ML-Project

September 22, 2024

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore') #to avoid the warning messages
```

[2]:
```
import pandas as pd
sleep_dataset = r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv"
df = pd.read_csv(sleep_dataset)
print(df.head())
```

```
   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

   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

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

[3]:
```
print(df.tail())
```

```
     Person ID  Gender  Age Occupation  Sleep Duration  Quality of Sleep  \
369        370  Female   59      Nurse             8.1                 9
370        371  Female   59      Nurse             8.0                 9
```

```
371        372  Female   59      Nurse             8.1                 9
372        373  Female   59      Nurse             8.1                 9
373        374  Female   59      Nurse             8.1                 9

     Physical Activity Level  Stress Level BMI Category Blood Pressure  \
369                       75             3   Overweight         140/95
370                       75             3   Overweight         140/95
371                       75             3   Overweight         140/95
372                       75             3   Overweight         140/95
373                       75             3   Overweight         140/95

     Heart Rate  Daily Steps Sleep Disorder
369          68         7000    Sleep Apnea
370          68         7000    Sleep Apnea
371          68         7000    Sleep Apnea
372          68         7000    Sleep Apnea
373          68         7000    Sleep Apnea
```

[4]:
```
print(df.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           155 non-null    object
dtypes: float64(1), int64(7), object(5)
memory usage: 38.1+ KB
None
```

[5]:
```
shape = df.shape
print(shape)
```

```
(374, 13)
```

[6]:
```
print(df.describe())
```

	Person ID	Age	Sleep Duration	Quality of Sleep	\
count	374.000000	374.000000	374.000000	374.000000	
mean	187.500000	42.184492	7.132086	7.312834	
std	108.108742	8.673133	0.795657	1.196956	
min	1.000000	27.000000	5.800000	4.000000	
25%	94.250000	35.250000	6.400000	6.000000	
50%	187.500000	43.000000	7.200000	7.000000	
75%	280.750000	50.000000	7.800000	8.000000	
max	374.000000	59.000000	8.500000	9.000000	

	Physical Activity Level	Stress Level	Heart Rate	Daily Steps
count	374.000000	374.000000	374.000000	374.000000
mean	59.171123	5.385027	70.165775	6816.844920
std	20.830804	1.774526	4.135676	1617.915679
min	30.000000	3.000000	65.000000	3000.000000
25%	45.000000	4.000000	68.000000	5600.000000
50%	60.000000	5.000000	70.000000	7000.000000
75%	75.000000	7.000000	72.000000	8000.000000
max	90.000000	8.000000	86.000000	10000.000000

[7]:
```
df.describe(include ='all').T
```

[7]:

	count	unique	top	freq	mean	\
Person ID	374.0	NaN	NaN	NaN	187.5	
Gender	374	2	Male	189	NaN	
Age	374.0	NaN	NaN	NaN	42.184492	
Occupation	374	11	Nurse	73	NaN	
Sleep Duration	374.0	NaN	NaN	NaN	7.132086	
Quality of Sleep	374.0	NaN	NaN	NaN	7.312834	
Physical Activity Level	374.0	NaN	NaN	NaN	59.171123	
Stress Level	374.0	NaN	NaN	NaN	5.385027	
BMI Category	374	4	Normal	195	NaN	
Blood Pressure	374	25	130/85	99	NaN	
Heart Rate	374.0	NaN	NaN	NaN	70.165775	
Daily Steps	374.0	NaN	NaN	NaN	6816.84492	
Sleep Disorder	155	2	Sleep Apnea	78	NaN	

	std	min	25%	50%	75%	max
Person ID	108.108742	1.0	94.25	187.5	280.75	374.0
Gender	NaN	NaN	NaN	NaN	NaN	NaN
Age	8.673133	27.0	35.25	43.0	50.0	59.0
Occupation	NaN	NaN	NaN	NaN	NaN	NaN
Sleep Duration	0.795657	5.8	6.4	7.2	7.8	8.5
Quality of Sleep	1.196956	4.0	6.0	7.0	8.0	9.0
Physical Activity Level	20.830804	30.0	45.0	60.0	75.0	90.0
Stress Level	1.774526	3.0	4.0	5.0	7.0	8.0
BMI Category	NaN	NaN	NaN	NaN	NaN	NaN

```
Blood Pressure                  NaN     NaN     NaN     NaN     NaN      NaN
Heart Rate                 4.135676    65.0    68.0    70.0    72.0     86.0
Daily Steps             1617.915679  3000.0  5600.0  7000.0  8000.0  10000.0
Sleep Disorder                  NaN     NaN     NaN     NaN     NaN      NaN
```

[8]:
```
df.isnull().any()
```

[8]:
```
Person ID                 False
Gender                    False
Age                       False
Occupation                False
Sleep Duration            False
Quality of Sleep          False
Physical Activity Level   False
Stress Level              False
BMI Category              False
Blood Pressure            False
Heart Rate                False
Daily Steps               False
Sleep Disorder             True
dtype: bool
```

[9]:
```
df.isnull().sum()
```

[9]:
```
Person ID                   0
Gender                      0
Age                         0
Occupation                  0
Sleep Duration              0
Quality of Sleep            0
Physical Activity Level     0
Stress Level                0
BMI Category                0
Blood Pressure              0
Heart Rate                  0
Daily Steps                 0
Sleep Disorder            219
dtype: int64
```

[10]:
```
df['Sleep Disorder'].fillna('None', inplace=True)
```

[11]:
```
df.isnull().sum()
```

[11]:
```
Person ID                 0
Gender                    0
Age                       0
Occupation                0
```

```
Sleep Duration             0
Quality of Sleep           0
Physical Activity Level    0
Stress Level               0
BMI Category               0
Blood Pressure             0
Heart Rate                 0
Daily Steps                0
Sleep Disorder             0
dtype: int64
```

[12]:
```
df.duplicated().sum()
```

[12]:
```
np.int64(0)
```

[13]:
```
df.drop_duplicates()
```

[13]:
```
     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
..         …     …  …                   …              …
369        370  Female   59                 Nurse             8.1
370        371  Female   59                 Nurse             8.0
371        372  Female   59                 Nurse             8.1
372        373  Female   59                 Nurse             8.1
373        374  Female   59                 Nurse             8.1

     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
..                …                       …            …          …
369                 9                       75             3   Overweight
370                 9                       75             3   Overweight
371                 9                       75             3   Overweight
372                 9                       75             3   Overweight
373                 9                       75             3   Overweight

    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
..             …            …             …          …
369         140/95          68          7000    Sleep Apnea
370         140/95          68          7000    Sleep Apnea
371         140/95          68          7000    Sleep Apnea
372         140/95          68          7000    Sleep Apnea
373         140/95          68          7000    Sleep Apnea

[374 rows x 13 columns]
```

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

[14]:
```
np.int64(0)
```

[15]:
```
for i, column in enumerate(df.columns):
    unique_values = df[column].unique()
    print(f"'{column}': {unique_values}\n")
```

```
'Person ID': [  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270
 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324
 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342
 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
 361 362 363 364 365 366 367 368 369 370 371 372 373 374]

'Gender': ['Male' 'Female']

'Age': [27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 48 49 50 51 52
 53 54 55 56 57 58 59]

'Occupation': ['Software Engineer' 'Doctor' 'Sales Representative' 'Teacher'
```

```
'Nurse'
 'Engineer' 'Accountant' 'Scientist' 'Lawyer' 'Salesperson' 'Manager']

'Sleep Duration': [6.1 6.2 5.9 6.3 7.8 6.  6.5 7.6 7.7 7.9 6.4 7.5 7.2 5.8 6.7
7.3 7.4 7.1
 6.6 6.9 8.  6.8 8.1 8.3 8.5 8.4 8.2]

'Quality of Sleep': [6 4 7 5 8 9]

'Physical Activity Level': [42 60 30 40 75 35 45 50 32 70 80 55 90 47 65 85]

'Stress Level': [6 8 7 4 3 5]

'BMI Category': ['Overweight' 'Normal' 'Obese' 'Normal Weight']

'Blood Pressure': ['126/83' '125/80' '140/90' '120/80' '132/87' '130/86'
'117/76' '118/76'
 '128/85' '131/86' '128/84' '115/75' '135/88' '129/84' '130/85' '115/78'
 '119/77' '121/79' '125/82' '135/90' '122/80' '142/92' '140/95' '139/91'
 '118/75']

'Heart Rate': [77 75 85 82 70 80 78 69 72 68 76 81 65 84 74 67 73 83 86]

'Daily Steps': [ 4200 10000  3000  3500  8000  4000  4100  6800  5000  7000
5500  5200
  5600  3300  4800  7500  7300  6200  6000  3700]

'Sleep Disorder': ['None' 'Sleep Apnea' 'Insomnia']
```

[16]:
```
df['BMI Category'] = df['BMI Category'].replace({'Normal Weight':␣
↪'Underweight'})
df['BMI Category'].value_counts()
```

[16]:
```
BMI Category
Normal         195
Overweight     148
Underweight     21
Obese           10
Name: count, dtype: int64
```

[17]:
```
df.dtypes
```

[17]:
```
Person ID                  int64
Gender                    object
Age                        int64
Occupation                object
Sleep Duration           float64
```

```
Quality of Sleep             int64
Physical Activity Level      int64
Stress Level                 int64
BMI Category                object
Blood Pressure              object
Heart Rate                   int64
Daily Steps                  int64
Sleep Disorder              object
dtype: object
```

[18]:
```
cat_df = df.select_dtypes(include=[object]).columns
```

[19]:
```
df[cat_df] = df[cat_df].astype("category")
```

[20]:
```
df[cat_df] = df[cat_df].astype("category")
df
```

[20]:

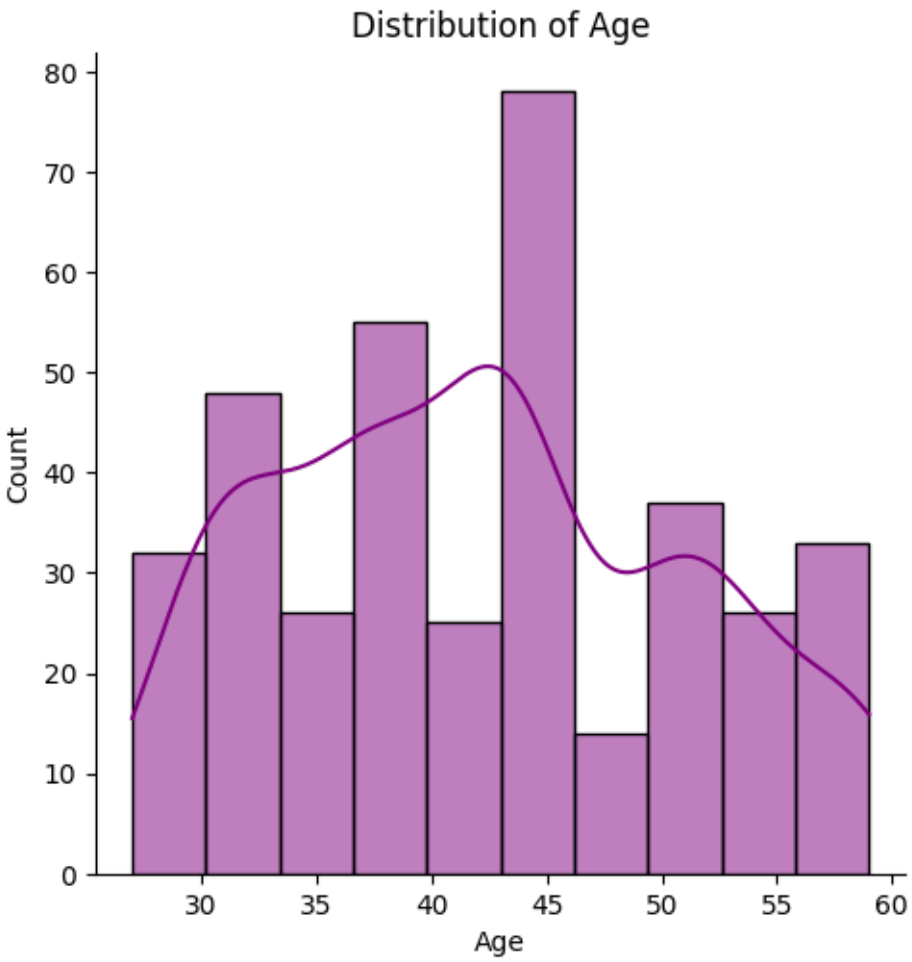
	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate	Daily Steps	Sleep Disorder
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77	4200	None
1	2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None
2	3	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese				
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese				
..	…	…	…	…	…	…	…	…	…				
369	370	Female	59	Nurse	8.1	9	75	3	Overweight				
370	371	Female	59	Nurse	8.0	9	75	3	Overweight				
371	372	Female	59	Nurse	8.1	9	75	3	Overweight				
372	373	Female	59	Nurse	8.1	9	75	3	Overweight				
373	374	Female	59	Nurse	8.1	9	75	3	Overweight				

```
3           140/90          85          3000    Sleep Apnea
4           140/90          85          3000    Sleep Apnea
..             …            …           …            …
369         140/95          68          7000    Sleep Apnea
370         140/95          68          7000    Sleep Apnea
371         140/95          68          7000    Sleep Apnea
372         140/95          68          7000    Sleep Apnea
373         140/95          68          7000    Sleep Apnea

[374 rows x 13 columns]
```

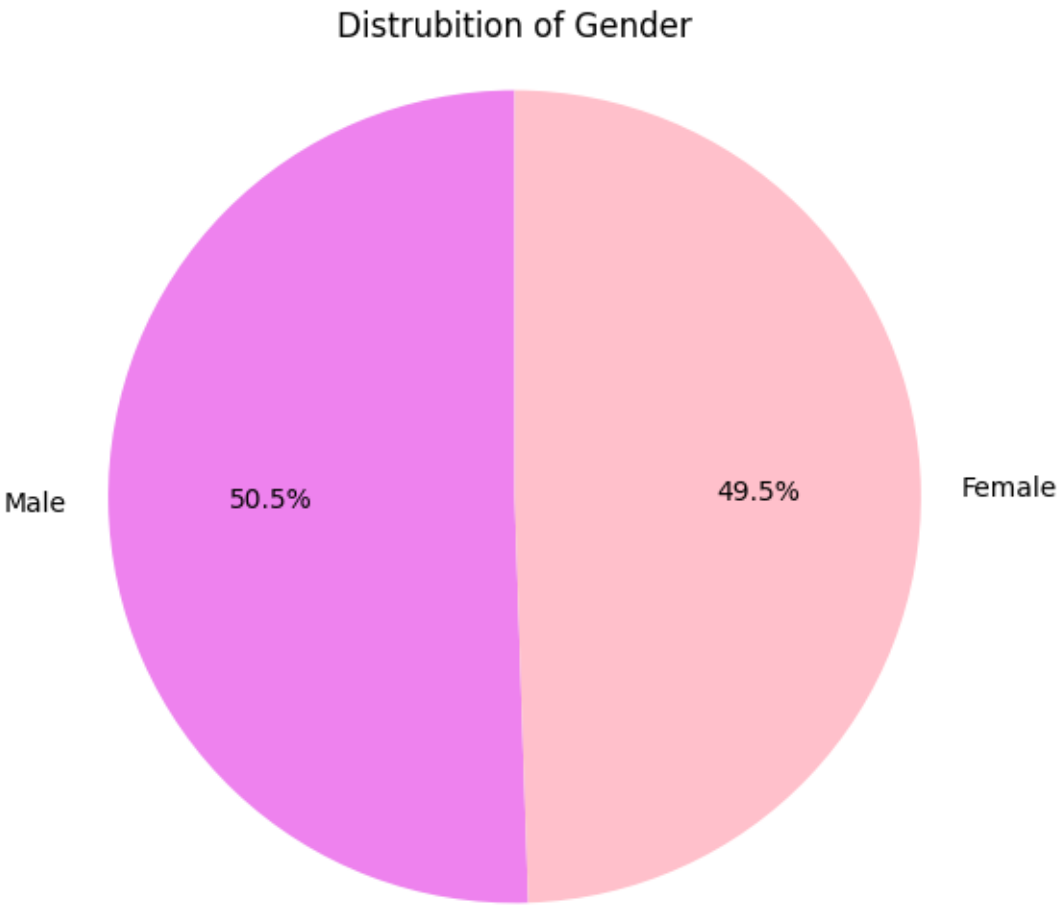
[21]:
```
df.info()
```

```
<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    category
 2   Age                      374 non-null    int64
 3   Occupation               374 non-null    category
 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    category
 9   Blood Pressure           374 non-null    category
 10  Heart Rate               374 non-null    int64
 11  Daily Steps              374 non-null    int64
 12  Sleep Disorder           374 non-null    category
dtypes: category(5), float64(1), int64(7)
memory usage: 26.9 KB
```

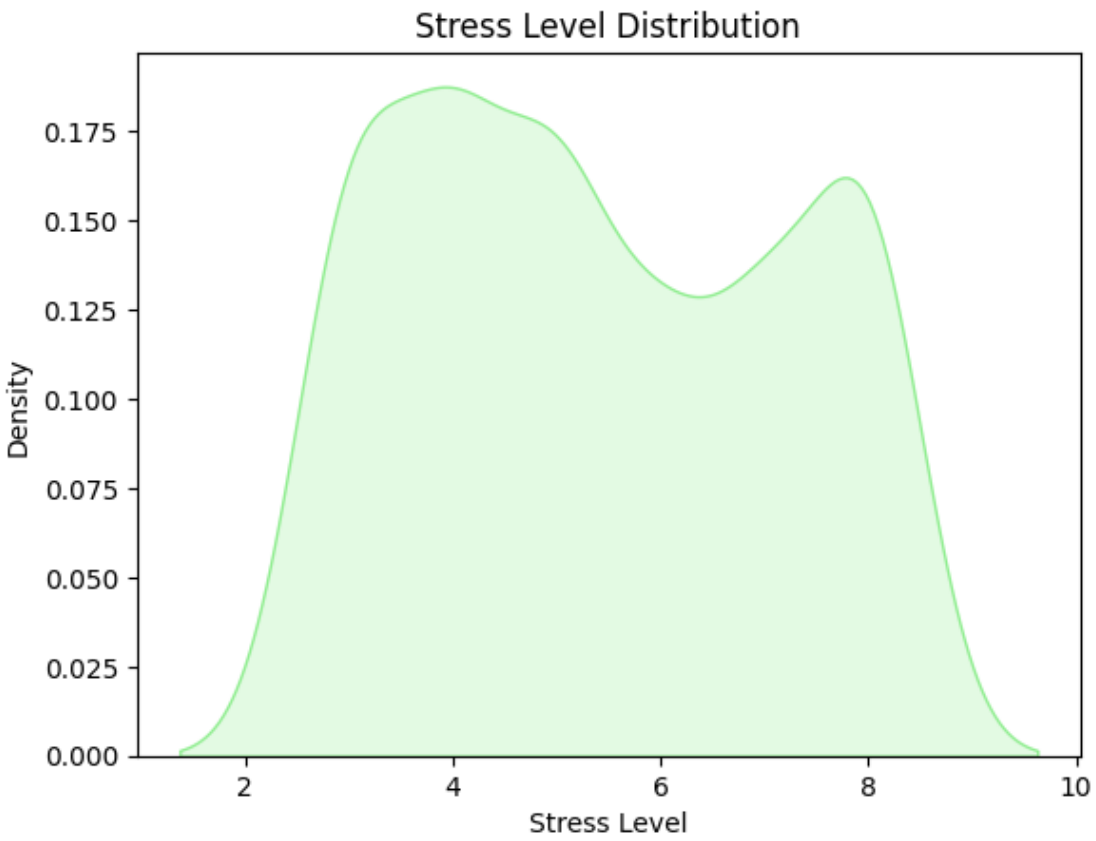
[29]:
```
#Data Visualization
sns.displot(df.Age ,kde=True,color="purple")
plt.title('Distribution of Age');
```



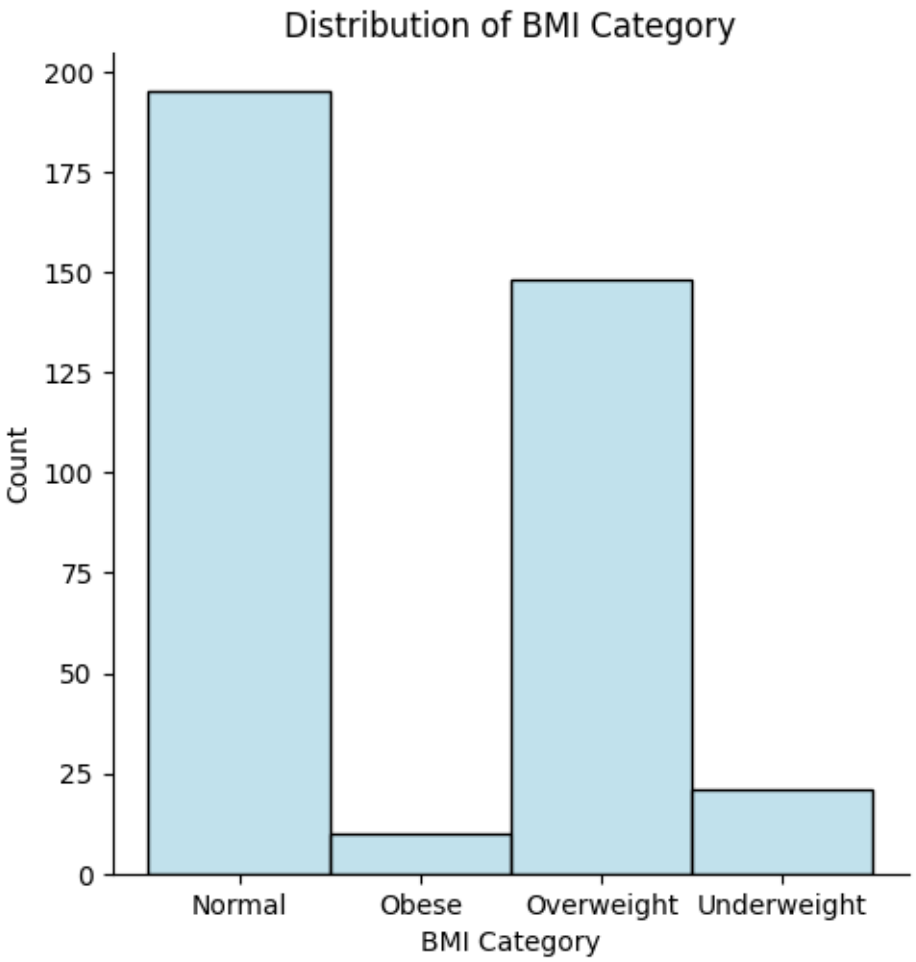
[30]:
```
Gender_Count=df["Gender"].value_counts()
plt.figure(figsize=(6,6))
plt.pie(Gender_Count,labels=Gender_Count.index,autopct="%1.
  ↪1f%%",colors=['violet','pink'], startangle=90)
plt.title("Distrubition of Gender");
plt.axis("equal");
```



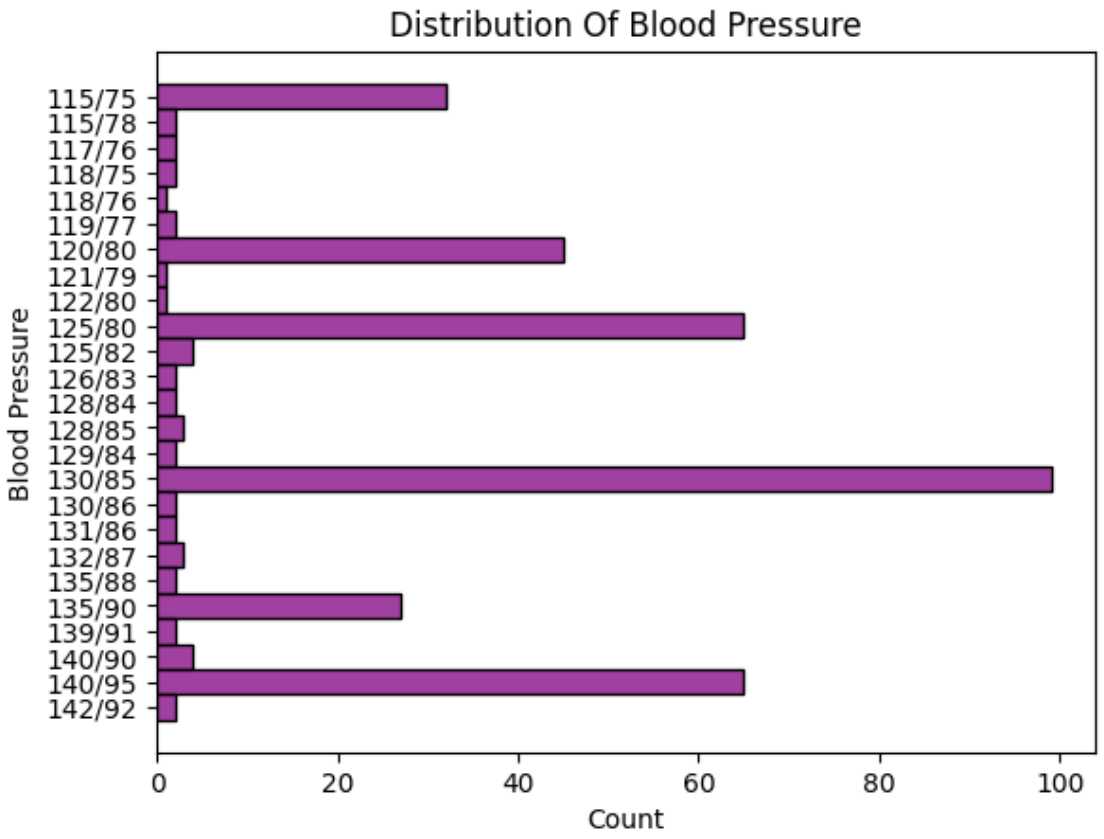
[31]:
```
sns.kdeplot(df['Stress Level'], shade=True, color='lightgreen')
plt.title('Stress Level Distribution');
```



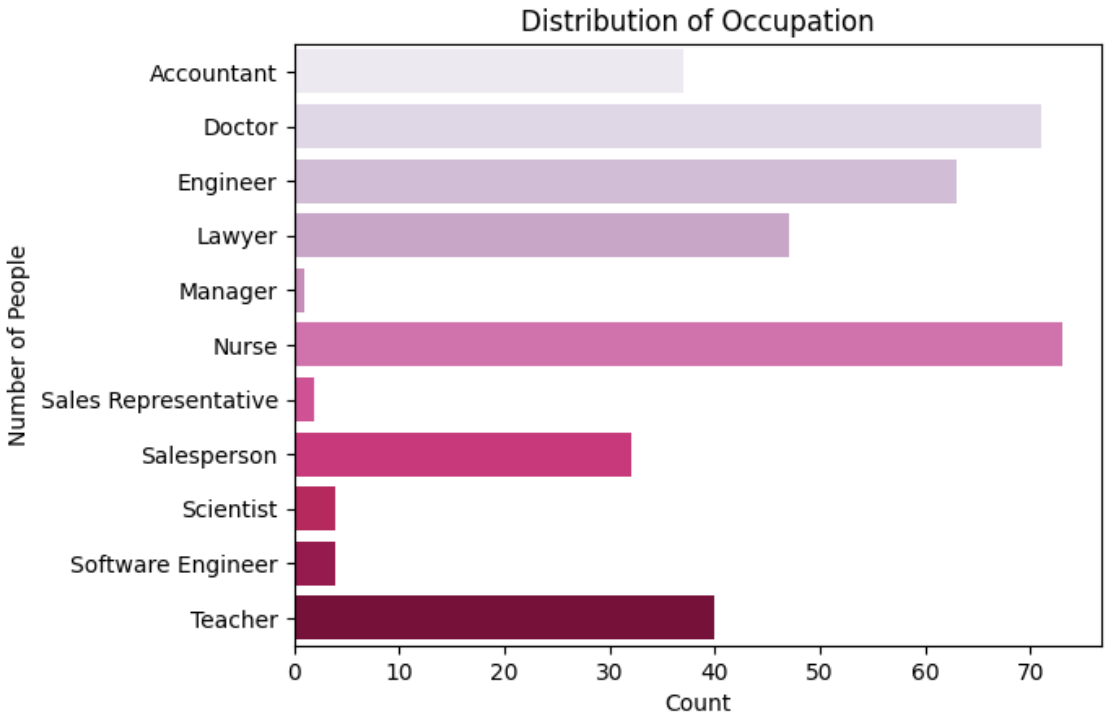
[33]:
```
sns.displot(df["BMI Category"], kde=False, color="lightblue").
↪set(title="Distribution of BMI Category");
```



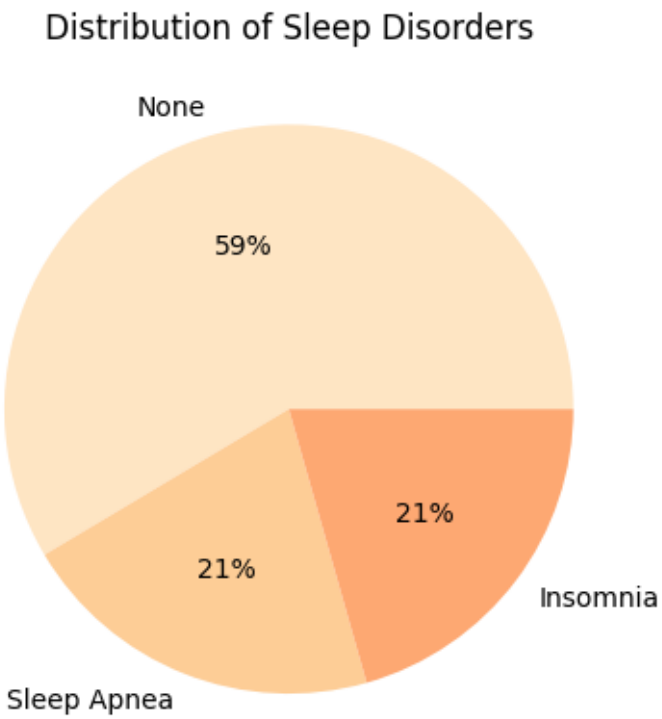
[40]:
```
sns.histplot(data=df, y="Blood Pressure", kde=False, color="purple")
plt.xlabel('Count')
plt.ylabel('Blood Pressure')
plt.title('Distribution Of Blood Pressure');
```



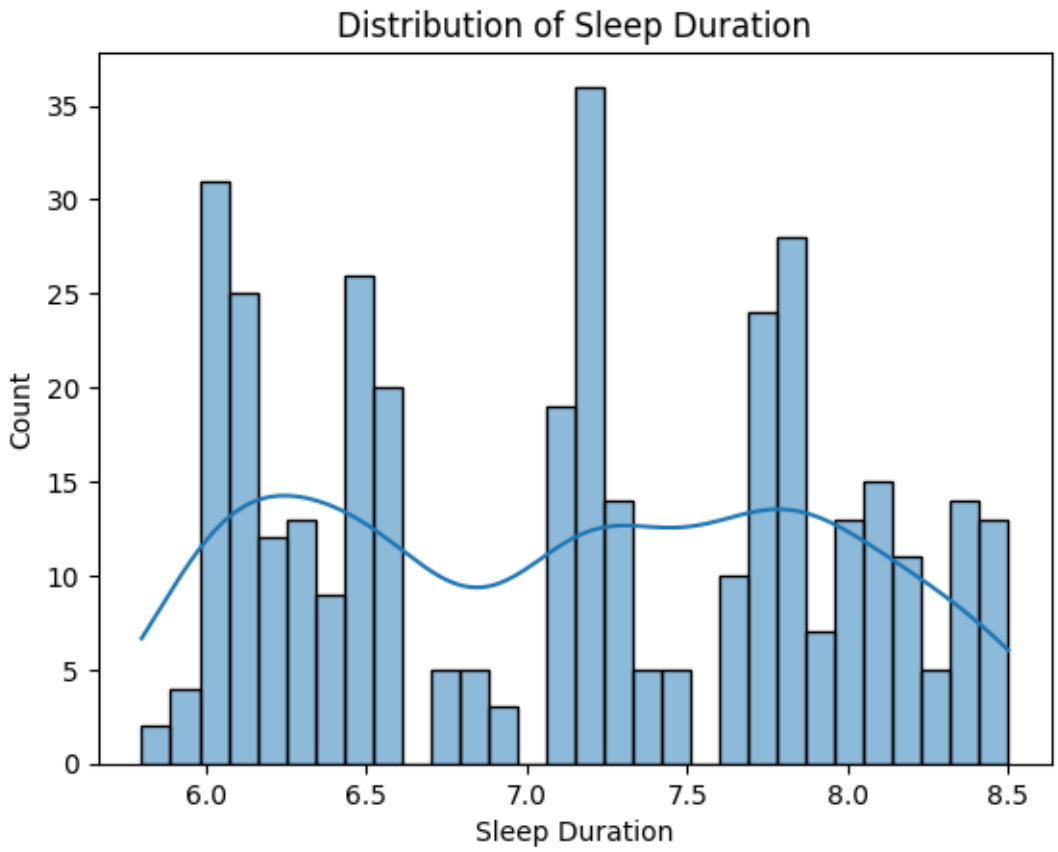
[39]:
```
occupation_counts = df['Occupation'].value_counts().reset_index()
occupation_counts.columns = ['Occupation', 'Count']

ocu = sns.barplot(x='Count', y='Occupation', data=occupation_counts,␣
↪palette="PuRd")
ocu.set_title("Distribution of Occupation");
ocu.set_ylabel("Number of People");
```



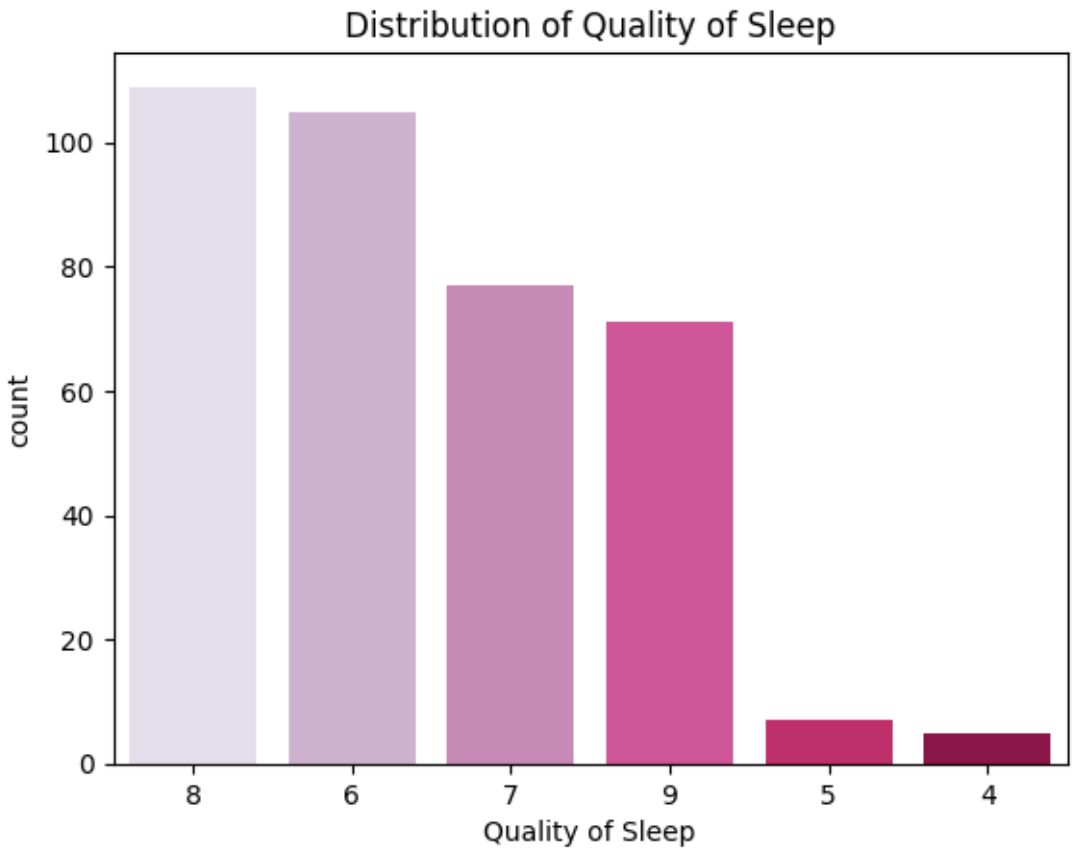
[46]:
```
sleep_disorder = df['Sleep Disorder'].value_counts()
plt.pie(sleep_disorder, labels=sleep_disorder.index, autopct='%.0f%%',␣
  ↪colors=sns.color_palette("OrRd"))
plt.title('Distribution of Sleep Disorders');
```



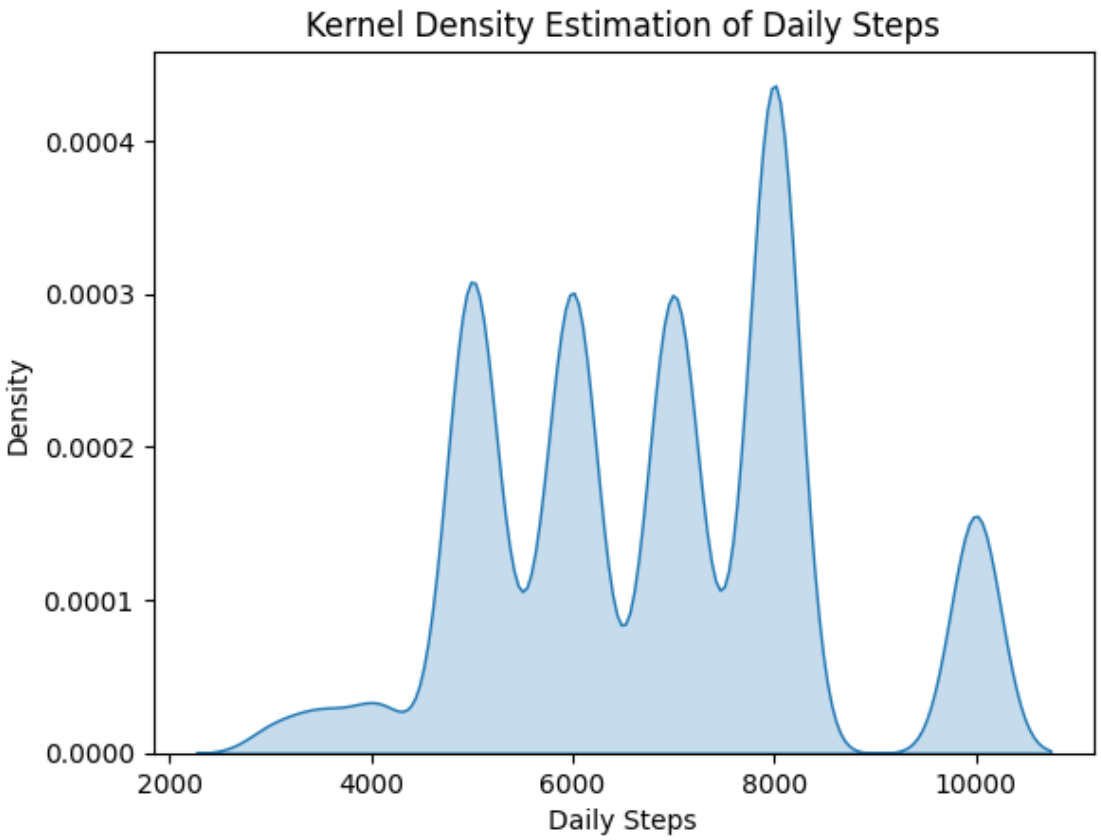
[47]:
```
sns.histplot(df['Sleep Duration'], bins=30, kde=True)
plt.title('Distribution of Sleep Duration');
```



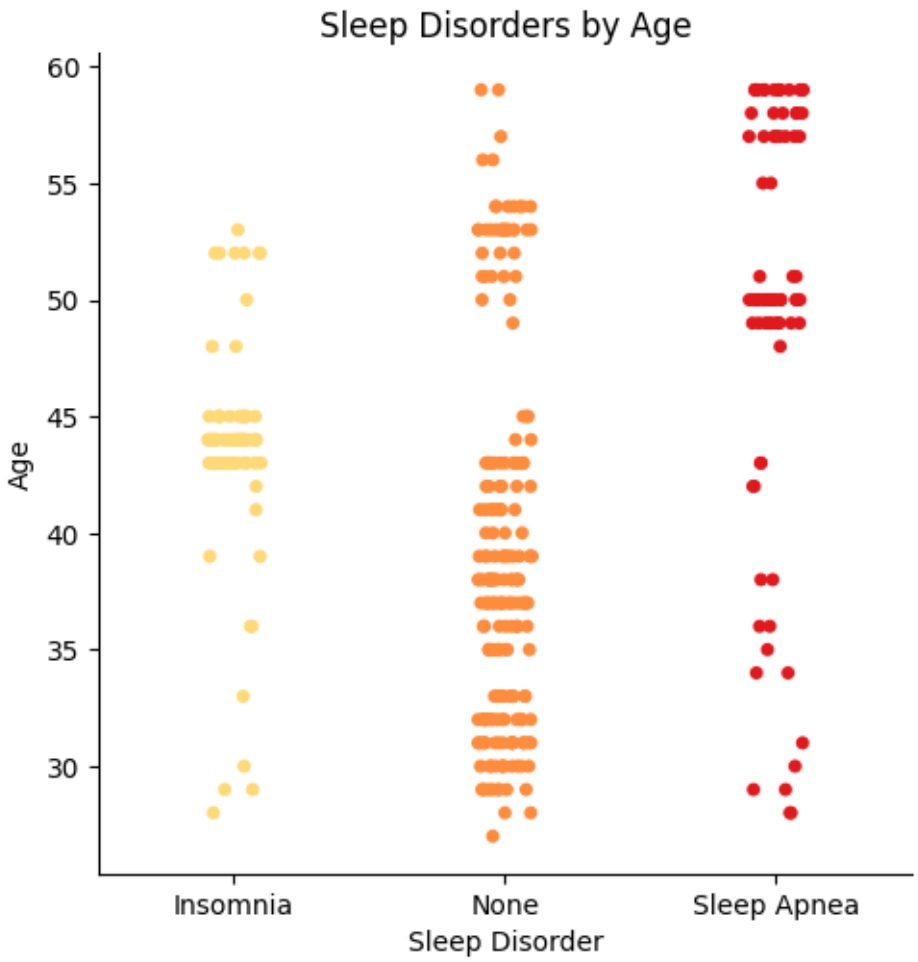
[48]:
```
sns.countplot(x='Quality of Sleep', data=df, palette = "PuRd",␣
  ↪order=df['Quality of Sleep'].value_counts().index)
plt.title('Distribution of Quality of Sleep');
```



[50]:
```
sns.kdeplot(x='Daily Steps', data=df, fill=True, bw_adjust=0.5)
plt.title('Kernel Density Estimation of Daily Steps');
```

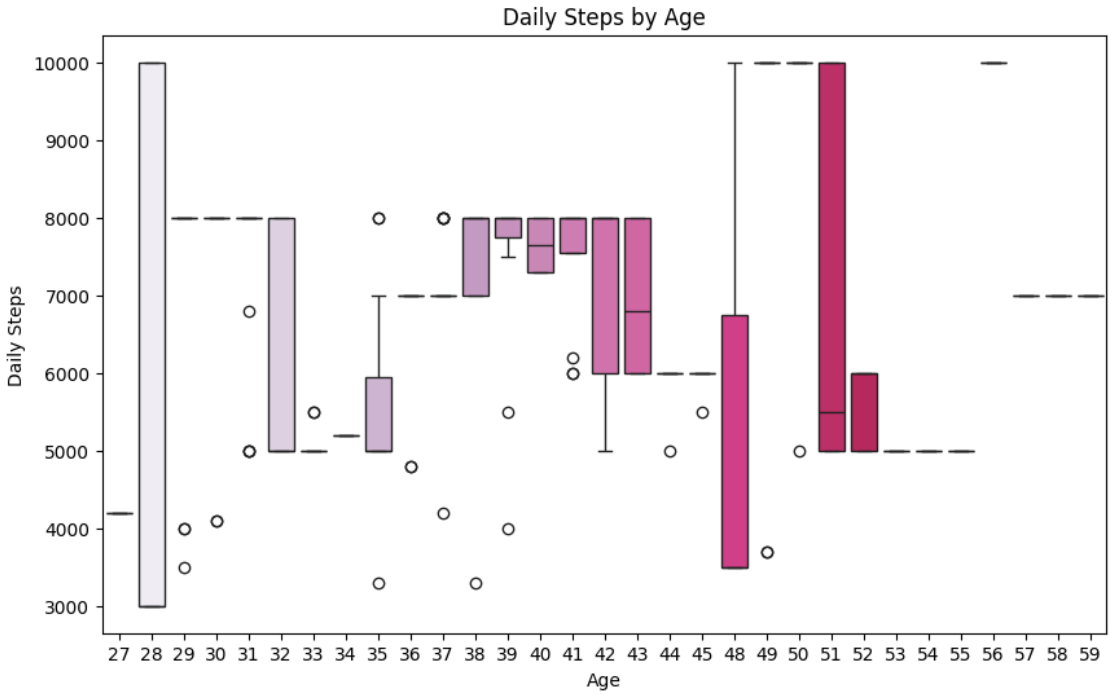


[55]:
```
sns.catplot(y=df.Age,x="Sleep Disorder", data=df,palette="YlOrRd");
plt.title('Sleep Disorders by Age');
```

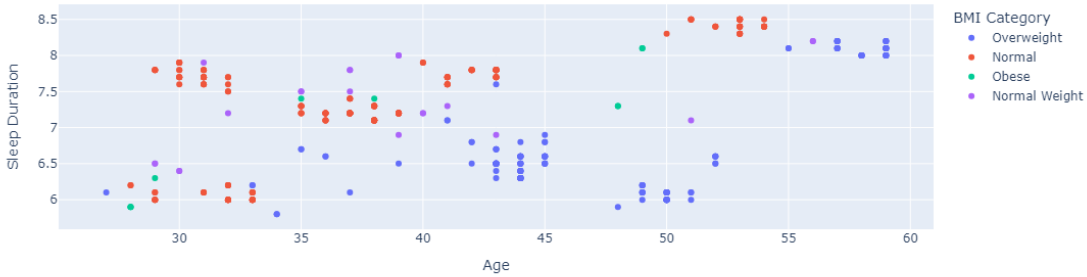


[60]:
```
sns.catplot(x="Sleep Disorder", y="Age", hue="Gender", kind="violin", data=df,␣
↪palette="YlGnBu")
plt.title("Violin Plot of Age by Sleep Disorder and Gender");
```

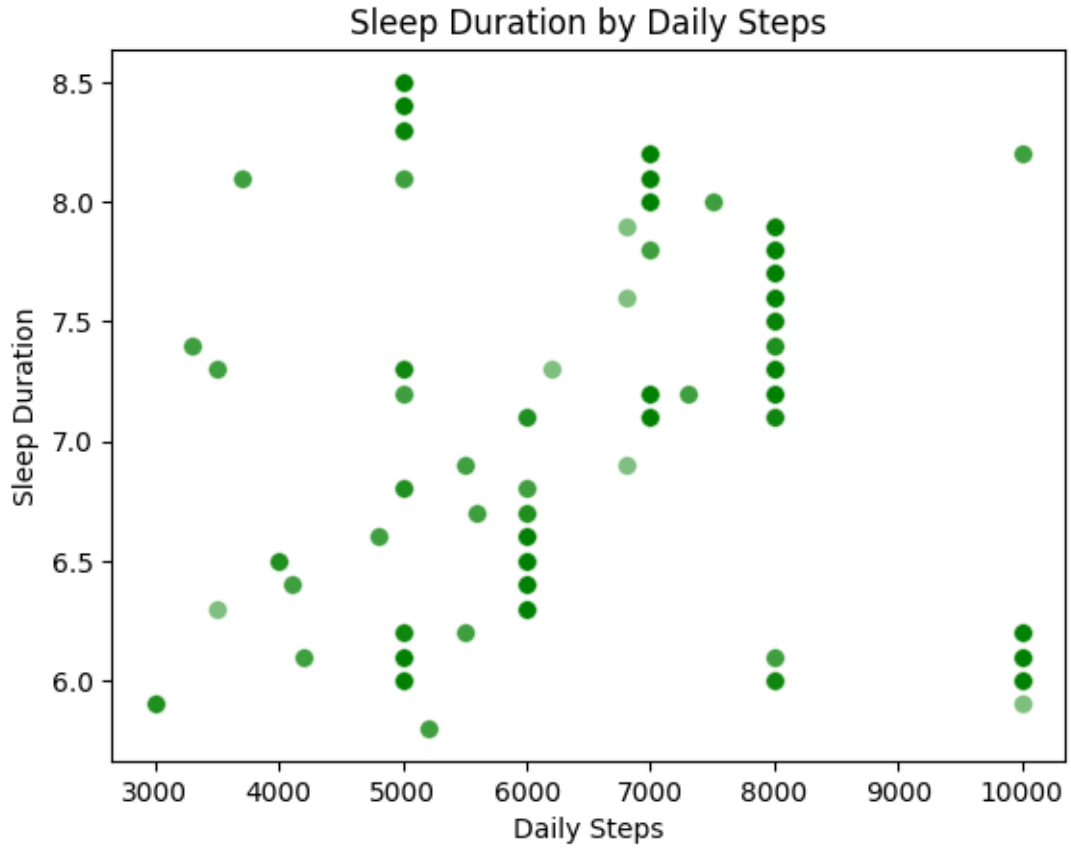
[61]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x=df.Age,y="Daily Steps",data=df,palette="PuRd").set_title("Daily␣
↪Steps by Age");
```



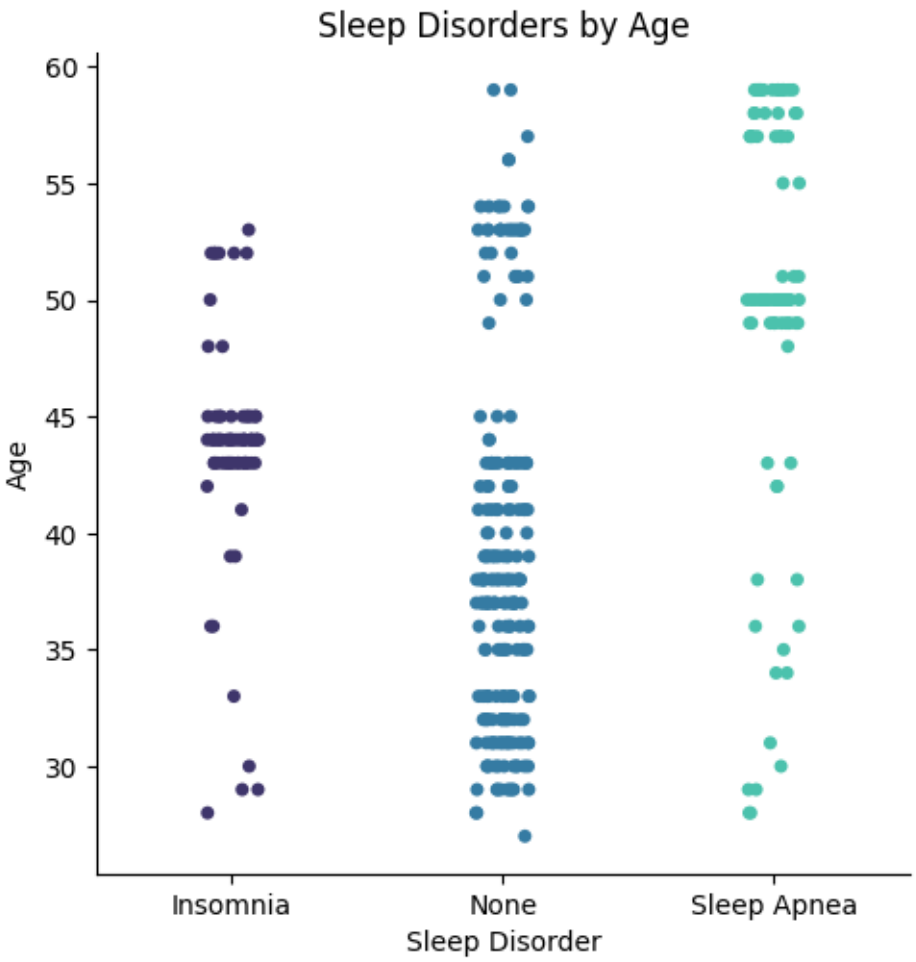
[4]:
```
import pandas as pd
import plotly.express as px
def plotly_scatter(df, numerical_column_one, numerical_column_two,
                   color=None, row=None, col=None):
    fig = px.scatter(df,
                     x=numerical_column_one,
                     y=numerical_column_two,
                     facet_col=col,
                     color=color,
                     facet_row=row,
                     height=600
                     )
    fig.update_yaxes(showticklabels=True, matches=None)
    fig.update_xaxes(showticklabels=True, matches=None)
    fig.show()
sleep_dataset = r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv"
df = pd.read_csv(sleep_dataset)
plotly_scatter(df, "Age", "Sleep Duration", color="BMI Category")
```



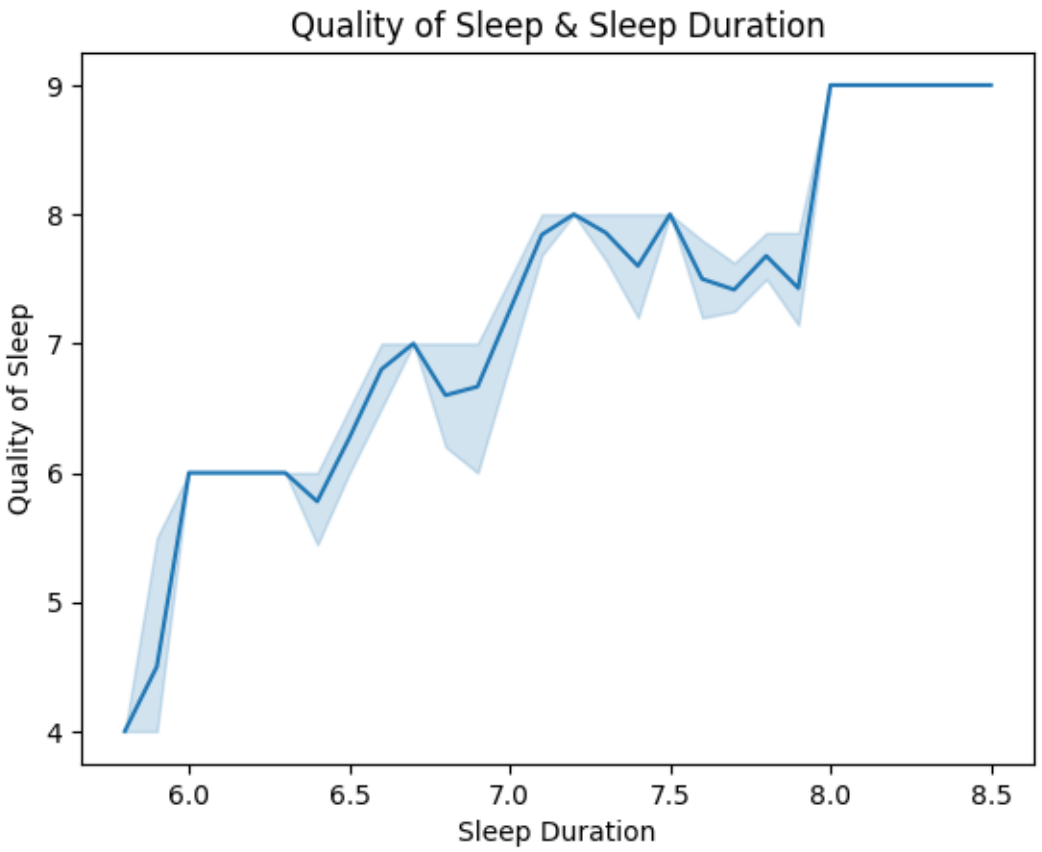
[52]:
```
sns.scatterplot(x='Daily Steps', y='Sleep Duration', data=df, alpha=0.5, s=50,␣
↪color='green')
plt.title('Sleep Duration by Daily Steps');
```



[53]:
```
sns.catplot(y=df.Age,x="Sleep Disorder", data=df,palette="mako");
plt.title('Sleep Disorders by Age');
```

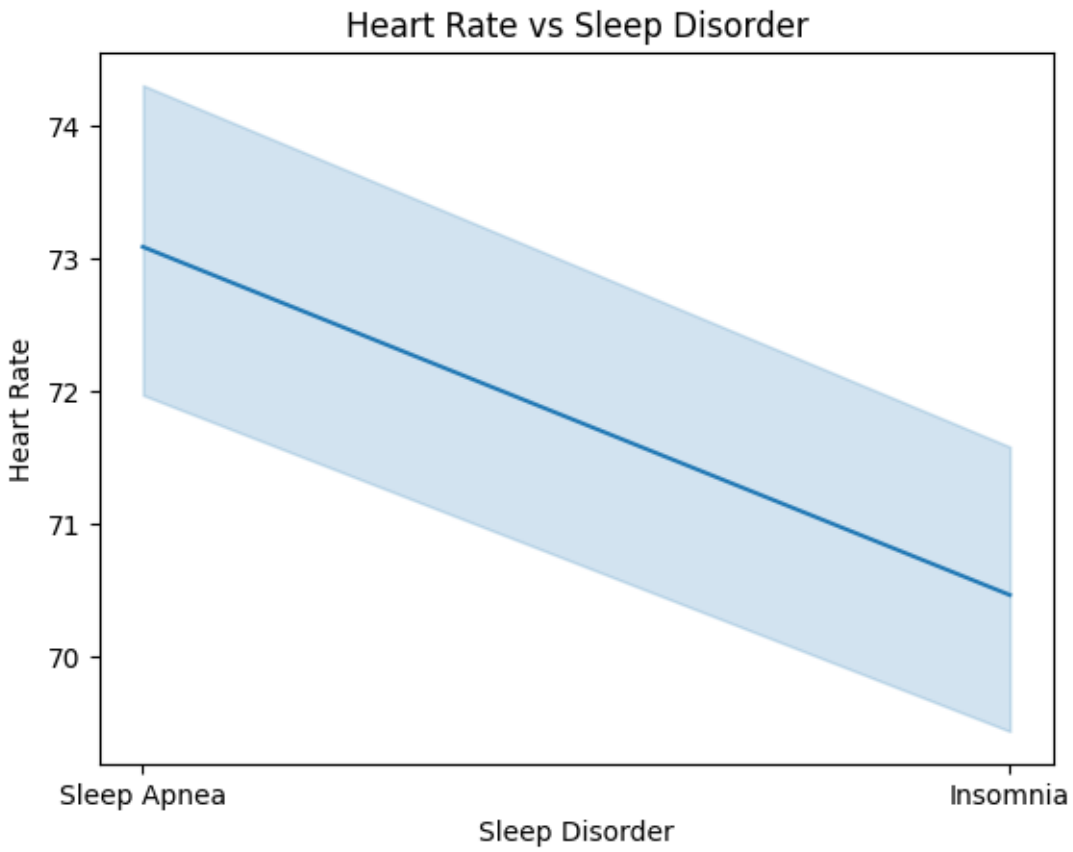


[22]:
```
sns.lineplot(x="Sleep Duration", y="Quality of Sleep", data=df);
plt.title("Quality of Sleep & Sleep Duration");
```

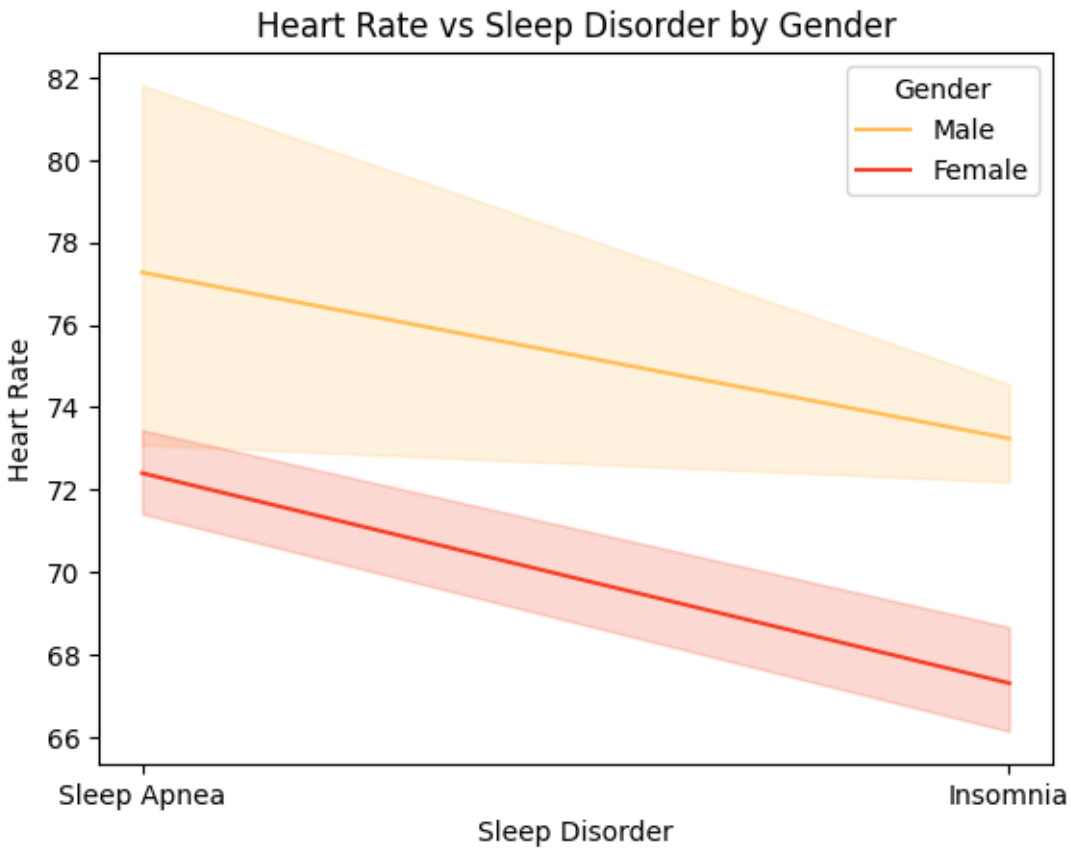


[23]:
```
sns.lineplot(x='Sleep Disorder',
             y='Heart Rate',
             palette ="PuRd",
             data=df)
plt.title('Heart Rate vs Sleep Disorder');
```

```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\4118055715.py:1:
UserWarning:

Ignoring `palette` because no `hue` variable has been assigned.
```

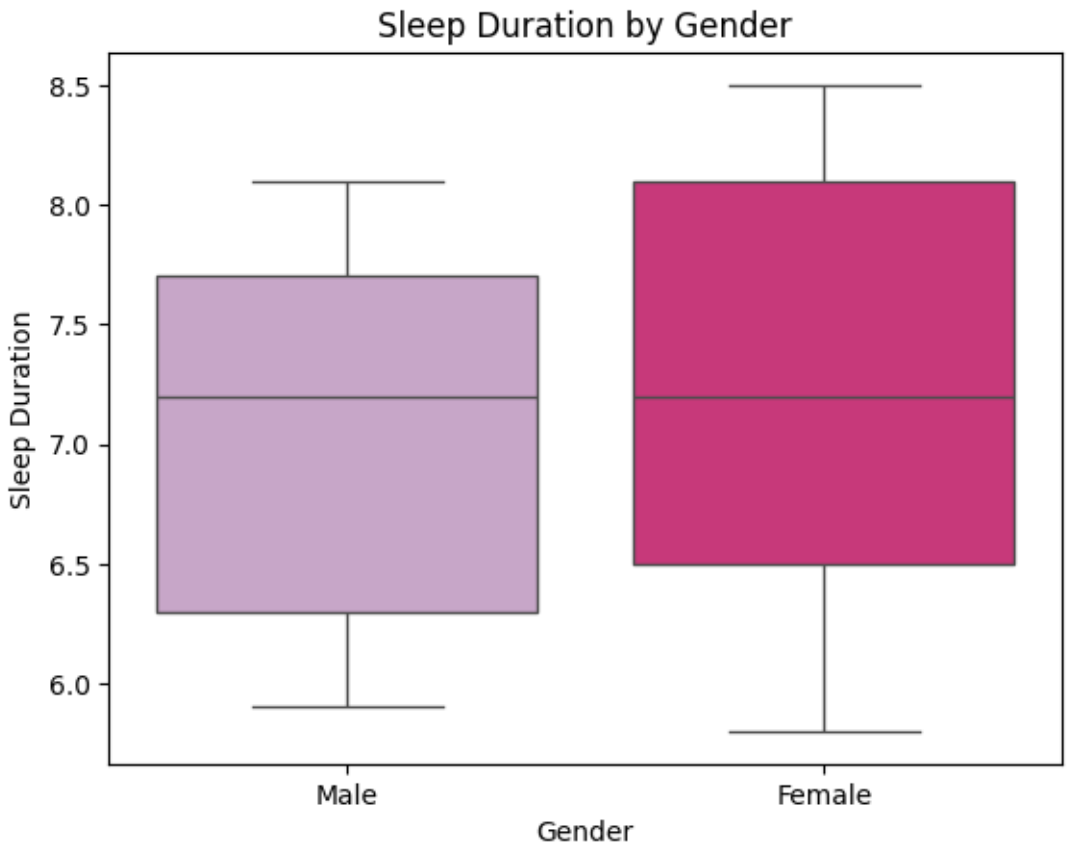


[20]:
```
sns.lineplot(x='Sleep Disorder',
             y='Heart Rate',
             hue="Gender",
             palette ="YlOrRd",
             markers=True,
             dashes=False,
             data=df)
plt.title('Heart Rate vs Sleep Disorder by Gender');
```



[17]:
```
sns.boxplot(x='Gender', y='Sleep Duration', palette = "PuRd", data=df)
plt.title('Sleep Duration by Gender');
```

```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\1385702253.py:1:
FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```



[24]:
```
df['Age_grp'] = pd.cut(df['Age'], [20, 30, 40, 50, 60], labels=['20s', '30s',␣
↪'40s', '50s'])
```

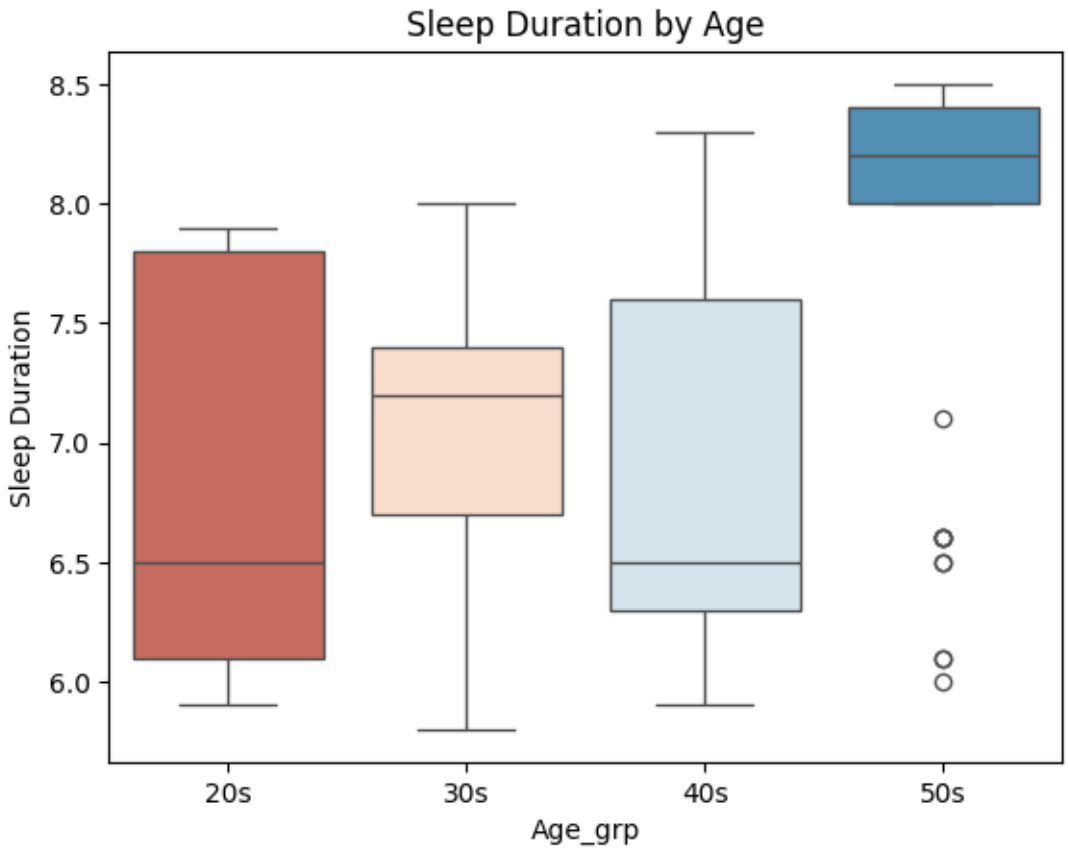
[25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 14 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
```

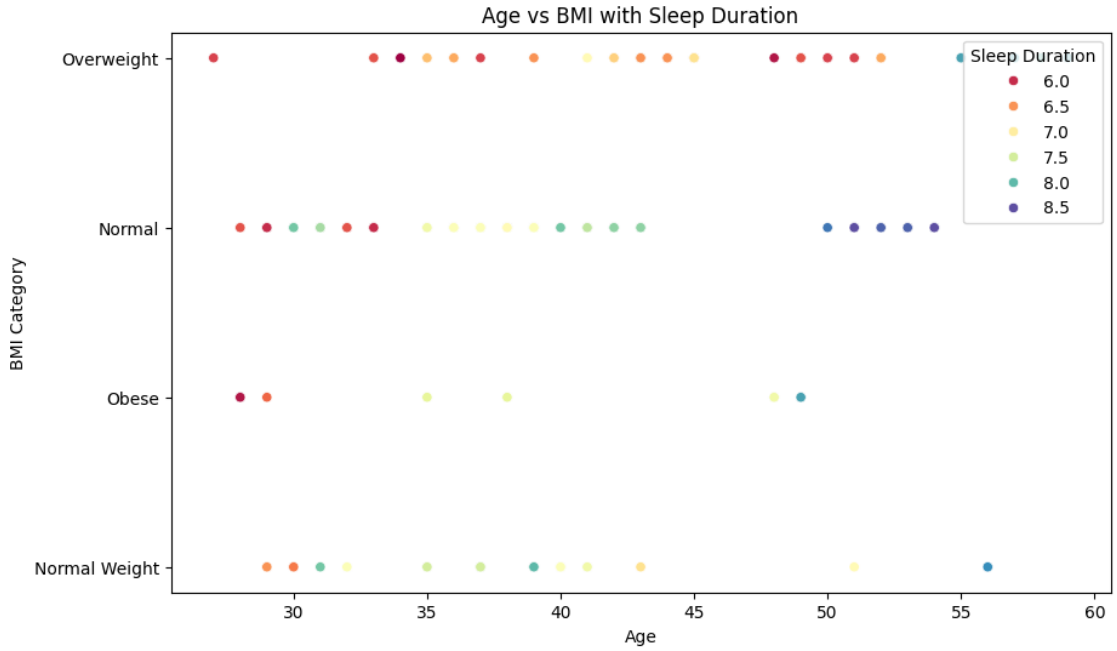
```
 10  Heart Rate               374 non-null    int64
 11  Daily Steps              374 non-null    int64
 12  Sleep Disorder           155 non-null    object
 13  Age_grp                  374 non-null    category
dtypes: category(1), float64(1), int64(7), object(5)
memory usage: 38.7+ KB
```

[28]:
```
sns.boxplot(x='Age_grp', y='Sleep Duration', palette = "RdBu", data=df)
plt.title('Sleep Duration by Age');
```

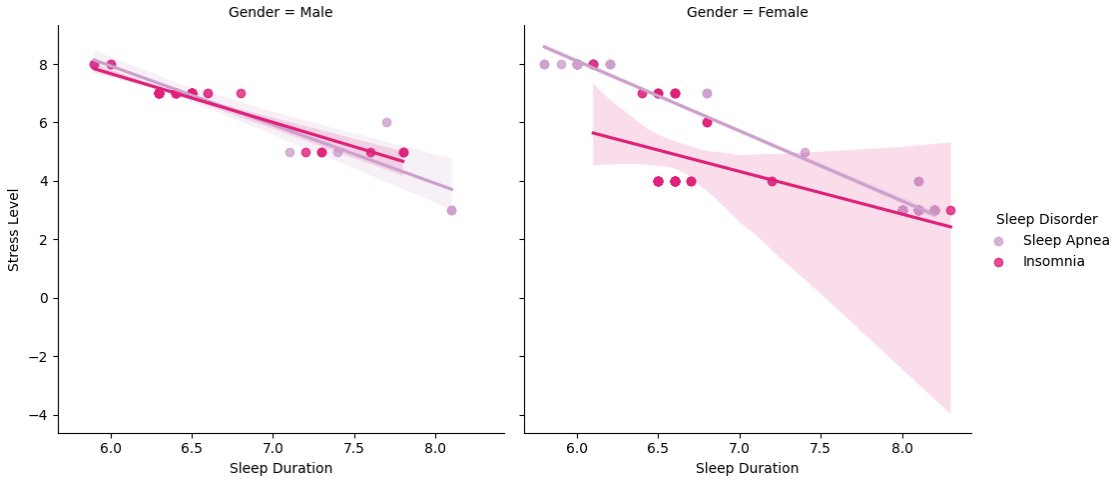
```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\2670058239.py:1:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```



[31]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Age', y='BMI Category', hue='Sleep Duration', data=df,␣
↪palette='Spectral')
plt.legend(title='Sleep Duration', loc='upper right')
plt.title('Age vs BMI with Sleep Duration');
```

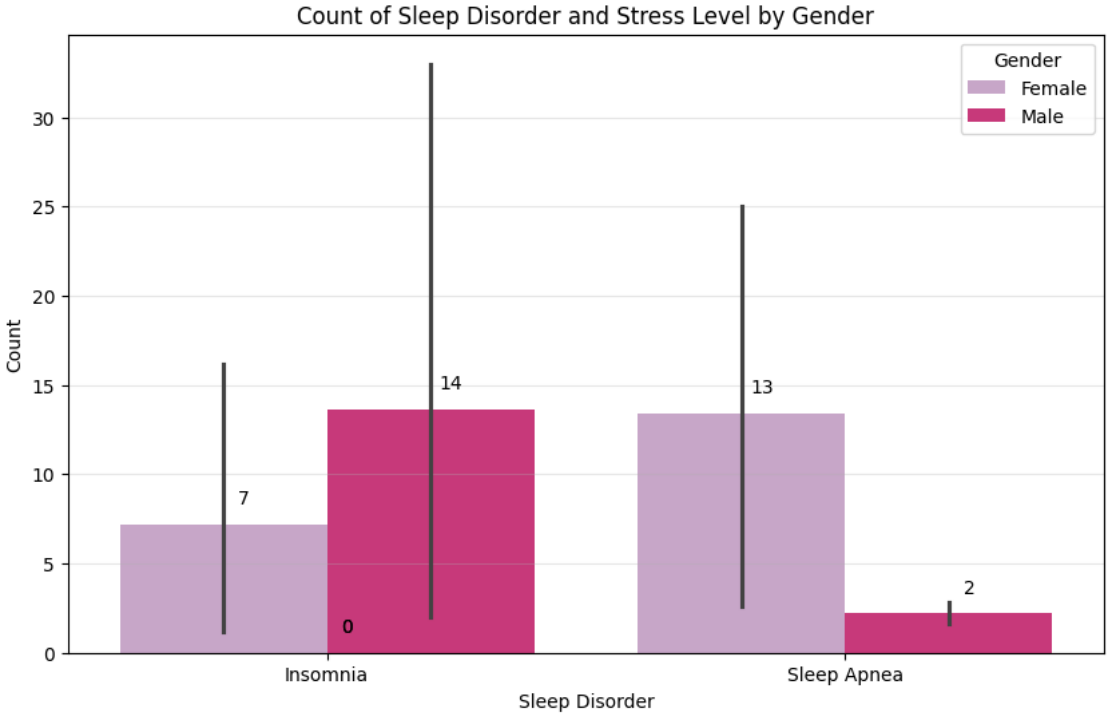


[32]:
```
sns.lmplot(x="Sleep Duration", y="Stress Level", hue="Sleep Disorder",␣
↪col="Gender", palette="PuRd",data=df);
```



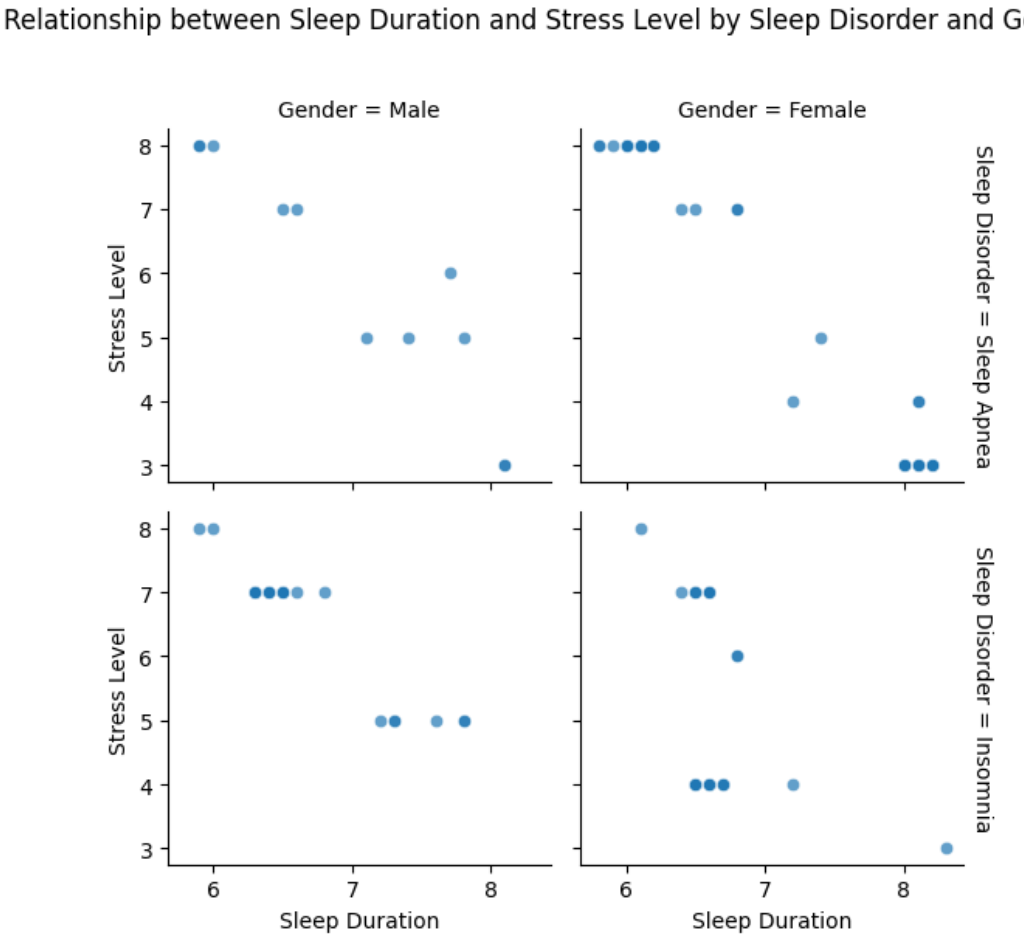
[33]:
```
# Grouping the data by "Sleep Disorder", "Stress Level", and "Gender"
grouped_data = df.groupby(["Sleep Disorder", "Stress Level", "Gender"]).size().
 ↪reset_index(name="Count")
```

[35]:
```
plt.figure(figsize=(10, 6))
ax = sns.barplot(x="Sleep Disorder", y="Count", hue="Gender", palette="PuRd",␣
 ↪data=grouped_data)
plt.title("Count of Sleep Disorder and Stress Level by Gender")
plt.grid(axis='y', alpha=0.3)
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 = 'bottom',
                xytext = (11, 9),
                textcoords = 'offset points')
```



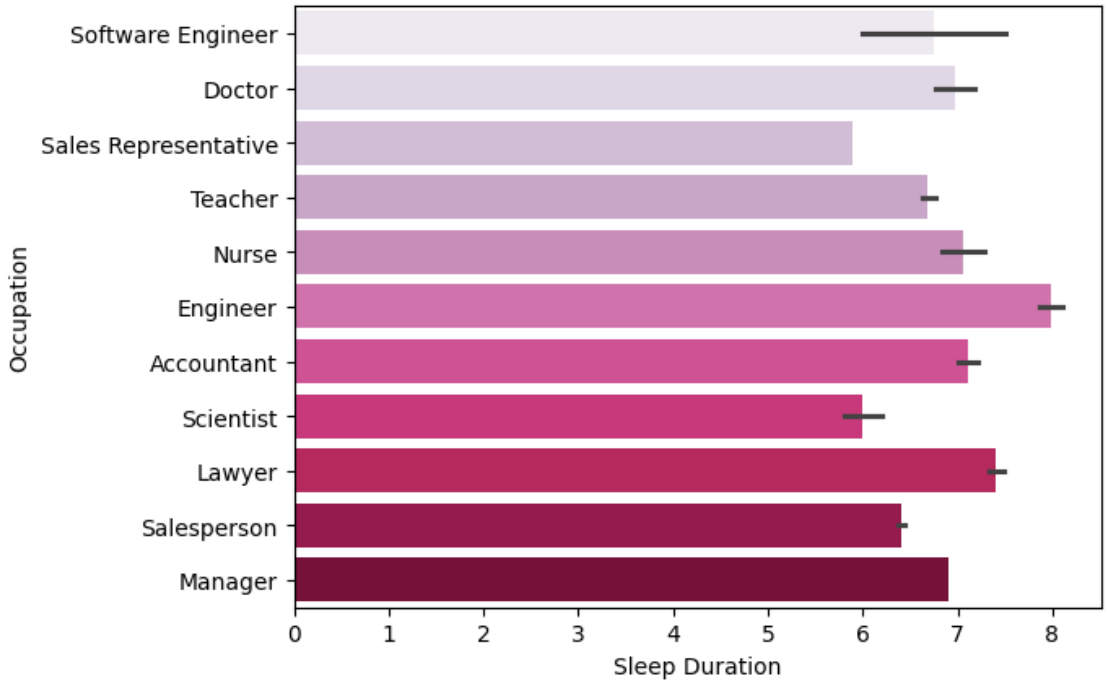
[37]:
```
#Facet Drid creation
g = sns.FacetGrid(df, col="Gender", row="Sleep Disorder", palette = 'PuRd',␣
 ↪margin_titles=True)
#Mapping a scatter plot to the Facet Grid
g.map(sns.scatterplot, "Sleep Duration", "Stress Level", alpha=0.7)
#Setting of the titles
```

```
g.fig.suptitle('Relationship between Sleep Duration and Stress Level by Sleep␣
 ↪Disorder and Gender', y=1.02)
g.set_axis_labels("Sleep Duration", "Stress Level")
#Adjusting the corresponding layout
plt.tight_layout()
plt.show()
```



[38]:
```
sns.barplot(x='Sleep Duration', y='Occupation', data=df ,palette="PuRd");
```

```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\3792947875.py:1:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.
```

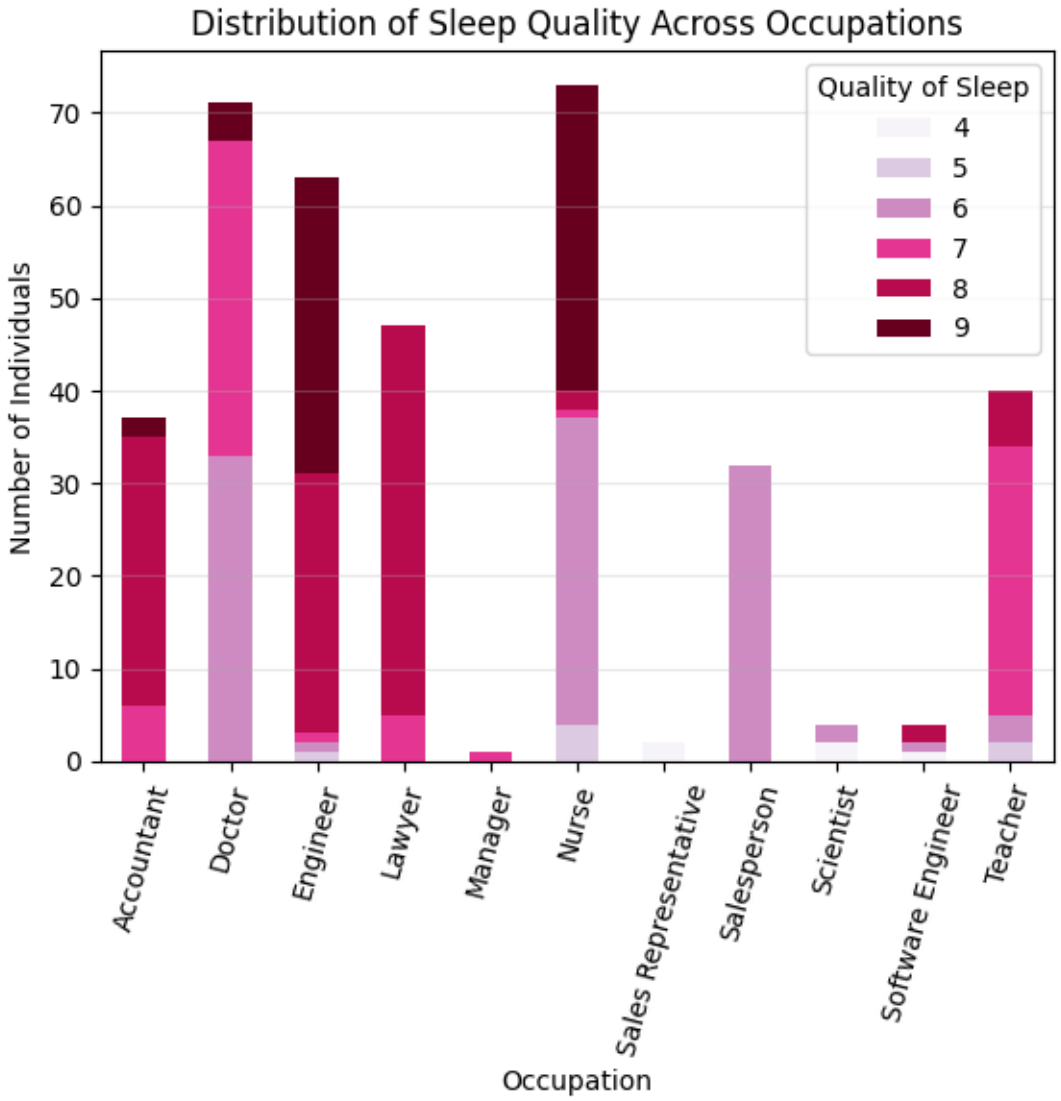


[39]:
```
occupation_counts = df['Occupation'].value_counts()
occupation_counts
```

[39]:
```
Occupation
Nurse                    73
Doctor                   71
Engineer                 63
Lawyer                   47
Teacher                  40
Accountant               37
Salesperson              32
Scientist                 4
Software Engineer         4
Sales Representative      2
Manager                   1
Name: count, dtype: int64
```

[40]:
```
occupation_counts = df['Occupation'].value_counts()
occupation_sleep_counts = df.groupby(['Occupation', 'Quality of Sleep']).size().
 ↪unstack(fill_value=0)
occupation_sleep_counts.plot(kind='bar', stacked=True, cmap='PuRd')
plt.title('Distribution of Sleep Quality Across Occupations')
plt.ylabel('Number of Individuals')
```

```
plt.xticks(rotation=75)
plt.legend(title='Quality of Sleep', loc='upper right')
plt.grid(axis='y', alpha=0.3)
plt.show()
print('\n',occupation_sleep_counts)
```

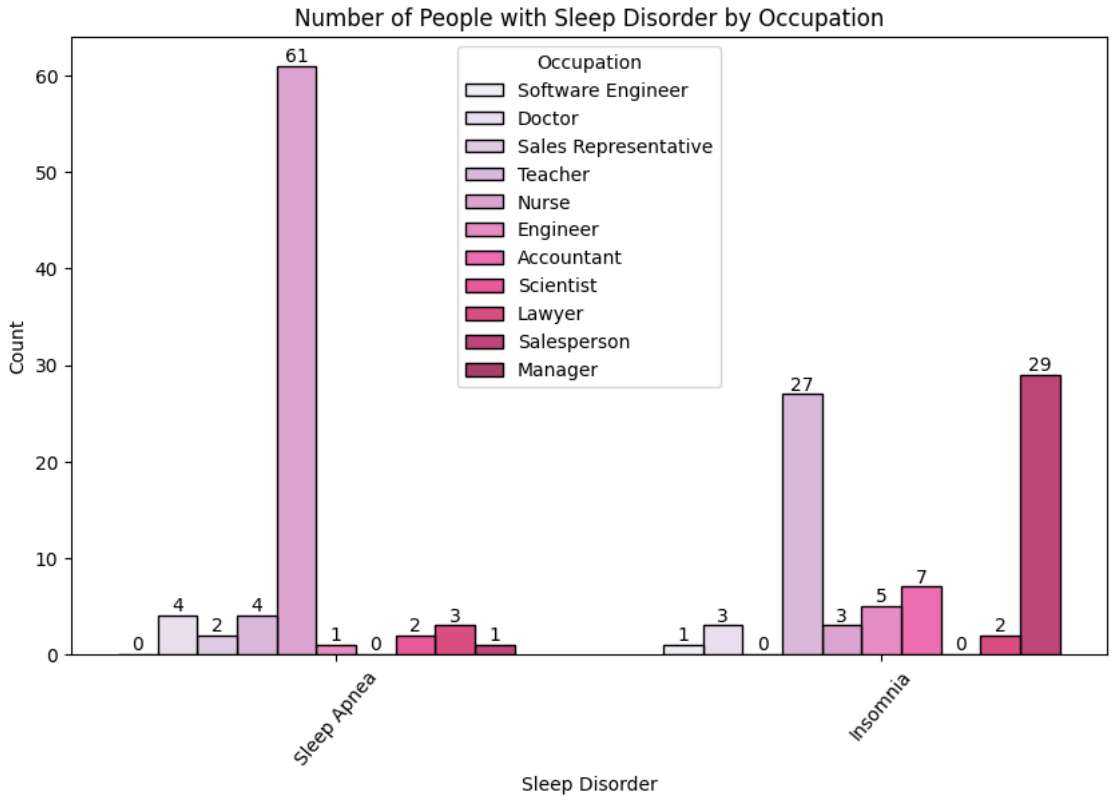


Quality of Sleep	4	5	6	7	8	9
Occupation						
Accountant	0	0	0	6	29	2
Doctor	0	0	33	34	0	4
Engineer	0	1	1	1	28	32
Lawyer	0	0	0	5	42	0

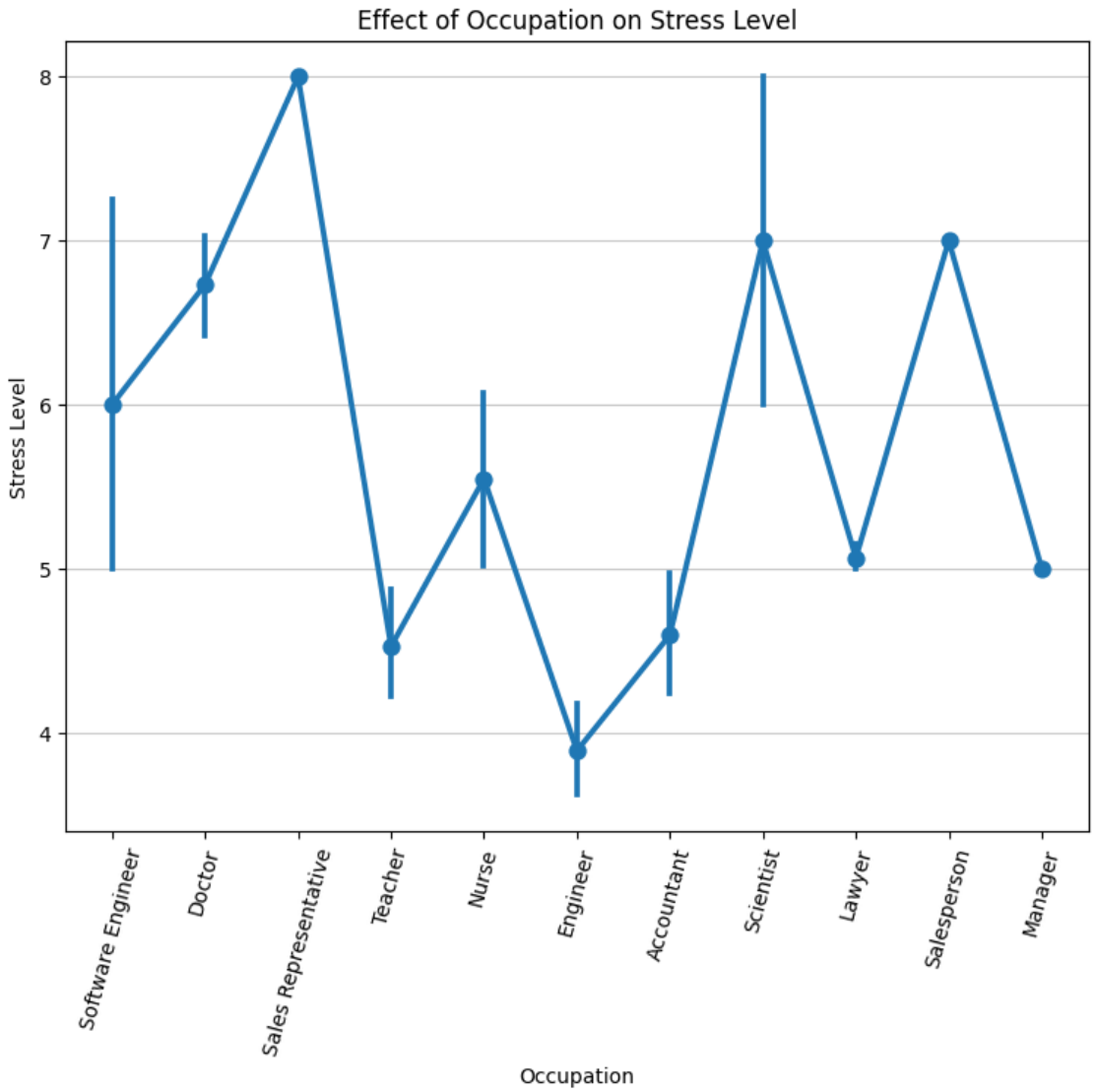
```
Manager                0  0   0   1   0   0
Nurse                  0  4  33   1   2  33
Sales Representative   2  0   0   0   0   0
Salesperson            0  0  32   0   0   0
Scientist              2  0   2   0   0   0
Software Engineer      1  0   1   0   2   0
Teacher                0  2   3  29   6   0
```

[42]:
```
contingency_table = pd.crosstab(df['Sleep Disorder'], df['Occupation'])
print("Contingency Table:")
print(contingency_table)
print("\n")
plt.figure(figsize=(10,6))
ax = sns.histplot(data=df, x=df["Sleep Disorder"], hue=df['Occupation'],␣
 ↪multiple="dodge",palette = "PuRd", shrink=.8)
plt.title("Number of People with Sleep Disorder by Occupation");
plt.xticks(rotation=50);
for i in ax.containers:
        ax.bar_label(i);
```

```
Contingency Table:
Occupation      Accountant  Doctor  Engineer  Lawyer  Nurse  \
Sleep Disorder
Insomnia                 7       3         5       2      3
Sleep Apnea              0       4         1       3     61

Occupation      Sales Representative  Salesperson  Scientist  \
Sleep Disorder
Insomnia                           0           29          0
Sleep Apnea                        2            1          2

Occupation      Software Engineer  Teacher
Sleep Disorder
Insomnia                        1       27
Sleep Apnea                     0        4
```



[43]:
```
plt.figure(figsize=(9, 7))
sns.pointplot(x='Occupation', y='Stress Level', data=df)
plt.title('Effect of Occupation on Stress Level')
plt.xticks(rotation=75)
plt.grid(axis='y', linestyle='-', alpha=0.7)
plt.show()
```



[45]:
```
occupations = df["Occupation"].unique()
num_occupations = len(occupations)
#Creating the subplot
fig, axes = plt.subplots(nrows=1, ncols=num_occupations,␣
 ↪figsize=(6*num_occupations, 6), sharey=True)
#Subplot for all occupation
for ax, occupation in zip(axes, occupations):
    sns.barplot(x="Sleep Disorder", y="Stress Level", hue="Gender", palette =␣
 ↪"PuRd", data=df[df["Occupation"] == occupation], ax=ax, ci=None)
    ax.set_title(occupation)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
```

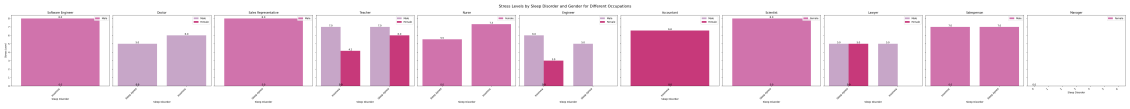
```
    ax.legend(loc='upper right')
    for p in ax.patches:
        height = p.get_height()
        ax.text(p.get_x() + p.get_width() / 2, height, f'{height:.1f}',␣
 ↪ha="center", va="bottom")
plt.suptitle('Stress Levels by Sleep Disorder and Gender for Different␣
 ↪Occupations', fontsize=16)
plt.tight_layout()
```

```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:
```

```
set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:
```

```
The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.
```

```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:7:
FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.


C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\46337155.py:9:
UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.
```



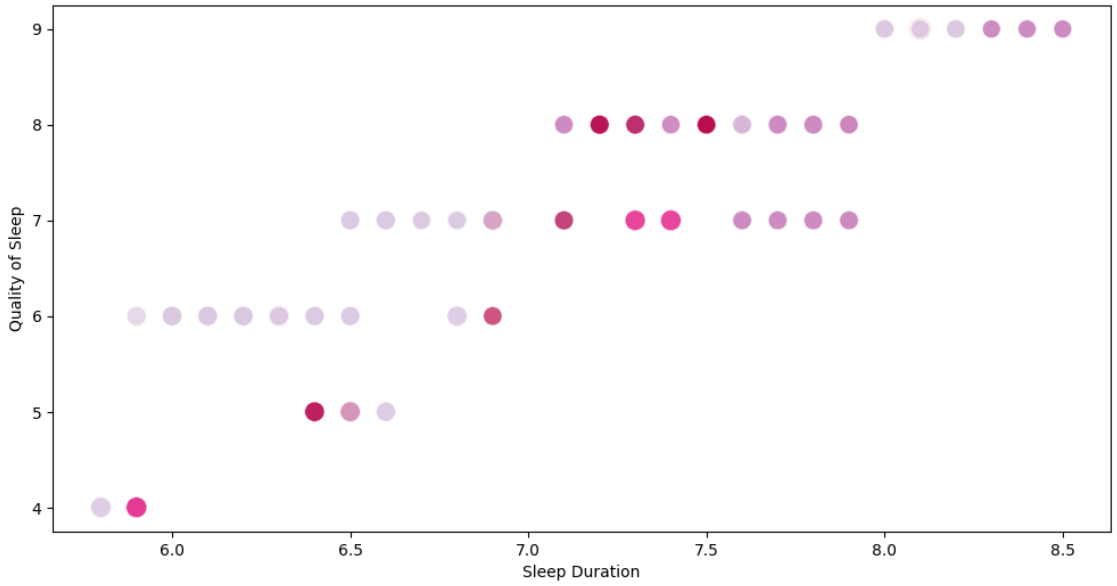
[48]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
#Ensuring that the Heart Rate is converted to numeric value before␣
 ↪multiplication
heart_rate_numeric = df['Heart Rate'].astype(float)  # Assuming Heart Rate is␣
 ↪numeric
scatter_plot = sns.scatterplot(
    x='Sleep Duration',
    y='Quality of Sleep',
    data=df,
    hue='BMI Category',
    palette='PuRd',
    s=heart_rate_numeric * 2,
    edgecolor='w',
    alpha=0.7,
    linewidth=0.5
)
plt.xticks(rotation=45)
```

```
plt.title('Relationship between Sleep Duration and Quality of Sleep by BMI␣
 ↪Category', fontweight="bold")
plt.xlabel('Sleep Duration', fontsize=12)
plt.ylabel('Quality of Sleep', fontsize=12)
plt.legend(title='BMI Category', fontsize=10)
plt.grid(True, linestyle='--', alpha=0.7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[48], line 6
      4 #Ensuring that the Heart Rate is converted to numeric value before␣
 ↪multiplication
      5 heart_rate_numeric = df['Heart Rate'].astype(float)  # Assuming Heart␣
 ↪Rate is numeric
----> 6 scatter_plot = sns.scatterplot(
      7     x='Sleep Duration',
      8     y='Quality of Sleep',
      9     data=df,
     10     hue='BMI Category',
     11     palette='PuRd',
     12     s=heart_rate_numeric * 2,
     13     edgecolor='w',
     14     alpha=0.7,
     15     linewidth=0.5
     16 )
     17 plt.xticks(rotation=45)
     18 plt.title('Relationship between Sleep Duration and Quality of Sleep by␣
 ↪BMI Category', fontweight="bold")

File␣
 ↪~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\relationa..
 ↪py:636, in scatterplot(data, x, y, hue, size, style, palette, hue_order,␣
 ↪hue_norm, sizes, size_order, size_norm, markers, style_order, legend, ax,␣
 ↪**kwargs)
    633 color = kwargs.pop("color", None)
    634 kwargs["color"] = _default_color(ax.scatter, hue, color, kwargs)
--> 636 p.plot(ax, kwargs)
    638 return ax

File␣
 ↪~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\relationa..
 ↪py:464, in _ScatterPlotter.plot(self, ax, kws)
    462 if self.legend:
    463     attrs = {"hue": "color", "size": "s", "style": None}
--> 464     self.add_legend_data(ax, _scatter_legend_artist, kws, attrs)
    465     handles, _ = ax.get_legend_handles_labels()
    466     if handles:
```

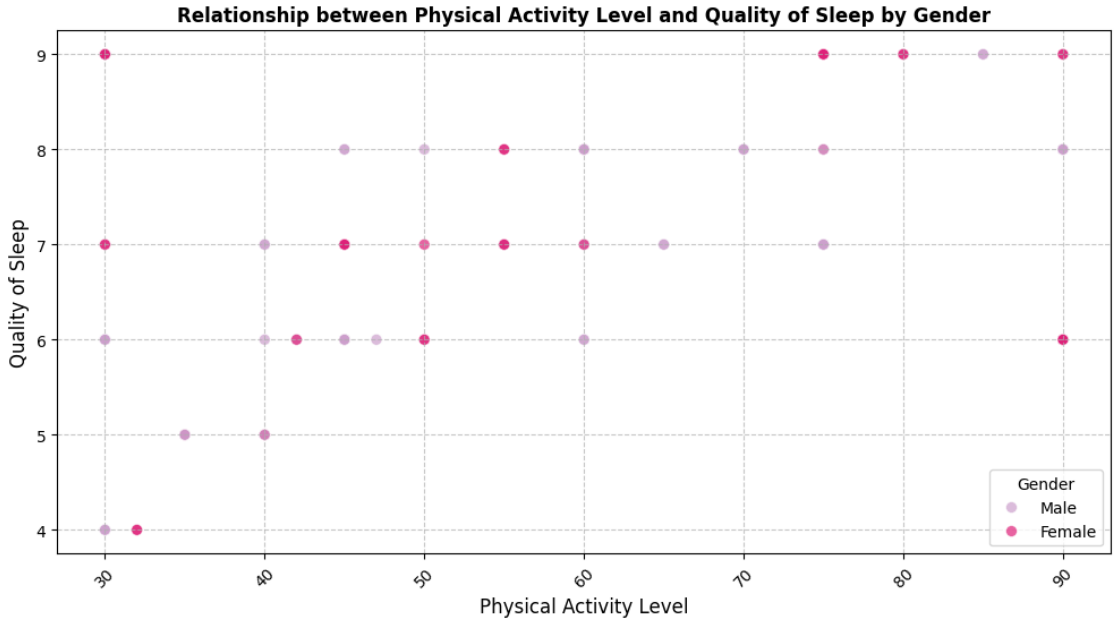
```
File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_base.
 ↪py:1267, in VectorPlotter.add_legend_data(self, ax, func, common_kws, attrs,␣
 ↪semantic_kws)
   1265     if attr in kws:
   1266         level_kws[attr] = kws[attr]
-> 1267 artist = func(label=label, **{"color": ".2", **common_kws, **level_kws}
   1268 if _version_predates(mpl, "3.5.0"):
   1269     if isinstance(artist, mpl.lines.Line2D):

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\utils.
 ↪py:880, in _scatter_legend_artist(**kws)
    877     else:
    878         line_kws["markeredgecolor"] = edgecolor
--> 880 return mpl.lines.Line2D([], [], **line_kws)

File␣
 ↪~\AppData\Local\Programs\Python\Python311\Lib\site-packages\matplotlib\lines.
 ↪py:393, in Line2D.__init__(self, xdata, ydata, linewidth, linestyle, color,␣
 ↪gapcolor, marker, markersize, markeredgewidth, markeredgecolor,␣
 ↪markerfacecolor, markerfacecoloralt, fillstyle, antialiased, dash_capstyle,␣
 ↪solid_capstyle, dash_joinstyle, solid_joinstyle, pickradius, drawstyle,␣
 ↪markevery, **kwargs)
    391 self.set_markevery(markevery)
    392 self.set_antialiased(antialiased)
--> 393 self.set_markersize(markersize)
    395 self._markeredgecolor = None
    396 self._markeredgewidth = None

File␣
 ↪~\AppData\Local\Programs\Python\Python311\Lib\site-packages\matplotlib\lines.
 ↪py:1270, in Line2D.set_markersize(self, sz)
   1261 def set_markersize(self, sz):
   1262     """
   1263     Set the marker size in points.
   1264
   (…)
   1268         Marker size, in points.
   1269     """
-> 1270     sz = float(sz)
   1271     if self._markersize != sz:
   1272         self.stale = True

File␣
 ↪~\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\serie
 ↪py:248, in _coerce_method.<locals>.wrapper(self)
    240     warnings.warn(
    241         f"Calling {converter.__name__} on a single element Series is "
    242         "deprecated and will raise a TypeError in the future. "
   (…)
    245         stacklevel=find_stack_level(),
```

```
    246     )
    247     return converter(self.iloc[0])
--> 248 raise TypeError(f"cannot convert the series to {converter}")

TypeError: cannot convert the series to <class 'float'>
```

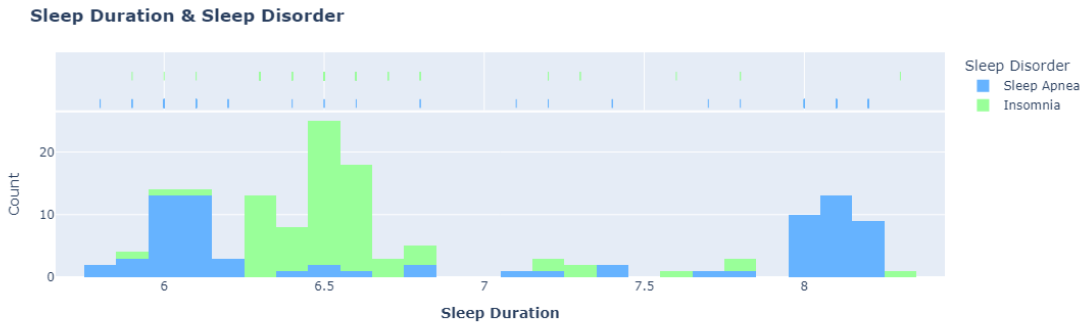


[49]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
#Assuming that we want marker size to be constant for all data points
scatter_plot = sns.scatterplot(
    x='Physical Activity Level',
    y='Quality of Sleep',
    data=df,
    hue='Gender',
    palette='PuRd',
    s=50,
    edgecolor='w',
    alpha=0.7,
    linewidth=0.5
)
plt.xticks(rotation=45)
plt.title('Relationship between Physical Activity Level and Quality of Sleep by␣
 ↪Gender', fontweight="bold")
plt.xlabel('Physical Activity Level', fontsize=12)
plt.ylabel('Quality of Sleep', fontsize=12)
plt.legend(title='Gender', fontsize=10)
```

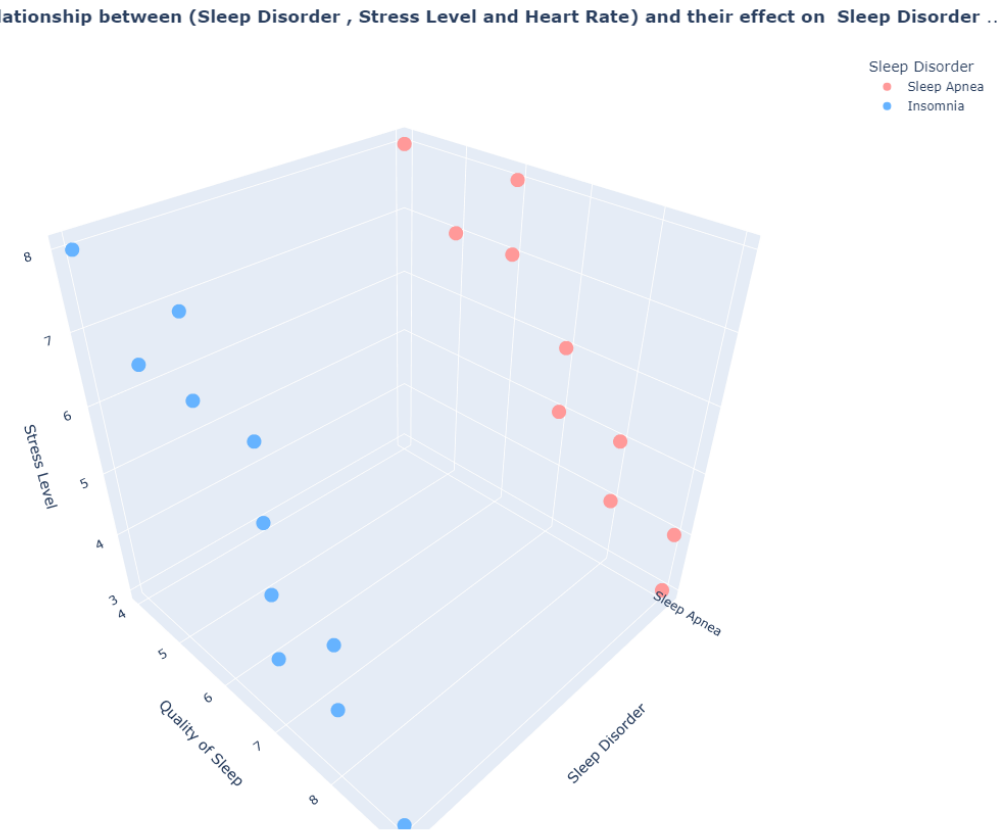
```
plt.grid(True, linestyle='--', alpha=0.7)
```



[53]:
```
my_palette = ["#ff9999", "#66b3ff", "#99ff99", "#ffcc99", "#c2c2f0", "#fdc086",␣
 ↪"#f26d6d", "#d44e4e", "#9e4e4e"]
fig = px.histogram(df, x='Sleep Duration', color='Sleep Disorder',␣
 ↪marginal='rug', nbins=30,color_discrete_sequence=my_palette)
fig.update_layout(title='<b>Sleep Duration & Sleep Disorder<b>',
                  xaxis=dict(title='<b>Sleep Duration<b>'),
                  yaxis=dict(title='Count'),
                  legend=dict(title='Sleep Disorder'),
                  showlegend=True)
fig.show()
```



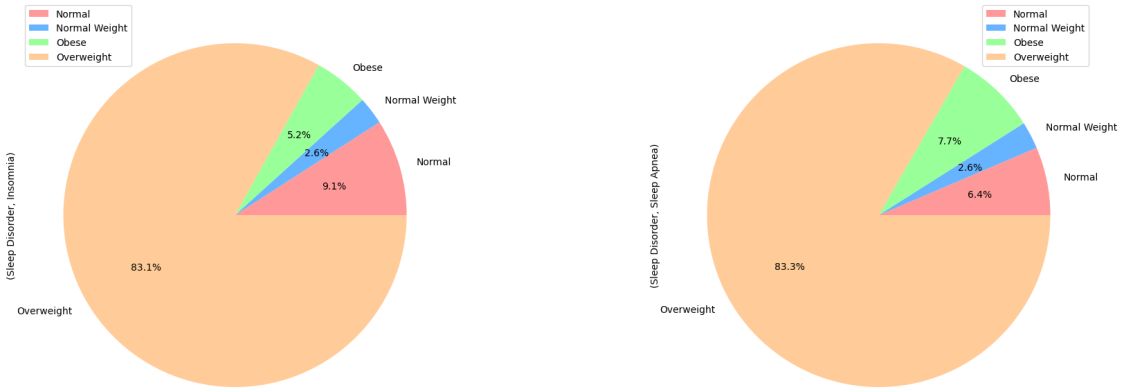
[54]:
```
fig=px.scatter_3d(df,x='Sleep Disorder',y='Quality of Sleep',z='Stress Level',
                  color='Sleep Disorder',width=1000,height=900,
                  color_discrete_sequence=['white','#ff9999', '#66b3ff'])
fig.update_layout(title='<b>The relationship between (Sleep Disorder , Stress␣
 ↪Level and Heart Rate) and their effect on  Sleep Disorder</b> ..',
                  showlegend=True)
fig.show()
```



[55]:
```
import matplotlib.pyplot as plt
#Pivot table
pivot_df = df.pivot_table(index='BMI Category',
                          columns='Sleep Disorder',
                          aggfunc={'Sleep Disorder':'count'})
#Pie chart
fig, ax = plt.subplots(figsize=(20, 10))
pivot_df.plot.pie(autopct ='%1.1f%%',
                  subplots=True,
                  ax=ax,
```

```
                    colors=["#ff9999", "#66b3ff", "#99ff99", "#ffcc99"])
#Adjusting the spacing between subplots
plt.subplots_adjust(wspace=0.5)
```

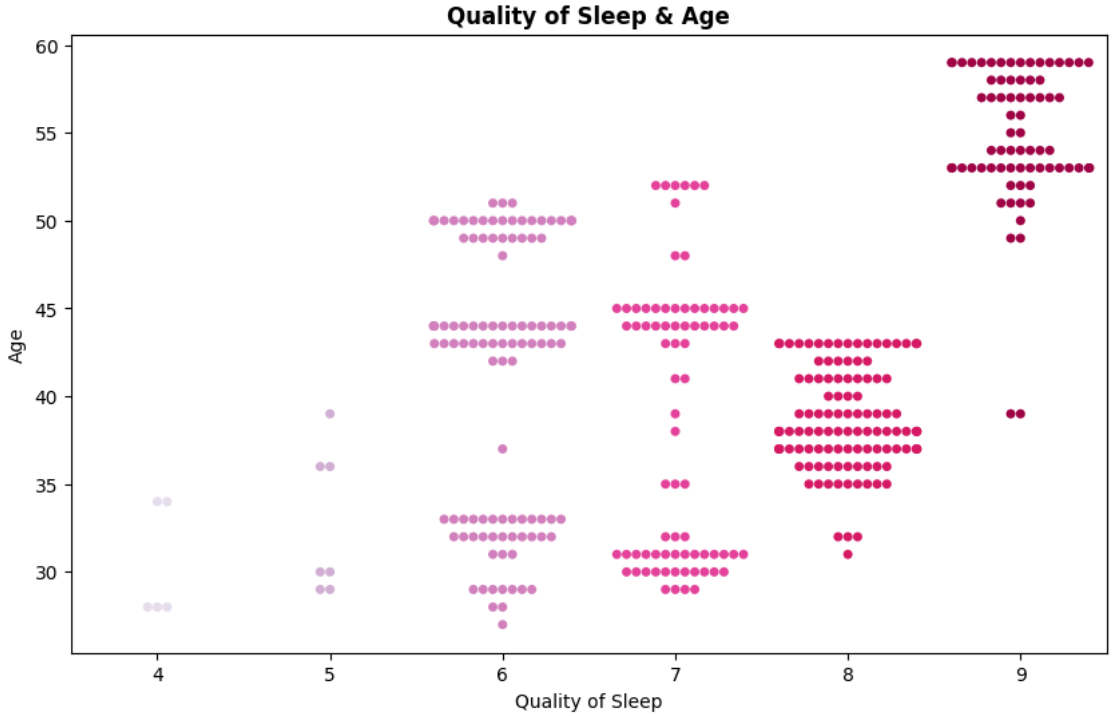
```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\680864722.py:8:
UserWarning:

To output multiple subplots, the figure containing the passed axes is being
cleared.
```



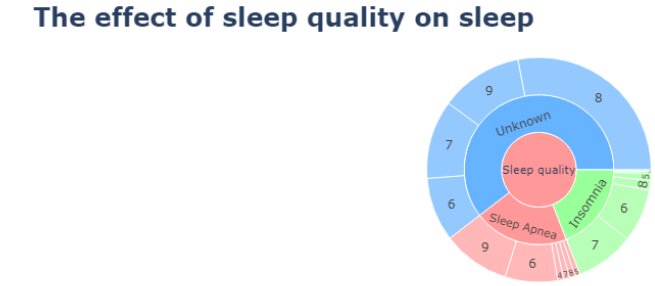
[56]:
```
plt.figure(figsize=(10, 6))
sns.swarmplot(x='Quality of Sleep', y='Age', data=df,palette="PuRd")
plt.title('Quality of Sleep & Age', fontweight='bold')
plt.xlabel('Quality of Sleep')
plt.ylabel('Age')
plt.show()
```

```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_24052\2352893511.py:2:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\categorical.py:3399: UserWarning:

6.7% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.
```

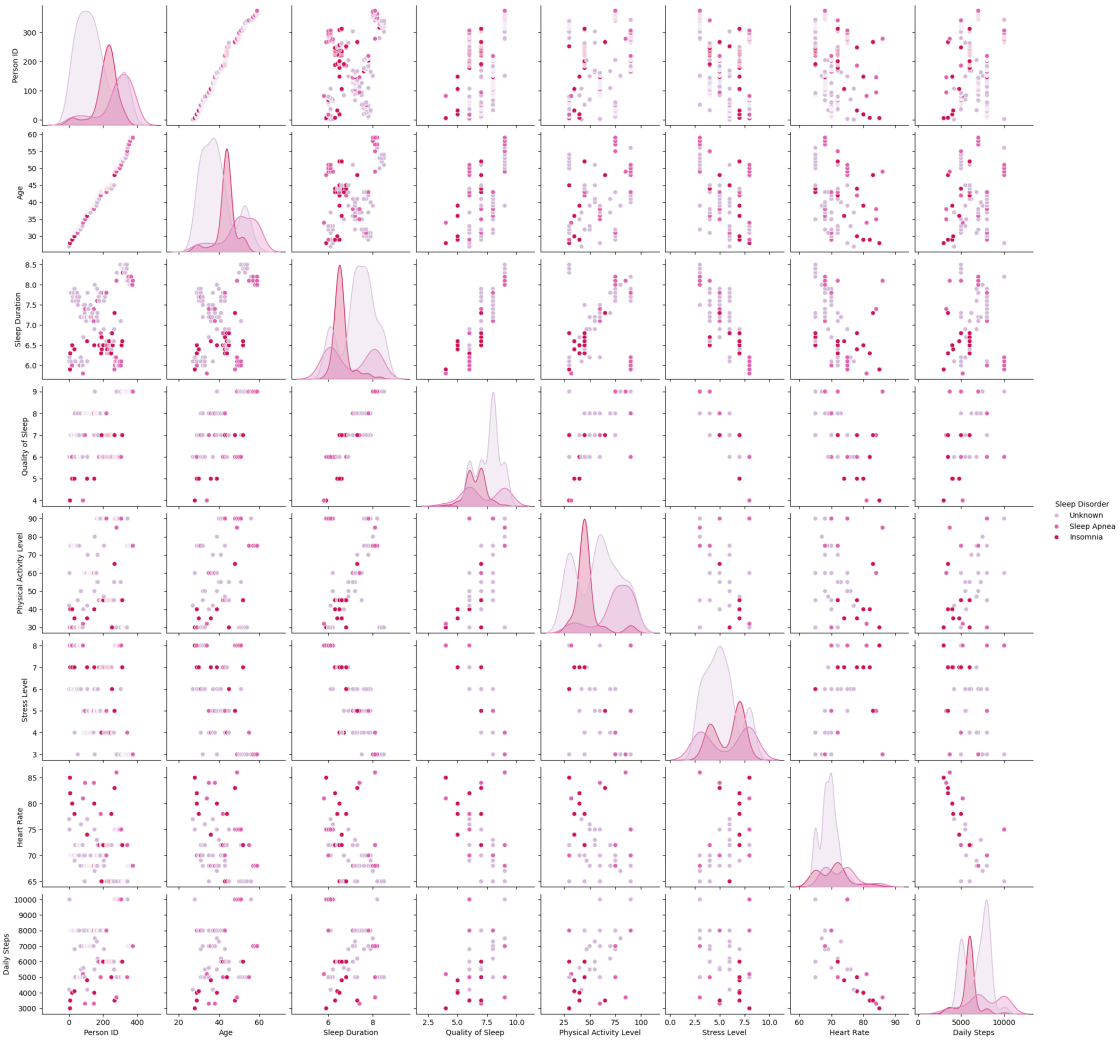
```
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\categorical.py:3399: UserWarning:

9.2% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.
```



[59]:
```
import pandas as pd
import plotly.express as px
df["Sleep Disorder"] = df["Sleep Disorder"].fillna("Unknown")  # Replace None␣
 ↪with Unknown
fig = px.sunburst(df, path=[px.Constant('Sleep quality'), 'Sleep Disorder',␣
 ↪'Quality of Sleep'],
                  color='Sleep Disorder', values='Sleep Duration',
                  color_discrete_sequence=["#ff9999", "#66b3ff", "#99ff99"])
fig.update_layout(title='<b>The effect of sleep quality on sleep </b>',
                  title_font={'size': 25})
fig.show()
```



[60]:
```
sns.pairplot(df,hue="Sleep Disorder",palette="PuRd");
```



[5]:
```
import pandas as pd
from scipy import stats
def detect_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return outliers
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
for column in numeric_columns:
    outliers_iqr = detect_outliers_iqr(df, column)
    if not outliers_iqr.empty:
        outlier_percentage_iqr = len(outliers_iqr) / len(df) * 100
        print(f"There are outliers in the {column} column. Outlier percentage:␣
 ↪{outlier_percentage_iqr:.2f}%")
```

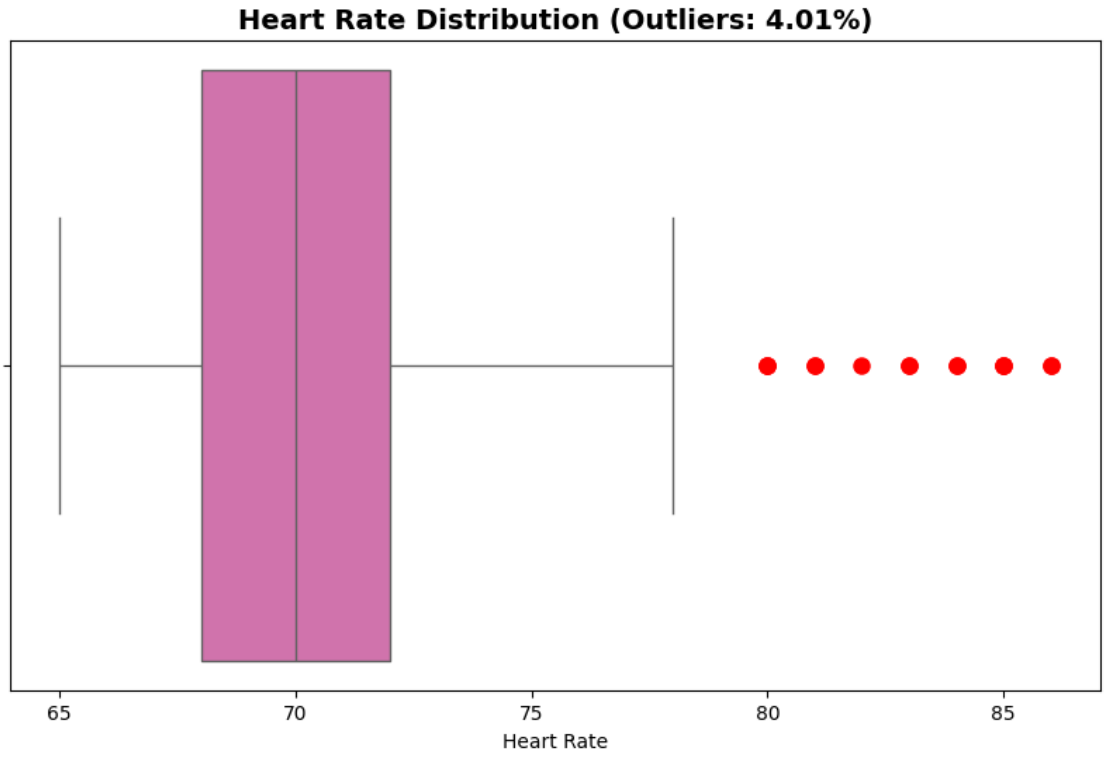
```
There are outliers in the Heart Rate column. Outlier percentage: 4.01%
```

[7]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#Calculating the outliers
Q1 = df['Heart Rate'].quantile(0.25)
print("Q1:", Q1)
Q3 = df['Heart Rate'].quantile(0.75)
print("Q3:", Q3)
IQR = Q3 - Q1
print("IQR: ", IQR)
lower_bound = Q1 - 1.5 * IQR
print("Lower Bound: ", lower_bound)
upper_bound = Q3 + 1.5 * IQR
print("Upper Bound: ", upper_bound)
#Identifying outliers
outliers = df[(df['Heart Rate'] < lower_bound) | (df['Heart Rate'] >␣
 ↪upper_bound)]
outlier_percentage = len(outliers) / len(df) * 100
#Creating the boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x=df['Heart Rate'], palette="PuRd")
#Title with outlier percentage
plt.title(f'Heart Rate Distribution (Outliers: {outlier_percentage:.2f}%)',␣
 ↪fontsize=14, fontweight='bold')
#Highlighting outliers
```

```
for outlier in outliers['Heart Rate']:
    plt.plot(outlier, 0, 'ro', markersize=8)
plt.show()
```

```
Q1: 68.0
Q3: 72.0
IQR:  4.0
Lower Bound:  62.0
Upper Bound:  78.0
```

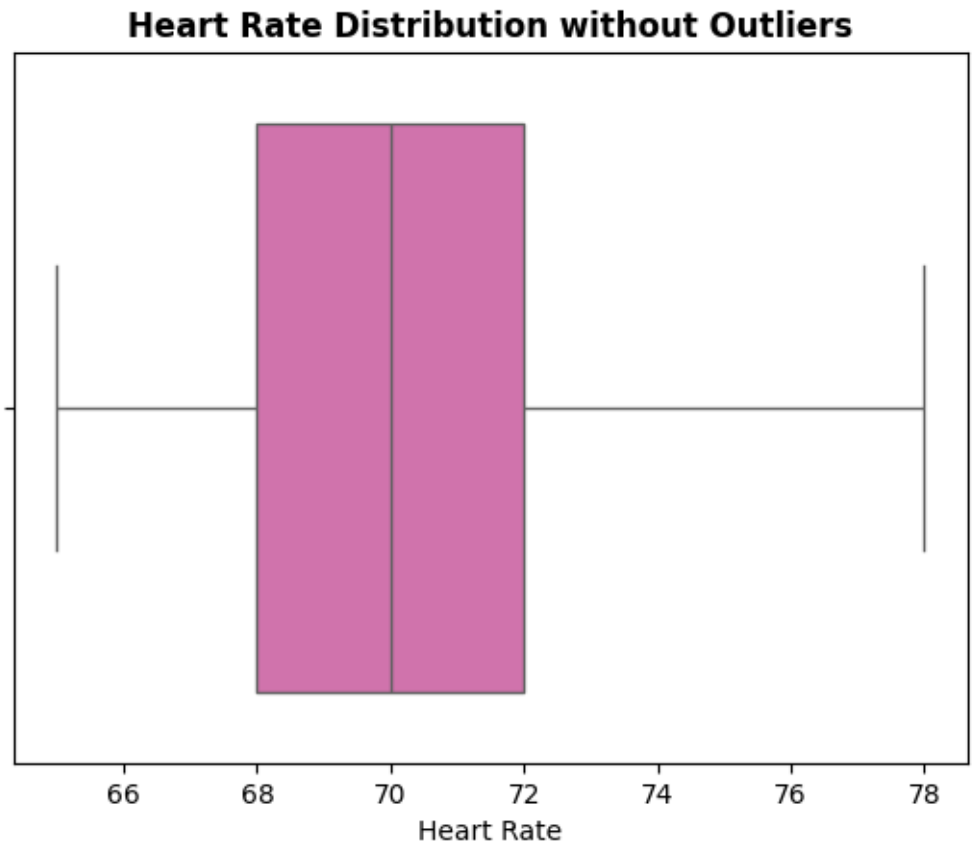
```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_6132\1238621586.py:20:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x=df['Heart Rate'], palette="PuRd")
```



[9]:
```
import numpy as np
df['Heart Rate'] = np.where(df['Heart Rate'] < lower_bound, lower_bound,␣
 ↪df['Heart Rate'])
```

```
df['Heart Rate'] = np.where(df['Heart Rate'] > upper_bound, upper_bound,␣
 ↪df['Heart Rate'])
sns.boxplot(x=df['Heart Rate'], palette="PuRd")
plt.title(f'Heart Rate Distribution without Outliers', fontweight='bold');
```

```
C:\Users\Rvdeekshitha\AppData\Local\Temp\ipykernel_6132\4124537699.py:4:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x=df['Heart Rate'], palette="PuRd")
```



[8]:
```
import pandas as pd
def detect_rare_categories(df, column, threshold=0.05):
    category_counts = df[column].value_counts(normalize=True)
    rare_categories = category_counts[category_counts < threshold]
    return rare_categories
```

```
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
categorical_columns = df.select_dtypes(include=['object']).columns
for column in categorical_columns:
    df[column] = df[column].astype('category')
for column in df.columns:
    if df[column].dtype == 'category':
        rare_categories = detect_rare_categories(df, column)
        if not rare_categories.empty:
            print(f"Rare categories in the {column} column:")
            print(rare_categories)
            print("")
```

```
Rare categories in the Occupation column:
Occupation
Scientist               0.010695
Software Engineer       0.010695
Sales Representative    0.005348
Manager                 0.002674
Name: proportion, dtype: float64

Rare categories in the BMI Category column:
BMI Category
Obese    0.026738
Name: proportion, dtype: float64

Rare categories in the Blood Pressure column:
Blood Pressure
125/82    0.010695
140/90    0.010695
132/87    0.008021
128/85    0.008021
130/86    0.005348
126/83    0.005348
129/84    0.005348
117/76    0.005348
118/75    0.005348
115/78    0.005348
119/77    0.005348
135/88    0.005348
131/86    0.005348
139/91    0.005348
128/84    0.005348
142/92    0.005348
122/80    0.002674
118/76    0.002674
121/79    0.002674
Name: proportion, dtype: float64
```

[5]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Checking for categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns
#Performing the label encoding
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Printing the encoded DataFrame
print(df.head())
#Printing the mapping between original categories and encoded values
for col in categorical_columns:
    print(f"Column: {col}")
    print(le.classes_)
    print()
```

```
   Person ID  Gender  Age  Occupation  Sleep Duration  Quality of Sleep  \
0          1       1   27           9             6.1                 6
1          2       1   28           1             6.2                 6
2          3       1   28           1             6.2                 6
3          4       1   28           6             5.9                 4
4          5       1   28           6             5.9                 4

   Physical Activity Level  Stress Level  BMI Category  Blood Pressure  \
0                       42             6             3              11
1                       60             8             0               9
2                       60             8             0               9
3                       30             8             2              22
4                       30             8             2              22

   Heart Rate  Daily Steps  Sleep Disorder
0          77         4200               2
1          75        10000               2
2          75        10000               2
3          85         3000               1
4          85         3000               1
Column: Gender
['Insomnia' 'Sleep Apnea' nan]

Column: Occupation
['Insomnia' 'Sleep Apnea' nan]

Column: BMI Category
['Insomnia' 'Sleep Apnea' nan]
```

```
Column: Blood Pressure
['Insomnia' 'Sleep Apnea' nan]

Column: Sleep Disorder
['Insomnia' 'Sleep Apnea' nan]
```

[6]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in df:
    df[col] = le.fit_transform(df[col])
df
```

[6]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	\
0	0	1	0	9	3	2	
1	1	1	1	1	4	2	
2	2	1	1	1	4	2	
3	3	1	1	6	1	0	
4	4	1	1	6	1	0	
..	…	…	…	…	…	…	
369	369	0	30	5	22	5	
370	370	0	30	5	21	5	
371	371	0	30	5	22	5	
372	372	0	30	5	22	5	
373	373	0	30	5	22	5	

	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	\
0	4	3	3	11	
1	9	5	0	9	
2	9	5	0	9	
3	0	5	2	22	
4	0	5	2	22	
..	…	…	…	…	
369	12	0	3	23	
370	12	0	3	23	
371	12	0	3	23	
372	12	0	3	23	
373	12	0	3	23	

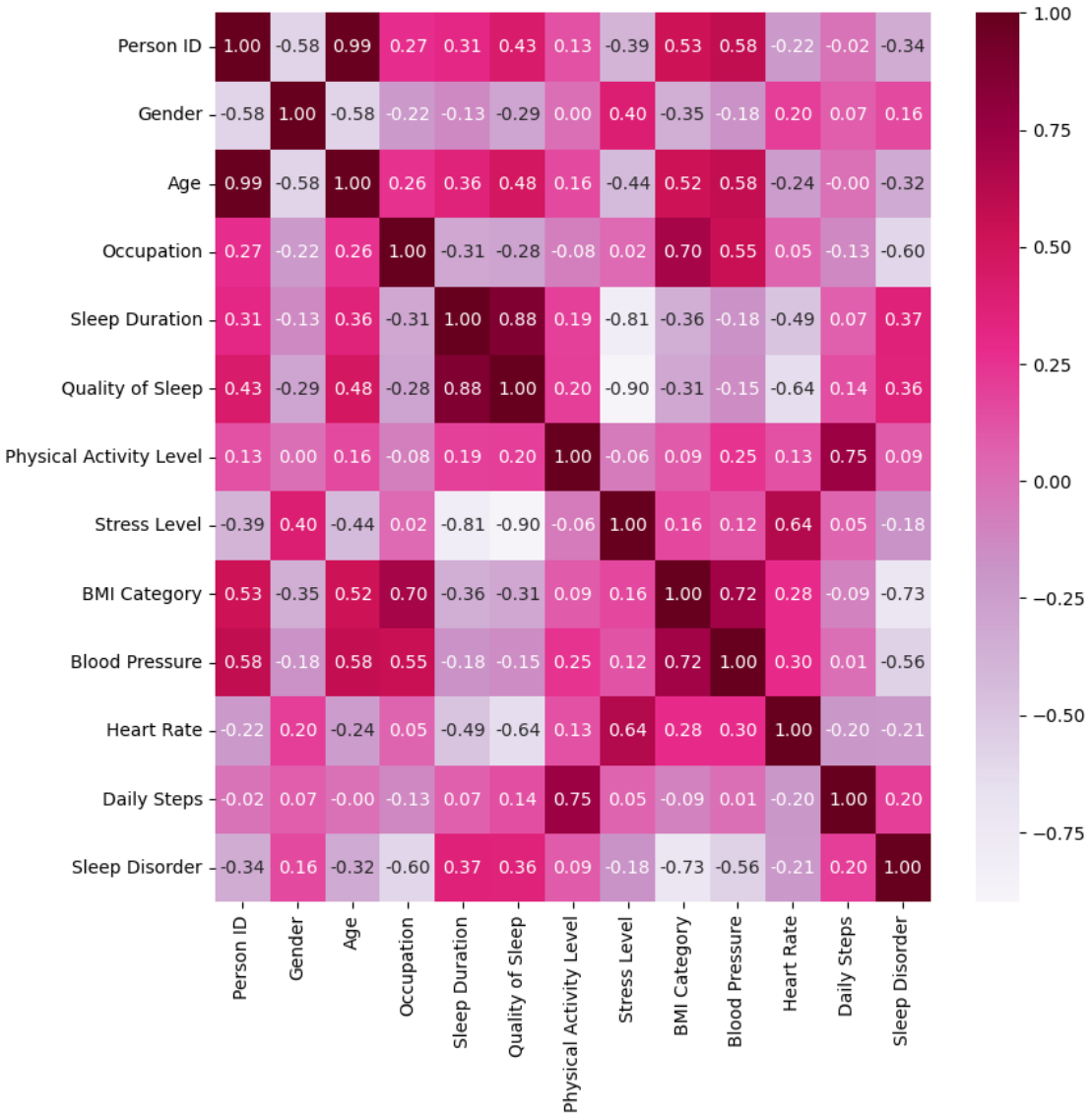
	Heart Rate	Daily Steps	Sleep Disorder
0	10	6	2
1	8	19	2
2	8	19	2
3	17	0	1
4	17	0	1
..	…	…	…

```
369          2           15              1
370          2           15              1
371          2           15              1
372          2           15              1
373          2           15              1

[374 rows x 13 columns]
```

[7]:
```
df_corr = df.corr()
df_corr["Sleep Duration"].sort_values(ascending = False)
```

[7]:
```
Sleep Duration             1.000000
Quality of Sleep           0.879352
Sleep Disorder             0.367677
Age                        0.356898
Person ID                  0.311105
Physical Activity Level    0.192503
Daily Steps                0.072468
Gender                    -0.129468
Blood Pressure            -0.179067
Occupation                -0.312653
BMI Category              -0.360772
Heart Rate                -0.488306
Stress Level              -0.810712
Name: Sleep Duration, dtype: float64
```

[10]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (9, 9))
sns.heatmap(df.corr(numeric_only=True),fmt = ".2f",annot=True,cmap="PuRd");
plt.show()
```



[11]:
```
#Feature-Scaling
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
numeric_features = df.select_dtypes(include=['int64', 'float64']).columns
# Normalization
scaler = MinMaxScaler()
df[numeric_features] = scaler.fit_transform(df[numeric_features])
```

[13]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
#Assuming your DataFrame is named 'df' and 'Sleep Disorder' is the target column
X = df.drop('Sleep Disorder', axis=1)  # Features
y = df['Sleep Disorder']  # Target variable
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
print("X_train:")
print(X_train.head())
print("\nX_test:")
print(X_test.head())
print("\ny_train:")
print(y_train.head())
print("\ny_test:")
print(y_test.head())
```

```
X_train:
     Person ID  Gender     Age         Occupation  Sleep Duration  \
192   0.514745    Male  0.5000        Salesperson        0.259259
75    0.201072    Male  0.1875             Doctor        0.074074
84    0.225201    Male  0.2500  Software Engineer        0.629630
362   0.970509  Female  1.0000              Nurse        0.888889
16    0.042895  Female  0.0625              Nurse        0.259259

     Quality of Sleep  Physical Activity Level  Stress Level   BMI Category  \
192               0.4                 0.250000           0.8     Overweight
75                0.4                 0.000000           1.0         Normal
84                0.8                 0.500000           0.4  Normal Weight
362               1.0                 0.750000           0.0     Overweight
16                0.2                 0.166667           0.8  Normal Weight

    Blood Pressure  Heart Rate  Daily Steps
192         130/85    0.333333     0.428571
75          125/80    0.333333     0.285714
84          120/80    0.238095     0.714286
362         140/95    0.142857     0.571429
16          132/87    0.714286     0.142857

X_test:
     Person ID  Gender      Age Occupation  Sleep Duration  Quality of Sleep  \
329   0.882038  Female  0.81250   Engineer        1.000000               1.0
33    0.088472    Male  0.12500     Doctor        0.111111               0.4
15    0.040214    Male  0.06250     Doctor        0.074074               0.4
325   0.871314  Female  0.81250   Engineer        1.000000               1.0
57    0.152815    Male  0.15625     Doctor        0.074074               0.4

     Physical Activity Level  Stress Level BMI Category Blood Pressure  \
329                      0.0           0.0       Normal         125/80
33                       0.0           1.0       Normal         125/80
```

```
15                      0.0           1.0       Normal         120/80
325                     0.0           0.0       Normal         125/80
57                      0.0           1.0       Normal         125/80

     Heart Rate  Daily Steps
329    0.000000     0.285714
33     0.333333     0.285714
15     0.238095     0.714286
325    0.000000     0.285714
57     0.333333     0.285714

y_train:
192       Insomnia
75             NaN
84             NaN
362    Sleep Apnea
16     Sleep Apnea
Name: Sleep Disorder, dtype: object

y_test:
329    NaN
33     NaN
15     NaN
325    NaN
57     NaN
Name: Sleep Disorder, dtype: object
```

[19]:
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
#Load your data
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify numerical and categorical columns
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns
#Handle categorical features
le = LabelEncoder()
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)  # Assuming 'Sleep Disorder' is your␣
 ↪target
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
#Create a scaler object
scaler = StandardScaler()
#Fit the scaler to the training data on numerical columns only
scaler.fit(X_train[numeric_cols])
#Transform both training and testing data on numerical columns only
X_train_scaled = scaler.transform(X_train[numeric_cols])
X_test_scaled = scaler.transform(X_test[numeric_cols])
#Combine scaled numerical features with original categorical features
X_train_scaled_with_categorical = pd.concat([pd.DataFrame(X_train_scaled,␣
 ↪columns=numeric_cols), df[categorical_cols]], axis=1)
X_test_scaled_with_categorical = pd.concat([pd.DataFrame(X_test_scaled,␣
 ↪columns=numeric_cols), df[categorical_cols]], axis=1)
#Print the scaled data (optional)
print("Scaled Training Data:")
print(X_train_scaled_with_categorical.head())
print("Scaled Testing Data:")
print(X_test_scaled_with_categorical.head())
```

```
Scaled Training Data:
   Person ID       Age  Sleep Duration  Quality of Sleep  \
0   0.013228  0.055811       -0.833457         -1.155098
1  -1.083373 -1.111141       -1.468533         -1.155098
2  -0.999019 -0.877751        0.436694          0.537913
3   1.606580  1.922934        1.325801          1.384419
4  -1.636359 -1.577922       -0.833457         -2.001604

   Physical Activity Level  Stress Level  Heart Rate  Daily Steps  Gender  \
0                -0.728644      0.956601    0.481081    -0.554643       1
1                -1.450369      1.522984    0.481081    -1.180447       1
2                -0.006920     -0.176166   -0.014076     0.696966       1
3                 0.714805     -1.308933   -0.509234     0.071162       1
4                -0.969219      0.956601    2.461710    -1.806251       1

   Occupation  BMI Category  Blood Pressure  Sleep Disorder
0           9             3              11               2
1           1             0               9               2
2           1             0               9               2
3           6             2              22               1
4           6             2              22               1
Scaled Testing Data:
   Person ID       Age  Sleep Duration  Quality of Sleep  \
0   1.297282  1.222763        1.706846          1.384419
1  -1.477024 -1.344532       -1.341518         -1.155098
2  -1.645732 -1.577922       -1.468533         -1.155098
3   1.259792  1.222763        1.706846          1.384419
4  -1.252080 -1.227836       -1.468533         -1.155098
```

```
   Physical Activity Level  Stress Level  Heart Rate  Daily Steps  Gender  \
0                -1.450369     -1.308933   -1.251970    -1.180447       1
1                -1.450369      1.522984    0.481081    -1.180447       1
2                -1.450369      1.522984   -0.014076     0.696966       1
3                -1.450369     -1.308933   -1.251970    -1.180447       1
4                -1.450369      1.522984    0.481081    -1.180447       1

   Occupation  BMI Category  Blood Pressure  Sleep Disorder
0           9             3              11               2
1           1             0               9               2
2           1             0               9               2
3           6             2              22               1
4           6             2              22               1
```

[27]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Load your data
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify categorical features (assuming the Sleep Disorder column is not␣
 ↪categorical)
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features using LabelEncoder
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Create a Decision Tree classifier
dt_model = DecisionTreeClassifier()
#Train the model
dt_model.fit(X_train, y_train)
#Make predictions on the testing set
y_pred = dt_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("Decision Tree Model:")
```

```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Decision Tree Model:
Accuracy: 0.9066666666666666
Precision: 0.8750661375661376
Recall: 0.8541666666666666
F1-score: 0.8632575757575758
```

[30]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Load your data
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Create a Random Forest classifier
rf_model = RandomForestClassifier()
#Train the model
rf_model.fit(X_train, y_train)
#Make predictions on the testing set
y_pred = rf_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')  # Corrected call
print("Random Forest Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Random Forest Model:
Accuracy: 0.88
Precision: 0.8409738409738411
Recall: 0.8255813953488372
F1-score: 0.8296146044624746
```

[33]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Load your data
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Create an SVM classifier
svm_model = SVC()
```

```
#Train the model
svm_model.fit(X_train, y_train)
#Make predictions on the testing set
y_pred = svm_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')  # Corrected call
print("SVM Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
SVM Model:
Accuracy: 0.64
Precision: 0.6441176470588236
Recall: 0.4375
F1-score: 0.4222794222794222
```

[34]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Load your data
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Create a Naive Bayes classifier
nb_model = GaussianNB()
#Train the model
nb_model.fit(X_train, y_train)
#Make predictions on the testing set
y_pred = nb_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("Naive Bayes Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Naive Bayes Model:
Accuracy: 0.8666666666666667
Precision: 0.8312447786131996
Recall: 0.8309108527131782
```

```
F1-score: 0.8280112044817928
```

[35]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Load your data
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Create a Logistic Regression classifier
lr_model = LogisticRegression()
#Train the model
lr_model.fit(X_train, y_train)
#Make predictions on the testing set
y_pred = lr_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("Logistic Regression Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Logistic Regression Model:
Accuracy: 0.84
Precision: 0.8194570287593543
Recall: 0.7892441860465116
F1-score: 0.7912253338609928

C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression  # Adjust max_iter if needed
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Load your data
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Encode categorical features
categorical_columns = df.select_dtypes(include=['object']).columns
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Create and train models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),  # Increase␣
 ↪iterations if needed
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1_score = f1_score(y_test, y_pred, average='macro')

    print(f"{model_name}:")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1-score: {f1_score:.2f}")
    print()
```

```
Logistic Regression:
Accuracy: 0.91
Precision: 0.89
Recall: 0.87
F1-score: 0.87
```

```
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[2], line 35
     33 precision = precision_score(y_test, y_pred, average='macro')
     34 recall = recall_score(y_test, y_pred, average='macro')
---> 35 f1_score = f1_score(y_test, y_pred, average='macro')
     37 print(f"{model_name}:")
     38 print(f"Accuracy: {accuracy:.2f}")

TypeError: 'numpy.float64' object is not callable
```

[6]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Loading the dataset
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify categorical features
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Define hyperparameter grid
```

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}
#Create a Random Forest classifier
rf_model = RandomForestClassifier()
#Create a GridSearchCV object
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy')  #␣
 ↪Adjust scoring metric as needed
#Fit the grid search to the training data
grid_search.fit(X_train, y_train)
#Get the best parameters and model
best_params = grid_search.best_params_
best_rf_model = grid_search.best_estimator_
#Make predictions using the best model
y_pred = best_rf_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("Random Forest Model (Grid Search):")
print("Best Parameters:", best_params)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Random Forest Model (Grid Search):
Best Parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimators':
300}
Accuracy: 0.88
Precision: 0.8409738409738411
Recall: 0.8255813953488372
F1-score: 0.8296146044624746
```

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Loading the dataset
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify categorical features
```

```
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Define hyperparameter distributions
param_distributions = {
    'n_estimators': range(100, 500, 50),
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': range(2, 11)
}
#Create a Random Forest classifier
rf_model = RandomForestClassifier()
#Create a RandomizedSearchCV object
random_search = RandomizedSearchCV(rf_model, param_distributions, n_iter=100,␣
 ↪cv=5, scoring='accuracy')
#Fit the random search to the training data
random_search.fit(X_train, y_train)
#Get the best parameters and model
best_params = random_search.best_params_
best_rf_model = random_search.best_estimator_
#Make predictions using the best model
y_pred = best_rf_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("Random Forest Model (Random Search):")
print("Best Parameters:", best_params)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Random Forest Model (Random Search):
Best Parameters: {'n_estimators': 100, 'min_samples_split': 8, 'max_depth':
None}
Accuracy: 0.88
Precision: 0.8409738409738411
Recall: 0.8255813953488372
```

```
F1-score: 0.8296146044624746
```

[5]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score  # Import necessary metrics
from sklearn.model_selection import cross_val_score
from hyperopt import fmin, tpe, Trials, hp
#Loading the dataset
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify categorical features
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Define objective function for Bayesian Optimization
def objective(params):
    model = RandomForestClassifier(**params)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    return -scores.mean()  # Minimize negative accuracy
#Define parameter space
space = {
    'n_estimators': hp.choice('n_estimators', [100, 200, 300]),  # Adjust␣
 ↪choices if needed
    'max_depth': hp.choice('max_depth', [None, 5, 10]),
    'min_samples_split': hp.choice('min_samples_split', [2, 5, 10])
}
#Run Bayesian Optimization
trials = Trials()
best_params = fmin(objective, space, algo=tpe.suggest, max_evals=10,␣
 ↪trials=trials)
#Ensure valid hyperparameters
if 'n_estimators' in best_params:
    best_params['n_estimators'] = int(max(1, best_params['n_estimators']))
if 'min_samples_split' in best_params:
    best_params['min_samples_split'] = max(2,␣
 ↪int(best_params['min_samples_split']))
```

```
#Create the best model
best_rf_model = RandomForestClassifier(**best_params)
best_rf_model.fit(X_train, y_train)
#Make predictions and evaluate
y_pred = best_rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("Random Forest Model (Bayesian Optimization):")
print("Best Parameters:", best_params)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
100%|                          | 10/10 [00:29<00:00,
2.91s/trial, best loss: -0.9163841807909605]
Random Forest Model (Bayesian Optimization):
Best Parameters: {'max_depth': np.int64(1), 'min_samples_split': 2,
'n_estimators': 1}
Accuracy: 0.76
Precision: 0.49393939393939396
Recall: 0.625
F1-score: 0.5517762660619804

C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[9]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Loading the dataset
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify categorical features
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
```

```
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Create a base model (e.g., Decision Tree)
base_model = DecisionTreeClassifier()
#Create a BaggingClassifier (using 'estimator' argument)
bagging_model = BaggingClassifier(estimator=base_model, n_estimators=100)
#Train the model
bagging_model.fit(X_train, y_train)
#Make predictions
y_pred = bagging_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("Bagging Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Bagging Model:
Accuracy: 0.8933333333333333
Precision: 0.8587980646804176
Recall: 0.8464147286821705
F1-score: 0.8511320097526994
```

[12]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Loading the dataset
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify categorical features
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
```

```
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Create a base model (e.g., Decision Tree)
base_model = DecisionTreeClassifier()
#Create an AdaBoostClassifier (using 'estimator' argument)
ada_boost_model = AdaBoostClassifier(estimator=base_model, n_estimators=100)
#Train the model
ada_boost_model.fit(X_train, y_train)
#Make predictions
y_pred = ada_boost_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("AdaBoost Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
AdaBoost Model:
Accuracy: 0.9066666666666666
Precision: 0.8750661375661376
Recall: 0.8541666666666666
F1-score: 0.8632575757575758
```

```
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

[15]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression  # Corrected import
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Loading the dataset
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
```

```
#Identify categorical features
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Create base models
base_models = [
    ('rf', RandomForestClassifier()),
    ('lr', LogisticRegression()),
    ('svm', SVC())
]
#Create a meta-model (e.g., Logistic Regression)
meta_model = LogisticRegression()
#Create a StackingClassifier
stacking_model = StackingClassifier(estimators=base_models,␣
 ↪final_estimator=meta_model)
#Train the model
stacking_model.fit(X_train, y_train)
#Make predictions
y_pred = stacking_model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
print("Stacking Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
```

```
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

Stacking Model:
Accuracy: 0.88
Precision: 0.8409738409738411
Recall: 0.8255813953488372
F1-score: 0.8296146044624746
```

[18]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix
#Loading the dataset
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify categorical features
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Create and train a Random Forest model (replace with your desired model)
model = RandomForestClassifier()
model.fit(X_train, y_train)
#Make predictions
y_pred = model.predict(X_test)
#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1_score = f1_score(y_test, y_pred, average='macro')
confusion_mat = confusion_matrix(y_test, y_pred)
```

```
#Printing the results
print("Model Evaluation:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
print("Confusion Matrix:\n", confusion_mat)
```

```
Model Evaluation:
Accuracy: 0.88
Precision: 0.8409738409738411
Recall: 0.8255813953488372
F1-score: 0.8296146044624746
Confusion Matrix:
 [[13  2  1]
 [ 4 11  1]
 [ 1  0 42]]
```

[25]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
 ↪BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
#Loading the dataset
df = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")
#Identify categorical features
categorical_columns = df.select_dtypes(include=['object']).columns
#Encode categorical features
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
#Split data into features and target
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
#Scale features (optional, but might help convergence)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)
#Train models
dt_model = DecisionTreeClassifier().fit(X_train, y_train)
rf_model = RandomForestClassifier().fit(X_train, y_train)
svm_model = SVC().fit(X_train, y_train)
nb_model = GaussianNB().fit(X_train, y_train)
lr_model = LogisticRegression(max_iter=1000).fit(X_train, y_train)  # Increase␣
 ↪max_iter
#Evaluate models
models = {
    "Decision Tree": dt_model.predict(X_test),
    "Random Forest": rf_model.predict(X_test),
    "SVM": svm_model.predict(X_test),
    "Naive Bayes": nb_model.predict(X_test),
    "Logistic Regression": lr_model.predict(X_test)
}
for model_name, y_pred in models.items():
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1_score = f1_score(y_test, y_pred, average='macro')  # Use a descriptive␣
 ↪variable name

    print(f"{model_name}:")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1-score: {f1_score:.2f}")
    print()
#Selecting the best model based on the evaluation criteria
best_model = None
best_score = 0
for model_name, y_pred in models.items():
    score = accuracy_score(y_test, y_pred)
    if score > best_score:
        best_model = model_name
        best_score = score
print("Best Model:", best_model)
```

```
Decision Tree:
Accuracy: 0.89
Precision: 0.86
Recall: 0.85
F1-score: 0.85
```

```
TypeError                                 Traceback (most recent call last)
Cell In[25], line 45
     43 precision = precision_score(y_test, y_pred, average='macro')
     44 recall = recall_score(y_test, y_pred, average='macro')
---> 45 f1_score = f1_score(y_test, y_pred, average='macro')  # Use a␣
 ↪descriptive variable name
     47 print(f"{model_name}:")
     48 print(f"Accuracy: {accuracy:.2f}")

TypeError: 'numpy.float64' object is not callable
```

[1]:
```python
import streamlit as st
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
#1.Loading the dataset
@st.cache_data
def load_data():
    data = pd.read_csv(r"C:\Users\Rvdeekshitha\Desktop\ML␣
 ↪Project\Sleep_health_and_lifestyle_dataset.csv")  # Replace with your␣
 ↪dataset file path
    return data
#2.Preprocess data (encoding, handling missing values, and splitting)
def preprocess_data(df):
    # Handle missing values
    # For numerical columns, we use median filling
    num_cols = df.select_dtypes(include=['float64', 'int64']).columns
    cat_cols = df.select_dtypes(include=['object']).columns

    #Impute numerical columns with median
    num_imputer = SimpleImputer(strategy='median')
    df[num_cols] = num_imputer.fit_transform(df[num_cols])

    #Impute categorical columns with the most frequent value
    cat_imputer = SimpleImputer(strategy='most_frequent')
    df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])

    #Set target as 'Sleep Disorder'
    y = df['Sleep Disorder']

    #Drop the target column from features
    X = df.drop(columns=['Sleep Disorder'])

    #One-hot encode categorical variables
```

```
    X_encoded = pd.get_dummies(X, drop_first=True)

    #Split into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_encoded, y,␣
 ↪test_size=0.3, random_state=42)

    return X_train, X_test, y_train, y_test, X_encoded.columns

#3.Train Random Forest Model
def train_model(X_train, y_train):
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)
    return model

#4.Take user input from Streamlit
def user_input_features():
    gender = st.selectbox('Gender', ('Male', 'Female'))
    age = st.slider('Age', 1, 100, 25)
    occupation = st.selectbox('Occupation', ('Student', 'Professional',␣
 ↪'Other'))
    sleep_duration = st.slider('Sleep Duration (hours)', 1, 12, 7)
    quality_of_sleep = st.slider('Quality of Sleep (1-5)', 1, 5, 3)
    physical_activity_level = st.slider('Physical Activity Level (1-5)', 1, 5,␣
 ↪3)
    stress_level = st.slider('Stress Level (1-5)', 1, 5, 3)
    bmi_category = st.selectbox('BMI Category', ('Underweight', 'Normal',␣
 ↪'Overweight', 'Obese'))
    blood_pressure = st.slider('Blood Pressure', 60, 200, 120)
    heart_rate = st.slider('Heart Rate', 40, 200, 70)
    daily_steps = st.slider('Daily Steps', 0, 30000, 5000)

    #User input dictionary
    user_data = {
        'Gender': gender,
        'Age': age,
        'Occupation': occupation,
        'Sleep Duration': sleep_duration,
        'Quality of Sleep': quality_of_sleep,
        'Physical Activity Level': physical_activity_level,
        'Stress Level': stress_level,
        'BMI Category': bmi_category,
        'Blood Pressure': blood_pressure,
        'Heart Rate': heart_rate,
        'Daily Steps': daily_steps
    }

    #Convert to DataFrame for consistency
```

```
    user_input_df = pd.DataFrame(user_data, index=[0])
    return user_input_df

#5.Preprocess user input to match model features
def preprocess_user_input(user_input, feature_columns):
    #One-hot encode the user input
    user_input_encoded = pd.get_dummies(user_input, drop_first=True)

    #Align the columns of user input to match the columns from the training data
    #Add missing columns and set them to 0
    aligned_user_input = user_input_encoded.reindex(columns=feature_columns,␣
 ↪fill_value=0)

    return aligned_user_input

#6.Main function to run the Streamlit app
def main():
    st.title('Sleep Disorder Prediction App')

    #Loading the dataset
    df = load_data()
    st.write("Dataset Overview:", df.head())

    #Get user input
    user_input = user_input_features()
    st.write("User Input:", user_input)

    #Preprocess data and split into train/test sets
    X_train, X_test, y_train, y_test, feature_columns = preprocess_data(df)

    #Train the Random Forest model
    model = train_model(X_train, y_train)

    #Preprocess user input to match model features
    input_encoded = preprocess_user_input(user_input, feature_columns)

    #Make prediction
    prediction = model.predict(input_encoded)

    #Display prediction
    st.write(f"Prediction: {'Sleep Disorder Detected' if prediction[0] == 1␣
 ↪else 'No Sleep Disorder Detected'}")

    #Test model accuracy
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    st.write(f"Model Accuracy: {accuracy * 100:.2f}%")
```

```
if __name__ == '__main__':
    main()
```

```
2024-09-22 10:37:55.504 WARNING streamlit.runtime.caching.cache_data_api: No
runtime found, using MemoryCacheStorageManager
2024-09-22 10:37:55.512 WARNING
streamlit.runtime.scriptrunner_utils.script_run_context: Thread 'MainThread':
missing ScriptRunContext! This warning can be ignored when running in bare mode.
2024-09-22 10:37:56.442
  Warning: to view this Streamlit app on a browser, run it with the
following
  command:

    streamlit run
C:\Users\Rvdeekshitha\AppData\Local\Programs\Python\Python311\Lib\site-
packages\ipykernel_launcher.py [ARGUMENTS]
2024-09-22 10:37:56.443 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.443 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.444 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.445 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.448 No runtime found, using MemoryCacheStorageManager
2024-09-22 10:37:56.471 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.472 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.473 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.473 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.473 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.473 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.531 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.531 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.540 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.542 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.542 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
```

```
2024-09-22 10:37:56.547 Session state does not function when running a script
without `streamlit run`
2024-09-22 10:37:56.548 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.551 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.553 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.554 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.558 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.560 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.562 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.565 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.568 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.570 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.571 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.572 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.573 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.574 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.575 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.576 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.578 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.579 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.581 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.582 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.583 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.584 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.586 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
```

```
2024-09-22 10:37:56.586 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.587 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.588 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.589 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.590 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.591 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.594 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.596 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.598 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.600 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.602 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.604 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.606 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.607 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.608 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.609 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.610 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.611 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.612 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.614 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.616 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.619 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.620 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.621 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
```

```
2024-09-22 10:37:56.623 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.626 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.628 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.882 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.883 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.883 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.884 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.893 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.893 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.897 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
2024-09-22 10:37:56.898 Thread 'MainThread': missing ScriptRunContext! This
warning can be ignored when running in bare mode.
```

[ ]: