SLEEP DISORDER PREDICTION

Sleep disorders are increasingly recognized as significant health concerns affecting millions worldwide. Identifying and addressing these disorders early can significantly improve individuals' quality of life and overall health outcomes. Our project aims to leverage machine learning techniques, specifically linear regression, to predict the likelihood of sleep disorders based on various factors. This project utilizes a dataset containing various sleep-related parameters such as sleep duration, quality of sleep, stress levels, and other lifestyle factors.

In [1]:
```
import numpy as np
import pandas as pd

#for visualizaion
import matplotlib.pyplot as plt
import seaborn as sns

#vif
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import shapiro, kstest, normaltest #hypothesis testing

from sklearn.model_selection import train_test_split

#linear regression
from sklearn.linear_model import LinearRegression

#evaluation metrics for regression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Problem Statement To predict if the patient suffers from sleep disorder Target Column: Sleep Disorder

In [2]:
```
df = pd.read_csv("SDP_dataset.csv")
df
```

Out[2]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Heart Rate	Daily Steps
0	1	Male	27	ASD	6.1	6	42	6	Overweight	77	4200
1	2	Male	28	Dr	6.2	6	60	8	Normal	75	10000
2	3	Male	28	Dr	6.2	6	60	8	Normal	75	10000
3	4	Male	28	Sales Rep	5.9	4	30	8	Obese	85	3000
4	5	Male	28	Sales Rep	5.9	4	30	8	Obese	85	3000
...	...	...	...	...	...	...	...	...	...	...	...
369	370	Female	59	Hospital Worker	8.1	9	75	3	Overweight	68	7000
370	371	Female	59	Hospital Worker	8.0	9	75	3	Overweight	68	7000
371	372	Female	59	Hospital Worker	8.1	9	75	3	Overweight	68	7000
372	373	Female	59	Hospital Worker	8.1	9	75	3	Overweight	68	7000
373	374	Female	59	Hospital Worker	8.1	9	75	3	Overweight	68	7000

374 rows × 12 columns

In [3]:
```
df1=df.drop(["Gender", "Age", "Occupation", "Physical Activity Level", "BMI Category",
df1
```

Out[3]:

	Sleep Duration	Quality of Sleep	Stress Level	Sleep Disorder
0	6.1	6	6	No Disorder
1	6.2	6	8	No Disorder
2	6.2	6	8	No Disorder
3	5.9	4	8	Sleep Apnea
4	5.9	4	8	Sleep Apnea
...	...	...	...	...
369	8.1	9	3	Sleep Apnea
370	8.0	9	3	Sleep Apnea
371	8.1	9	3	Sleep Apnea
372	8.1	9	3	Sleep Apnea
373	8.1	9	3	Sleep Apnea

374 rows × 4 columns

Exploratory Data Analysis (EDA)

In [4]:
```
# Check the no of rows and columns of the data
df1.shape
```

Out[4]:
```
(374, 4)
```

In [5]:
```
# Check for information
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Sleep Duration    374 non-null    float64
 1   Quality of Sleep  374 non-null    int64  
 2   Stress Level      374 non-null    int64  
 3   Sleep Disorder    374 non-null    object 
dtypes: float64(1), int64(2), object(1)
memory usage: 11.8+ KB
```

In [6]:
```
# Check for statistical information

df1.describe()
```

Out[6]:

	Sleep Duration	Quality of Sleep	Stress Level
count	374.000000	374.000000	374.000000
mean	7.132086	7.312834	5.385027
std	0.795657	1.196956	1.774526
min	5.800000	4.000000	3.000000
25%	6.400000	6.000000	4.000000
50%	7.200000	7.000000	5.000000
75%	7.800000	8.000000	7.000000
max	8.500000	9.000000	8.000000

In [7]:
```
# Check for no. of categories in target column

df1["Sleep Disorder"].unique()
```

Out[7]:
```
array(['No Disorder', 'Sleep Apnea', 'Insomnia'], dtype=object)
```

In [8]:
```
# Count for each category in the target column

df1["Sleep Disorder"].value_counts()
```

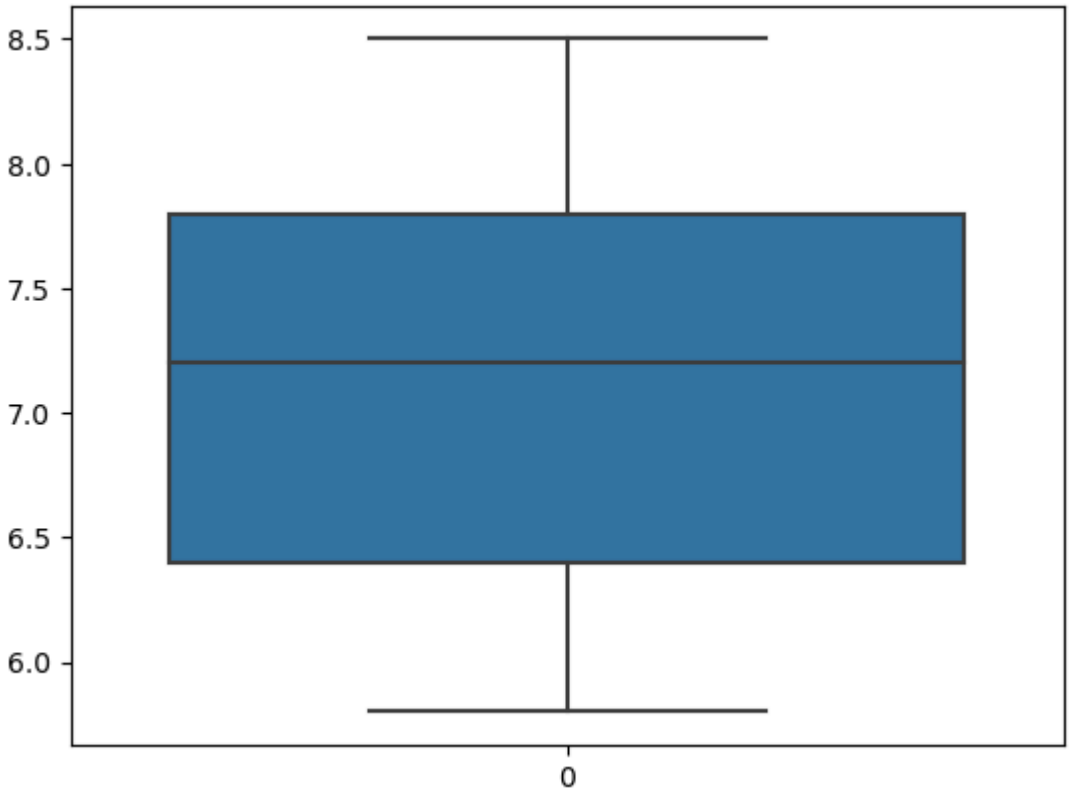
Out[8]:
```
Sleep Disorder
No Disorder    219
Sleep Apnea     78
Insomnia        77
Name: count, dtype: int64
```

In [9]:
```
# Check for missing values

df1.isnull().sum()
```

Out[9]:
```
Sleep Duration      0
Quality of Sleep    0
Stress Level        0
Sleep Disorder      0
dtype: int64
```

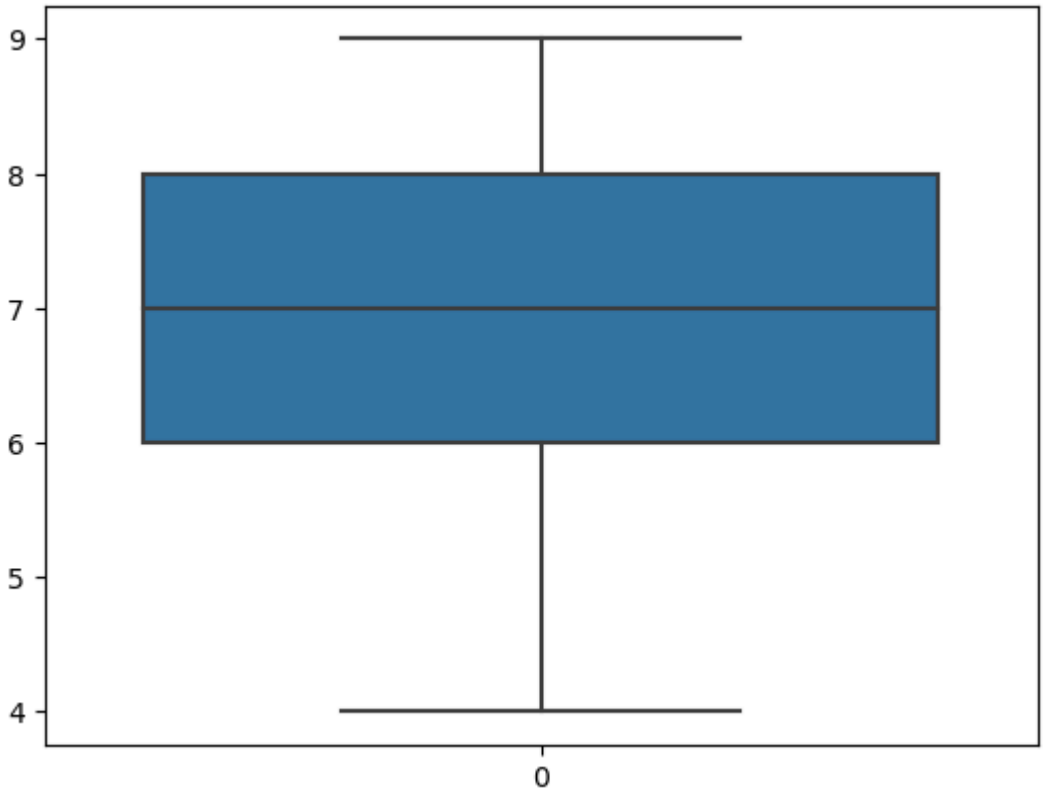
In [12]:
```
# Check For Outliers

sns.boxplot(df1["Sleep Duration"])
```

Out[12]:
```
<Axes: >
```



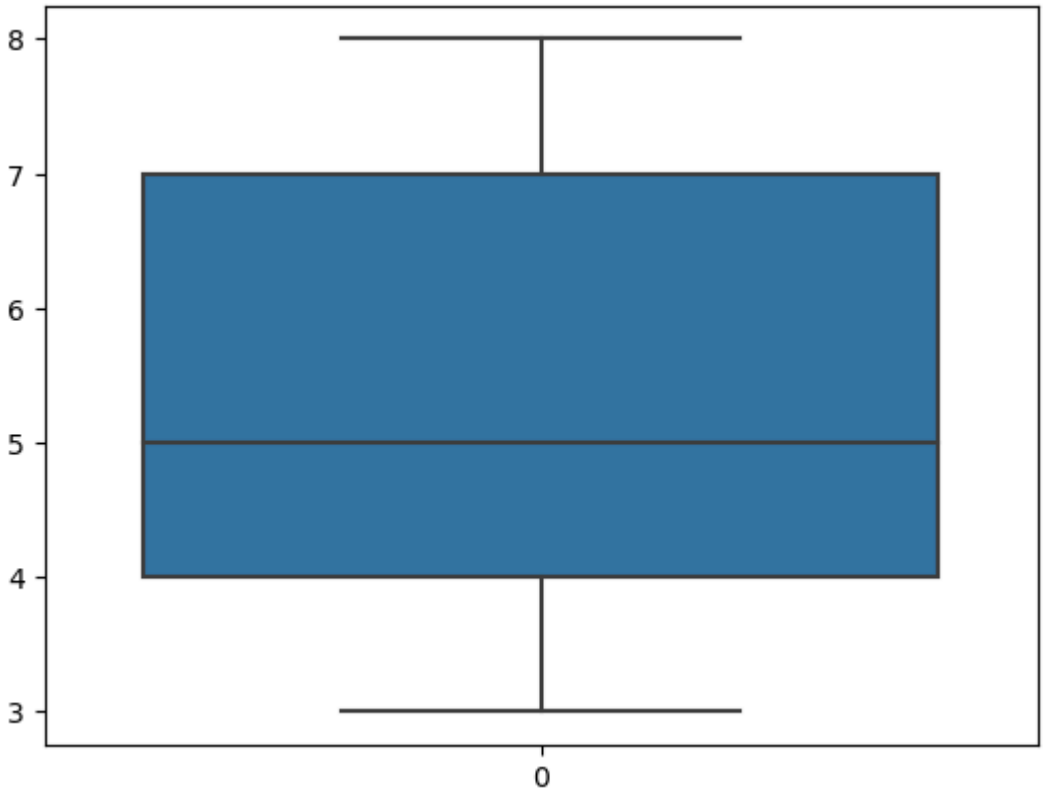
In [10]:
```
sns.boxplot(df1["Quality of Sleep"])
```

Out[10]: <Axes: >



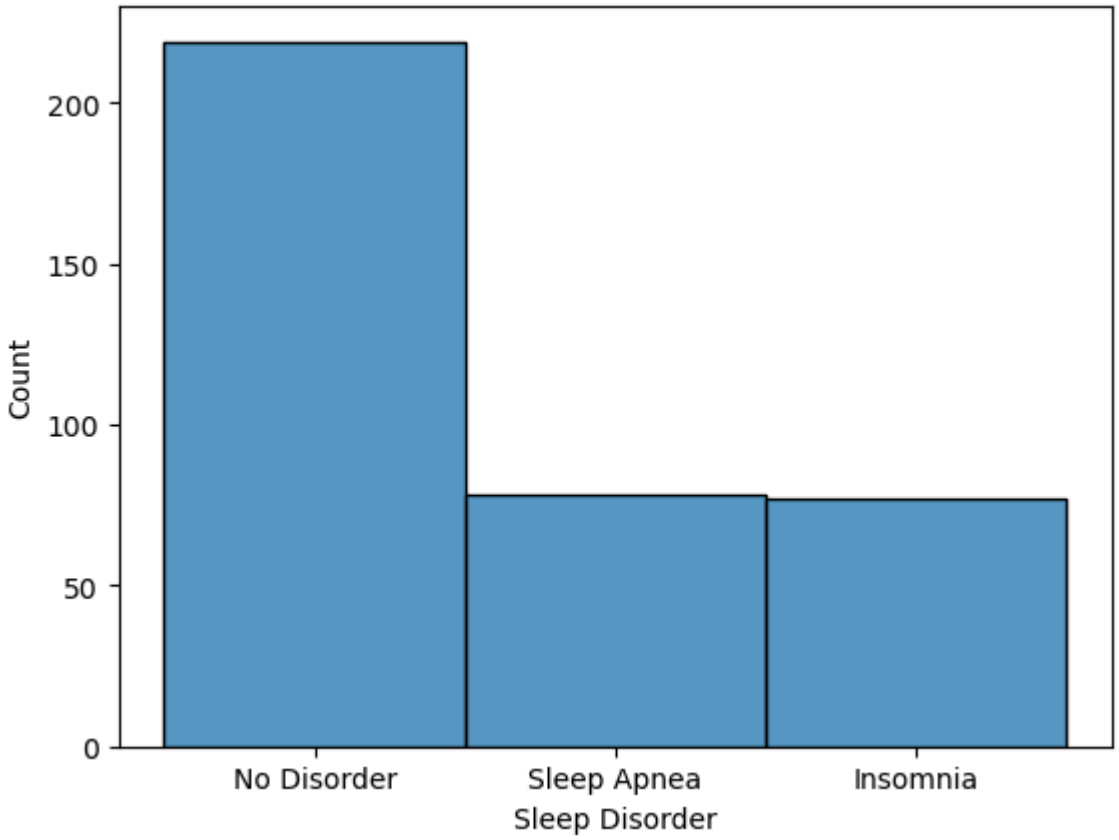
In [11]:
```
sns.boxplot(df["Stress Level"])
```

Out[11]: <Axes: >



In [12]:
```
sns.histplot(df["Sleep Disorder"])
```

Out[12]:
```
<Axes: xlabel='Sleep Disorder', ylabel='Count'>
```



Feature Engineering

In [13]:
```
# Convert whole data in numerical format

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Sleep Duration    374 non-null    float64
 1   Quality of Sleep  374 non-null    int64
 2   Stress Level      374 non-null    int64
 3   Sleep Disorder    374 non-null    object
dtypes: float64(1), int64(2), object(1)
memory usage: 11.8+ KB
```

In [14]:
```
df1["Sleep Disorder"].unique()
```

Out[14]:
```
array(['No Disorder', 'Sleep Apnea', 'Insomnia'], dtype=object)
```

In [15]:
```
df1["Sleep Disorder"].replace({'No Disorder': 0, 'Sleep Apnea': 1, 'Insomnia': 2}, inp
df1
```

Out[15]:

	Sleep Duration	Quality of Sleep	Stress Level	Sleep Disorder
0	6.1	6	6	0
1	6.2	6	8	0
2	6.2	6	8	0
3	5.9	4	8	1
4	5.9	4	8	1
...	...	...	...	...
369	8.1	9	3	1
370	8.0	9	3	1
371	8.1	9	3	1
372	8.1	9	3	1
373	8.1	9	3	1

374 rows × 4 columns

In [16]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Sleep Duration    374 non-null    float64
 1   Quality of Sleep  374 non-null    int64  
 2   Stress Level      374 non-null    int64  
 3   Sleep Disorder    374 non-null    int64  
dtypes: float64(1), int64(3)
memory usage: 11.8 KB
```

Feature Selection

Assumption 1: Linearity

There should be a linear relationship between independant and dependant variables. Therefore, to check the same we use correlation

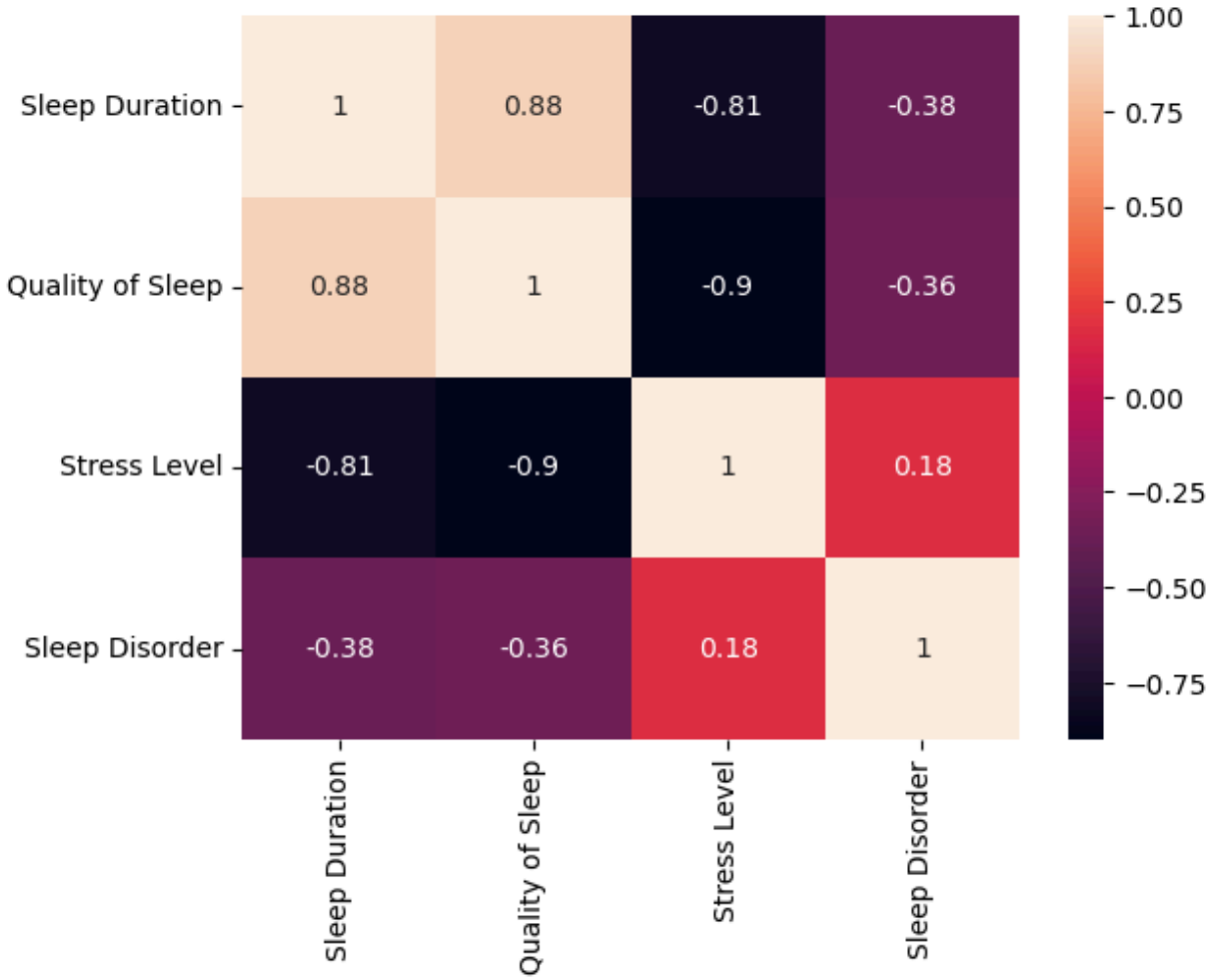
In [17]:
```
# Correlation

df1.corr()
```

Out[17]:

	Sleep Duration	Quality of Sleep	Stress Level	Sleep Disorder
Sleep Duration	1.000000	0.883213	-0.811023	-0.382045
Quality of Sleep	0.883213	1.000000	-0.898752	-0.357477
Stress Level	-0.811023	-0.898752	1.000000	0.181296
Sleep Disorder	-0.382045	-0.357477	0.181296	1.000000

In [18]:
```
# Plotting Heatmap for correlation

sns.heatmap(df1.corr(), annot = True)  # annot: to display no. on the map
```

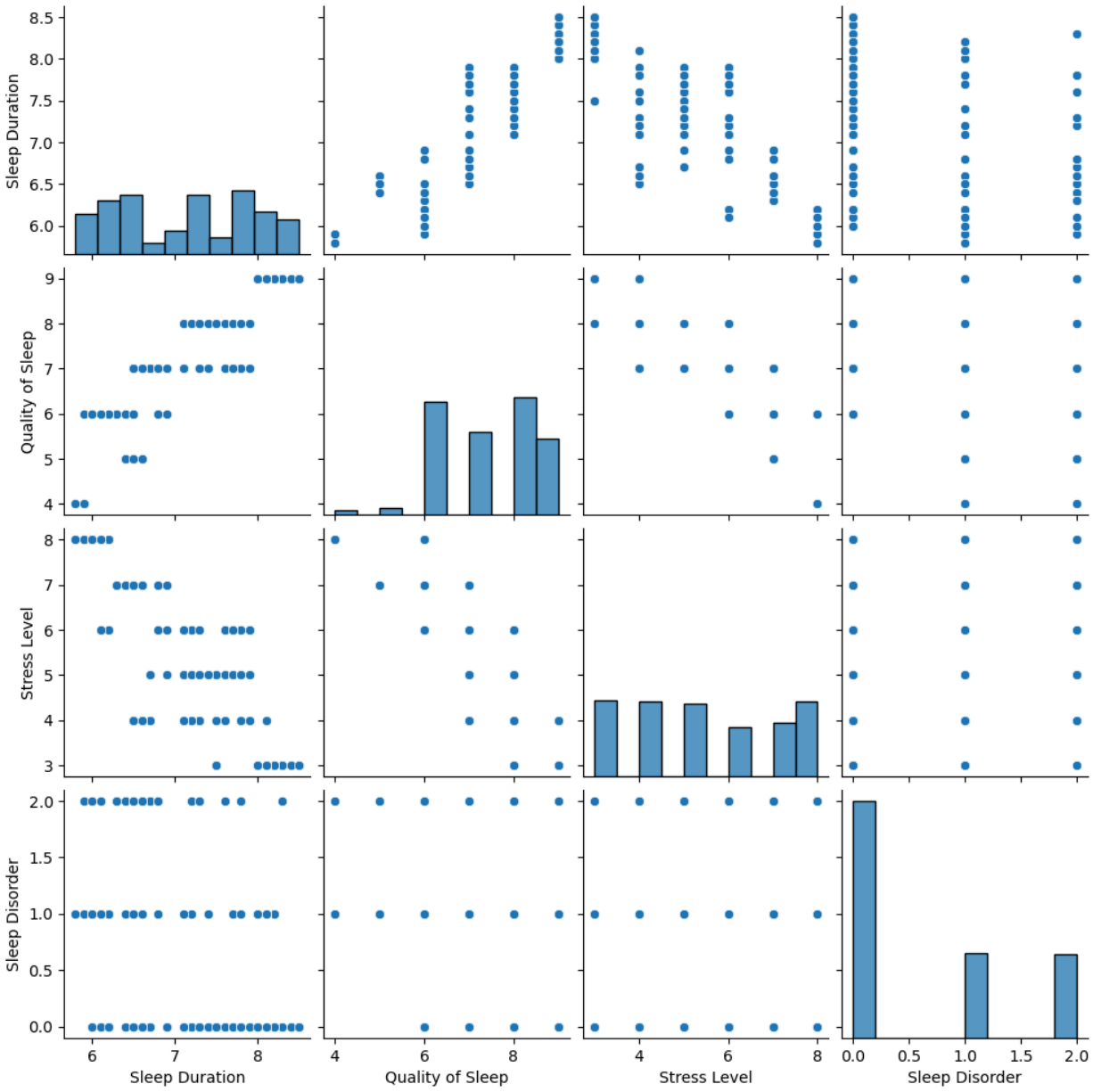
Out[18]:
```
<Axes: >
```



In [22]:
```
# Pairplot

sns.pairplot(df1)
```

```
D:\ANACONDA3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layou
t has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x26f13753f90>
```



Assumption 2: No Multicolinearit

There should not be strong corelation between 2 independant features We check this by vif (variance_inflation_factor) vif = 1/R2_score

In [19]:
```
df2 = df1.iloc[:,:3]
df2
```

Out[19]:

	Sleep Duration	Quality of Sleep	Stress Level
0	6.1	6	6
1	6.2	6	8
2	6.2	6	8
3	5.9	4	8
4	5.9	4	8
...	...	...	...
369	8.1	9	3
370	8.0	9	3
371	8.1	9	3
372	8.1	9	3
373	8.1	9	3

374 rows × 3 columns

In [20]:
```
vif_df = pd.DataFrame()
vif_df["Features"] = df2.columns

vif_df
```

Out[20]:

	Features
0	Sleep Duration
1	Quality of Sleep
2	Stress Level

In [21]:
```
vif_list = []

for i in range(df2.shape[1]):
    vif = variance_inflation_factor(df2.to_numpy(),i)
    vif_list.append(vif)
```

In [22]:
```
vif_list
```

Out[22]:
```
[309.59378021280764, 244.18460309441568, 10.26620737778925]
```

In [23]:
```
vif_df["VIF"] = vif_list

vif_df
```

Out[23]:

	Features	VIF
0	Sleep Duration	309.593780
1	Quality of Sleep	244.184603
2	Stress Level	10.266207

In [24]:
```
x = df1.iloc[:,:3]  # independent features

y = df1["Sleep Disorder"] # Target Feature
```

In [25]:
```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.2)
```

In [26]:
```
xtrain.shape
```

Out[26]:
```
(299, 3)
```

In [27]:
```
ytrain.shape
```

Out[27]:
```
(299,)
```

In [28]:
```
xtest.shape
```

Out[28]:
```
(75, 3)
```

In [29]:
```
ytest.shape
```

Out[29]:
```
(75,)
```

Model Training

In [30]:
```
lin_reg = LinearRegression()

lin_reg
```

Out[30]:
```
LinearRegression
LinearRegression()
```

In [31]:
```
lin_reg_model = lin_reg.fit(xtrain, ytrain)
lin_reg_model
```

Out[31]:
```
LinearRegression
LinearRegression()
```

Assumption 3: Normality of Residual

In this assumption we try to visualize the error using: check for assumption 3: visualization: kdeplot/histplot qqplot hypothesis testing: shapiro kstest normaltest

In [32]:
```
ytrain
```

Out[32]:
```
25     0
342    0
309    2
33     0
305    1
      ..
190    2
341    0
317    0
181    0
368    1
Name: Sleep Disorder, Length: 299, dtype: int64
```

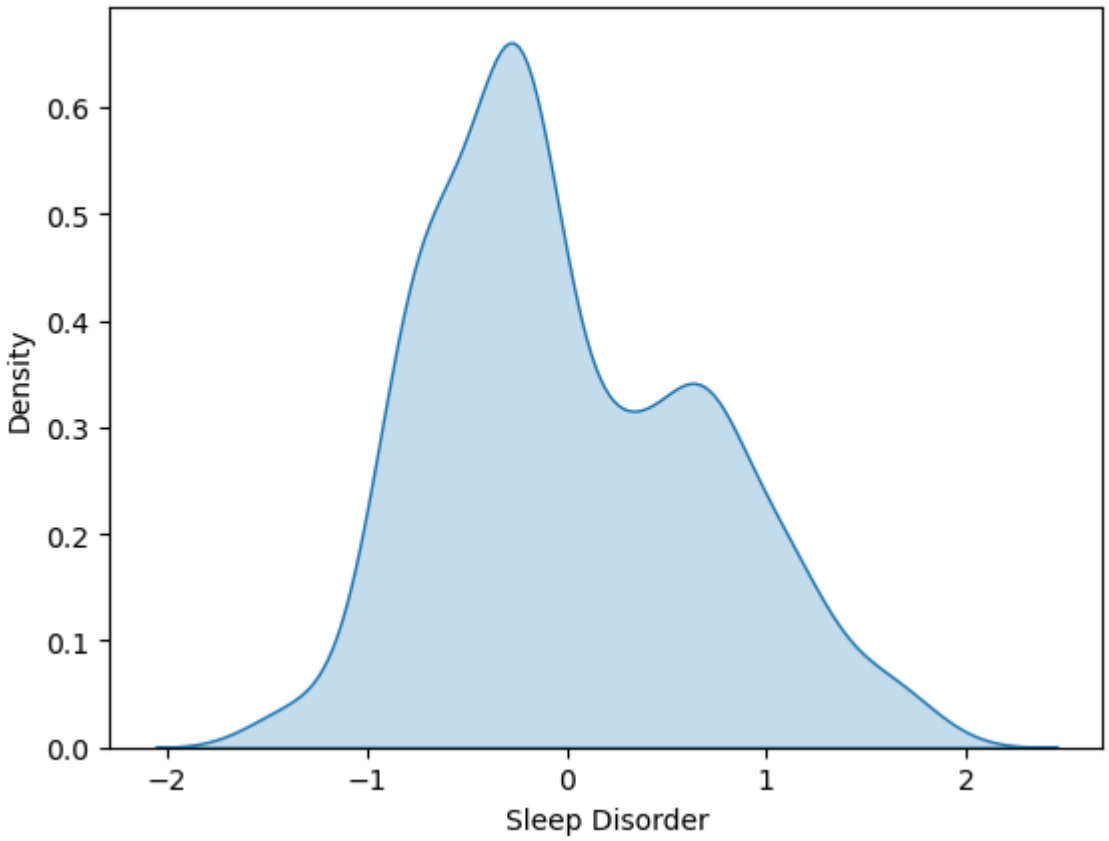
In [33]:
```
ytrain_predict = lin_reg_model.predict(xtrain)
```

In [34]:
```
ytrain_predict
```

Out[34]:
```
array([ 0.22980203,  0.23236999,  0.4216351 ,  0.7255982 ,  0.7255982 ,
        0.44735163,  0.97504512,  0.67108202,  0.4216351 ,  0.68548184,
        0.77703128,  0.94624547,  0.16653709,  0.93492876,  0.31003475,
        0.23236999,  0.77446331,  1.41067403,  0.20665346,  0.76571457,
        0.76571457,  0.76571457,  0.23236999,  0.32700254,  0.7255982 ,
        0.76006348,  1.04087802,  0.23236999,  0.76571457,  0.16653709,
        1.38495749,  0.7255982 ,  0.23236999,  0.76571457,  0.15213727,
        0.15213727,  1.41067403,  0.76571457,  0.07755563,  1.45079039,
        0.15213727,  0.31003475,  1.38495749,  0.7255982 ,  1.45079039,
        0.97504512,  0.7255982 ,  1.34484113,  0.24676982,  0.31260272,
        0.7255982 ,  0.31260272,  0.31003475,  1.45079039,  0.1120209 ,
        0.31260272,  0.27248636,  1.41067403,  0.15213727,  0.68548184,
        0.1120209 ,  0.15213727,  0.73691491,  0.23236999,  0.76571457,
        0.97504512,  0.26991839,  0.32700254,  0.16653709,  0.20665346,
        0.40723527,  0.7255982 ,  0.16653709,  0.93492876,  0.15213727,
        0.16653709,  0.32700254,  0.76571457,  0.20665346,  1.67195281,
        0.76571457,  0.76571457,  0.16653709,  0.24676982,  0.7255982 ,
        0.19225363,  0.1120209 ,  0.7255982 ,  0.97504512,  0.77703128,
        0.36711891,  0.4216351 ,  0.26991839,  0.73691491,  0.73691491,
        0.73434695,  0.7255982 ,  0.07755563,  0.20665346,  0.34140237,
        0.35015111,  1.37055767,  0.73691491,  0.31260272,  0.20665346,
        0.7255982 ,  0.40723527,  0.15213727,  0.40723527,  1.28775697,
        0.15213727,  0.31003475,  0.23236999,  0.35015111,  0.69679855,
        0.27248636, -0.05719329,  0.27248636,  0.8948124 ,  0.1120209 ,
        0.36711891,  0.76006348,  0.44735163,  1.3679897 ,  0.15213727,
        0.93492876,  0.23236999,  0.80017985,  0.31260272,  0.77703128,
        0.35015111,  0.76571457,  0.67108202,  1.32787334,  0.19225363,
        0.68548184,  0.73691491,  0.7255982 ,  0.19225363,  0.77703128,
        0.73691491,  1.3679897 ,  0.36711891,  0.68548184,  0.26991839,
        0.24676982,  0.16653709,  0.28688618,  0.94624547,  0.68548184,
        0.80017985,  0.8948124 ,  0.23236999,  0.31003475,  0.16653709,
        0.76571457,  0.40723527,  0.19225363,  0.8948124 ,  0.16653709,
        1.32787334,  0.8948124 ,  0.16653709,  0.76571457,  0.20665346,
        0.15213727,  0.31003475,  0.27248636,  0.7255982 ,  0.03743926,
        1.41067403,  0.27248636,  0.77446331,  0.31260272,  1.28775697,
        1.41067403,  0.26991839,  0.40723527,  0.76571457,  1.41067403,
        0.31260272,  0.12642073,  0.26991839,  0.68548184,  0.1120209 ,
        0.26991839,  0.26991839,  0.36711891,  0.76571457,  0.7255982 ,
        0.19225363,  0.8948124 ,  0.61656583,  0.77703128,  0.31003475,
        0.97504512,  0.73691491,  0.69679855,  0.76571457,  0.7255982 ,
        0.93492876,  0.55073293,  0.26991839,  0.16653709,  0.1120209 ,
        0.7255982 ,  1.37055767,  0.31260272,  0.77703128,  0.73691491,
        1.45079039,  0.55073293,  0.77703128,  0.69679855,  0.36711891,
        0.76571457,  1.71206917,  0.31003475,  0.15213727,  1.45079039,
        1.41067403,  0.77703128,  0.76571457,  0.1120209 ,  0.40723527,
        0.44735163,  0.31260272,  0.9606453 ,  0.7255982 ,  0.31003475,
        1.41067403,  0.31003475,  1.67195281,  1.41067403,  0.49621674,
        1.32787334,  0.1120209 ,  1.41067403,  0.76571457,  0.16653709,
        0.27248636,  0.23236999,  0.93492876,  0.57644946,  0.93492876,
        0.27248636,  0.76571457,  0.73691491,  0.73691491,  0.76571457,
        0.8948124 ,  1.04087802,  0.77703128,  0.73691491,  0.68548184,
        0.31260272,  0.28688618,  0.40723527,  0.15213727,  0.55073293,
        1.45079039,  0.73691491,  0.73691491,  0.8948124 ,  0.20665346,
        0.35015111,  0.68548184,  0.97504512,  0.27248636,  0.22980203,
        0.40723527,  0.77703128,  0.7255982 ,  0.26991839,  0.76571457,
        0.73691491,  0.20665346,  0.8948124 ,  1.67195281,  0.76571457,
        0.46175146,  0.8948124 ,  0.31260272,  0.7255982 ,  0.40723527,
        0.4216351 ,  0.27248636,  0.22980203,  0.68548184,  0.73691491,
        0.45610038,  0.76571457,  0.76571457,  0.36711891,  1.37055767,
        0.23236999,  0.1120209 ,  0.16653709,  0.27248636])
```

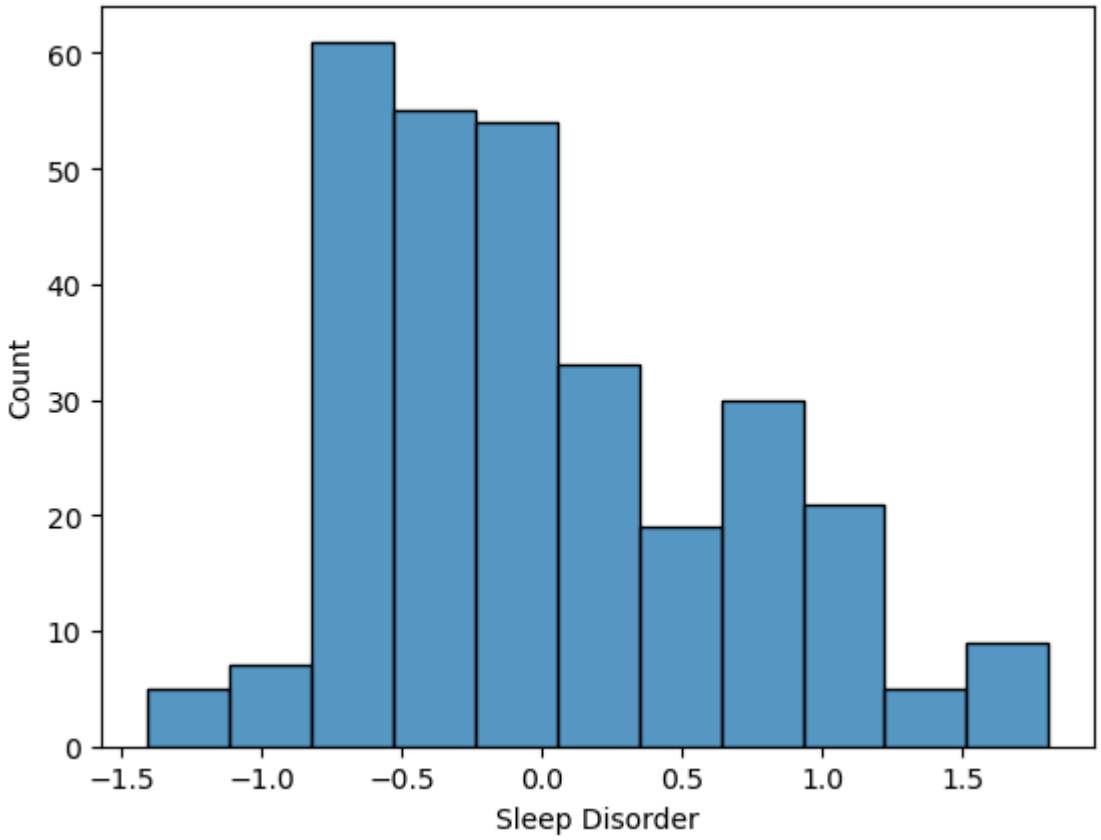
In [35]:
```
# Assuming you have fitted a linear regression model and obtained predictions
y_pred = lin_reg_model.predict(xtrain)

# Compute residuals
residual = ytrain - y_pred

# Plot kernel density estimate of residuals
import seaborn as sns
sns.kdeplot(residual, fill=True)
```

Out[35]:
```
<Axes: xlabel='Sleep Disorder', ylabel='Density'>
```



In [36]:
```
sns.histplot(residual)
```

Out[36]:
```
<Axes: xlabel='Sleep Disorder', ylabel='Count'>
```



Hypothesis Testing

In [37]:
```
stats, p_val = shapiro(residual)
print(stats,p_val)

if p_val>=0.05:
    print("DATA is normally distributed")
    print("Null hypothesis H0 is true")


else:
    print("Data is not normally distributed")
    print("alternative hypothesis H1 is true")
```

```
0.9524638652801514 2.8675071206407665e-08
Data is not normally distributed
alternative hypothesis H1 is true
```

In [38]:
```
df_float = df2.astype(float)
df_float
```

Out[38]:

	Sleep Duration	Quality of Sleep	Stress Level
0	6.1	6.0	6.0
1	6.2	6.0	8.0
2	6.2	6.0	8.0
3	5.9	4.0	8.0
4	5.9	4.0	8.0
...	...	...	...
369	8.1	9.0	3.0
370	8.0	9.0	3.0
371	8.1	9.0	3.0
372	8.1	9.0	3.0
373	8.1	9.0	3.0

374 rows × 3 columns

In [39]:
```
df3 = np.log(df_float)
df3
```

Out[39]:

	Sleep Duration	Quality of Sleep	Stress Level
0	1.808289	1.791759	1.791759
1	1.824549	1.791759	2.079442
2	1.824549	1.791759	2.079442
3	1.774952	1.386294	2.079442
4	1.774952	1.386294	2.079442
...	...	...	...
369	2.091864	2.197225	1.098612
370	2.079442	2.197225	1.098612
371	2.091864	2.197225	1.098612
372	2.091864	2.197225	1.098612
373	2.091864	2.197225	1.098612

374 rows × 3 columns

In [40]:
```
# residuals replaced with your actual residual
residuals = np.random.normal(loc = 0, scale = 1, size = 100)


#shapiro-wilk test for normality
statistic, p_value = shapiro(residuals)
print("Shapiro-wilk test statistic:", statistic)
print("p-value:", p_value)
```

```
# check if the residuals are normally distributed
if p_value >= 0.05:
    print("residuals are normally distributed")
else:
    print("residuals are not normally distributed")
```

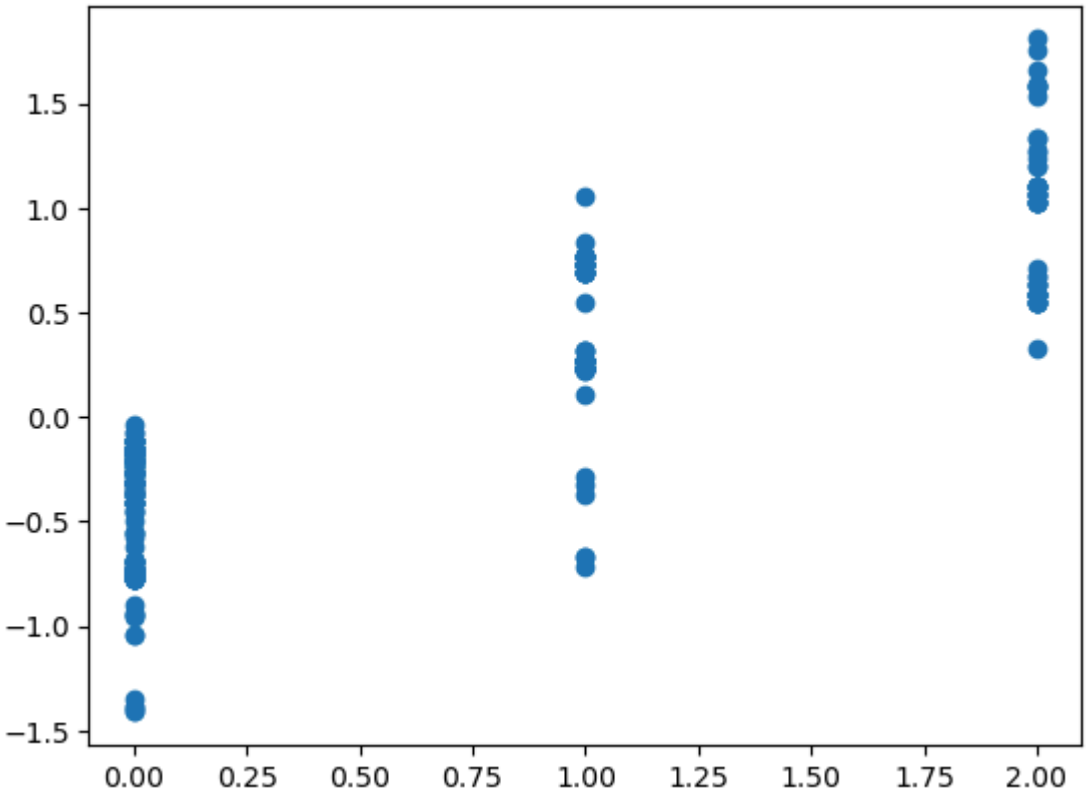
```
Shapiro-wilk test statistic: 0.9802603721618652
p-value: 0.13928022980690002
residuals are normally distributed
```

Assumption 4: Homoscadasticity

error variance = constant

In [41]:
```
plt.scatter(ytrain, residual)    #matplotlib visualization
```

Out[41]:
```
<matplotlib.collections.PathCollection at 0x1db6cc27010>
```



Model Evaluation

In [42]:
```
mse = mean_squared_error(ytrain,ytrain_predict)
print(f"Mean Squared error : {mse}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared error : {rmse}")

mae = mean_absolute_error(ytrain,ytrain_predict)
print(f"Mean Absolute error : {mae}")
```

```
r2score = r2_score(ytrain,ytrain_predict)
print(f"R2 score : {r2score}")
```

```
Mean Squared error : 0.46047490730821394
Root Mean Squared error : 0.6785830143086503
Mean Absolute error : 0.5692451605135611
R2 score : 0.25107485831280674
```

Model Evaluation for Testing:

In [43]:
```
ytest
```

Out[43]:
```
163    0
245    2
356    1
55     0
258    2
      ..
112    0
235    2
42     0
159    0
354    1
Name: Sleep Disorder, Length: 75, dtype: int64
```

In [44]:
```
ytest_predict = lin_reg_model.predict(xtest)
ytest_predict
```

Out[44]:
```
array([ 0.12642073,  1.45079039,  0.31260272,  0.76571457,  1.41067403,
        0.07755563,  0.7255982 ,  0.20665346,  0.73691491,  0.46175146,
        0.97504512,  0.73691491,  0.97504512,  0.36711891,  0.76571457,
        0.93492876,  0.16653709,  1.41067403,  0.49621674,  0.1120209 ,
        0.1120209 ,  0.15213727,  0.1120209 ,  0.22980203,  0.16653709,
        0.63096565,  0.15213727,  0.97504512,  0.76571457,  0.73691491,
        0.31003475,  1.45079039,  1.71206917,  0.97504512,  0.68548184,
        0.1120209 ,  0.76571457,  0.97504512,  0.16653709,  0.40723527,
        1.41067403,  0.69679855,  0.35015111,  0.8948124 , -0.05719329,
        0.8948124 ,  0.80583093,  1.34484113,  1.41067403,  0.73691491,
        0.27248636,  0.77703128,  0.76571457,  0.76571457,  0.27248636,
        0.16653709,  0.8948124 ,  0.24676982,  0.44735163,  0.27248636,
        0.97504512,  0.77703128,  0.31003475,  0.73691491,  0.20665346,
        0.7255982 ,  0.8948124 ,  0.73691491,  0.31003475,  0.7255982 ,
        0.73691491,  0.97504512,  0.31003475,  0.40723527,  0.31260272])
```

In [45]:
```
mse = mean_squared_error(ytest,ytest_predict)
print(f"Mean Squared error : {mse}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared error : {rmse}")

mae = mean_absolute_error(ytest,ytest_predict)
print(f"Mean Absolute error : {mae}")

r2score = r2_score(ytest,ytest_predict)
print(f"R2 score : {r2score}")
```

```
Mean Squared error : 0.5624104925597672
Root Mean Squared error : 0.7499403259991871
Mean Absolute error : 0.6406265285308256
R2 score : 0.26701598224080414
```

Save Model into pickle file

In [46]:
```
import pickle
```

In [47]:
```
with open("lin_model.pkl","wb") as f:
    pickle.dump(lin_reg_model,f)
```

In [48]:
```
def predictions(Sleep_Duration, Quality_of_Sleep, Stress_Level):
    # Create a DataFrame with the provided sleep parameters
    test_df = pd.DataFrame({"Sleep Duration": [Sleep_Duration],
                            "Quality of Sleep": [Quality_of_Sleep],
                            "Stress Level": [Stress_Level]
                            })
    print(test_df)

    # Load the trained model
    with open("lin_model.pkl", "rb") as f:
        final_model = pickle.load(f)

        # Predict sleep disorder
        sleep_disorder = final_model.predict(test_df)

        print(f"Sleep Disorder: {sleep_disorder}")
```

In [49]:
```
# Example usage:
predictions(6.1, 6, 6)
```

```
   Sleep Duration  Quality of Sleep  Stress Level
0             6.1                 6             6
Sleep Disorder: [1.38495749]
```

Model Training using Decision tree

In [64]:
```
from sklearn.tree import DecisionTreeRegressor

dt_reg = DecisionTreeRegressor()
dt_reg_model = dt_reg.fit(xtrain, ytrain)
dt_reg_model
```

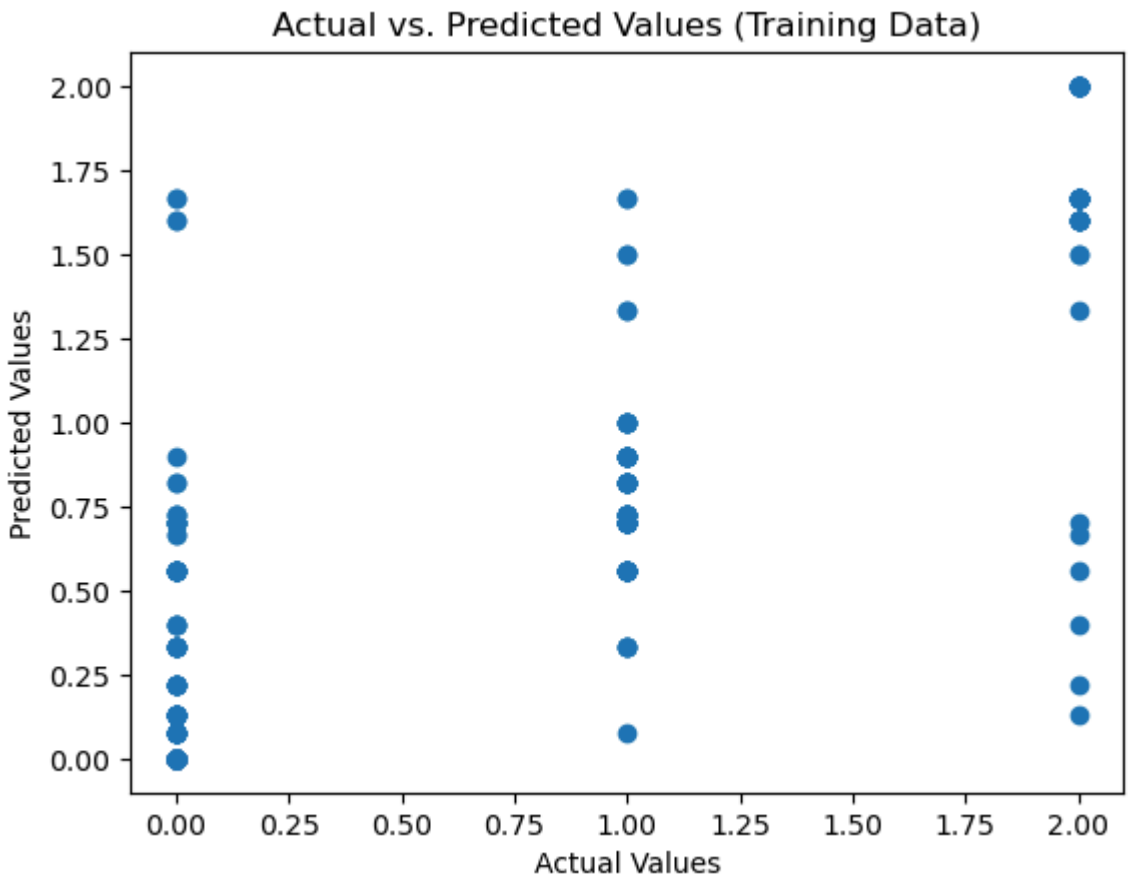
Out[64]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

Model Evaluation

In [61]:
```
ytrain_pred = dt_reg_model.predict(xtrain)
```

In [66]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import seaborn as sns
import matplotlib.pyplot as plt

## For Training Data
mse = mean_squared_error(ytrain, ytrain_pred)
print(f"Mean Squared Error (MSE) : {mse}")
print("*" * 60)

mae = mean_absolute_error(ytrain, ytrain_pred)
print(f"Mean Absolute Error (MAE) : {mae}")
print("*" * 60)

r2 = r2_score(ytrain, ytrain_pred)
print(f"R-squared (R2) Score : {r2}")
print("*" * 60)

plt.scatter(ytrain, ytrain_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values (Training Data)")
plt.show()
```

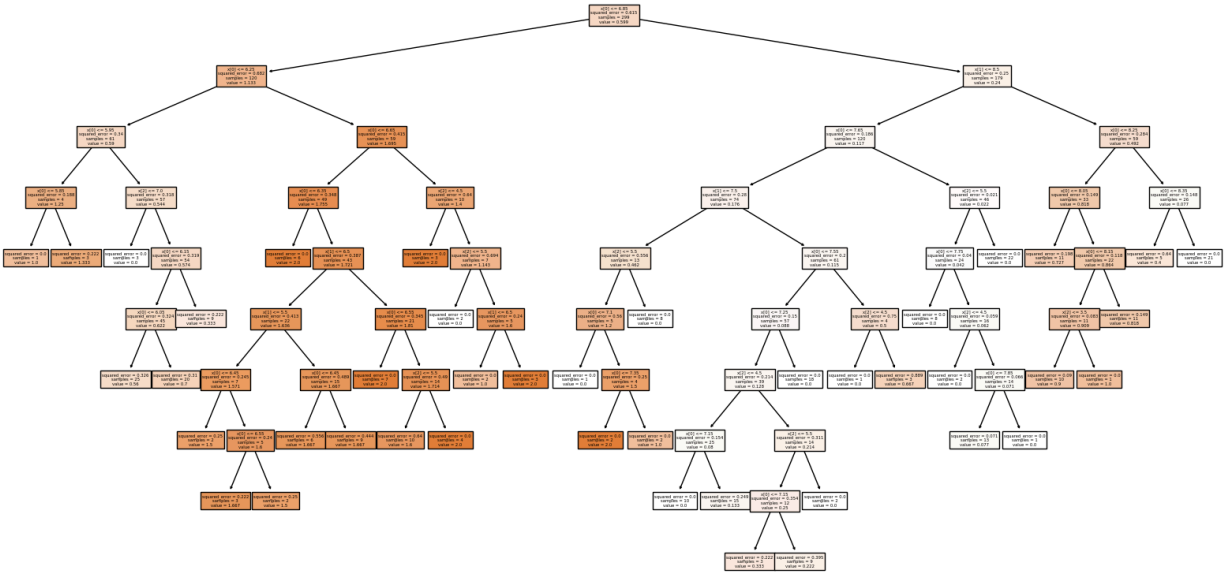
```
Mean Squared Error (MSE) : 0.17354564424798538
************************************************************
Mean Absolute Error (MAE) : 0.2410637894249934
************************************************************
R-squared (R2) Score : 0.7177420655396932
************************************************************
```



In [68]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

## For Training Data
mse = mean_squared_error(ytrain, ytrain_pred)
print(f"Mean Squared Error (MSE) : {mse}")
print("*" * 60)

mae = mean_absolute_error(ytrain, ytrain_pred)
print(f"Mean Absolute Error (MAE) : {mae}")
print("*" * 60)

r2 = r2_score(ytrain, ytrain_pred)
print(f"R-squared (R2) Score : {r2}")
print("*" * 60)
```

```
Mean Squared Error (MSE) : 0.17354564424798538
************************************************************
Mean Absolute Error (MAE) : 0.2410637894249934
************************************************************
R-squared (R2) Score : 0.7177420655396932
************************************************************
```

In [69]:
```
from sklearn.tree import plot_tree

# Assuming dt_reg_model is your trained DecisionTreeRegressor model
plt.figure(figsize=(20,10))
plot_tree(dt_reg_model, filled=True, class_names=["Maligant","Benign"])
plt.savefig("dt.png")
plt.show()
```



In [70]:
```
ytest_pred = dt_reg_model.predict(xtest)
```

In [71]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

## For Testing Data
mse = mean_squared_error(ytest, ytest_pred)
print(f"Mean Squared Error (MSE) : {mse}")
print("*" * 60)
```

```
mae = mean_absolute_error(ytest, ytest_pred)
print(f"Mean Absolute Error (MAE) : {mae}")
print("*" * 60)

r2 = r2_score(ytest, ytest_pred)
print(f"R-squared (R2) Score : {r2}")
print("*" * 60)
```

```
Mean Squared Error (MSE) : 0.20198027863276036
************************************************************
Mean Absolute Error (MAE) : 0.2483986531986532
************************************************************
R-squared (R2) Score : 0.7367611058134207
************************************************************
```

Note: As Decision Tree Always Overfits on its Training data Overfitting : Low Bias and High Variance To overcome it we do Hyperparameter Tunning

Hyperparameter Tunning

In [72]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor

# Define hyperparameters to tune
hyperparameters = {
    "min_samples_split": [2, 6],
    "min_samples_leaf": [11, 4],
    "max_depth": [None, 10, 20],
    "max_features": ["auto", "sqrt", "log2"]
}

# Initialize Decision Tree Regressor
dt_reg = DecisionTreeRegressor()

# Initialize GridSearchCV with the decision tree regressor model and hyperparameters
gscv = GridSearchCV(dt_reg, hyperparameters, cv=5)

# Fit the GridSearchCV object to the training data
gscv.fit(xtrain, ytrain)

# Best parameters found by GridSearchCV
print("Best parameters:", gscv.best_params_)

# Best estimator found by GridSearchCV
dt_reg_hyp_model = gscv.best_estimator_
```

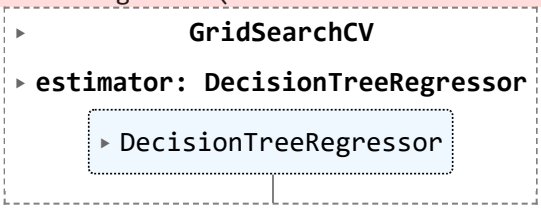
```
Best parameters: {'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 4, 'mi
n_samples_split': 2}
```

```
D:\ANACONDA3\Lib\site-packages\sklearn\model_selection\_validation.py:425: FitFailedW
arning:
60 fits failed out of a total of 180.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score
='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "D:\ANACONDA3\Lib\site-packages\sklearn\model_selection\_validation.py", line
732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "D:\ANACONDA3\Lib\site-packages\sklearn\base.py", line 1144, in wrapper
    estimator._validate_params()
  File "D:\ANACONDA3\Lib\site-packages\sklearn\base.py", line 637, in _validate_param
s
    validate_parameter_constraints(
  File "D:\ANACONDA3\Lib\site-packages\sklearn\utils\_param_validation.py", line 95,
in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter o
f DecisionTreeRegressor must be an int in the range [1, inf), a float in the range
(0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
D:\ANACONDA3\Lib\site-packages\sklearn\model_selection\_search.py:976: UserWarning: O
ne or more of the test scores are non-finite: [       nan        nan        nan
nan 0.49288814 0.42298604
 0.4381298  0.52267302 0.46363183 0.3825466  0.49774617 0.49920131
        nan        nan        nan        nan 0.46738027 0.52067029
 0.48934327 0.47266155 0.39592888 0.33604999 0.54744134 0.53614586
        nan        nan        nan        nan 0.4534483  0.48017041
 0.47631348 0.47314733 0.43657271 0.43166837 0.47664473 0.47815007]
  warnings.warn(
```

In [73]:
```
gscv = GridSearchCV(dt_reg_model,hyperparameters,cv=5)
gscv.fit(xtrain,ytrain)
```

```
D:\ANACONDA3\Lib\site-packages\sklearn\model_selection\_validation.py:425: FitFailedW
arning:
60 fits failed out of a total of 180.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score
='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "D:\ANACONDA3\Lib\site-packages\sklearn\model_selection\_validation.py", line
732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "D:\ANACONDA3\Lib\site-packages\sklearn\base.py", line 1144, in wrapper
    estimator._validate_params()
  File "D:\ANACONDA3\Lib\site-packages\sklearn\base.py", line 637, in _validate_param
s
    validate_parameter_constraints(
  File "D:\ANACONDA3\Lib\site-packages\sklearn\utils\_param_validation.py", line 95,
in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter o
f DecisionTreeRegressor must be an int in the range [1, inf), a float in the range
(0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
D:\ANACONDA3\Lib\site-packages\sklearn\model_selection\_search.py:976: UserWarning: O
ne or more of the test scores are non-finite: [       nan        nan        nan
nan 0.43109428 0.41152785
 0.46983786 0.50539404 0.44550037 0.40402339 0.47918159 0.5348678
        nan        nan        nan        nan 0.52310642 0.33957569
 0.48529627 0.47107627 0.44474896 0.47792606 0.46255052 0.56976013
        nan        nan        nan        nan 0.37632604 0.44281375
 0.50062493 0.54695552 0.46108486 0.51915228 0.43545733 0.46594143]
  warnings.warn(
```

Out[73]:



In [74]:

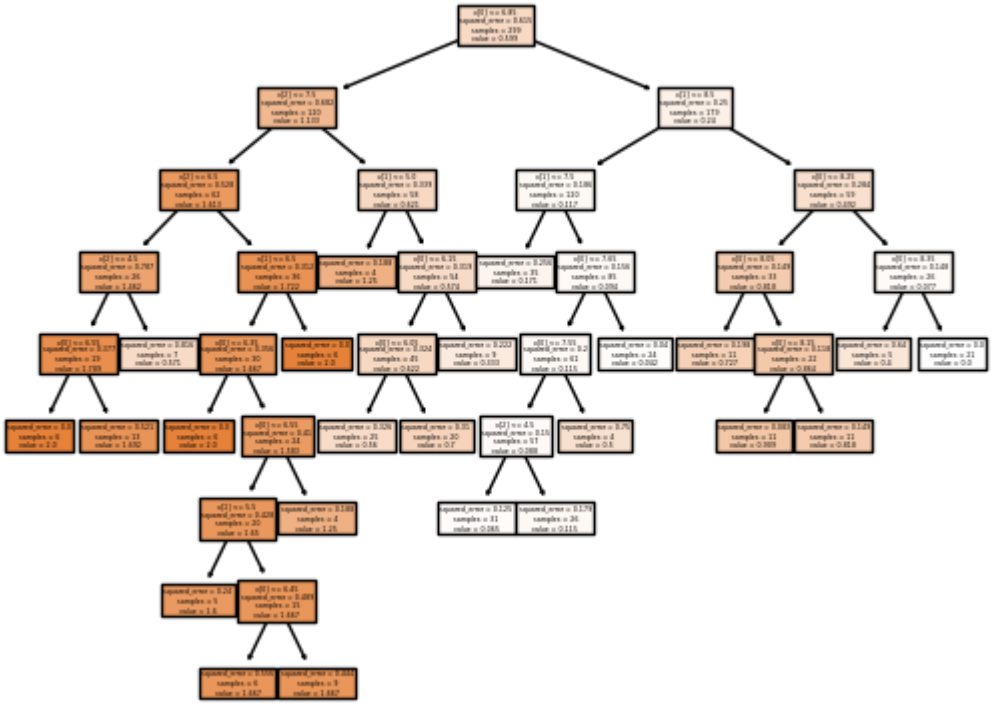
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# For Training Data
ytrain_pred = dt_reg_hyp_model.predict(xtrain)  # Assuming dt_reg_hyp_model is your tr
mse = mean_squared_error(ytrain, ytrain_pred)
print(f"Mean Squared Error (MSE) : {mse}")
print("*" * 60)

mae = mean_absolute_error(ytrain, ytrain_pred)
print(f"Mean Absolute Error (MAE) : {mae}")
print("*" * 60)

r2 = r2_score(ytrain, ytrain_pred)
print(f"R-squared (R2) Score : {r2}")
print("*" * 60)
```

```
Mean Squared Error (MSE) : 0.22828996101895013
************************************************************
Mean Absolute Error (MAE) : 0.3034539441929668
************************************************************
R-squared (R2) Score : 0.6287048681950378
************************************************************
```

In [75]:
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# For Testing Data
ytest_pred = dt_reg_hyp_model.predict(xtest)  # Assuming dt_reg_hyp_model is your trai

mse = mean_squared_error(ytest, ytest_pred)
print(f"Mean Squared Error (MSE) : {mse}")
print("*" * 60)

mae = mean_absolute_error(ytest, ytest_pred)
print(f"Mean Absolute Error (MAE) : {mae}")
print("*" * 60)

r2 = r2_score(ytest, ytest_pred)
print(f"R-squared (R2) Score : {r2}")
print("*" * 60)
```

```
Mean Squared Error (MSE) : 0.20309556400669555
************************************************************
Mean Absolute Error (MAE) : 0.26368442883281595
************************************************************
R-squared (R2) Score : 0.7353075654453981
************************************************************
```

In [76]:
```python
plot_tree(dt_reg_hyp_model,class_names=["0","1"], filled=True)
plt.savefig("dt_hyp.png")
```



In [ ]: