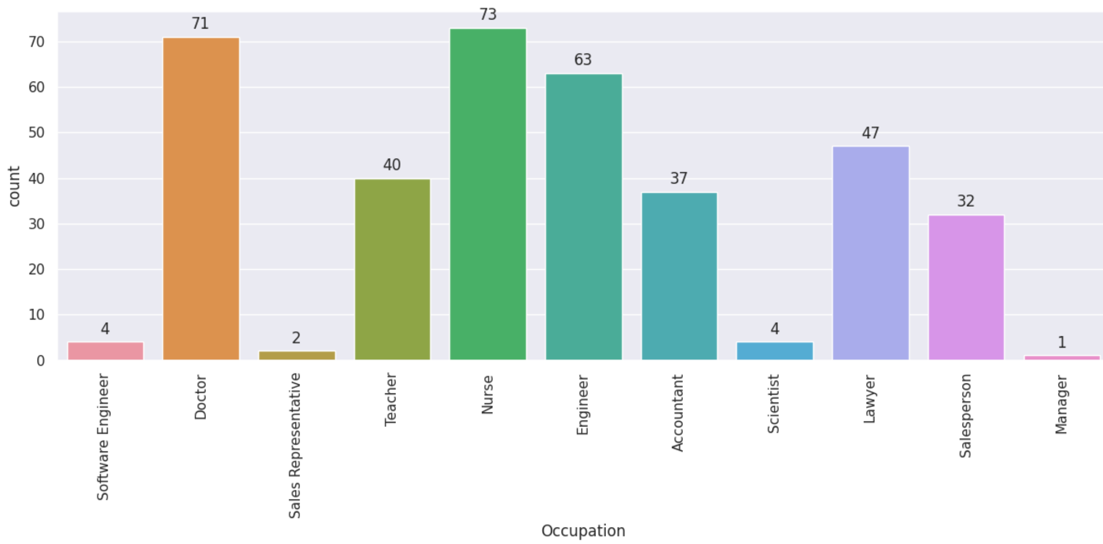



```
ax = sns.countplot(x = 'Sleep Disorder', data = data)

for p in ax.patches:
    ax.annotate(format(p.get_height(), '.0f'), (p.get_x() +
p.get_width() / 2., p.get_height()), ha = 'center', va = 'center',
xytext = (0, 10), textcoords = 'offset points')

plt.show()
```

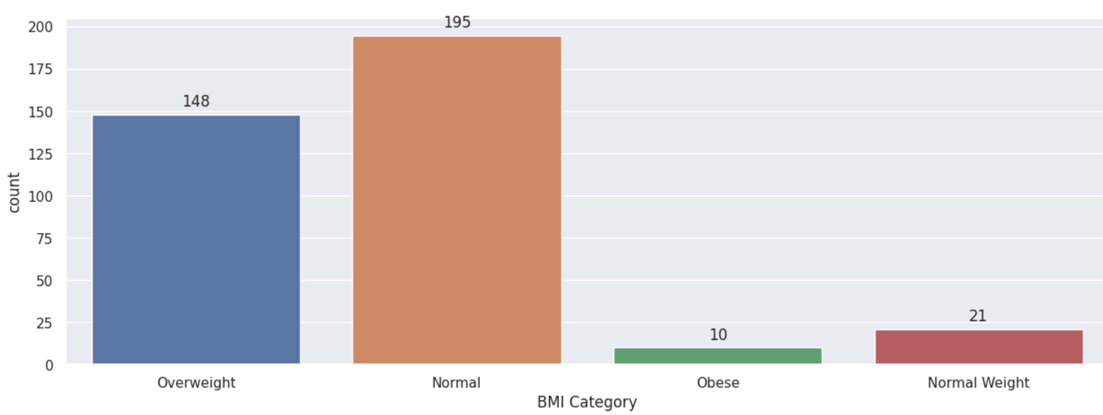



```
# Create a countplot chart
ax = sns.countplot(x='Occupation', data=data)

# Swap the labels on the x-axis
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)

# # Adding annotations
for p in ax.patches:
    ax.annotate(format(p.get_height(), '.0f'), (p.get_x() +
p.get_width() / 2., p.get_height()), ha='center', va='center',
xytext=(0, 10), textcoords='offset points')
```

```
plt.show()
```

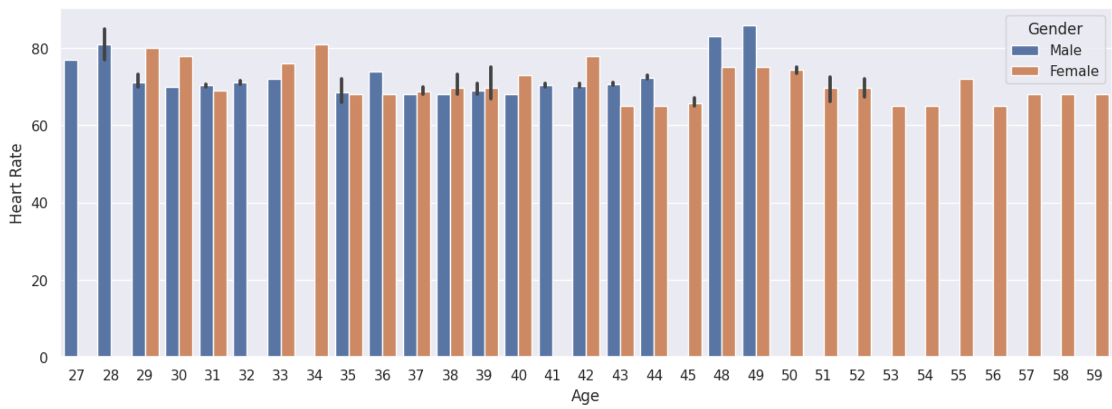



```
ax = sns.countplot(x = 'BMI Category', data = data)

for p in ax.patches:
    ax.annotate(format(p.get_height(), '.0f'), (p.get_x() +
p.get_width() / 2., p.get_height()), ha = 'center', va = 'center',
xytext = (0, 10), textcoords = 'offset points')

plt.show()
```



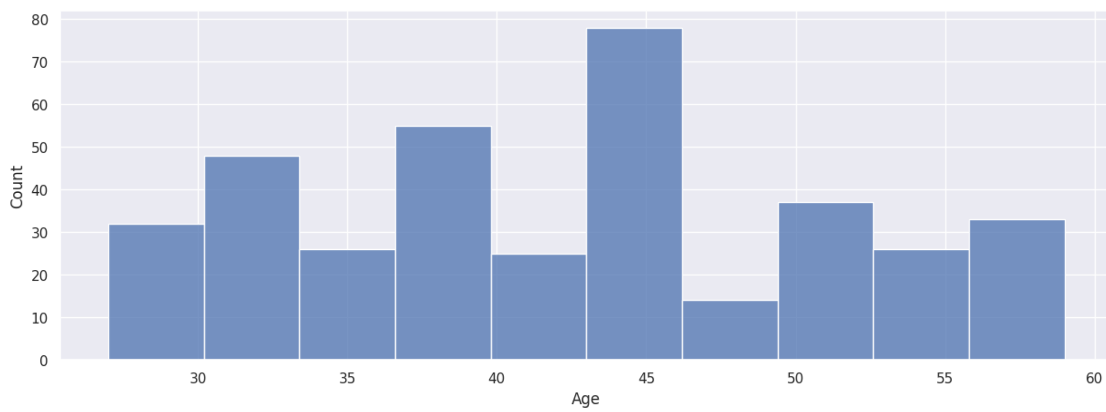

```
sales_state = data.groupby(['Age','Gender'],as_index=False)['Heart
Rate'].sum().sort_values(by='Heart Rate',ascending=False)

sns.set(rc={'figure.figsize':(15,5)})
sns.barplot(data=data,x='Age',y='Heart Rate',hue='Gender')
```

```
<Axes: xlabel='Age', ylabel='Heart Rate'>
```



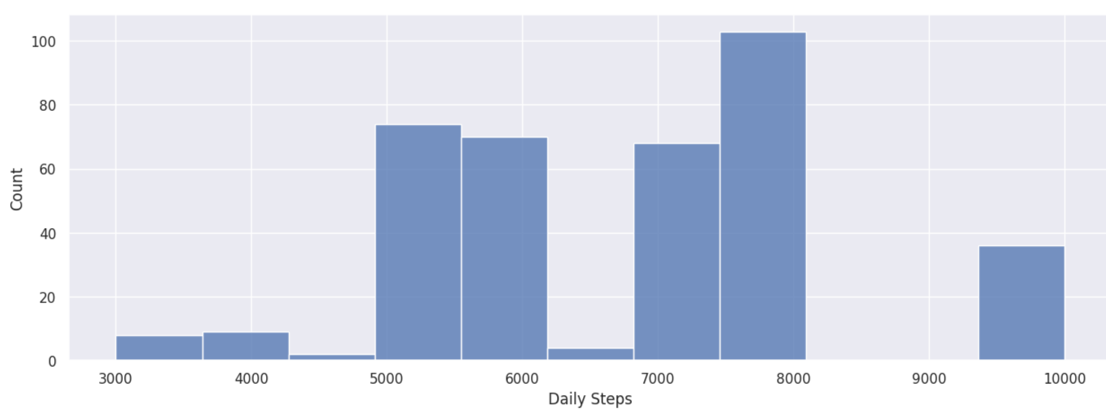

```
sns.histplot(data=data['Age'], kde=False)
```

```
<Axes: xlabel='Age', ylabel='Count'>
```



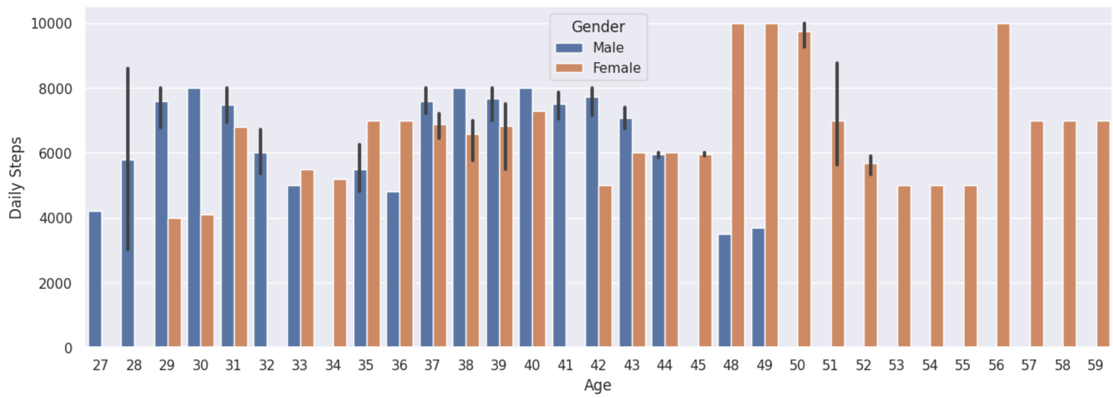

```
sns.histplot(data=data['Daily Steps'], kde=False)
```

```
<Axes: xlabel='Daily Steps', ylabel='Count'>
```




```
sales_state = data.groupby(['Age','Gender'],as_index=False)['Daily
Steps'].sum().sort_values(by='Daily Steps',ascending=False)

sns.set(rc={'figure.figsize':(15,5)})
sns.barplot(data=data,x='Age',y='Daily Steps',hue='Gender')
```

```
<Axes: xlabel='Age', ylabel='Daily Steps'>
```




```
#Mean of Heart Rate
data['Heart Rate'].mean()
```

```
70.16577540106952
```

```
data['Gender'] = [1 if value == "Male" else 0 for value in
data['Gender']]

# 1 = Man, 0 = Woman
data.head(30)
```

```
    Person ID  Gender  Age            Occupation  Sleep Duration  \
0           1       1   27     Software Engineer             6.1
1           2       1   28                Doctor             6.2
2           3       1   28                Doctor             6.2
3           4       1   28  Sales Representative             5.9
4           5       1   28  Sales Representative             5.9
5           6       1   28     Software Engineer             5.9
6           7       1   29               Teacher             6.3
7           8       1   29                Doctor             7.8
8           9       1   29                Doctor             7.8
9          10       1   29                Doctor             7.8
10         11       1   29                Doctor             6.1
11         12       1   29                Doctor             7.8
12         13       1   29                Doctor             6.1
13         14       1   29                Doctor             6.0
14         15       1   29                Doctor             6.0
15         16       1   29                Doctor             6.0
16         17       0   29                 Nurse             6.5
17         18       1   29                Doctor             6.0
18         19       0   29                 Nurse             6.5
19         20       1   30                Doctor             7.6
20         21       1   30                Doctor             7.7
21         22       1   30                Doctor             7.7
22         23       1   30                Doctor             7.7
23         24       1   30                Doctor             7.7
24         25       1   30                Doctor             7.8
```

```
25          26       1   30                 Doctor            7.9
26          27       1   30                 Doctor            7.8
27          28       1   30                 Doctor            7.9
28          29       1   30                 Doctor            7.9
29          30       1   30                 Doctor            7.9

    Quality of Sleep  Physical Activity Level  Stress Level   BMI
Category  \
0                  6                       42             6
Overweight
1                  6                       60             8
Normal
2                  6                       60             8
Normal
3                  4                       30             8
Obese
4                  4                       30             8
Obese
5                  4                       30             8
Obese
6                  6                       40             7
Obese
7                  7                       75             6
Normal
8                  7                       75             6
Normal
9                  7                       75             6
Normal
10                 6                       30             8
Normal
11                 7                       75             6
Normal
12                 6                       30             8
Normal
13                 6                       30             8
Normal
14                 6                       30             8
Normal
15                 6                       30             8
Normal
16                 5                       40             7  Normal
Weight
17                 6                       30             8
Normal
18                 5                       40             7  Normal
Weight
19                 7                       75             6
Normal
20                 7                       75             6
Normal
```

21	7	75	6
Normal			
22	7	75	6
Normal			
23	7	75	6
Normal			
24	7	75	6
Normal			
25	7	75	6
Normal			
26	7	75	6
Normal			
27	7	75	6
Normal			
28	7	75	6
Normal			
29	7	75	6
Normal			

	Blood Pressure	Heart Rate	Daily Steps	Sleep Disorder
0	126/83	77	4200	None
1	125/80	75	10000	None
2	125/80	75	10000	None
3	140/90	85	3000	Sleep Apnea
4	140/90	85	3000	Sleep Apnea
5	140/90	85	3000	Insomnia
6	140/90	82	3500	Insomnia
7	120/80	70	8000	None
8	120/80	70	8000	None
9	120/80	70	8000	None
10	120/80	70	8000	None
11	120/80	70	8000	None
12	120/80	70	8000	None
13	120/80	70	8000	None
14	120/80	70	8000	None
15	120/80	70	8000	None
16	132/87	80	4000	Sleep Apnea
17	120/80	70	8000	Sleep Apnea
18	132/87	80	4000	Insomnia
19	120/80	70	8000	None
20	120/80	70	8000	None
21	120/80	70	8000	None
22	120/80	70	8000	None
23	120/80	70	8000	None
24	120/80	70	8000	None
25	120/80	70	8000	None
26	120/80	70	8000	None
27	120/80	70	8000	None
28	120/80	70	8000	None
29	120/80	70	8000	None

```python
data['Sleep Disorder'] = [2 if value == "Insomnia" else 1 if value ==
'Sleep Apnea' else 0 for value in data['Sleep Disorder']]

data.head(10)
```

```
   Person ID  Gender  Age            Occupation  Sleep Duration  \
0          1       1   27     Software Engineer             6.1
1          2       1   28                Doctor             6.2
2          3       1   28                Doctor             6.2
3          4       1   28  Sales Representative             5.9
4          5       1   28  Sales Representative             5.9
5          6       1   28     Software Engineer             5.9
6          7       1   29               Teacher             6.3
7          8       1   29                Doctor             7.8
8          9       1   29                Doctor             7.8
9         10       1   29                Doctor             7.8

   Quality of Sleep  Physical Activity Level  Stress Level BMI
Category  \
0                 6                       42             6
Overweight
1                 6                       60             8
Normal
2                 6                       60             8
Normal
3                 4                       30             8
Obese
4                 4                       30             8
Obese
5                 4                       30             8
Obese
6                 6                       40             7
Obese
7                 7                       75             6
Normal
8                 7                       75             6
Normal
9                 7                       75             6
Normal

  Blood Pressure  Heart Rate  Daily Steps  Sleep Disorder
0         126/83          77         4200               0
1         125/80          75        10000               0
2         125/80          75        10000               0
3         140/90          85         3000               1
4         140/90          85         3000               1
5         140/90          85         3000               2
6         140/90          82         3500               2
7         120/80          70         8000               0
```

```
8         120/80          70         8000               0
9         120/80          70         8000               0
```

```python
data['Sleep Disorder'].unique()
```

```
array([0, 1, 2])
```

```python
data['Gender'].unique()
```

```
array([1, 0])
```

```python
data['BMI Category'] = data['BMI Category'].replace('Normal Weight', 
'Normal')
```

```python
data.head(30)
```

```
    Person ID  Gender  Age            Occupation  Sleep Duration  \
0           1       1   27     Software Engineer             6.1   
1           2       1   28                Doctor             6.2   
2           3       1   28                Doctor             6.2   
3           4       1   28  Sales Representative             5.9   
4           5       1   28  Sales Representative             5.9   
5           6       1   28     Software Engineer             5.9   
6           7       1   29               Teacher             6.3   
7           8       1   29                Doctor             7.8   
8           9       1   29                Doctor             7.8   
9          10       1   29                Doctor             7.8   
10         11       1   29                Doctor             6.1   
11         12       1   29                Doctor             7.8   
12         13       1   29                Doctor             6.1   
13         14       1   29                Doctor             6.0   
14         15       1   29                Doctor             6.0   
15         16       1   29                Doctor             6.0   
16         17       0   29                 Nurse             6.5   
17         18       1   29                Doctor             6.0   
18         19       0   29                 Nurse             6.5   
19         20       1   30                Doctor             7.6   
20         21       1   30                Doctor             7.7   
21         22       1   30                Doctor             7.7   
22         23       1   30                Doctor             7.7   
23         24       1   30                Doctor             7.7   
24         25       1   30                Doctor             7.8   
25         26       1   30                Doctor             7.9   
26         27       1   30                Doctor             7.8   
27         28       1   30                Doctor             7.9   
28         29       1   30                Doctor             7.9   
29         30       1   30                Doctor             7.9   

    Quality of Sleep  Physical Activity Level  Stress Level BMI 
Category  \
0                  6                       42             6    
Overweight
```

1 Normal	6	60	8
2 Normal	6	60	8
3 Obese	4	30	8
4 Obese	4	30	8
5 Obese	4	30	8
6 Obese	6	40	7
7 Normal	7	75	6
8 Normal	7	75	6
9 Normal	7	75	6
10 Normal	6	30	8
11 Normal	7	75	6
12 Normal	6	30	8
13 Normal	6	30	8
14 Normal	6	30	8
15 Normal	6	30	8
16 Normal	5	40	7
17 Normal	6	30	8
18 Normal	5	40	7
19 Normal	7	75	6
20 Normal	7	75	6
21 Normal	7	75	6
22 Normal	7	75	6
23 Normal	7	75	6
24 Normal	7	75	6
25 Normal	7	75	6

```
26                 7                       75             6
Normal
27                 7                       75             6
Normal
28                 7                       75             6
Normal
29                 7                       75             6
Normal

   Blood Pressure  Heart Rate  Daily Steps  Sleep Disorder
0          126/83          77         4200               0
1          125/80          75        10000               0
2          125/80          75        10000               0
3          140/90          85         3000               1
4          140/90          85         3000               1
5          140/90          85         3000               2
6          140/90          82         3500               2
7          120/80          70         8000               0
8          120/80          70         8000               0
9          120/80          70         8000               0
10         120/80          70         8000               0
11         120/80          70         8000               0
12         120/80          70         8000               0
13         120/80          70         8000               0
14         120/80          70         8000               0
15         120/80          70         8000               0
16         132/87          80         4000               1
17         120/80          70         8000               1
18         132/87          80         4000               2
19         120/80          70         8000               0
20         120/80          70         8000               0
21         120/80          70         8000               0
22         120/80          70         8000               0
23         120/80          70         8000               0
24         120/80          70         8000               0
25         120/80          70         8000               0
26         120/80          70         8000               0
27         120/80          70         8000               0
28         120/80          70         8000               0
29         120/80          70         8000               0
```

```
data['BMI Category'] = [2 if value == "Overweight" else 1 if value ==
'Obese' else 0 for value in data['BMI Category']]

data.head(30)
```

```
    Person ID  Gender  Age            Occupation  Sleep Duration  \
0           1       1   27     Software Engineer             6.1
1           2       1   28                Doctor             6.2
2           3       1   28                Doctor             6.2
3           4       1   28  Sales Representative             5.9
```

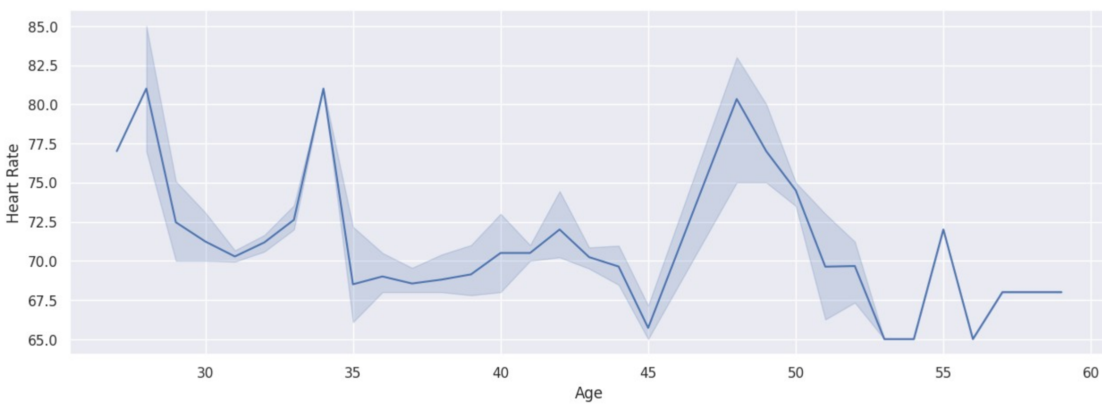
```
4            5       1   28  Sales Representative             5.9
5            6       1   28     Software Engineer             5.9
6            7       1   29               Teacher             6.3
7            8       1   29                Doctor             7.8
8            9       1   29                Doctor             7.8
9           10       1   29                Doctor             7.8
10          11       1   29                Doctor             6.1
11          12       1   29                Doctor             7.8
12          13       1   29                Doctor             6.1
13          14       1   29                Doctor             6.0
14          15       1   29                Doctor             6.0
15          16       1   29                Doctor             6.0
16          17       0   29                 Nurse             6.5
17          18       1   29                Doctor             6.0
18          19       0   29                 Nurse             6.5
19          20       1   30                Doctor             7.6
20          21       1   30                Doctor             7.7
21          22       1   30                Doctor             7.7
22          23       1   30                Doctor             7.7
23          24       1   30                Doctor             7.7
24          25       1   30                Doctor             7.8
25          26       1   30                Doctor             7.9
26          27       1   30                Doctor             7.8
27          28       1   30                Doctor             7.9
28          29       1   30                Doctor             7.9
29          30       1   30                Doctor             7.9

    Quality of Sleep  Physical Activity Level  Stress Level  BMI
Category  \
0                  6                       42             6
2
1                  6                       60             8
0
2                  6                       60             8
0
3                  4                       30             8
1
4                  4                       30             8
1
5                  4                       30             8
1
6                  6                       40             7
1
7                  7                       75             6
0
8                  7                       75             6
0
9                  7                       75             6
0
10                 6                       30             8
```

```
0
11                 7                      75              6
0
12                 6                      30              8
0
13                 6                      30              8
0
14                 6                      30              8
0
15                 6                      30              8
0
16                 5                      40              7
0
17                 6                      30              8
0
18                 5                      40              7
0
19                 7                      75              6
0
20                 7                      75              6
0
21                 7                      75              6
0
22                 7                      75              6
0
23                 7                      75              6
0
24                 7                      75              6
0
25                 7                      75              6
0
26                 7                      75              6
0
27                 7                      75              6
0
28                 7                      75              6
0
29                 7                      75              6
0

   Blood Pressure  Heart Rate  Daily Steps  Sleep Disorder
0          126/83          77         4200               0
1          125/80          75        10000               0
2          125/80          75        10000               0
3          140/90          85         3000               1
4          140/90          85         3000               1
5          140/90          85         3000               2
6          140/90          82         3500               2
7          120/80          70         8000               0
8          120/80          70         8000               0
```

9	120/80	70	8000	0
10	120/80	70	8000	0
11	120/80	70	8000	0
12	120/80	70	8000	0
13	120/80	70	8000	0
14	120/80	70	8000	0
15	120/80	70	8000	0
16	132/87	80	4000	1
17	120/80	70	8000	1
18	132/87	80	4000	2
19	120/80	70	8000	0
20	120/80	70	8000	0
21	120/80	70	8000	0
22	120/80	70	8000	0
23	120/80	70	8000	0
24	120/80	70	8000	0
25	120/80	70	8000	0
26	120/80	70	8000	0
27	120/80	70	8000	0
28	120/80	70	8000	0
29	120/80	70	8000	0

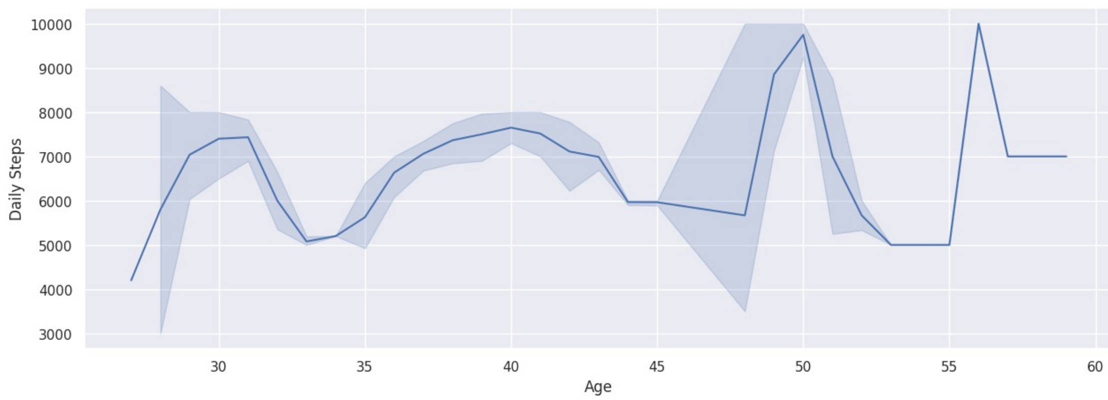
```
sns.lineplot(data=data, x='Age', y='Heart Rate')
```

```
<Axes: xlabel='Age', ylabel='Heart Rate'>
```



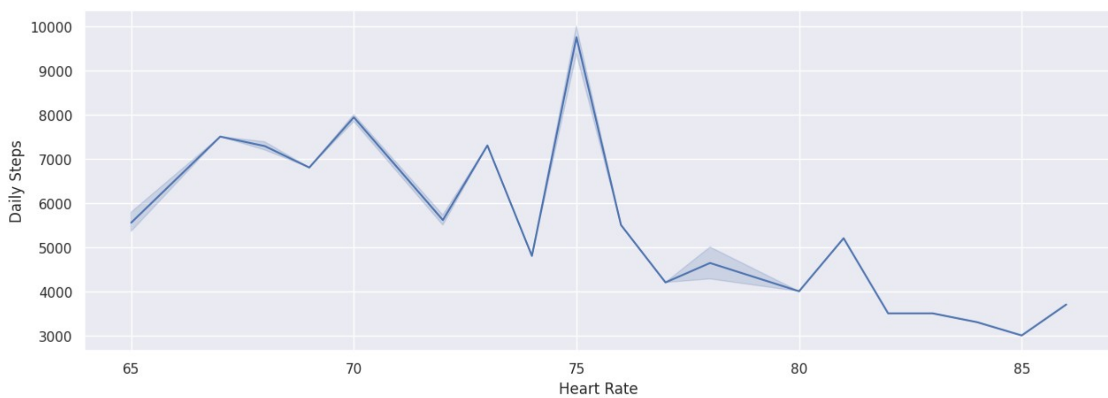

```
sns.lineplot(data=data, x='Age', y='Daily Steps')
```

```
<Axes: xlabel='Age', ylabel='Daily Steps'>
```



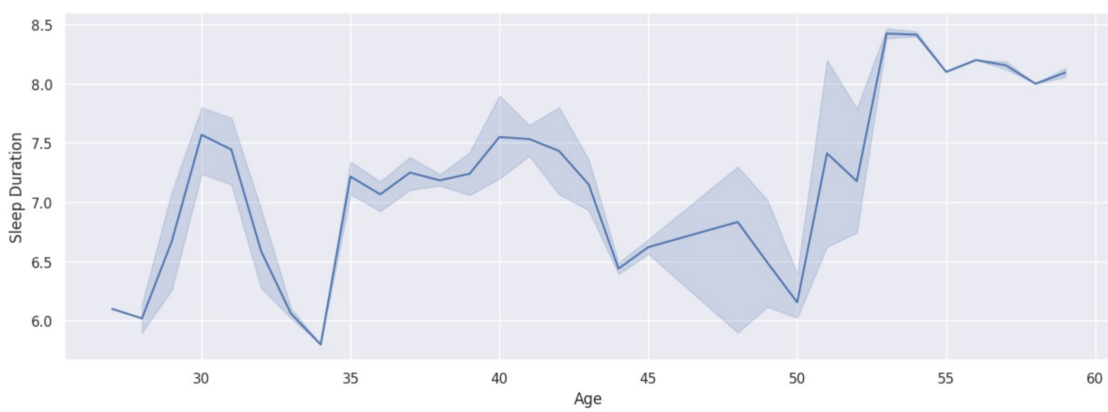

```
sns.lineplot(data=data, x='Heart Rate', y='Daily Steps')
```

```
<Axes: xlabel='Heart Rate', ylabel='Daily Steps'>
```



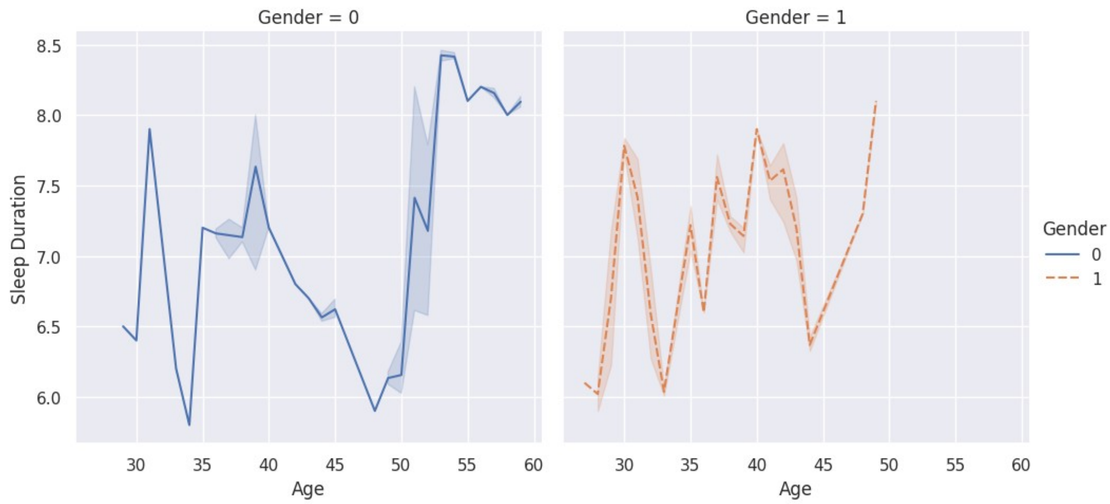

```
sns.lineplot(data=data, x='Age', y='Sleep Duration')
```

```
<Axes: xlabel='Age', ylabel='Sleep Duration'>
```




```
sns.relplot(
    data=data, x="Age", y="Sleep Duration",
    col="Gender", hue="Gender", style="Gender",
    kind="line"
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f0d8f07fe80>
```



Summary

There are 189 men and 185 women in the dataset. The predominant people who have no sleep problem are 219, while people with sleep problems are as many as 78 (Sleep Apnea) and 77 (Insomnia). The three most predominant occupations of these people are respectively: Nurses, Doctors and Engineers. As for their weight, most are normal (195) or overweight (148). As for heart rate, men aged 48-49 have the biggest problems, with both genders having similar heart rates that average around 70.2. People aged 50 and 56-58 care the most about taking daily steps (more than 10,000), which also translates into a reduced heart rate from those who do not do such physical activity daily. Those aged 53-58 get the most sleep. In contrast, those under 30 and between 30 and 34 years old get the least sleep (6-6.5h). People aged 50, also sleep less than 7h a day. As for the difference between men and women, generally speaking, women sleep fewer hours a day than men (they rarely exceed 8h of sleep a day).

Logistic Regression

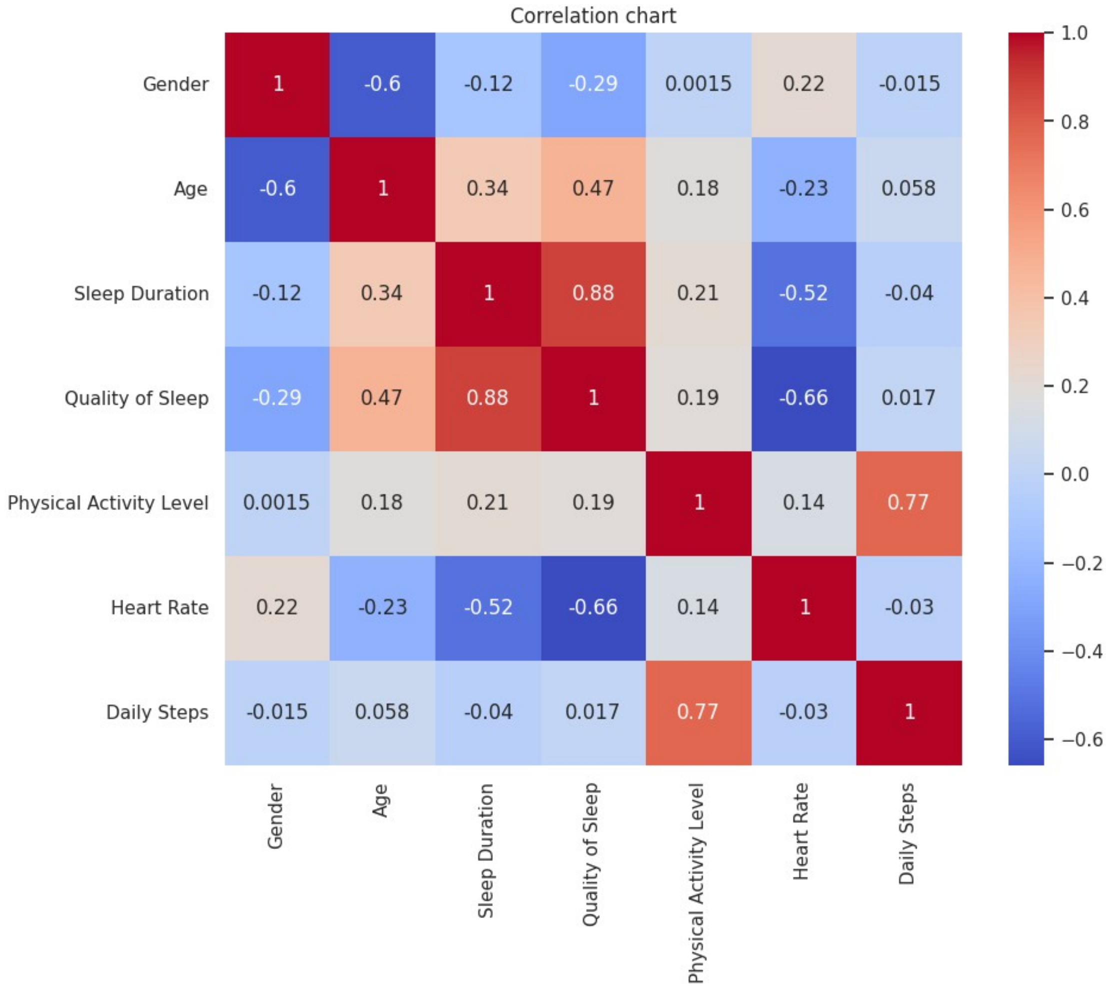
```
#Assign new variable with selected columns for binary modeling.
data1 = data[['Gender','Age','Sleep Duration','Quality of
Sleep','Physical Activity Level','Heart Rate','Daily Steps']]

data1['Gender'].unique()

array([1, 0])

data1.head(30)

    Gender  Age  Sleep Duration  Quality of Sleep  Physical Activity
Level  \
0        1   27             6.1                 6
42
1        1   28             6.2                 6
```

60				
2	1	28	6.2	6
60				
3	1	28	5.9	4
30				
4	1	28	5.9	4
30				
5	1	28	5.9	4
30				
6	1	29	6.3	6
40				
7	1	29	7.8	7
75				
8	1	29	7.8	7
75				
9	1	29	7.8	7
75				
10	1	29	6.1	6
30				
11	1	29	7.8	7
75				
12	1	29	6.1	6
30				
13	1	29	6.0	6
30				
14	1	29	6.0	6
30				
15	1	29	6.0	6
30				
16	0	29	6.5	5
40				
17	1	29	6.0	6
30				
18	0	29	6.5	5
40				
19	1	30	7.6	7
75				
20	1	30	7.7	7
75				
21	1	30	7.7	7
75				
22	1	30	7.7	7
75				
23	1	30	7.7	7
75				
24	1	30	7.8	7
75				
25	1	30	7.9	7
75				
26	1	30	7.8	7

```
75
27       1   30            7.9                 7
75
28       1   30            7.9                 7
75
29       1   30            7.9                 7
75

    Heart Rate  Daily Steps
0           77         4200
1           75        10000
2           75        10000
3           85         3000
4           85         3000
5           85         3000
6           82         3500
7           70         8000
8           70         8000
9           70         8000
10          70         8000
11          70         8000
12          70         8000
13          70         8000
14          70         8000
15          70         8000
16          80         4000
17          70         8000
18          80         4000
19          70         8000
20          70         8000
21          70         8000
22          70         8000
23          70         8000
24          70         8000
25          70         8000
26          70         8000
27          70         8000
28          70         8000
29          70         8000
```

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Generate a correlation matrix
correlation_matrix = data1.corr()

# Correlation chart
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
```

```
plt.title("Correlation chart")
plt.show()
```




```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

#Divide dataset into training and test dataset
X = data1.drop('Gender', axis=1)
y = data1['Gender']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)

#Display maximum number of columns and all columns will be displayed
in full width.
pd.set_option('display.max_columns', None)

# display of crop dimensions
print('Collection sizes:')
print(f'Training collection: {X_train.shape}, attrition:
```

```
{y_train.shape}')
print(f'Test collection: {X_test.shape}, attrition: {y_test.shape}')
```

```
Collection sizes:
Training collection: (299, 6), attrition: (299,)
Test collection: (75, 6), attrition: (75,)
```

```
# Fitting a logistic regression model
logistic_model = LogisticRegression(max_iter=500)
logistic_model.fit(X_train, y_train)
```

```
LogisticRegression(max_iter=500)
```

```
#Generate predictions for test data
y_pred = logistic_model.predict(X_test)

#Display the first 10 lines
y_pred[1:10,]
```

```
array([1, 1, 0, 1, 0, 1, 1, 0, 1])
```

```
#Class prediction for test data
y_proba = logistic_model.predict_proba(X_test)

#Display the first 10 rows of class probability predictions for the
test data (pos and neg, respectively)
y_proba[1:10,]
```

```
array([[0.16402812, 0.83597188],
       [0.22074956, 0.77925044],
       [0.95032916, 0.04967084],
       [0.19822033, 0.80177967],
       [0.69011941, 0.30988059],
       [0.23349705, 0.76650295],
       [0.3163536 , 0.6836464 ],
       [0.96017011, 0.03982989],
       [0.40340015, 0.59659985]])
```

```
# Creating a data frame
result_df = pd.DataFrame()
result_df['gender'] = y_test
result_df['expected class'] = y_pred
result_df['0'] = y_proba[:, 0]
result_df['1'] = y_proba[:, 1]

#Display the first 10 lines.
result_df.head(10)
```

	gender	expected class	0	1
329	0	0	0.950329	0.049671
33	1	1	0.164028	0.835972
15	1	1	0.220750	0.779250
325	0	0	0.950329	0.049671

```
57         1                1  0.198220  0.801780
239        1                0  0.690119  0.309881
76         1                1  0.233497  0.766503
119        0                1  0.316354  0.683646
332        0                0  0.960170  0.039830
126        1                1  0.403400  0.596600
```

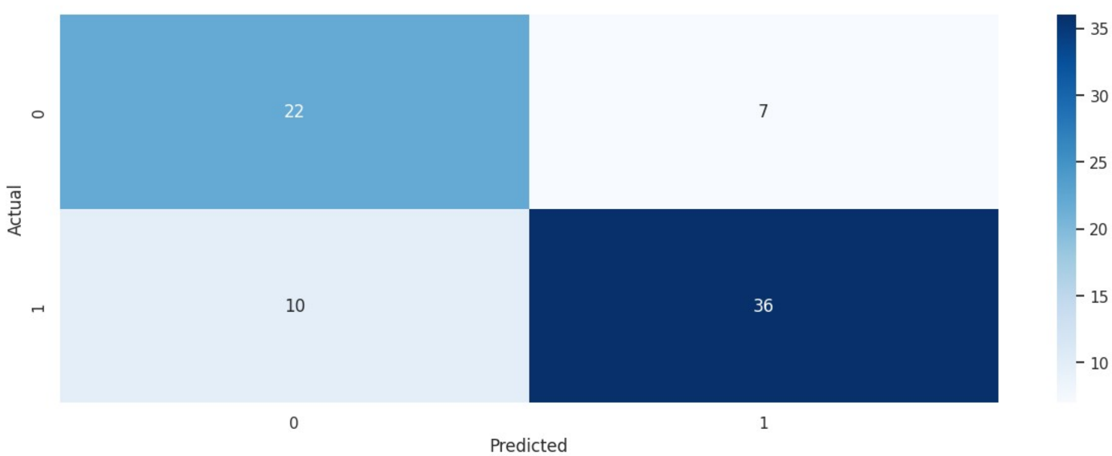
```
#Model Evaluation
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the model: {:.2f}%".format(accuracy*100))
```

```
Accuracy of the model: 77.33%
```

```
from sklearn.metrics import confusion_matrix
# Creating a confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[22  7]
 [10 36]]
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

y_pred = logistic_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

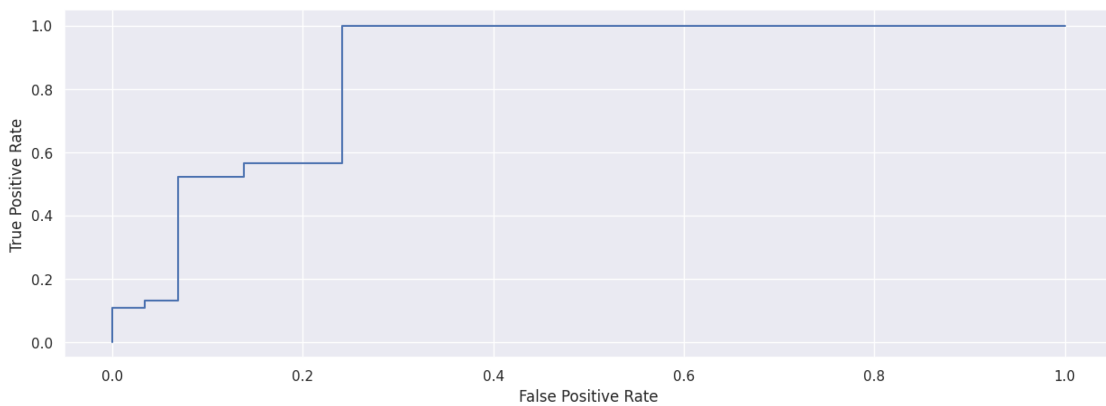



```
#Roc auc
from sklearn.metrics import roc_auc_score
```

```python
# Calculation of ROC AUC for a test set
y_prob = logistic_model.predict_proba(X_test)[:, 1] # we select the
column for the positive class
roc_auc = roc_auc_score(y_test, y_prob)
print("The ROC AUC is: {:.2f}".format(roc_auc))
```

```
The ROC AUC is: 0.86
```

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, RocCurveDisplay

y_pred = logistic_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.show()
```



Summary

Our model obtained an ROC-AUC score of `0.86`, indicating a well-performing model that can distinguish the gender of a given test subject.

Neural network model

In the model on neural networks, we will try to train our model to correctly predict heart rate values in our dataset.

```python
data.head()
```

```
   Person ID  Gender  Age           Occupation  Sleep Duration  \
0          1       1   27     Software Engineer             6.1
1          2       1   28                Doctor             6.2
2          3       1   28                Doctor             6.2
3          4       1   28  Sales Representative             5.9
4          5       1   28  Sales Representative             5.9

   Quality of Sleep  Physical Activity Level  Stress Level  BMI
Category  \
```

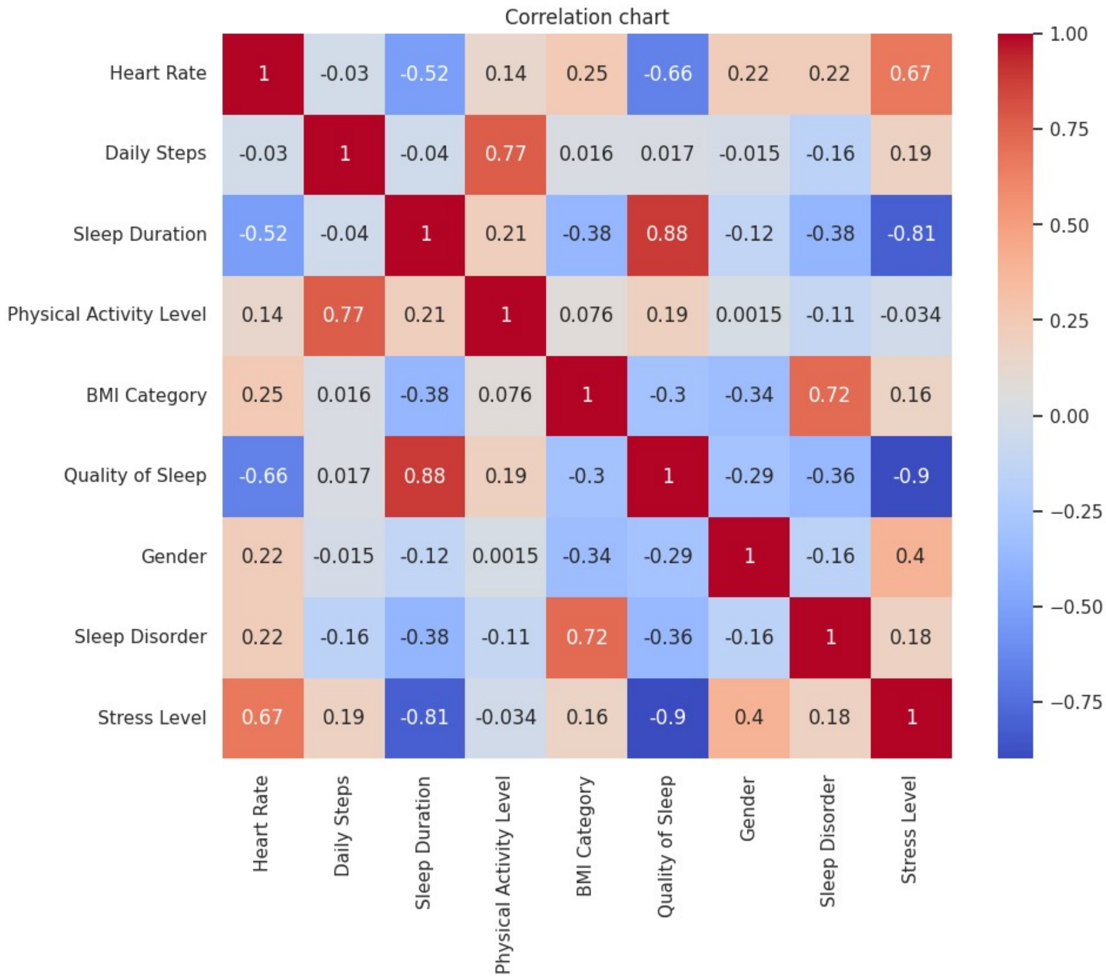
```
0                  6                       42             6
2
1                  6                       60             8
0
2                  6                       60             8
0
3                  4                       30             8
1
4                  4                       30             8
1

  Blood Pressure  Heart Rate  Daily Steps  Sleep Disorder
0         126/83          77         4200               0
1         125/80          75        10000               0
2         125/80          75        10000               0
3         140/90          85         3000               1
4         140/90          85         3000               1
```

```
data2 = data[['Heart Rate','Daily Steps','Sleep Duration','Physical
Activity Level','BMI Category','Quality of Sleep','Gender','Sleep
Disorder','Stress Level']]

data2.head()
```

```
   Heart Rate  Daily Steps  Sleep Duration  Physical Activity Level  \
0          77         4200             6.1                       42
1          75        10000             6.2                       60
2          75        10000             6.2                       60
3          85         3000             5.9                       30
4          85         3000             5.9                       30

   BMI Category  Quality of Sleep  Gender  Sleep Disorder  Stress
Level
0             2                 6       1               0
6
1             0                 6       1               0
8
2             0                 6       1               0
8
3             1                 4       1               1
8
4             1                 4       1               1
8
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Generate a correlation matrix
correlation_matrix = data2.corr()
```

```python
# Correlation chart
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation chart")
plt.show()
```




```python
#Creation of a new data frame without Heart Rate
X = data2.drop('Heart Rate', axis = 1)

#Dimension of variable X
X.shape
```

```
(374, 8)
```

```python
#Convert the data from the 'Heart Rate' column to a numpy array and check the dimensions
y = data2['Heart Rate'].to_numpy()
y.shape
```

```
(374,)
```

```
#Convert data from data without MonthlyIncome to numpy array and check
dimensions
X = X.to_numpy()
X.shape
```

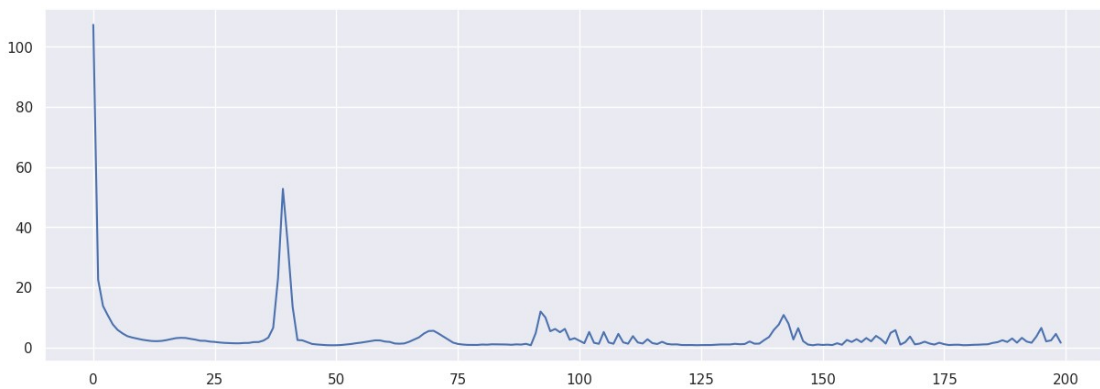
```
(374, 8)
```

```
import copy

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
import tqdm
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler


# We are splitting the data into a training set and a test set
X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y,
train_size=0.8, shuffle=True)

# We standardize the data using the StandardScaler class
scaler = StandardScaler()
scaler.fit(X_train_raw)
X_train = scaler.transform(X_train_raw)
X_test = scaler.transform(X_test_raw)

# We are converting the data to 2D PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).reshape(-1, 1)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32).reshape(-1, 1)

# Defines a neural network model using the `nn.Sequential()` class.
model = nn.Sequential(
    nn.Linear(8, 128),
    nn.ReLU(),
    nn.Linear(128, 64),
    nn.ReLU(),
    nn.Linear(64, 32),
    nn.ReLU(),
    nn.Linear(32, 1)
)

# We define the loss function as mean square error (MSELoss) using the
function `nn.MSELoss()`.
```

```python
loss_fn = nn.MSELoss()  # mean square error
optimizer = optim.Adam(model.parameters(), lr=0.01)

#Determine the number of epochs (n_epochs) equal to 200 and the batch
size (batch_size) equal to 1
n_epochs = 200
batch_size = 8

#Initialize the batch_start list as a sequence of indices from 0 to
the length of the training set
batch_start = torch.arange(0, len(X_train), batch_size)

#Let's initialize the best result of the mean squared error (best_mse)
to infinity
best_mse = np.inf
best_weights = None
history = []

#Starts a training loop, iterating through epochs
for epoch in range(n_epochs):
    model.train()
    with tqdm.tqdm(batch_start, unit="batch", mininterval=0,
disable=True) as bar:
        bar.set_description(f"Epoch {epoch}")
        for start in bar:

            X_batch = X_train[start:start+batch_size]
            y_batch = y_train[start:start+batch_size]


            y_pred = model(X_batch)
            loss = loss_fn(y_pred, y_batch)

            optimizer.zero_grad()
            loss.backward()
            #We update the weights
            optimizer.step()

            bar.set_postfix(mse=float(loss))
    #Evaluate accuracy for each era
    model.eval()

    y_pred = model(X_test)
    mse = loss_fn(y_pred, y_test)
    mse = float(mse)
    history.append(mse)
    if mse < best_mse:
        best_mse = mse
        best_weights = copy.deepcopy(model.state_dict())
```

```
#Return model and display best accuracy
model.load_state_dict(best_weights)
print("MSE: %.2f" % best_mse)
print("RMSE: %.2f" % np.sqrt(best_mse))
plt.plot(history)
plt.show()

model.eval()
with torch.no_grad():
    #We test inference based on 5 samples
    for i in range(5):
        X_sample = X_test_raw[i: i+1]
        X_sample = scaler.transform(X_sample)
        X_sample = torch.tensor(X_sample, dtype=torch.float32)
        y_pred = model(X_sample)
        print(f"{X_test_raw[i]} -> {y_pred[0].numpy()} (expected
{y_test[i].numpy()})")
```

```
MSE: 0.72
RMSE: 0.85
```




```
[6.0e+03 6.7e+00 4.5e+01 2.0e+00 7.0e+00 0.0e+00 2.0e+00 4.0e+00] ->
[65.00131] (expected [65.])
[6.0e+03 6.5e+00 4.5e+01 2.0e+00 6.0e+00 1.0e+00 2.0e+00 7.0e+00] ->
[71.1698] (expected [72.])
[6.8e+03 7.9e+00 7.5e+01 0.0e+00 8.0e+00 0.0e+00 0.0e+00 4.0e+00] ->
[69.24059] (expected [69.])
[8.0e+03 7.8e+00 9.0e+01 0.0e+00 8.0e+00 1.0e+00 0.0e+00 5.0e+00] ->
[69.47745] (expected [70.])
[7.0e+03 8.0e+00 7.5e+01 2.0e+00 9.0e+00 0.0e+00 1.0e+00 3.0e+00] ->
[67.729416] (expected [68.])
```

```
# Model evaluation on training set
model.eval()
with torch.no_grad():
    y_pred_train = model(X_train)
    train_mse = loss_fn(y_pred_train, y_train)
```

```
    train_mse = float(train_mse)
    print("Train MSE: %.2f" % train_mse)
    print("Train RMSE: %.2f" % np.sqrt(train_mse))

# Model evaluation on test set
model.eval()
with torch.no_grad():
    y_pred_test = model(X_test)
    test_mse = loss_fn(y_pred_test, y_test)
    test_mse = float(test_mse)
    print("Test MSE: %.2f" % test_mse)
    print("Test RMSE: %.2f" % np.sqrt(test_mse))
```

```
Train MSE: 0.52
Train RMSE: 0.72
Test MSE: 0.72
Test RMSE: 0.85
```

Comparing the evaluation of the model on both the training and test sets, we conclude that neither under-training nor over-training occurred.

Evaluation of model on neural networks

```
y_pred = model(X_test)
y_pred = y_pred.view(-1, 1)  #Flattening the y_pred tensor to a one-
dimensional form

mse = loss_fn(y_pred, y_test)
mse = float(mse)

from sklearn.metrics import mean_absolute_error, r2_score,
mean_squared_error

y_pred_np = y_pred.detach().numpy()
y_test_np = y_test.detach().numpy()

mae = mean_absolute_error(y_test_np, y_pred_np)
r2 = r2_score(y_test_np, y_pred_np)
rmse = np.sqrt(mean_squared_error(y_test_np, y_pred_np))

print(f"Average absolute error is: {mae}")
print(f"The coefficient of R^2 is: {r2}")
print(f"Mean squared error is: {rmse}")
```

```
Average absolute error is: 0.536781907081604
The coefficient of R^2 is: 0.9094620054387987
Mean squared error is: 0.8496408462524414
```

Cross-Validation Algorithm

```
import copy
import numpy as np
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error

# Define the number of folds for cross-validation
n_folds = 5

# Convert the data to numpy arrays
X = X_train.numpy()
y = y_train.numpy()

# Initialize lists to store the MSE scores for each fold
train_mse_scores = []
test_mse_scores = []

# Perform cross-validation
kfold = KFold(n_splits=n_folds, shuffle=True)
for fold, (train_index, test_index) in enumerate(kfold.split(X)):
    print(f"Fold: {fold+1}/{n_folds}")

    # Split the data into training and test sets for the current fold
    X_train_fold, X_test_fold = X[train_index], X[test_index]
    y_train_fold, y_test_fold = y[train_index], y[test_index]

    # Convert the data back to PyTorch tensors
    X_train_fold = torch.tensor(X_train_fold, dtype=torch.float32)
    y_train_fold = torch.tensor(y_train_fold,
dtype=torch.float32).reshape(-1, 1)
    X_test_fold = torch.tensor(X_test_fold, dtype=torch.float32)
    y_test_fold = torch.tensor(y_test_fold,
dtype=torch.float32).reshape(-1, 1)

    # Create a new instance of the model for each fold
    model = nn.Sequential(
        nn.Linear(8, 128),
        nn.ReLU(),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.Linear(64, 32),
        nn.ReLU(),
        nn.Linear(32, 1)
    )

    # Define the loss function and optimizer
    loss_fn = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```python
    # Train the model
    n_epochs = 200
    for epoch in range(n_epochs):
        model.train()
        optimizer.zero_grad()

        y_pred = model(X_train_fold)
        loss = loss_fn(y_pred, y_train_fold)
        loss.backward()
        optimizer.step()

    # Evaluate the model on the training and test sets
    model.eval()

    with torch.no_grad():
        y_train_pred = model(X_train_fold)
        train_mse = mean_squared_error(y_train_fold.numpy(),
y_train_pred.numpy())
        train_mse_scores.append(train_mse)

        y_test_pred = model(X_test_fold)
        test_mse = mean_squared_error(y_test_fold.numpy(),
y_test_pred.numpy())
        test_mse_scores.append(test_mse)

# Calculate the mean and standard deviation of the MSE scores
train_mse_mean = np.mean(train_mse_scores)
train_mse_std = np.std(train_mse_scores)
test_mse_mean = np.mean(test_mse_scores)
test_mse_std = np.std(test_mse_scores)

# Print the results
print("Train MSE (mean): {:.2f}".format(train_mse_mean))
print("Train MSE (std): {:.2f}".format(train_mse_std))
print("Test MSE (mean): {:.2f}".format(test_mse_mean))
print("Test MSE (std): {:.2f}".format(test_mse_std))
```

```
Fold: 1/5
Fold: 2/5
Fold: 3/5
Fold: 4/5
Fold: 5/5
Train MSE (mean): 2.12
Train MSE (std): 0.27
Test MSE (mean): 4.81
Test MSE (std): 2.10
```

Using cross-validation for this model, we find that the model is somewhat overtrained, so we leave our earlier well-trained model.

```
X_test_numpy = X_test.squeeze().detach().numpy()
```

```
X_test_numpy.shape
```

```
(75, 8)
```

```
y_test_numpy = y_test.squeeze().detach().numpy()
```

```
y_test_numpy.shape
```

```
(75,)
```

```
y_test_numpy = y_test.squeeze().detach().numpy()
```

```
y_pred.shape
```

```
torch.Size([240, 1])
```

```
y_test_numpy.shape
```

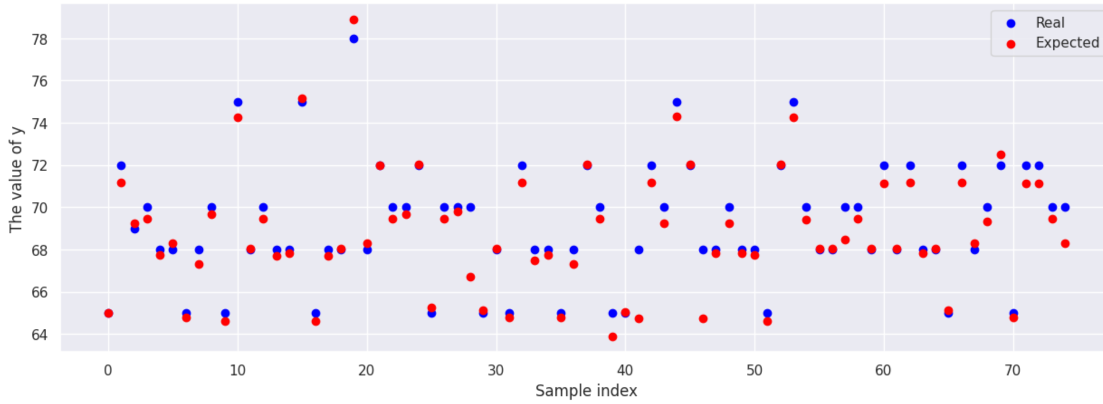
```
(75,)
```

```
print("X_test.shape:", X_test.shape)
print("y_test_numpy.shape:", y_test_numpy.shape)
```

```
X_test.shape: torch.Size([75, 8])
y_test_numpy.shape: (75,)
```

Real vs Expected chart

```
import matplotlib.pyplot as plt

#Predicting y-values for test data
y_pred = model(X_test).detach().numpy()

#Creating a chart
plt.scatter(range(len(y_test_numpy)), y_test_numpy, color='blue',
label='Real')
plt.scatter(range(len(y_test_numpy)), y_pred_np, color='red',
label='Expected')
plt.xlabel('Sample index')
plt.ylabel('The value of y')
plt.legend()
plt.show()
```



As we can see from the chart above comparing the values predicted by our model with the actual values, we can conclude that our model almost perfectly predicts the heart rate (`Heart Rate`) value.

Tuning the model

```
import copy
from sklearn.model_selection import GridSearchCV

#A multivariate matrix for the variable X_train
X_train

tensor([[ 1.9013, -1.2230,  1.4700,  ..., -1.0034,  0.4674,  1.4103],
        [ 1.9013, -1.2230,  1.4700,  ..., -1.0034,  0.4674,  1.4103],
        [ 0.7123,  0.8893,  0.7603,  ...,  0.9967, -0.7804,  0.2933],
        ...,
        [ 0.7123,  0.7651,  1.4700,  ...,  0.9967, -0.7804, -0.2652],
        [-0.4766, -0.7260, -0.6591,  ...,  0.9967,  1.7152,  0.8518],
        [ 0.7123, -1.2230, -1.3688,  ...,  0.9967, -0.7804,  1.4103]])

#Multivariate matrix for the y_train variable
y_train

tensor([[75.],
        [75.],
        [70.],
        [65.],
        [70.],
        [65.],
        [68.],
        [74.],
        [68.],
        [65.],
        [70.],
        [68.],
        [68.],
        [70.],
        [70.],
        [68.],
```

```
[72.],
[70.],
[68.],
[68.],
[77.],
[75.],
[65.],
[75.],
[68.],
[68.],
[68.],
[83.],
[68.],
[65.],
[65.],
[70.],
[67.],
[68.],
[65.],
[68.],
[68.],
[72.],
[70.],
[65.],
[65.],
[65.],
[84.],
[65.],
[75.],
[85.],
[65.],
[83.],
[81.],
[65.],
[68.],
[68.],
[70.],
[75.],
[70.],
[70.],
[86.],
[70.],
[68.],
[68.],
[70.],
[70.],
[70.],
[72.],
[65.],
[72.],
```

```
[75.],
[72.],
[75.],
[65.],
[70.],
[72.],
[70.],
[72.],
[65.],
[65.],
[70.],
[75.],
[68.],
[65.],
[70.],
[72.],
[65.],
[72.],
[72.],
[75.],
[70.],
[65.],
[68.],
[70.],
[65.],
[72.],
[68.],
[68.],
[68.],
[68.],
[70.],
[72.],
[65.],
[75.],
[72.],
[75.],
[70.],
[65.],
[72.],
[68.],
[70.],
[72.],
[75.],
[70.],
[72.],
[72.],
[68.],
[70.],
[77.],
[68.],
```

```
[68.],
[72.],
[78.],
[72.],
[65.],
[70.],
[65.],
[75.],
[72.],
[80.],
[78.],
[72.],
[70.],
[70.],
[72.],
[72.],
[65.],
[72.],
[72.],
[70.],
[68.],
[78.],
[68.],
[85.],
[68.],
[72.],
[65.],
[68.],
[80.],
[73.],
[65.],
[65.],
[82.],
[72.],
[75.],
[72.],
[70.],
[65.],
[68.],
[68.],
[65.],
[75.],
[70.],
[65.],
[75.],
[75.],
[68.],
[72.],
[68.],
[70.],
```

```
[65.],
[72.],
[72.],
[72.],
[72.],
[72.],
[70.],
[68.],
[72.],
[70.],
[68.],
[68.],
[65.],
[65.],
[75.],
[72.],
[70.],
[68.],
[70.],
[65.],
[72.],
[69.],
[72.],
[75.],
[70.],
[65.],
[70.],
[75.],
[72.],
[68.],
[72.],
[72.],
[72.],
[70.],
[68.],
[70.],
[67.],
[72.],
[70.],
[68.],
[68.],
[70.],
[68.],
[75.],
[72.],
[72.],
[72.],
[75.],
[65.],
[75.],
```

```
[70.],
[68.],
[68.],
[72.],
[65.],
[72.],
[75.],
[70.],
[73.],
[65.],
[65.],
[68.],
[68.],
[68.],
[76.],
[68.],
[70.],
[75.],
[68.],
[70.],
[65.],
[68.],
[72.],
[65.],
[68.],
[70.],
[68.],
[70.],
[65.],
[84.],
[65.],
[68.],
[65.],
[72.],
[68.],
[80.],
[65.],
[75.],
[68.],
[68.],
[65.],
[75.],
[68.],
[81.],
[68.],
[68.],
[68.],
[70.],
[85.],
[65.],
```

```
       [65.],
       [65.],
       [78.],
       [65.],
       [65.],
       [68.],
       [75.],
       [75.],
       [68.],
       [70.],
       [70.],
       [72.],
       [75.],
       [86.],
       [72.],
       [68.],
       [72.],
       [76.],
       [68.],
       [65.],
       [68.],
       [74.],
       [70.],
       [72.],
       [70.],
       [68.],
       [70.],
       [75.],
       [70.],
       [70.],
       [70.],
       [72.],
       [70.]])
```

```python
#pip install skorch

import copy
from sklearn.model_selection import GridSearchCV
from skorch import NeuralNetRegressor

#Define the model
class CustomModel(nn.Module):
    def __init__(self, hidden_sizes):
        super(CustomModel, self).__init__()
        layers = []
        input_size = 8
        for size in hidden_sizes:
            layers.append(nn.Linear(input_size, size))
            layers.append(nn.ReLU())
            input_size = size
```

```
        layers.append(nn.Linear(input_size, 1))
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)

# We are creating an instance of a custom model
model = CustomModel(hidden_sizes=(128,))


# We define the parameters of the Grid algorithm
param_grid = {
    'module__hidden_sizes': [(32, 64, 128), (128,64,32), (64,128, 32),
(128, 32, 64)],
    'batch_size': [1,8,16]
}

# We create a NeuralNetRegressor function with a custom model of our
choice
net = NeuralNetRegressor(
    module=model,
    criterion=nn.MSELoss,
    optimizer=optim.Adam,
    lr=0.01
)

# We define the GridSearchCV object
grid_search = GridSearchCV(estimator=net, param_grid=param_grid,
scoring='neg_mean_squared_error', cv=5)

# We define the GridSearchCV object
grid_search.fit(X_train, y_train)

# We choose the best model and its parameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

# We are training our best model
best_model.fit(X_train, y_train)

# We evaluate the best model for the test data
y_pred = best_model.predict(X_test)
mse = nn.MSELoss()(torch.tensor(y_pred), y_test)
mse = float(mse)
rmse = np.sqrt(mse)

print("Best Parameters:", best_params)
print("Best MSE:", mse)
print("Best RMSE:", rmse)
```

```
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      451.3301       39.7120  1.1260
      2       30.6367       34.9429  0.9715
      3       31.7470       27.0987  0.3198
      4       30.1712       30.3370  0.3016
      5       16.0802       51.2438  0.3192
      6       46.3404       16.6813  0.3295
      7       14.4101       10.9584  0.3218
      8        6.3353        6.4533  0.3281
      9        6.1103        6.3276  0.3265
     10       14.5731        6.5588  0.3284
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      539.6226       74.0125  0.2981
      2       27.1386       20.9518  0.3017
      3       12.1703        9.2937  0.3036
      4       25.2094       35.5859  0.2921
      5       35.8084       23.4012  0.3370
      6       13.5705       13.4162  0.3446
      7       13.0934       10.6328  0.3188
      8       11.5259       10.6023  0.3367
      9       11.9112       11.9052  0.3315
     10       23.6397        7.5049  0.3169
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      644.8016       29.6024  0.3112
      2       21.2382       19.0869  0.3521
      3       11.1366        9.3925  0.3033
      4       27.6133       34.4770  0.3122
      5       55.4981       48.9617  0.3346
      6       27.4425       18.7001  0.3176
      7       14.3713       12.2480  0.3320
      8       11.2468        9.2926  0.3414
      9        9.8118        8.3007  0.3209
     10       10.9083        6.6305  0.3262
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      498.8382       17.6612  0.3153
      2       14.9039       16.6673  0.2901
      3       13.4150       21.2504  0.3896
      4       23.5144       27.9212  0.4211
      5       31.6392       52.1620  0.4574
      6       34.7544       46.5022  0.4621
      7       19.6427       27.1388  0.4439
      8        9.4393       26.8290  0.4781
      9       15.0062       51.6668  0.3170
     10       22.1672       36.5702  0.3186
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
```

```
      1      493.0324      267.4842  0.3375
      2       85.6523       45.6087  0.2998
      3       19.9754       16.3398  0.2974
      4       11.8500       29.0215  0.3138
      5       16.4819      139.2552  0.3225
      6      147.9238       77.0446  0.3312
      7        9.1254        9.9248  0.3319
      8        6.3430       14.3371  0.3224
      9        6.0951       23.8786  0.3207
     10        7.7499       18.0704  0.3309
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      455.8328       50.9938  0.2898
      2       54.4383       19.5610  0.2943
      3       33.1468       33.6418  0.3008
      4       74.9911       85.6269  0.3110
      5       97.5719       37.1405  0.3121
      6       25.0664       28.9939  0.3099
      7       31.8338       20.8128  0.3165
      8       11.8621       11.1763  0.3087
      9        6.4696        9.5377  0.3167
     10        8.2498        6.1951  0.3316
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      495.7108       40.3383  0.2958
      2       51.3816       30.0852  0.3297
      3       22.9974       25.4296  0.3146
      4       26.6452       56.1555  0.2993
      5       41.2633       49.7303  0.3375
      6       27.2383       46.7368  0.3308
      7       19.2873       36.2157  0.3158
      8      112.3005       33.7112  0.3298
      9       28.2822       23.4095  0.3225
     10       16.5850       14.3362  0.4745
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      557.3904       33.2927  0.4200
      2       23.1796       16.5411  0.4447
      3       18.8692       68.0179  0.4255
      4       56.8721      236.5787  0.4244
      5      100.4138      135.4182  0.4602
      6       20.1893       21.5947  0.3184
      7       11.3095       12.5501  0.3253
      8        7.9616        6.5123  0.3085
      9       10.3934        6.2799  0.3170
     10       15.7892       21.4418  0.3128
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      552.5406       40.9192  0.2892
      2       28.5732       48.0006  0.3035
```

```
      3       31.1857       32.5232  0.3056
      4       20.1063       18.7971  0.3600
      5       15.7581       51.3672  0.3621
      6       25.7632      123.0330  0.3242
      7       37.1724       67.4067  0.3237
      8       23.9933       39.8114  0.3282
      9       18.5205       46.5137  0.3137
     10       21.1716       77.7532  0.3191
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      494.2799      139.1496  0.2956
      2       54.1106       60.6752  0.3000
      3       24.9334       19.3308  0.3024
      4       24.8711      154.0759  0.2926
      5      143.8127       63.7811  0.3126
      6       20.1711       11.8505  0.3159
      7        6.3673        9.5569  0.3352
      8        3.6210       13.3232  0.3127
      9       13.1013       46.3412  0.3203
     10       30.2056       30.2433  0.3240
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      547.2962       37.8599  0.2909
      2       34.2705       20.8980  0.3062
      3       29.8338       39.6022  0.3208
      4       49.7567       61.4486  0.3180
      5       49.5398       33.4010  0.3259
      6       19.5777       25.5310  0.3375
      7       10.9997       17.4183  0.5415
      8       15.6453       17.5878  0.4919
      9       16.0314       32.7981  0.4597
     10       50.7871      107.9216  0.4987
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      608.6425       62.4576  0.4219
      2       40.4581       15.9830  0.3241
      3       38.8974       42.6371  0.3333
      4       18.3085       14.6467  0.3013
      5       24.7243       17.6967  0.3293
      6        8.7412       13.5928  0.3595
      7       24.5715       25.1576  0.3312
      8       83.4686       59.0610  0.3194
      9       39.3215       40.5273  0.3461
     10       16.7299       15.2460  0.3252
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      455.0243       19.8460  0.2916
      2       17.8002       19.4878  0.3093
      3       19.1401       32.3135  0.3089
      4      101.2583      227.7103  0.2933
```

```
      5      168.1279      11.1088  0.3357
      6        8.5551       9.2585  0.3224
      7        7.9494       7.7494  0.3231
      8        7.7555       7.6787  0.4018
      9        9.2509      10.2888  0.3320
     10       22.4054      20.9086  0.3152
  epoch    train_loss   valid_loss     dur
-------  ------------  ------------  ------
      1      454.1448      33.0042  0.3021
      2       32.4986      30.1729  0.3071
      3       19.4044      31.5559  0.2959
      4       25.5521      55.7723  0.3023
      5       46.9373     115.1532  0.3315
      6       43.8991      35.7241  0.3191
      7       13.7247      18.3835  0.3462
      8       33.2957      10.1203  0.3199
      9       10.3541      15.6790  0.3168
     10       14.8043      12.1523  0.3190
  epoch    train_loss   valid_loss     dur
-------  ------------  ------------  ------
      1      562.3941      46.1951  0.3058
      2       30.8257      14.5927  0.3340
      3       23.4721     140.9447  0.4437
      4       40.7750      20.7549  0.4259
      5       28.9089      58.9428  0.4622
      6       34.5133      39.0188  0.4480
      7       18.9299      12.1692  0.4804
      8        8.7680      32.7178  0.4241
      9       76.4595     786.2772  0.3310
     10      167.4370      24.6331  0.3339
  epoch    train_loss   valid_loss     dur
-------  ------------  ------------  ------
      1      467.4257      43.5372  0.2986
      2       47.6335      19.3019  0.2906
      3       25.7219      20.7478  0.2934
      4       51.1273      87.4171  0.3083
      5      137.3868      57.7459  0.3081
      6       39.3373      21.5537  0.2982
      7       14.0825       6.6468  0.3084
      8       11.4764       5.7390  0.3236
      9       13.4503       9.5450  0.3011
     10        9.3538       9.8842  0.3193
  epoch    train_loss   valid_loss     dur
-------  ------------  ------------  ------
      1      578.8201      65.8373  0.3175
      2       36.8352      10.0677  0.3041
      3       36.0060      42.4009  0.3266
      4       18.8885       7.6725  0.3067
      5       29.3183      48.9639  0.3273
      6       53.7786      68.4393  0.3180
```

```
      7       13.4366       18.4950  0.3408
      8       19.6145       13.8817  0.3028
      9       59.6923       21.2207  0.3074
     10       10.1485        8.0518  0.3203
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      569.1576       33.8167  0.2848
      2       21.2857       11.8636  0.2940
      3       22.8710       89.5147  0.3030
      4       40.9434      198.3086  0.3193
      5      114.4250      153.5020  0.3612
      6       40.6768       20.0500  0.3110
      7        8.3943        9.1196  0.3127
      8        6.7487        7.1883  0.3092
      9       10.0250       18.6862  0.3386
     10       35.3585       67.1546  0.4965
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      558.1662       41.9278  0.4162
      2       26.4435       26.1613  0.4266
      3       20.2336       17.0230  0.4203
      4       12.5719       13.5487  0.4602
      5       15.2101       74.3030  0.4487
      6       31.3171       78.1241  0.3292
      7       35.0494       41.5749  0.3257
      8       28.4962       34.8095  0.3275
      9       17.1303       23.8256  0.3180
     10       10.7115       22.7829  0.3118
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1      661.4837       80.1701  0.2876
      2       26.9156       43.6249  0.2885
      3       19.5523       10.4326  0.2891
      4        9.9832       61.0768  0.3060
      5       43.0500       80.2361  0.3113
      6      154.7735       80.5966  0.2996
      7       13.7982       13.7552  0.3194
      8        7.8170       10.9305  0.3395
      9        6.8549       11.1137  0.3066
     10        3.0113       12.9747  0.3183
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2076.4789      489.5720  0.0411
      2      243.5945       88.9846  0.0434
      3       81.4161       21.8682  0.0439
      4       36.2408       17.7938  0.0492
      5       14.6608        6.8519  0.0452
      6        9.7100        5.1428  0.0472
      7        6.9714        4.1783  0.0415
      8        5.5173        3.3904  0.0411
```

```
      9        4.5215        2.8870  0.0426
     10        3.8126        2.6638  0.0452
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2151.9616      771.2498  0.0405
      2      318.6212      110.3596  0.0469
      3       89.2342       84.1703  0.0533
      4       46.1526       26.3325  0.0446
      5       16.0765       19.0795  0.0471
      6       11.2889       11.6082  0.0435
      7        6.2181        8.8207  0.0444
      8        4.5047        7.4769  0.0568
      9        3.6171        6.7413  0.0437
     10        3.0524        6.4662  0.0484
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2064.7068      689.1932  0.0433
      2      223.4737      108.8028  0.0444
      3       48.0185       46.9166  0.0443
      4       21.4375       31.6562  0.0461
      5       10.0705       19.3448  0.0478
      6        6.2800       15.2061  0.0489
      7        5.0818       14.5101  0.0430
      8        4.1021       13.3377  0.0481
      9        3.4151       11.2287  0.0453
     10        2.5531       10.3900  0.0461
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2099.3092      561.3239  0.0407
      2      217.7952      145.6739  0.0466
      3       52.7433       42.7796  0.0494
      4       23.1394       26.8599  0.0437
      5       15.5841       17.5485  0.0449
      6       11.9087       13.9531  0.0417
      7        9.0427       12.2334  0.0440
      8        7.3203       11.1033  0.0452
      9        6.0191       10.6369  0.0539
     10        5.2777       10.9967  0.0434
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2132.1267      490.2888  0.0427
      2      211.2496      197.9349  0.0464
      3       79.3518       99.2243  0.0453
      4       43.2704       70.1123  0.0439
      5       29.4090       49.4516  0.0479
      6       20.4703       34.1014  0.0472
      7       13.5274       23.6930  0.0480
      8        8.9599       18.1338  0.0429
      9        6.4776       15.0958  0.0451
     10        5.2904       13.3707  0.0442
```

```
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2627.8231      149.4695  0.0413
      2      198.7402       66.4908  0.0449
      3       50.9657       21.6601  0.0485
      4       25.1645       10.6134  0.0491
      5       13.8963        6.4342  0.0451
      6        9.0262        4.3363  0.0470
      7        6.1635        3.1850  0.0481
      8        4.5141        2.2712  0.0511
      9        3.3972        1.6771  0.0559
     10        2.6550        1.2808  0.0488
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2160.0879      729.8386  0.0438
      2      320.3719       90.9018  0.0459
      3       54.5454       40.0533  0.0458
      4       32.0901       18.6967  0.0499
      5       14.8339       12.0664  0.0625
      6        8.9894        9.8622  0.0675
      7        5.8711        8.7378  0.0610
      8        4.5002        8.1710  0.0525
      9        3.6040        7.8263  0.0478
     10        3.0041        7.6383  0.0475
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2113.6449      698.4320  0.0522
      2      256.3248       90.8024  0.0474
      3       37.5519       37.9056  0.0452
      4       18.8021       31.3803  0.0483
      5       11.8194       20.4826  0.0451
      6        7.8793       16.6847  0.0525
      7        5.9229       14.2569  0.0425
      8        4.6272       13.0312  0.0558
      9        3.7902       12.3033  0.0462
     10        3.2230       11.8867  0.0425
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2115.2650      474.7176  0.0409
      2      195.0032      126.2407  0.0430
      3       38.1934       33.0385  0.0461
      4       18.6586       23.1612  0.0430
      5       12.1347       17.3159  0.0484
      6        9.0084       14.5925  0.0424
      7        6.6187       13.3714  0.0439
      8        5.2651       12.1193  0.0429
      9        4.3820       11.4553  0.0421
     10        3.6613       10.4353  0.0433
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
```

```
      1     2004.7895      368.6263  0.0401
      2      136.6449       93.0830  0.0493
      3       35.1638       43.0899  0.0431
      4       17.4142       26.1422  0.0424
      5        8.0779       18.2381  0.0450
      6        5.0085       14.3579  0.0431
      7        4.1081       12.9021  0.0423
      8        3.4189       11.8069  0.0437
      9        2.7298       11.0969  0.0471
     10        2.1812       10.6284  0.0412
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2248.2186      403.0103  0.0419
      2      280.0554       76.5219  0.0692
      3       68.5949       22.4278  0.0643
      4       31.2524       15.1338  0.0618
      5       16.6274        6.4963  0.0666
      6        8.3382        5.3139  0.0706
      7        6.5055        4.6442  0.0611
      8        5.3506        3.7684  0.0599
      9        4.3741        3.0895  0.0702
     10        3.6755        2.6644  0.0584
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2686.3844      204.2690  0.0585
      2      181.4431      104.7297  0.0616
      3       64.9061       54.3717  0.0615
      4       29.2314       30.9397  0.0650
      5       16.0021       20.6373  0.0726
      6        9.2705       16.2159  0.0606
      7        5.8438       14.2740  0.0653
      8        4.2011       13.3158  0.0678
      9        3.2883       12.2586  0.0589
     10        2.6446       11.3868  0.0720
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     1942.7456      754.2504  0.0604
      2      224.0363       93.5311  0.0585
      3       41.7691       37.5517  0.0627
      4       25.1433       31.9190  0.0594
      5       12.9245       20.4887  0.0583
      6        8.2801       15.0249  0.0613
      7        6.2267       12.0509  0.0600
      8        4.7149       10.5817  0.0591
      9        3.8053        9.4579  0.0648
     10        3.1309        8.7662  0.0702
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     1867.5274      757.8856  0.0815
      2      231.5743      188.6330  0.0727
```

```
      3       58.4007       61.7622  0.0659
      4       25.5190       35.2190  0.0677
      5       17.3281       25.8708  0.0695
      6       12.7142       20.0377  0.0606
      7        9.6422       16.7934  0.0729
      8        7.7763       14.6375  0.0677
      9        6.5112       12.7646  0.0695
     10        5.4797       11.3076  0.0603
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2009.4795      387.8243  0.0495
      2      155.4907      110.7767  0.0502
      3       40.1955       42.4524  0.0458
      4       22.0432       29.0012  0.0446
      5        8.6226       18.4509  0.0541
      6        5.7049       14.2693  0.0453
      7        4.2742       11.8152  0.0537
      8        3.2667       10.3159  0.0435
      9        2.7295        9.7288  0.0466
     10        2.4895        9.4270  0.0437
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2161.6203      446.4538  0.0414
      2      293.1743       85.1584  0.0435
      3       66.7537       27.0745  0.0443
      4       38.3971       16.2915  0.0439
      5       15.9446        5.8741  0.0504
      6        8.5950        5.0128  0.0406
      7        6.3693        4.1976  0.0418
      8        4.9570        3.1957  0.0423
      9        3.8355        2.2567  0.0452
     10        3.0290        1.7106  0.0435
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     2094.9591      777.5960  0.0409
      2      307.9174      111.4703  0.0491
      3       61.8960       39.0044  0.0411
      4       27.8980       28.9898  0.0456
      5       15.7550       12.6305  0.0422
      6        7.1919        8.8106  0.0517
      7        4.5128        7.2742  0.0433
      8        3.4758        6.5898  0.0481
      9        2.7456        6.1497  0.0451
     10        2.2797        5.9101  0.0482
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     1893.8672      628.5495  0.0415
      2      166.4855       65.1464  0.0440
      3       34.5547       40.0226  0.0425
      4       22.5117       22.6243  0.0439
```

```
      5      11.6626       19.5463  0.0445
      6       7.4261       14.7327  0.0438
      7       5.4362       11.0501  0.0430
      8       4.0888        9.8521  0.0407
      9       3.4429        9.2393  0.0421
     10       3.0426        9.1978  0.0431
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1    2541.6394      265.7512  0.0398
      2     232.1855      115.1628  0.0426
      3      54.4664       37.6885  0.0424
      4      21.4594       19.5067  0.0434
      5      13.3967       14.1775  0.0459
      6       9.0063       11.2152  0.0522
      7       6.8977        9.8086  0.0625
      8       5.5907        8.9925  0.0463
      9       4.6989        8.6901  0.0440
     10       4.1063        8.4049  0.0444
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1    1921.7735      323.4641  0.0396
      2     108.8984       69.4423  0.0447
      3      29.7367       38.0669  0.0491
      4      15.6169       24.6349  0.0432
      5       8.9476       18.2243  0.0424
      6       6.2993       15.1529  0.0433
      7       5.5928       13.9704  0.0471
      8       4.9390       13.1060  0.0448
      9       4.3441       12.7796  0.0418
     10       3.7354       12.4083  0.0430
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1    3506.2360      294.0791  0.0223
      2     558.7654      659.2878  0.0304
      3     403.4795       95.4148  0.0256
      4     140.9487       70.2990  0.0302
      5      73.1552       63.9128  0.0284
      6      51.1433       18.2029  0.0271
      7      26.0644       12.1681  0.0244
      8      13.6065        5.7683  0.0316
      9       8.7877        3.0087  0.0232
     10       6.4862        2.3670  0.0254
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1    3778.9754      531.1197  0.0232
      2     541.4282      488.6443  0.0348
      3     313.5680       99.6287  0.0249
      4      90.6746       48.6511  0.0302
      5      41.6889       39.4437  0.0254
      6      28.5540       21.4176  0.0336
```

```
      7       15.2826       18.1741  0.0236
      8        8.0454       14.9914  0.0274
      9        5.6964       13.1533  0.0268
     10        4.4346       12.2138  0.0252
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3882.3165      684.2929  0.0231
      2      592.5492      558.2817  0.0249
      3      336.3025      172.0253  0.0238
      4      113.8705       67.0325  0.0290
      5       61.5773       50.9974  0.0268
      6       35.2460       27.8074  0.0250
      7       25.1413       26.9430  0.0271
      8       16.9177       16.6023  0.0233
      9       11.5954       13.6860  0.0231
     10        9.8687       11.9955  0.0280
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3851.5377      497.9771  0.0234
      2      467.6639      439.9367  0.0273
      3      295.1580      176.9258  0.0218
      4      107.6230       60.0173  0.0263
      5       51.7453       61.1738  0.0242
      6       55.9242       35.6998  0.0239
      7       28.3875       24.6874  0.0262
      8       14.7962       14.3118  0.0245
      9       12.6136       12.8594  0.0235
     10       10.2453       10.6528  0.0242
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3656.4202      386.5737  0.0231
      2      516.7392      638.3899  0.0261
      3      372.0994      138.7597  0.0264
      4      115.8311      150.4698  0.0269
      5       63.9594      107.3982  0.0265
      6       47.9027       64.0192  0.0258
      7       27.7282       55.7750  0.0299
      8       17.4064       43.5440  0.0252
      9       12.6830       35.8313  0.0290
     10        9.2706       29.7060  0.0238
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3481.1734      116.9905  0.0246
      2      543.4884      547.2411  0.0260
      3      345.1512      200.7584  0.0255
      4      101.4775       55.3963  0.0255
      5       64.3363       30.1878  0.0247
      6       42.4823       22.3801  0.0225
      7       19.3996        8.8075  0.0251
      8       14.3788        6.6285  0.0249
```

```
      9       11.6482        6.0676  0.0254
     10        8.8122        5.0204  0.0262
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3716.5033      433.9304  0.0234
      2      499.9764      528.7114  0.0291
      3      308.9653      162.9934  0.0275
      4      101.4348       73.6572  0.0265
      5       59.4104       65.5974  0.0236
      6       38.1291       30.7789  0.0239
      7       19.5190       22.6092  0.0239
      8       12.7602       16.7641  0.0226
      9        8.9719       13.9005  0.0259
     10        6.6548       12.1139  0.0255
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3543.3484      324.9866  0.0221
      2      530.0028      567.6709  0.0270
      3      333.0289      197.5227  0.0239
      4      110.3399       77.6641  0.0248
      5       52.5205       59.8570  0.0252
      6       39.4364       37.2870  0.0256
      7       23.7328       24.2399  0.0257
      8       13.0053       17.7950  0.0229
      9       10.1186       15.4711  0.0252
     10        8.5745       14.3743  0.0237
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3583.8563      236.3608  0.0253
      2      433.2259      495.7988  0.0328
      3      282.6771      161.0473  0.0318
      4       92.5376       72.9409  0.0276
      5       47.7804       61.2484  0.0255
      6       46.9807       43.3607  0.0249
      7       24.2854       25.2820  0.0215
      8       13.6462       19.2365  0.0255
      9       11.3447       17.6536  0.0232
     10        9.0731       14.8911  0.0262
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3458.1576      228.9757  0.0213
      2      480.4555      535.3881  0.0283
      3      287.8531      178.2825  0.0248
      4      108.8366       93.8531  0.0247
      5       50.6201       84.4262  0.0234
      6       39.9080       61.4259  0.0257
      7       23.4889       40.1995  0.0269
      8       14.5813       32.2620  0.0233
      9       10.0880       26.0743  0.0250
     10        7.7583       21.4118  0.0264
```

```
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3374.7703      550.4833  0.0238
      2      571.5563      526.5821  0.0260
      3      328.7951       87.1162  0.0271
      4      101.4160       47.8733  0.0267
      5       53.4416       31.9566  0.0256
      6       36.2836       13.4639  0.0235
      7       22.5792       10.5721  0.0258
      8       15.0256        8.7299  0.0284
      9       10.9007        6.4088  0.0241
     10        8.8556        5.5126  0.0257
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3547.7942      761.6600  0.0261
      2      546.0533      446.1845  0.0251
      3      308.8992      118.6934  0.0255
      4      118.4302       98.8573  0.0351
      5       63.8256       66.1404  0.0359
      6       43.7100       49.5001  0.0363
      7       29.0732       33.4539  0.0478
      8       20.9472       25.2731  0.0436
      9       15.3560       20.7971  0.0382
     10       11.7474       17.4452  0.0382
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3535.8388      292.6359  0.0304
      2      468.0213      474.1462  0.0377
      3      273.4661      103.4579  0.0322
      4       88.7233       55.7093  0.0274
      5       41.1363       55.0094  0.0253
      6       31.1353       29.4380  0.0288
      7       22.9205       25.2408  0.0238
      8       15.3633       18.0848  0.0263
      9       10.4485       15.0685  0.0283
     10        7.9868       13.6714  0.0274
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3349.7987      324.1317  0.0238
      2      513.8303      515.6401  0.0313
      3      299.0583      113.5345  0.0296
      4       98.7615       53.5557  0.0263
      5       49.6178       57.8488  0.0289
      6       41.9977       29.4138  0.0280
      7       25.3105       30.1301  0.0277
      8       14.8370       18.8651  0.0278
      9       10.9623       17.6897  0.0262
     10        8.8501       15.2028  0.0288
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
```

```
      1     3439.8045      253.1377  0.0305
      2      506.0477      609.8434  0.0282
      3      304.9828      165.1947  0.0251
      4      103.2929       91.8092  0.0269
      5       45.7619       75.1014  0.0267
      6       34.7833       50.0557  0.0260
      7       20.9856       34.4845  0.0367
      8       12.6575       26.5388  0.0263
      9        8.3235       22.3256  0.0247
     10        6.1311       18.5170  0.0265
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3998.0178     1027.5667  0.0234
      2      504.5031      253.5637  0.0236
      3      260.0781      218.6765  0.0253
      4       89.4893       61.8448  0.0270
      5       51.3939       24.4385  0.0243
      6       31.1107       22.7043  0.0260
      7       17.7644       13.2743  0.0250
      8       12.7036        9.2356  0.0246
      9        9.8779        8.0952  0.0280
     10        8.0293        7.1666  0.0236
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3745.4895      533.2282  0.0219
      2      539.5866      557.3505  0.0271
      3      346.4104      200.4063  0.0260
      4      108.3475      101.3351  0.0252
      5       68.7033       73.4681  0.0293
      6       45.7692       39.7068  0.0278
      7       23.5745       27.1249  0.0247
      8       15.8132       20.1250  0.0251
      9       11.7976       14.4117  0.0271
     10        8.8617       11.4612  0.0256
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3319.2238      462.9140  0.0212
      2      563.8268      579.1882  0.0244
      3      343.4895      116.4558  0.0239
      4      146.9430       84.0531  0.0280
      5       66.1144       76.4370  0.0249
      6       41.9305       42.4838  0.0262
      7       29.3257       32.1946  0.0238
      8       19.4198       25.1792  0.0267
      9       12.8354       19.5267  0.0292
     10        9.8448       17.2950  0.0263
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     4023.9145     1038.8955  0.0216
      2      564.0308      454.2438  0.0324
```

```
      3      459.6233      180.7246  0.0251
      4      123.0353      126.9449  0.0249
      5       83.1682       44.3945  0.0249
      6       54.9749       47.4398  0.0235
      7       30.3287       36.2275  0.0260
      8       21.4963       25.5168  0.0238
      9       19.2839       24.5674  0.0249
     10       13.2834       19.3253  0.0257
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     3779.8765      577.4721  0.0256
      2      534.4459      582.2245  0.0235
      3      386.7886      200.9842  0.0303
      4      119.2366      127.8596  0.0252
      5       58.9527       76.4602  0.0236
      6       40.0322       61.2021  0.0258
      7       23.7677       42.1911  0.0233
      8       12.9143       34.0573  0.0289
      9       10.0629       25.7101  0.0247
     10        7.4342       21.4792  0.0248
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     1804.5847      180.7729  0.0526
      2       89.1968       64.4864  0.0536
      3       32.4559       25.4212  0.0517
      4       23.4901       19.7270  0.0526
      5       16.0328       16.9739  0.0514
      6        9.3418       12.3237  0.0537
      7        5.1277        8.7428  0.0559
      8        3.5226        6.6390  0.0556
      9        2.6410        5.8650  0.0609
     10        2.0805        5.4852  0.0614
Re-initializing module because the following parameters were re-set:
hidden_sizes.
Re-initializing criterion.
Re-initializing optimizer.
  epoch    train_loss    valid_loss     dur
-------  ------------  ------------  ------
      1     1772.5155      185.6480  0.0549
      2       98.3659       59.7871  0.0568
      3       39.2578       25.5065  0.0551
      4       23.6418       18.1683  0.0549
      5       15.0928       15.5610  0.0533
      6       10.0698       13.0344  0.0547
      7        6.3694        9.6874  0.0540
      8        4.1893        7.1868  0.0713
      9        3.2654        5.8785  0.0812
     10        2.7195        5.1721  0.0760
Best Parameters: {'batch_size': 8, 'module__hidden_sizes': (128, 32,
64)}
```

```
Best MSE: 4.190647125244141
Best RMSE: 2.0471070136278025
```

Unfortunately, the `grid search` algorithm threw up worse parameters than my trained model - this is most likely due to an insufficient dataset.

```python
from sklearn.metrics import r2_score

# Predictions for test data using the best model from Grid Search
y_pred = best_model.predict(X_test)

# Calculation of the coefficient of determination (R^2)
r2 = r2_score(y_test, y_pred)

print("R^2 score:", r2)
```

```
R^2 score: 0.47441723158499405
```

R^2 also worse (previously it was over 0.97).

```python
X_train.shape
```

```
torch.Size([299, 8])
```

```python
y_train.shape
```

```
torch.Size([299, 1])
```

Summary

Best parameters:

`batch_size` -the number of samples (data records) used in one iteration while training the model should be 8. `module__hidden_sizes` - the neural network will have three hidden layers with 128, 64 and 32 neurons, respectively. The `MSE`, the root mean square error, which is a measure of the mean square of the difference between the predicted values of the model and the actual values, is `0.45` in our case. The `RMSE` (Root Mean Squared Error) is the square root of the MSE and is used to express the error in the same unit as the predicted values and in our case would be `0.67`.

Having such a correctly trained model, we can use it on other data. It should correctly identify our variables and predict well the value of the `Heart Rate` variable.

Random Forest Algorithm

We will now use the Random Forest algorithm to train our model to classify people well by their sleep disorders.

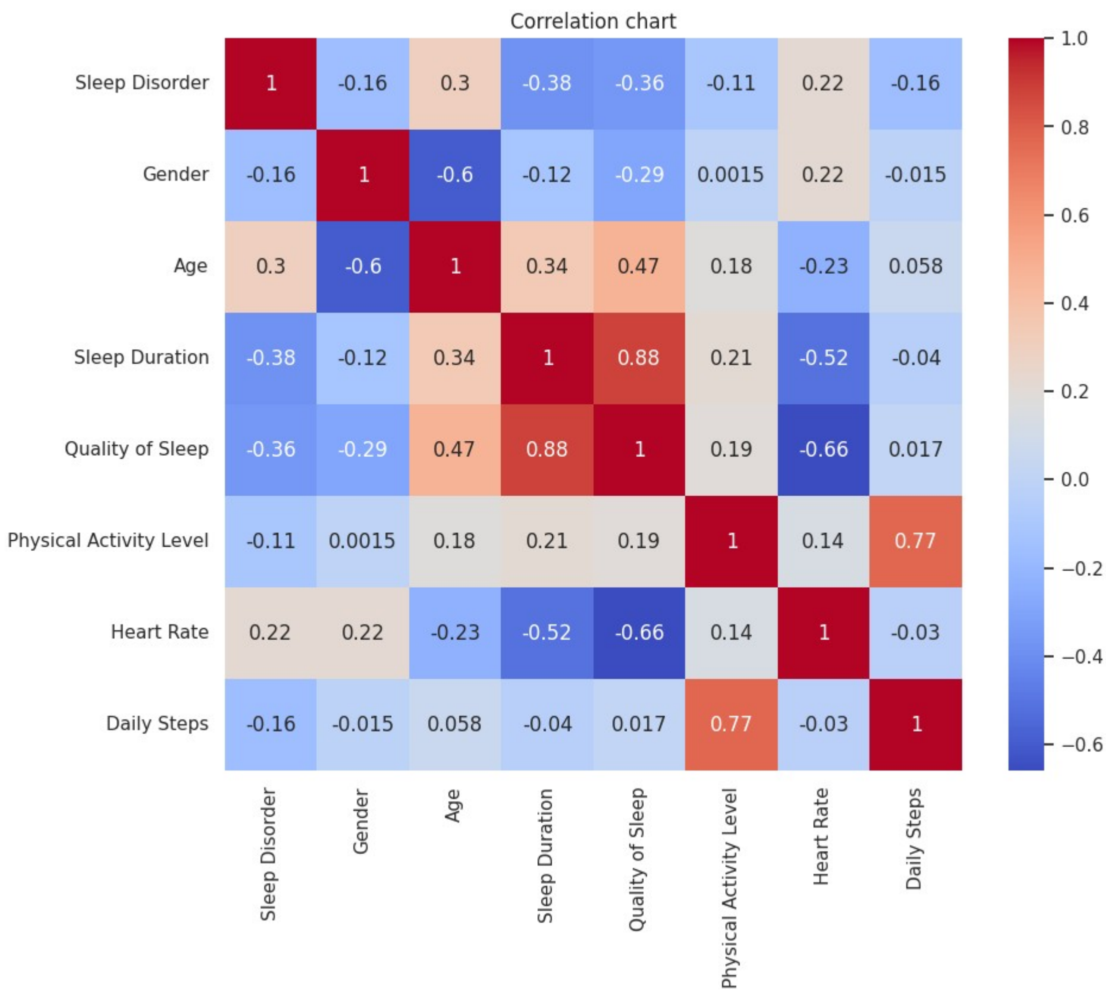
```python
#Assign new variable with selected columns for binary modeling.
data3 = data[['Sleep Disorder','Gender','Age','Sleep
```

```
Duration','Quality of Sleep','Physical Activity Level','Heart
Rate','Daily Steps']]
```

```
data3.head()
```

```
   Sleep Disorder  Gender  Age  Sleep Duration  Quality of Sleep  \
0               0       1   27             6.1                 6
1               0       1   28             6.2                 6
2               0       1   28             6.2                 6
3               1       1   28             5.9                 4
4               1       1   28             5.9                 4

   Physical Activity Level  Heart Rate  Daily Steps
0                       42          77         4200
1                       60          75        10000
2                       60          75        10000
3                       30          85         3000
4                       30          85         3000
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Generate a correlation matrix
correlation_matrix = data3.corr()

# Correlation chart
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation chart")
plt.show()
```




```
data3['Sleep Disorder'].unique()
```

```
array([0, 1, 2])
```

```
#Creation of a new data frame without Heart Rate
X = data3.drop('Sleep Disorder', axis = 1)
```

```
#Convert the data from the 'Heart Rate' column to a numpy array and
check the dimensions
y = data3['Sleep Disorder'].to_numpy()
y.shape
```

```
(374,)
```

```
#Convert data from data without MonthlyIncome to numpy array and check
dimensions
X = X.to_numpy()
X.shape
```

```
(374, 7)
```

```
# We are splitting the data into a training set and a test set
X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y,
train_size=0.8, shuffle=True)

# We standardize the data using the StandardScaler class
scaler = StandardScaler()
scaler.fit(X_train_raw)
X_train = scaler.transform(X_train_raw)
X_test = scaler.transform(X_test_raw)

# We are converting the data to 2D PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).reshape(-1, 1)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32).reshape(-1, 1)

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV

# Convert y_train to a one-dimensional array
y_train = np.ravel(y_train)

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV

# Definition of the set of hyperparameters to be searched
param_grid = {
    'n_estimators': [100, 200, 300], # Number of trees in the forest
    'max_depth': [None, 5, 10], # Maximum depth of the tree
    'min_samples_split': [2, 5, 10], # Minimum number of samples
required to split a node
    'min_samples_leaf': [1, 2, 4], # Minimum number of samples
required to create a leaf
    'max_features': ['sqrt'] # Number of features considered for
splitting
}

# Initialization of Random Forest model
model = RandomForestRegressor(random_state=42)

# Searching the grid of hyperparameters to find the best parameters
grid_search = GridSearchCV(model, param_grid,
scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train, y_train)

# best parameters found
best_params = grid_search.best_params_
```

```python
# Initialization of model with optimal parameters
model = RandomForestRegressor(**best_params, random_state=42)

# Training the model on the training data
model.fit(X_train, y_train)

# Prediction on test data
y_pred = model.predict(X_test)

# Calculation of the mean square error (MSE)
mse = mean_squared_error(y_test, y_pred)

print("Best parameters: ", best_params)
print("MSE: %.2f" % mse)
print("RMSE: %.2f" % np.sqrt(mse))
```

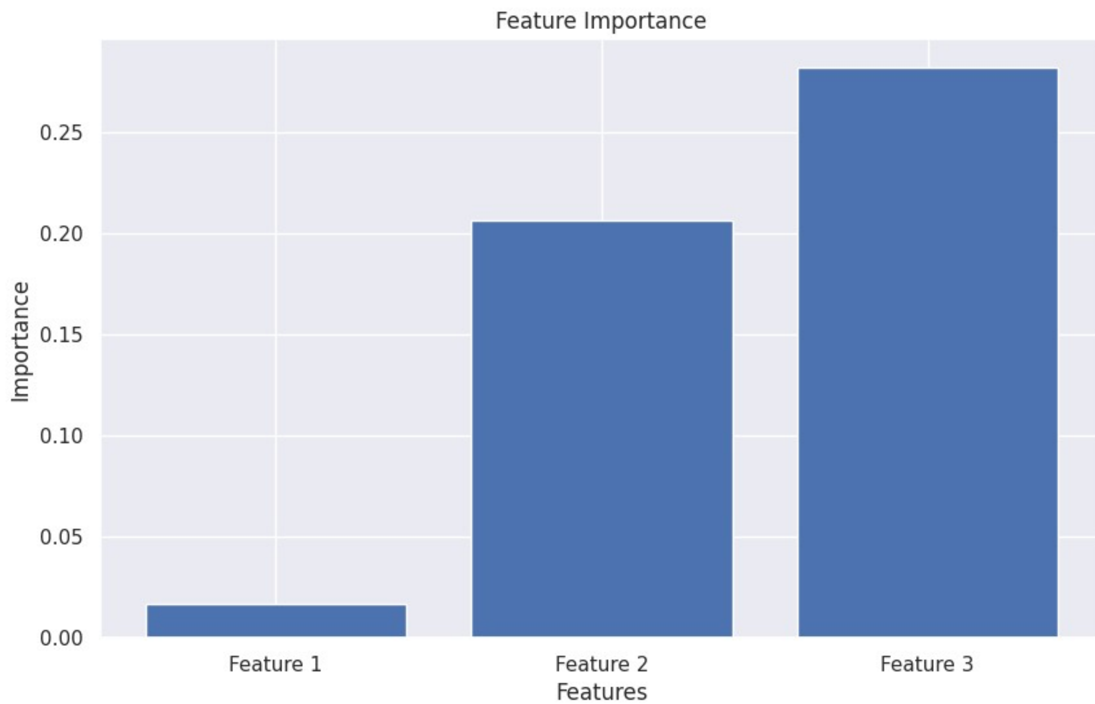
```
Best parameters:  {'max_depth': 5, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
MSE: 0.34
RMSE: 0.58
```

```python
from sklearn.metrics import r2_score, mean_absolute_error,
explained_variance_score

# R2 Score
r2 = r2_score(y_test, y_pred)

# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)

# Explained Variance Score
explained_variance = explained_variance_score(y_test, y_pred)

print("R2 Score: %.2f" % r2)
print("MAE: %.2f" % mae)
print("Explained Variance: %.2f" % explained_variance)
```

```
R2 Score: 0.48
MAE: 0.29
Explained Variance: 0.48
```

```python
import matplotlib.pyplot as plt
import numpy as np

# Pobranie ważności cech
importances = model.feature_importances_

# Utworzenie listy indeksów cech
feature_indices = np.arange(len(importances))
```

```python
# Wykres słupkowy ważności cech
plt.figure(figsize=(10, 6))
plt.bar(feature_indices[:3], importances[:3])
plt.xlabel('Features')
plt.ylabel('Importance')
plt.xticks(feature_indices[:3], ['Feature 1', 'Feature 2', 'Feature
3'])
plt.title('Feature Importance')
plt.show()
```




```python
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score

# Convert predicted probabilities to binary predictions
y_pred_binary = (y_pred > 0.5).astype(int)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_binary)

# Compute classification report
report = classification_report(y_test, y_pred_binary, zero_division=0)

# Compute accuracy
accuracy = accuracy_score(y_test, y_pred_binary)

print("Confusion Matrix:")
print(cm)
```

```python
print("\nClassification Report:")
print(report)
print("\nAccuracy: %.2f" % accuracy)
```

```
Confusion Matrix:
[[35  6  0]
 [ 2 16  0]
 [ 2 14  0]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      0.85      0.88        41
         1.0       0.44      0.89      0.59        18
         2.0       0.00      0.00      0.00        16

    accuracy                           0.68        75
   macro avg       0.45      0.58      0.49        75
weighted avg       0.60      0.68      0.62        75


Accuracy: 0.68
```

```python
model.score(X_test, y_test)
```

```
0.475514981529257
```

```python
data3['Sleep Disorder'].value_counts()
```

```
0    219
1     78
2     77
Name: Sleep Disorder, dtype: int64
```

We can suspect that, our model is able to predict correctly only class "0" due to the disparity between classes. As we can see when counting our labels, class "0" is dominant.

Oversampling using SMOTE

```python
#pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\pc\
anaconda3\lib\site-packages (0.10.1)
Requirement already satisfied: scipy>=1.3.2 in c:\users\pc\anaconda3\
lib\site-packages (from imbalanced-learn) (1.6.2)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\pc\
anaconda3\lib\site-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\pc\anaconda3\
lib\site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\pc\
anaconda3\lib\site-packages (from imbalanced-learn) (2.1.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\pc\anaconda3\
```

```
lib\site-packages (from imbalanced-learn) (1.20.1)
Note: you may need to restart the kernel to use updated packages.
```

```python
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Data preparation
data3 = data[['Sleep Disorder', 'Gender', 'Age', 'Sleep Duration',
'Quality of Sleep', 'Physical Activity Level', 'Heart Rate', 'Daily
Steps']]
X = data3.drop('Sleep Disorder', axis=1)
y = data3['Sleep Disorder']

# Division into training and test collection
X_train_raw, X_test_raw, y_train, y_test = train_test_split(X, y,
train_size=0.75, random_state=42)

# Data standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train_raw)
X_test = scaler.transform(X_test_raw)

# Oversampling using SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)

# Initialization of RandomForestClassifier model with adjusted class
weights
class_weights = dict(zip(range(3), ((y_train_resampled == 0).sum(),
(y_train_resampled == 1).sum(), (y_train_resampled == 2).sum())))
model = RandomForestClassifier(class_weight=class_weights,
random_state=42)

# Training the model on oversampled data
model.fit(X_train_resampled, y_train_resampled)

# Prediction on the test set
y_pred = model.predict(X_test)

# Classification report
print(classification_report(y_test, y_pred))
```

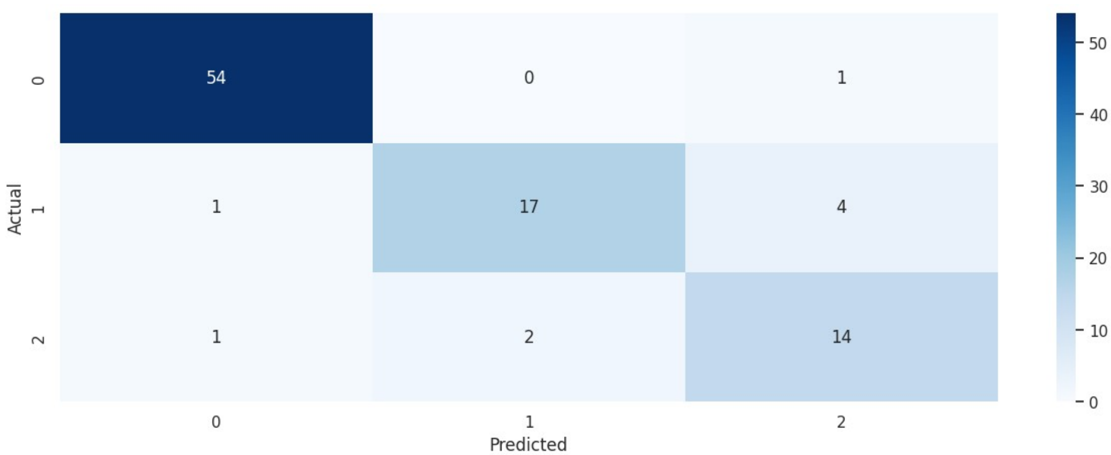
```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        55
           1       0.89      0.77      0.83        22
```

```
           2       0.74      0.82      0.78        17

    accuracy                           0.90        94
   macro avg       0.87      0.86      0.86        94
weighted avg       0.91      0.90      0.90        94
```

```
model.score(X_test, y_test)
```

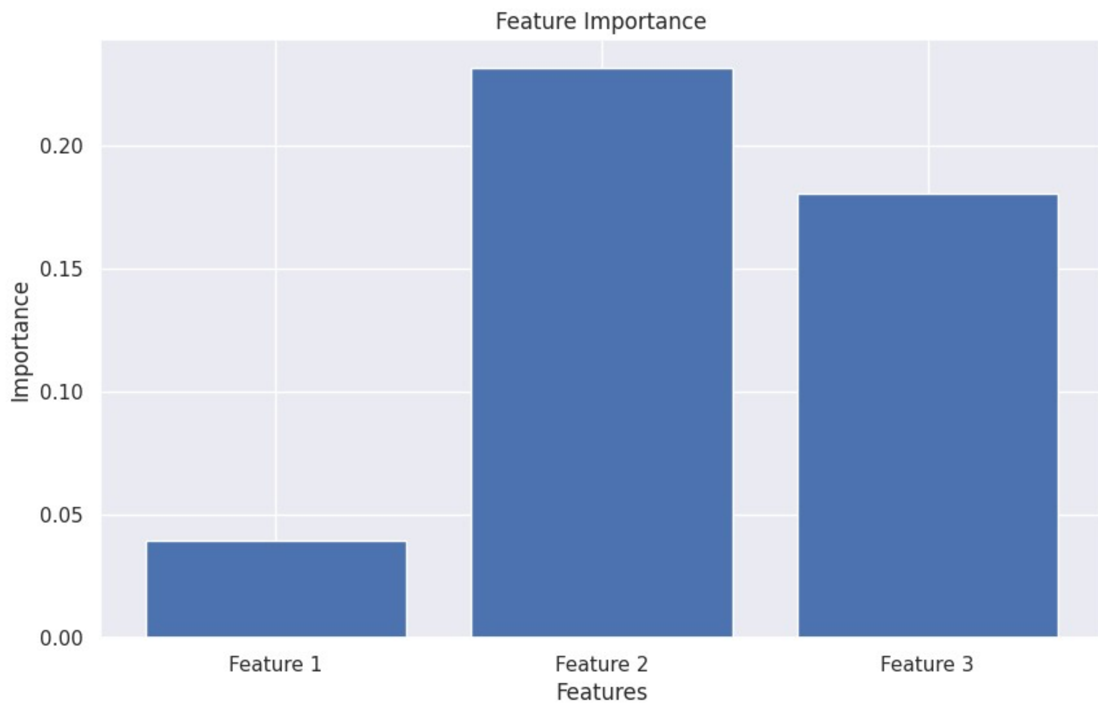
```
0.9042553191489362
```

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Prediction on the test set
y_pred = model.predict(X_test)

# Generate a confusion matrix
confusion_mtx = confusion_matrix(y_test, y_pred)

# Display the confusion matrix using Seaborn
sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

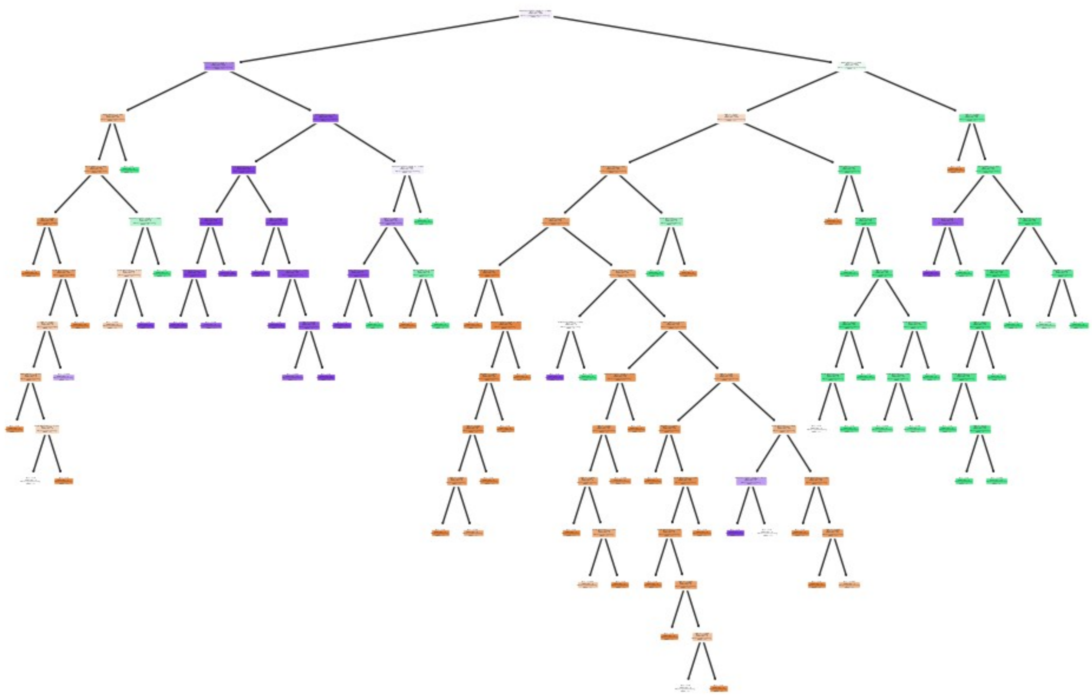



```
import matplotlib.pyplot as plt
import numpy as np

# Downloading the validity of features
importances = model.feature_importances_

# Create a list of feature indexes
feature_indices = np.arange(len(importances))

# Bar chart of the importance of features
```

```
plt.figure(figsize=(10, 6))
plt.bar(feature_indices[:3], importances[:3])
plt.xlabel('Features')
plt.ylabel('Importance')
plt.xticks(feature_indices[:3], ['Feature 1', 'Feature 2', 'Feature
3'])
plt.title('Feature Importance')
plt.show()
```




```
# Visualization of a the first tree from the Random Forest algorithm

from sklearn.tree import plot_tree

#Select the number of index
tree_index = 0

# Get the selected tree from the model
tree = model.estimators_[tree_index]

# Visualize the tree
plt.figure(figsize=(12, 8))
plot_tree(tree, feature_names=X.columns, class_names=[str(c) for c in
model.classes_], filled=True)
plt.show()
```



```
tree_values = model.estimators_[0].tree_
node_values = tree_values.value
print(node_values)
```

```
[[[25748. 25092. 29848.]]

 [[ 6560.  3608. 26732.]]

 [[ 5904.  1804.   820.]]

 [[ 5904.   656.   820.]]

 [[ 5576.     0.   656.]]

 [[ 1968.     0.     0.]]

 [[ 3608.     0.   656.]]

 [[ 1148.     0.   656.]]

 [[  984.     0.   328.]]

 [[  492.     0.     0.]]

 [[  492.     0.   328.]]

 [[  328.     0.   328.]]
```

```
[[  164.     0.     0.]]
[[  164.     0.   328.]]
[[ 2460.     0.     0.]]
[[  328.   656.   164.]]
[[  328.   164.   164.]]
[[  328.   164.     0.]]
[[    0.     0.   164.]]
[[    0.   492.     0.]]
[[    0.  1148.     0.]]
[[  656.  1804. 25912.]]
[[  492.     0. 23944.]]
[[  164.     0. 16728.]]
[[  164.     0.  7872.]]
[[    0.     0.  6396.]]
[[  164.     0.  1476.]]
[[    0.     0.  8856.]]
[[  328.     0.  7216.]]
[[    0.     0.  3772.]]
[[  328.     0.  3444.]]
[[    0.     0.   656.]]
[[  328.     0.  2788.]]
[[  328.     0.  2132.]]
[[    0.     0.   656.]]
[[  164.  1804.  1968.]]
```

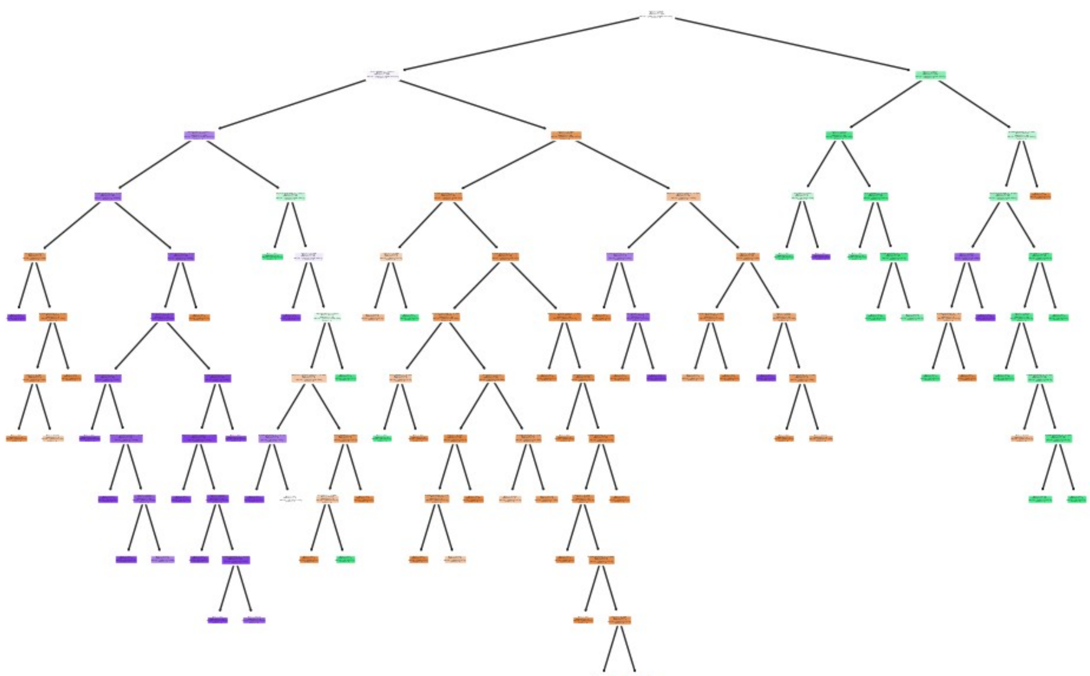
```
[[  164.   656.  1968.]]
[[    0.   164.  1968.]]
[[    0.     0.  1968.]]
[[    0.   164.     0.]]
[[  164.   492.     0.]]
[[  164.     0.     0.]]
[[    0.   492.     0.]]
[[    0.  1148.     0.]]
[[19188. 21484.  3116.]]
[[17712. 11480.  2132.]]
[[16564.  1804.  2132.]]
[[16236.  1148.  2132.]]
[[ 6232.   164.     0.]]
[[ 2460.     0.     0.]]
[[ 3772.   164.     0.]]
[[ 2788.   164.     0.]]
[[ 2132.   164.     0.]]
[[  820.   164.     0.]]
[[  164.     0.     0.]]
[[  656.   164.     0.]]
[[ 1312.     0.     0.]]
[[  656.     0.     0.]]
[[  984.     0.     0.]]
[[10004.   984.  2132.]]
```

```
[[    0.   492.   492.]]

[[    0.     0.   492.]]

[[    0.   492.     0.]]

[[10004.   492.  1640.]]

[[ 2624.   164.   164.]]

[[ 2460.   164.   164.]]

[[  656.   164.     0.]]

[[  164.     0.     0.]]

[[  492.   164.     0.]]

[[  328.   164.     0.]]

[[  164.     0.     0.]]

[[ 1804.     0.   164.]]

[[  164.     0.     0.]]

[[ 7380.   328.  1476.]]

[[ 3608.   328.     0.]]

[[  328.     0.     0.]]

[[ 3280.   328.     0.]]

[[ 2132.   328.     0.]]

[[  656.     0.     0.]]

[[ 1476.   328.     0.]]

[[  820.     0.     0.]]

[[  656.   328.     0.]]

[[  328.   328.     0.]]

[[  328.     0.     0.]]
```

```
[[ 1148.     0.     0.]]
[[ 3772.     0.  1476.]]
[[  492.     0.   984.]]
[[    0.     0.   492.]]
[[  492.     0.   492.]]
[[ 3280.     0.   492.]]
[[  984.     0.     0.]]
[[ 2296.     0.   492.]]
[[  984.     0.     0.]]
[[ 1312.     0.   492.]]
[[  328.   656.     0.]]
[[    0.   656.     0.]]
[[  328.     0.     0.]]
[[ 1148.  9676.     0.]]
[[  492.     0.     0.]]
[[  656.  9676.     0.]]
[[    0.  1476.     0.]]
[[  656.  8200.     0.]]
[[  164.  4756.     0.]]
[[  164.  1312.     0.]]
[[  164.   164.     0.]]
[[    0.  1148.     0.]]
[[    0.  3444.     0.]]
[[  492.  3444.     0.]]
```

```
[[  492.  2952.     0.]]

[[  164.  1148.     0.]]

[[  328.  1804.     0.]]

[[    0.   492.     0.]]

[[ 1476. 10004.   984.]]

[[  984.     0.     0.]]

[[  492. 10004.   984.]]

[[    0.   164.   984.]]

[[    0.     0.   984.]]

[[    0.   164.     0.]]

[[  492.  9840.     0.]]

[[  164.  7380.     0.]]

[[  164.  3444.     0.]]

[[  164.  2296.     0.]]

[[    0.   328.     0.]]

[[  164.  1968.     0.]]

[[    0.   164.     0.]]

[[  164.  1804.     0.]]

[[    0.  1148.     0.]]

[[    0.  3936.     0.]]

[[  328.  2460.     0.]]

[[  328.   820.     0.]]

[[    0.  1640.     0.]]]
```

```python
# Visualization of a the second tree from the Random Forest algorithm

from sklearn.tree import plot_tree

#Select the number of index
tree_index = 1

# Get the selected tree from the model
tree = model.estimators_[tree_index]

# Visualize the tree
plt.figure(figsize=(12, 8))
plot_tree(tree, feature_names=X.columns, class_names=[str(c) for c in
model.classes_], filled=True)
plt.show()
```



```python
tree_values = model.estimators_[1].tree_
node_values = tree_values.value
print(node_values)
```

```
[[[28044. 24600. 28044.]]

 [[22796.  4428. 24436.]]

 [[ 5904.  3280. 22632.]]

 [[ 4756.     0. 20828.]]
```

```
[[ 2296.     0.   328.]]
[[    0.     0.   164.]]
[[ 2296.     0.   164.]]
[[ 1476.     0.   164.]]
[[ 1148.     0.     0.]]
[[  328.     0.   164.]]
[[  820.     0.     0.]]
[[ 2460.     0. 20500.]]
[[  820.     0. 20500.]]
[[  656.     0.  7216.]]
[[    0.     0.  2788.]]
[[  656.     0.  4428.]]
[[    0.     0.   164.]]
[[  656.     0.  4264.]]
[[    0.     0.  2296.]]
[[  656.     0.  1968.]]
[[  164.     0. 13284.]]
[[  164.     0.  7216.]]
[[    0.     0.   492.]]
[[  164.     0.  6724.]]
[[    0.     0.   984.]]
[[  164.     0.  5740.]]
[[    0.     0.  4428.]]
[[  164.     0.  1312.]]
```

```
[[    0.     0.  6068.]]
[[ 1640.     0.     0.]]
[[ 1148.  3280.  1804.]]
[[    0.  1804.     0.]]
[[ 1148.  1476.  1804.]]
[[    0.     0.  1312.]]
[[ 1148.  1476.   492.]]
[[ 1148.   328.   492.]]
[[    0.   164.   492.]]
[[    0.     0.   328.]]
[[    0.   164.   164.]]
[[ 1148.   164.     0.]]
[[  328.   164.     0.]]
[[  328.     0.     0.]]
[[    0.   164.     0.]]
[[  820.     0.     0.]]
[[    0.  1148.     0.]]
[[16892.  1148.  1804.]]
[[13284.  1148.   164.]]
[[  820.   492.     0.]]
[[  820.   328.     0.]]
[[    0.   164.     0.]]
[[12464.   656.   164.]]
[[ 5576.   492.   164.]]
```

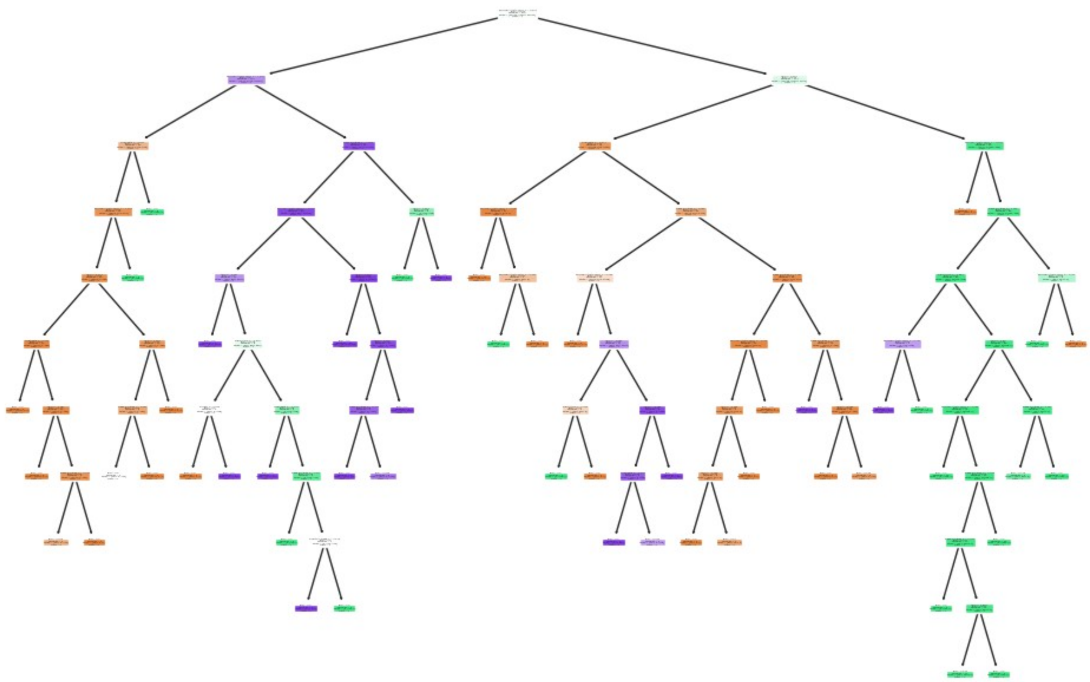
```
[[  328.   164.     0.]]

[[    0.   164.     0.]]

[[  328.     0.     0.]]

[[ 5248.   328.   164.]]

[[ 3772.   164.     0.]]

[[  820.   164.     0.]]

[[  492.     0.     0.]]

[[  328.   164.     0.]]

[[ 2952.     0.     0.]]

[[ 1476.   164.   164.]]

[[  328.   164.     0.]]

[[ 1148.     0.   164.]]

[[ 6888.   164.     0.]]

[[ 2460.     0.     0.]]

[[ 4428.   164.     0.]]

[[  492.     0.     0.]]

[[ 3936.   164.     0.]]

[[ 2624.   164.     0.]]

[[  492.     0.     0.]]

[[ 2132.   164.     0.]]

[[  656.     0.     0.]]

[[ 1476.   164.     0.]]

[[ 1312.   164.     0.]]

[[  164.     0.     0.]]
```

```
[[ 1312.     0.     0.]]
[[ 3608.     0.  1640.]]
[[  328.     0.   984.]]
[[  164.     0.     0.]]
[[  164.     0.   984.]]
[[  164.     0.     0.]]
[[    0.     0.   984.]]
[[ 3280.     0.   656.]]
[[ 1476.     0.   164.]]
[[  656.     0.   164.]]
[[  820.     0.     0.]]
[[ 1804.     0.   492.]]
[[    0.     0.   164.]]
[[ 1804.     0.   328.]]
[[  164.     0.     0.]]
[[ 1640.     0.   328.]]
[[ 5248. 20172.  3608.]]
[[  328. 11152.   328.]]
[[    0.   492.   328.]]
[[    0.   492.     0.]]
[[    0.     0.   328.]]
[[  328. 10660.     0.]]
[[    0.   984.     0.]]
[[  328.  9676.     0.]]
```

```
 [[    0.  8528.     0.]]

 [[  328.  1148.     0.]]

 [[ 4920.  9020.  3280.]]

 [[ 1148.  9020.  3280.]]

 [[  492.   164.  3280.]]

 [[  492.   164.     0.]]

 [[    0.   164.     0.]]

 [[  492.     0.     0.]]

 [[    0.     0.  3280.]]

 [[  656.  8856.     0.]]

 [[  656.  8692.     0.]]

 [[    0.  5904.     0.]]

 [[  656.  2788.     0.]]

 [[  492.   164.     0.]]

 [[  164.  2624.     0.]]

 [[  164.  2132.     0.]]

 [[    0.   492.     0.]]

 [[    0.   164.     0.]]

 [[ 3772.     0.     0.]]]
```

```python
# Visualization of a the second tree from the Random Forest algorithm

from sklearn.tree import plot_tree

#Select the number of index
tree_index = 2

# Get the selected tree from the model
tree = model.estimators_[tree_index]
```

```python
# Visualize the tree
plt.figure(figsize=(12, 8))
plot_tree(tree, feature_names=X.columns, class_names=[str(c) for c in
model.classes_], filled=True)
plt.show()
```



```python
tree_values = model.estimators_[2].tree_
node_values = tree_values.value
print(node_values)
```

```
[[[26732. 28372. 25584.]]

 [[ 7544.  4100. 22796.]]

 [[ 6888.  2460.   656.]]

 [[ 6888.   328.   656.]]

 [[ 6888.     0.   656.]]

 [[ 3772.     0.   164.]]

 [[ 1312.     0.     0.]]

 [[ 2460.     0.   164.]]
```

```
[[ 1312.     0.     0.]]

[[ 1148.     0.   164.]]

[[  492.     0.   164.]]

[[  656.     0.     0.]]

[[ 3116.     0.   492.]]

[[ 1640.     0.   492.]]

[[  492.     0.   492.]]

[[ 1148.     0.     0.]]

[[ 1476.     0.     0.]]

[[    0.   328.     0.]]

[[    0.  2132.     0.]]

[[  656.  1640. 22140.]]

[[  656.   820. 21812.]]

[[  328.   820.  2296.]]

[[    0.     0.  1640.]]

[[  328.   820.   656.]]

[[  328.     0.   328.]]

[[  328.     0.     0.]]

[[    0.     0.   328.]]

[[    0.   820.   328.]]

[[    0.     0.   164.]]

[[    0.   820.   164.]]

[[    0.   656.     0.]]

[[    0.   164.   164.]]
```

```
[[    0.     0.   164.]]
[[    0.   164.     0.]]
[[  328.     0. 19516.]]
[[    0.     0. 15416.]]
[[  328.     0.  4100.]]
[[  328.     0.  1640.]]
[[    0.     0.   492.]]
[[  328.     0.  1148.]]
[[    0.     0.  2460.]]
[[    0.   820.   328.]]
[[    0.   820.     0.]]
[[    0.     0.   328.]]
[[19188. 24272.  2788.]]
[[17548.   820.  2460.]]
[[ 8200.   328.     0.]]
[[ 7544.     0.     0.]]
[[  656.   328.     0.]]
[[    0.   328.     0.]]
[[  656.     0.     0.]]
[[ 9348.   492.  2460.]]
[[ 2788.   328.  1968.]]
[[ 2132.     0.     0.]]
[[  656.   328.  1968.]]
[[  492.   328.     0.]]
```

```
[[    0.   328.     0.]]

[[  492.     0.     0.]]

[[  164.     0.  1968.]]

[[  164.     0.   984.]]

[[    0.     0.   656.]]

[[  164.     0.   328.]]

[[    0.     0.   984.]]

[[ 6560.   164.   492.]]

[[ 3280.   164.     0.]]

[[ 1148.   164.     0.]]

[[  820.   164.     0.]]

[[  164.     0.     0.]]

[[  656.   164.     0.]]

[[  328.     0.     0.]]

[[ 2132.     0.     0.]]

[[ 3280.     0.   492.]]

[[    0.     0.   328.]]

[[ 3280.     0.   164.]]

[[ 2296.     0.     0.]]

[[  984.     0.   164.]]

[[ 1640. 23452.   328.]]

[[  328.     0.     0.]]

[[ 1312. 23452.   328.]]

[[  328. 21812.   328.]]
```

```
[[    0.   164.   328.]]

[[    0.     0.   328.]]

[[    0.   164.     0.]]

[[  328. 21648.     0.]]

[[  164. 18696.     0.]]

[[    0.  7708.     0.]]

[[  164. 10988.     0.]]

[[  164.  4592.     0.]]

[[    0.   984.     0.]]

[[  164.  3608.     0.]]

[[  164.  2624.     0.]]

[[    0.   984.     0.]]

[[    0.  6396.     0.]]

[[  164.  2952.     0.]]

[[  164.   820.     0.]]

[[    0.  2132.     0.]]

[[  984.  1640.     0.]]

[[    0.  1640.     0.]]

[[  984.     0.     0.]]]
```

Summary

In summary, we ran the Random Forest algorithm to train our model, which is correctly able to classify people by their sleep disorders.

Compiled by: Kwiek Kamil