SLEEP HEALTH AND LIFESTYLE ANALYSIS

Name : Anjana Vijayan

Organization : Entri Elevate

Date : 06 April 2025

Overview of Problem Statement:

According to the National Institutes of Health (NIH), quality sleep is as important to good health as diet and exercise. Regularly missing adequate quality sleep can raise the risk of a range of health issues, such as heart disease, stroke, obesity and dementia. The NIH recommends seven to nine hours of sleep a night for the average person.

Sleep health refers to the quality, duration, timing, and regularity of sleep, as well as the behaviors and habits that promote optimal sleep. It encompasses both the physiological aspects of sleep and the psychological and environmental factors that influence sleep patterns. It is essential for one's overall well-being and is associated with numerous physical, cognitive, and emotional benefits.

When sleep health is prioritized, it can lead to improved mood, cognitive function, immune function, cardiovascular health, and overall quality of life. Conversely, chronic sleep deprivation or poor sleep quality can contribute to a range of negative outcomes, including increased risk of obesity, diabetes, cardiovascular disease, depression, and impaired cognitive performance. Therefore, adopting habits and behaviors that support good sleep health is crucial for optimal functioning and well-being.

Objective:

⏳ **How is Fitness tied to Sleep Health**

Fitness plays a significant role in sleep health, as regular physical activity has been shown to positively influence both the quality and duration of sleep. Regular exercise has been associated with shorter sleep onset latency, meaning it may help individuals fall asleep more quickly after going to bed (including those with sleeping disorders). Beyond its direct effects on sleep, regular exercise is associated with numerous health benefits, including reduced risk of stress, obesity, cardiovascular disease, and depression, all of which can impact sleep health.

Data Description:

Dataset Overview: The Sleep Health and Lifestyle Dataset comprises 15000 rows and 13 columns, covering a wide range of variables related to sleep and daily habits. It includes details such as gender, age, occupation, sleep duration, quality of sleep, physical activity level, stress levels, BMI category, blood pressure, heart rate, daily steps, and the presence or absence of sleep disorders.

Source: The dataset can be downloaded from the link :Data

Recognizing Variables in the Dataset

1. **Person ID**: An identifier for each individual.
2. **Sex**: The sex of the person (Male/Female).
3. **Age**: The age of the person in years.
4. **Occupation**: The occupation or profession of the person
5. **Sleep Duration (hours)**: The number of hours the person sleeps per day.
6. **Quality of Sleep (scale: 1-10)**: A subjective rating of the quality of sleep, ranging from 1 to 10.
7. **Physical Activity Level (minutes/day)**: The number of minutes the person engages in physical activity daily.
8. **Stress Level (scale: 1-10)**: A subjective rating of the stress level experienced by the person, ranging from 1 to 10.
9. **BMI Category**: The BMI category of the person (e.g., Underweight, Normal, Overweight).
10. **Blood Pressure (systolic/diastolic)**: The blood pressure measurement of the person, indicated as systolic pressure over diastolic pressure.
11. **Heart Rate (bpm)**: The resting heart rate of the person in beats per minute.
12. **Daily Steps**: The number of steps the person takes per day.
13. **Sleep Disorder**: The presence or absence of a sleep disorder in the person (Healthy, Insomnia, Sleep Apnea).

Sleep Disorder

- **Healthy**: The individual does not exhibit any specific sleep disorder.
- **Insomnia**: The individual experiences difficulty falling asleep or staying asleep, leading to inadequate or poor-quality sleep.
- **Sleep Apnea**: The individual suffers from pauses in breathing during sleep, resulting in disrupted sleep patterns and potential health risks.

Importing necessary libraries

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
```

Loading the data set

```
[ ] df=pd.read_csv('/content/Sleep_Data_Sampled.csv')
    df
```

	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	35	Doctor	6.65	7	50	7	Normal Weight	120/80	71	7100	Healthy
1	2	Male	42	Teacher	6.90	8	52	4	Normal	135/90	66	7000	Healthy
2	3	Male	34	Software Engineer	6.95	7	66	6	Overweight	126/83	74	6100	Healthy
3	4	Male	32	Doctor	6.90	6	52	7	Normal	120/80	71	6500	Healthy
4	5	Male	37	Lawyer	6.85	7	60	6	Normal	125/80	71	6500	Healthy
...	...	...	...	...	...	...	...	...	...	...	...	...	...
14995	14996	Female	59	Nurse	8.10	9	75	3	Overweight	140/95	68	7000	Sleep Apnea
14996	14997	Female	59	Nurse	8.00	9	75	3	Overweight	140/95	68	7000	Sleep Apnea
14997	14998	Female	59	Nurse	8.10	9	75	3	Overweight	140/95	68	7000	Sleep Apnea
14998	14999	Female	59	Nurse	8.10	9	75	3	Overweight	140/95	68	7000	Sleep Apnea
14999	15000	Female	59	Nurse	8.10	9	75	3	Overweight	140/95	68	7000	Sleep Apnea

15000 rows × 13 columns

EXPLORING THE DATA

```
[ ] # Display the number of rows and columns
```

```
print("The Number of Rows :", df.shape[0])
print("The Number of Columns :", df.shape[1])
```

```
The Number of Rows : 15000
The Number of Columns : 13
```

```
#Identifying the types of data
numerical_columns=df.select_dtypes(include=["float","int"])
categorical_columns=df.select_dtypes(include="object")
print("Numerical Columns :",numerical_columns.columns)
print("Categorical Columns :",categorical_columns.columns)
```

```
Numerical Columns : Index(['Person ID', 'Age', 'Sleep Duration', 'Quality of Sleep',
       'Physical Activity Level', 'Stress Level', 'Heart Rate', 'Daily Steps'],
      dtype='object')
Categorical Columns : Index(['Gender', 'Occupation', 'BMI Category', 'Blood Pressure',
       'Sleep Disorder'],
      dtype='object')
```

```
#Checking the basic informations about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Person ID                15000 non-null  int64
 1   Gender                   15000 non-null  object
 2   Age                      15000 non-null  int64
 3   Occupation               15000 non-null  object
 4   Sleep Duration           15000 non-null  float64
 5   Quality of Sleep         15000 non-null  int64
 6   Physical Activity Level  15000 non-null  int64
 7   Stress Level             15000 non-null  int64
 8   BMI Category             15000 non-null  object
 9   Blood Pressure           15000 non-null  object
 10  Heart Rate               15000 non-null  int64
 11  Daily Steps              15000 non-null  int64
 12  Sleep Disorder           15000 non-null  object
dtypes: float64(1), int64(7), object(5)
memory usage: 1.5+ MB
```

DATA PREPROCESSING & ANALYSIS

Handling Missing Values

```
# To check the missing values in the dataset
df.isnull().sum()
```

	0
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

📌 The dataset not have any missing values.

Handling Duplicates

```
# To check the duplicated entries in the dataset.
df.duplicated().sum()
```

```
np.int64(0)
```

📌 The data not have any duplicated entries.

```
# we don't need the "person ID" column for further processing.so we are dropping the column.
df.drop("Person ID",axis=1,inplace=True)
```

```
#Checking descriptive statistics
df.describe()
```

	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate	Daily Steps
count	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000
mean	44.130667	6.997327	7.131267	59.925000	5.654800	70.857533	6795.080000
std	6.840091	0.615187	1.053111	16.814374	1.393568	3.614836	1329.706484
min	27.000000	5.800000	4.000000	30.000000	3.000000	65.000000	3000.000000
25%	40.000000	6.500000	6.000000	45.000000	4.000000	68.000000	6000.000000
50%	44.000000	7.000000	7.000000	60.000000	6.000000	70.000000	6500.000000
75%	48.000000	7.450000	8.000000	75.000000	6.000000	72.000000	7600.000000
max	59.000000	8.500000	9.000000	90.000000	8.000000	86.000000	10000.000000

📌 No major difference between mean and median of any column except "Daily Steps".We can see some difference between mean and median of "Daily Steps" column.Mean is greater than the median, which indicates that particular column is possitively skewed. The difference may also indivcates the presence of outliers in the data.

Handling Skewness

```
# To check the skewness of the data
skewness=df.select_dtypes(include='number').skew()
skewness
```

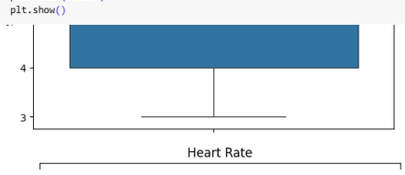
	0
Age	0.097985
Sleep Duration	0.346637
Quality of Sleep	-0.070315
Physical Activity Level	0.204935
Stress Level	-0.150428
Heart Rate	0.735076
Daily Steps	0.464187

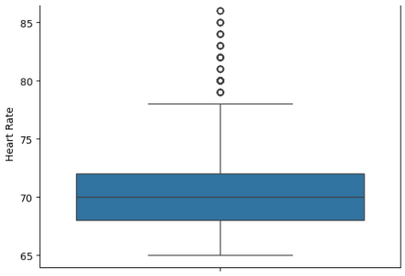
dtype: float64

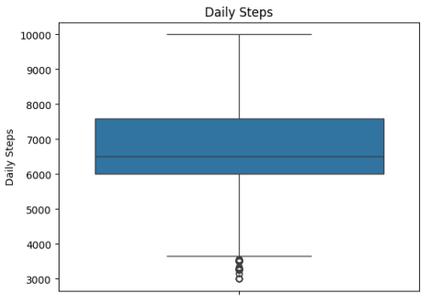
📌 Heart Rate column is moderately possitively skewed.

Handling Outliers

```
# checking outliers visually using boxplot
numerical_columns=df.select_dtypes(include=["float","int"])
for column in numerical_columns:
  sns.boxplot(df[column])
  plt.title(column)
  plt.show()
```





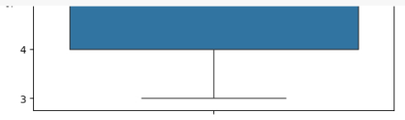


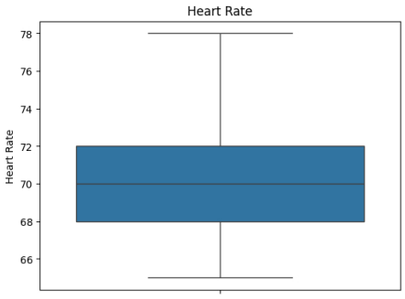
📌 Daily steps and Heart rate columns contains outliers. we have to treat it properly.

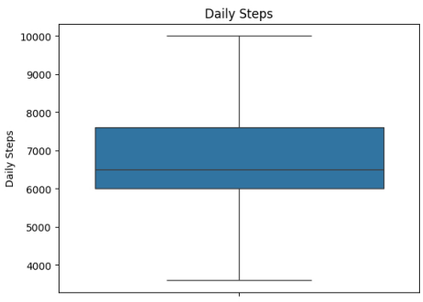
```
[ ] # To handle outliers using IQR Method
    def handling_outliers(df,columns):
      for column in columns:
        if df[column].dtype in ["float","int"]:
          Q1=df[column].quantile(0.25)
          Q3=df[column].quantile(0.75)
          IQR=Q3-Q1
          lower_bound=Q1-1.5*IQR
          upper_bound=Q3+1.5*IQR
          # To clip the outliers
          df[column]=df[column].clip(lower_bound,upper_bound)
      return df
```

```
[ ] # cleaning the data using the 'handle_ouliers' function.
    df=handling_outliers(df,df.columns)
```

```
[ ] # Visually checking whether the outliers are handled properly or not.
    numerical_columns=df.select_dtypes(include=["float","int"])
    for column in numerical_columns:
      sns.boxplot(df[column])
      plt.title(column)
      plt.show()
```







📌 The ouliers are handled properly.

```
[ ] # Checking skewness of the data after handling the outliers
    skewness=df.select_dtypes(include='number').skew()
    skewness
```

	0
Age	0.098136
Sleep Duration	0.346637
Quality of Sleep	-0.070315
Physical Activity Level	0.204935
Stress Level	-0.150428
Heart Rate	0.424716
Daily Steps	0.476226

dtype: float64

📌 Age,Sleep Duration,Physical Activity Level,Heart Rate and Daily Steps columns are slightly possitively skewed.Quality of Sleep and Stress Level columns are slighly negatively skewed.But the skewness values are closer to zero (between -0.5 to 0.5) so we don't need to apply any transformation to the data.

```
[ ] # displaying the cleaned data(df)
    df.head()
```

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate	Daily Steps	Sleep Disorder
0	Male	35	Doctor	6.65	7	50	7	Normal Weight	120/80	71	7100	Healthy
1	Male	42	Teacher	6.90	8	52	4	Normal	135/90	66	7000	Healthy
2	Male	34	Software Engineer	6.95	7	66	6	Overweight	126/83	74	6100	Healthy
3	Male	32	Doctor	6.90	6	52	7	Normal	120/80	71	6500	Healthy
4	Male	37	Lawyer	6.85	7	60	6	Normal	125/80	71	6500	Healthy

DATA ANALYSIS

Exploratory Data Analysis (EDA)

```
# checking unique values
unique__values=[]
for column in df.columns:
  count=df[column].value_counts().count()
  unique__values.append(count)
pd.DataFrame(unique__values,index=df.columns,columns=["Total Unique Values"])
```

	Total Unique Values
Gender	2
Age	32
Occupation	11
Sleep Duration	55
Quality of Sleep	6
Physical Activity Level	37
Stress Level	6
BMI Category	4
Blood Pressure	25
Heart Rate	14
Daily Steps	74
Sleep Disorder	3

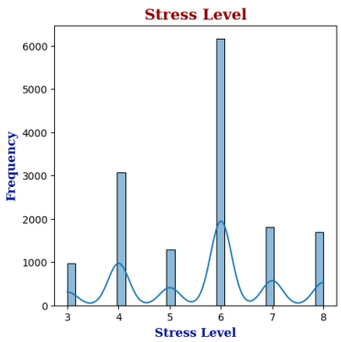
```
# Checking the statistical summery of the data
df.describe()
```

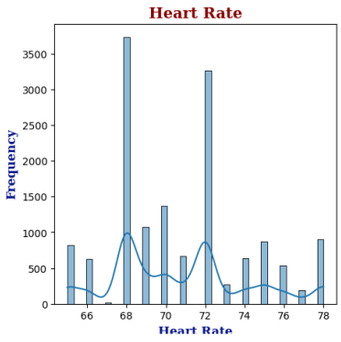
	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate	Daily Steps
count	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000
mean	44.130733	6.997327	7.131267	59.925000	5.654800	70.771533	6795.880000
std	6.839929	0.615187	1.053111	16.814374	1.393568	3.395188	1327.673607
min	28.000000	5.800000	4.000000	30.000000	3.000000	65.000000	3600.000000
25%	40.000000	6.500000	6.000000	45.000000	4.000000	68.000000	6000.000000
50%	44.000000	7.000000	7.000000	60.000000	6.000000	70.000000	6500.000000
75%	48.000000	7.450000	8.000000	75.000000	6.000000	72.000000	7600.000000
max	59.000000	8.500000	9.000000	90.000000	8.000000	78.000000	10000.000000

📌 The data seems uniformly distributed. We can not see any major difference between mean and median. A minute difference is there in 'Daily Steps" column which is due to the slight skewness in that particular column. We have already treated the outliers so the statistical summery gives an over view of cleaned data.

Univariate Analysis

```
# Analysing numerical variables using Histplot
title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
numerical_columns=df.select_dtypes(include=["float","int"])
for column in numerical_columns:
  plt.figure(figsize=(5,5))
  sns.histplot(df[column],kde=True)
  plt.title(column,fontdict=title_font)
  plt.xlabel(column,fontdict=axis_font)
  plt.ylabel("Frequency",fontdict=axis_font)
  plt.show()
```



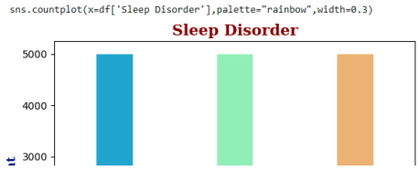


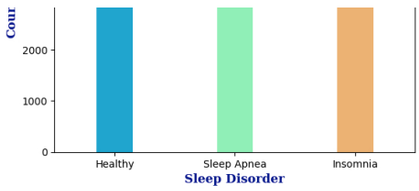
Insights:

- Age Distribution: The age distribution has uniform distribution with multiple peaks, indicating that the dataset might contain people from different age groups.
- Sleep Duration: The histogram shows a peak around 6-7 hours,that is 6.5 hrs. The majority seem to fall within this range..
- Physical Activity Level: The data has sharp peaks, indicating that certain activity levels are more common than others.
- Stress Level: The stress level data shows a similar pattern to sleep quality.Most of the data has a stress level of 6.
- Heart Rate: The heart rate histogram has multiple peaks.
- Daily Steps: The distribution is spread out, with higher frequencies in certain ranges. This could indicate that certain activity levels are more common.
- Some variables (Quality of Sleep, Stress Level, Physical Activity) seem to have discrete values rather than continuous distributions.
- There are multiple peaks in several distributions, which may indicate clustering or distinct groups within the dataset.
- Some data distributions appear skewed, suggesting that certain behaviors or characteristics are widespread..

```
#count plot for categorical columns-Sleep Disorder
title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
sns.countplot(x=df['Sleep Disorder'],palette="rainbow",width=0.3)
plt.title('Sleep Disorder',fontdict=title_font)
plt.xlabel('Sleep Disorder',fontdict=axis_font)
plt.ylabel('Count',fontdict=axis_font)
plt.show()
```

```
<ipython-input-20-95baa698f5ed>:4: 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.

  sns.countplot(x=df['Sleep Disorder'],palette="rainbow",width=0.3)
```



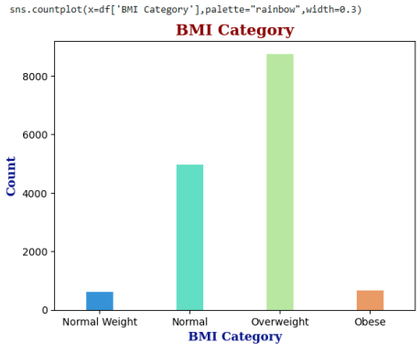


Insights:

- Sleep Disorder is the target column of this dataset. The data is equaly distributed among different classes like 'Healthy','Sleep Apnea' and 'Insomnia'.
- Tha data is balanced.

```
[ ] #count plot for categorical columns-BMI Category
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    sns.countplot(x=df['BMI Category'],palette="rainbow",width=0.3)
    plt.title('BMI Category',fontdict=title_font)
    plt.xlabel('BMI Category',fontdict=axis_font)
    plt.ylabel('Count',fontdict=axis_font)
    plt.show()
```

```
<ipython-input-21-7a5c56766cdf>:4: 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.

  sns.countplot(x=df['BMI Category'],palette="rainbow",width=0.3)
```

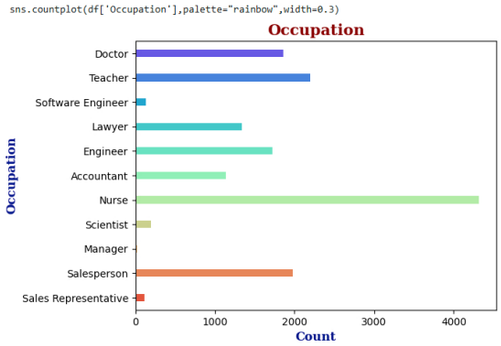


Insights:

- There 4 kinds of BMI categories.
- Most of the people belongs to the 'Overweight' category.
- 'Normal' and 'Obese' categories holds the second and'Obese' holds the third positions respectively.
- 'Normal Weight' category has the mininum number of entries.

```
[ ] #count plot for categorical columns-BMI Category
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    sns.countplot(df['Occupation'],palette="rainbow",width=0.3)
    plt.title('Occupation',fontdict=title_font)
    plt.xlabel('Count',fontdict=axis_font)
    plt.ylabel('Occupation',fontdict=axis_font)
    plt.show()
```

```
<ipython-input-22-3622560be6d1>: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.countplot(df['Occupation'],palette="rainbow",width=0.3)
```



Insights:

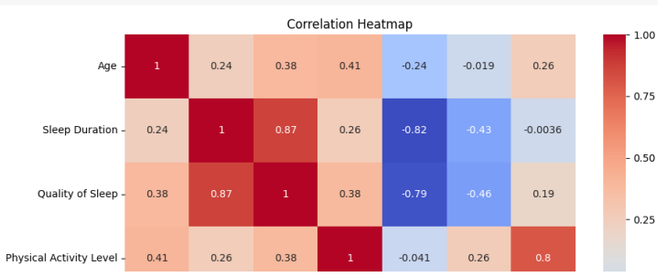
- 'Nurses' have the highest representation in the dataset, significantly higher than all other occupations.
- 'Teachers' , 'Salespersons' and 'Doctors' also have relatively high counts, suggesting these professions are well-represented in the dataset.
- 'Managers', 'Software Engineers', 'Scientists', and 'Sales Representatives' have the lowest representation.
- Since 'Nurses' and 'Doctors' have comparatively higher count, the data is highly representing healthcare sector compared to other professions.

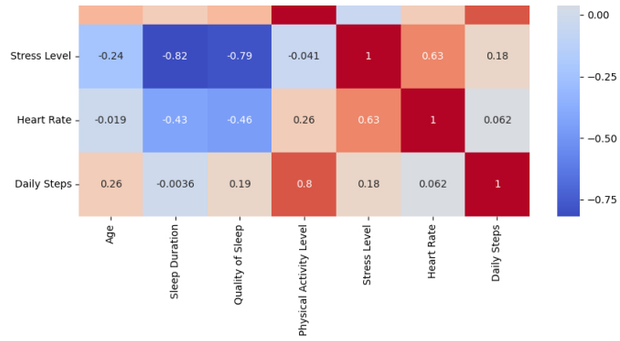
Bivariate Analysis

```
[ ] # checking the correlation between the features.
    correlation=df.select_dtypes("number").corr()
    correlation
```

	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate	Daily Steps
Age	1.000000	0.242350	0.382185	0.405466	-0.236079	-0.019318	0.257976
Sleep Duration	0.242350	1.000000	0.871070	0.258421	-0.819078	-0.426138	-0.003600
Quality of Sleep	0.382185	0.871070	1.000000	0.381601	-0.785485	-0.457982	0.188231
Physical Activity Level	0.405466	0.258421	0.381601	1.000000	-0.040732	0.261873	0.802956
Stress Level	-0.236079	-0.819078	-0.785485	-0.040732	1.000000	0.629591	0.184221
Heart Rate	-0.019318	-0.426138	-0.457982	0.261873	0.629591	1.000000	0.062251
Daily Steps	0.257976	-0.003600	0.188231	0.802956	0.184221	0.062251	1.000000

```
[ ] # visually analyzing the correlation between the features using Heatmap.
    plt.figure(figsize=(10,8))
    sns.heatmap(correlation, annot=True, cmap="coolwarm")
    plt.title("Correlation Heatmap")
    plt.show()
```





Insights:

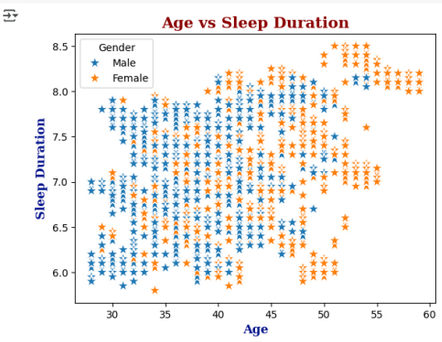
Strong Positive Correlations:

- Sleep duration has a high possitive correlation with Quality of sleep(0.87).
- Higher Physical activity levels correspond to more Daily steps.(0.80)
- Stress level and Heart rate has possitive correlation each other(0.63).Higher stress level associated with increased Heart rate.

Strong Negative Correlations:

- Sleep Duration & Stress Level (-0.82): People who sleep longer tend to have lower stress levels.
- Quality of Sleep & Stress Level (-0.79): Better sleep quality is associated to lower stress.
- Heart Rate & Quality of Sleep (-0.46): Higher heart rates are associated with poorer sleep quality.

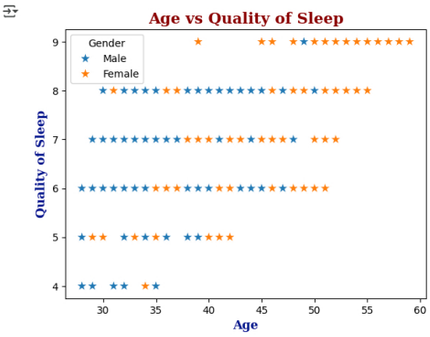
```
[ ] # Scatter plot-Age vs Sleep Duration
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    sns.scatterplot(data=df,x='Age',y='Sleep Duration',hue=df['Gender'],marker="*",s=150)
    plt.xlabel('Age',fontdict=axis_font)
    plt.ylabel('Sleep Duration',fontdict=axis_font)
    plt.title('Age vs Sleep Duration',fontdict=title_font)
    plt.show()
```



Insights:

- Yonger individuals have a wide range of sleep duration
- The sleep duration is maximum in the 55-60 age group, they are sleeping for almost 8.5 hrs.
- Both male and females exhibit a similar distribution in sleep duration,but females shows slightly longer sleep duration in the older ages.
- Between ages 30–50, sleep duration is mostly concentrated around 6.5 to 7.5 hours.

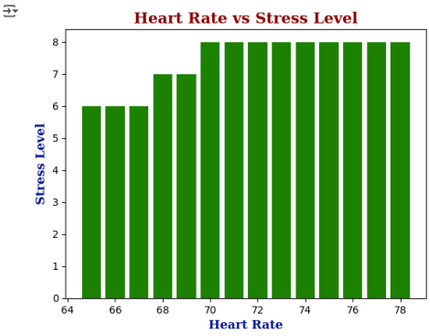
```
[ ] # Scatter Plot- Age vs Sleep Duration
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    sns.scatterplot(data=df,x="Age",y="Quality of Sleep",hue=df['Gender'],color="orange",marker="*",s=150)
    plt.xlabel('Age',fontdict=axis_font)
    plt.ylabel('Quality of Sleep',fontdict=axis_font)
    plt.title('Age vs Quality of Sleep',fontdict=title_font)
    plt.show()
```



Insights:

- we have seen in the scatterplot of Age vs Sleep Duration, the quality of sleep is maximum in the age group 55-60.
- In the older age groups, female shows high quality of sleep compared to male.
- Most of the people have the sleep quality lies in the range of 6-8.

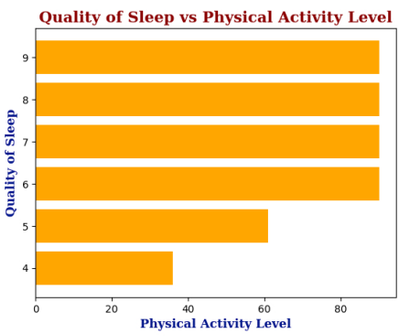
```
[ ] # Heart Rate vs Stress Level
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    plt.bar(df['Heart Rate'],df['Stress Level'],color="g")
    plt.xlabel('Heart Rate',fontdict=axis_font)
    plt.ylabel('Stress Level',fontdict=axis_font)
    plt.title('Heart Rate vs Stress Level',fontdict=title_font)
    plt.show()
```



Insights:

- The heart rate is increasing as the stress level increases.
- The people with high stress level also have an increased heart rate compared to others.
- When the Heart rate is 70 or higher,the Stress level reaches the maximum (8).

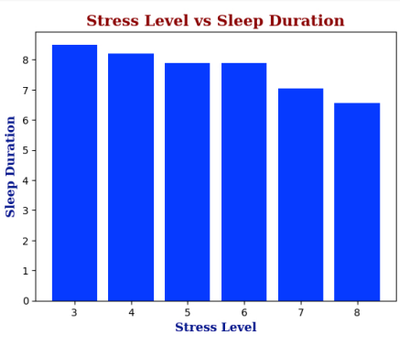
```
[ ] # Quality of Sleep vs Physical Activity Level
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    plt.barh(df['Quality of Sleep'],df['Physical Activity Level'],color="orange")
    plt.xlabel('Physical Activity Level',fontdict=axis_font)
    plt.ylabel('Quality of Sleep',fontdict=axis_font)
    plt.title('Quality of Sleep vs Physical Activity Level',fontdict=title_font)
    plt.show()
```



Insights:

- The quality of sleep is increasing with the intensity of physical activity increases.
- The people who is physically more active, they are getting good quality sleep .

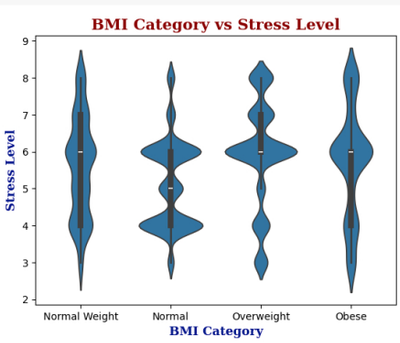
```
[ ] # Stress Level vs Sleep Duration
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    plt.bar(df['Stress Level'],df['Sleep Duration'],color="blue")
    plt.xlabel('Stress Level',fontdict=axis_font)
    plt.ylabel('Sleep Duration',fontdict=axis_font)
    plt.title('Stress Level vs Sleep Duration',fontdict=title_font)
    plt.show()
```



Insights:

- It is clearly visible that, people with high stress level have comparatively low sleep duration.
- As the stress level increases, the sleep duration decreases.

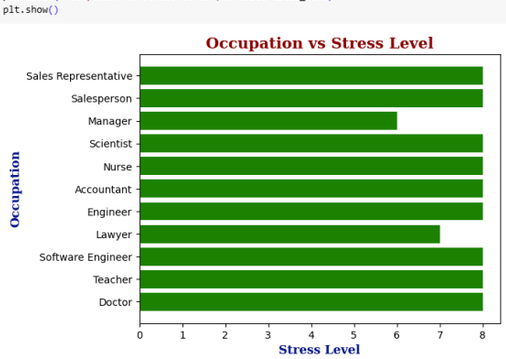
```
[ ] #violin plot - BMI Category vs Stress Level
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    sns.violinplot(x=df['BMI Category'],y=df['Stress Level'])
    plt.xlabel('BMI Category',fontdict=axis_font)
    plt.ylabel('Stress Level',fontdict=axis_font)
    plt.title('BMI Category vs Stress Level',fontdict=title_font)
    plt.show()
```



Insights:

- All BMI categories show a wide distribution of stress levels, indicating that individuals in each category experience a range of stress.
- The median stress level(6) appears higher for overweight and obese individuals compared to normal-weight individuals.

```
[ ] # Occupation vs Stress Level
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    plt.barh(df['Occupation'],df['Stress Level'],color="g")
    plt.xlabel('Stress Level',fontdict=axis_font)
    plt.ylabel('Occupation',fontdict=axis_font)
    plt.title('Occupation vs Stress Level',fontdict=title_font)
    plt.show()
```

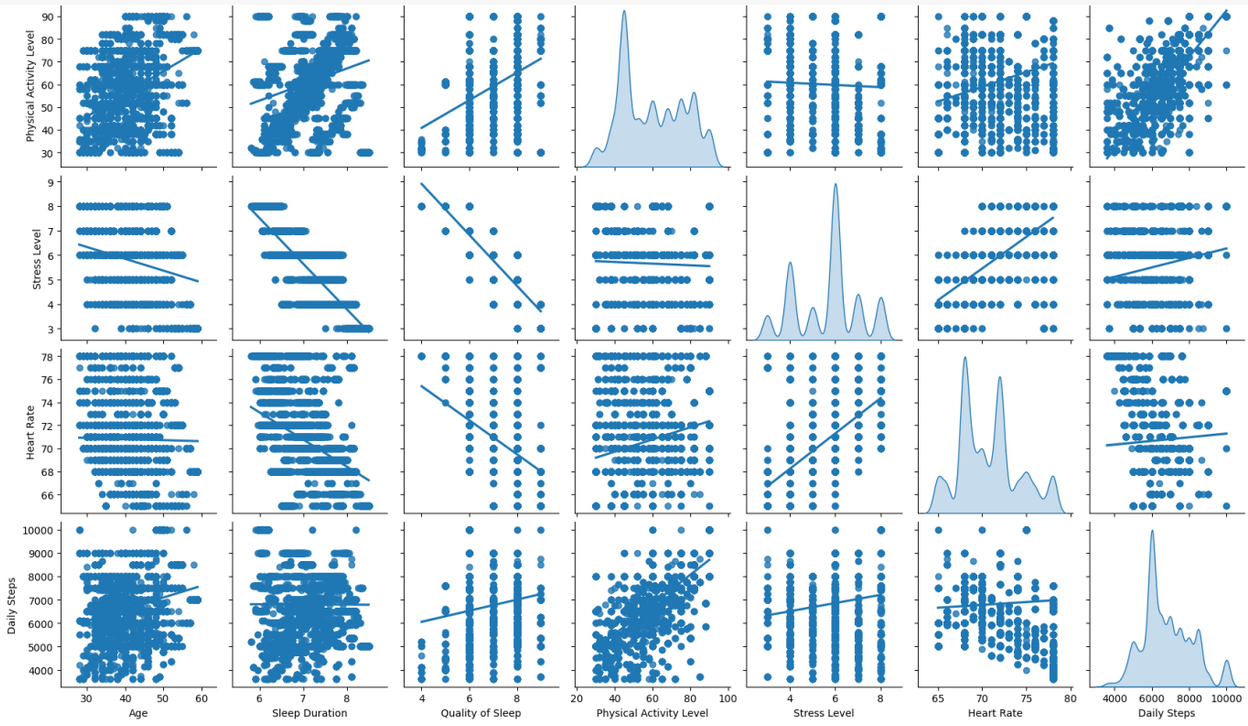


Insights:

- The stress level is high for almost all the occupation types.
- Managers and Lawyers have the lowest reported stress levels in this chart.
- So irrespective of the occupation type, the stress level is highly depends on other fitness factors of the body.

Multivariate Analysis

```
[ ] # pairplot
    plt.figure(figsize=(8,6))
    sns.pairplot(df,kind="reg",diag_kind="kde")
    plt.show()
```



Insights:

- we can see how the data is distributed among different features and how the features are related to eachother.
- Diagonal graphs are univarient kde plots.
- Age has a possitive correlation with Quality of sleep and slight negative correlation with stress level.
- Sleep duration has high possitive correlation with Quality of sleep and a high negative correlation with Stress level.
- Quality of sleep has possitive correlation with Age, Sleep duration, Physical activity level and daily steps. Bt it has a negative correlation with Stress level and Heart rate.
- Physical activity level has a possitive correlation with all other features except stress level.
- Stress level has negative correlation with all other features except Heart rate.
- Some variables show a clear linear relationship(Sleep Duration vs Quality of sleep)

```
[ ] # To analyse the class balance
    y=df['Sleep Disorder'] # target
    class_balance=df['Sleep Disorder'].value_counts()
    class_balance
```

Sleep Disorder	count
Healthy	5000
Sleep Apnea	5000
Insomnia	5000

dtype: int64

The data is equaly distributed in the 3 different classes. So the data is balanced.

FEATURE ENGINEERING

Feature Encoding

Encoding is the process of converting categorical or textual data into numerical formats. Most of the Machine Learning models can not work directly with non-numerical data, so encoding is one of the most essential steps to transform the data for Machine Learning algorithms. Label Encoding,One-Hot Encoding,Target Encoding etc are commonly used encoding techniques.I am using One-Hot Encoding technique for encoding my dataset.

```
[ ] # Importing One-Hot Encoder and Target Encoder
    from sklearn.preprocessing import OneHotEncoder
```

```
[ ] X=df.drop('Sleep Disorder',axis=1) #Features
    y=df['Sleep Disorder'] #Target
```

```
[ ] # analysing the categorical_features
    categorical_features=X.select_dtypes(include="object")
    categorical_features.head()
```

	Gender	Occupation	BMI Category	Blood Pressure
0	Male	Doctor	Normal Weight	120/80
1	Male	Teacher	Normal	135/90
2	Male	Software Engineer	Overweight	126/83
3	Male	Doctor	Normal	120/80
4	Male	Lawyer	Normal	125/80

```
[ ] # checking unique values in categorical features
    unique__values=[]
    for column in categorical_features.columns:
      count=categorical_features[column].value_counts().count()
      unique__values.append(count)
    pd.DataFrame(unique__values,index=categorical_features.columns,columns=["Total Unique Values"])
```

	Total Unique Values
Gender	2
Occupation	11
BMI Category	4
Blood Pressure	25

📌"Blood Pressure" contains 25 unique values."Gender" has 2, "Occupation" has 11 and "BMI Category" has 4 unique values. One Hot Encoding is using to encode categorical features such as "Gender","Occupation","Blood Pressure" and "BMI Category".

One-Hot Encoding

One-hot encoding transforms categorical values into a series of binary vectors, where each category is represented as a unique vector with a single 1 and all other values as 0.

```
[ ] #Encoding "Gender","Occupation", "Blood Pressure" and "BMI Category" columns using OneHotEncoder
    X_encoded=pd.get_dummies(X,columns=["Gender","Occupation","BMI Category", "Blood Pressure"],drop_first=True,dtype="int")
    X_encoded.head()
```

	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate	Daily Steps	Gender_Male	Occupation_Doctor	Occupation_Engineer	...	Blood Pressure_130/85	Blood Pressure_130/86	Blood Pressure_131/86	Blood Pressure_132/87	Blood Pressure_135/88	Blood Pressure_135/90	Blood Pressure_139/91	Pr

											...							
0	35	6.65	7	50	7	71	7100	1	1	0	...	0	0	0	0	0	0	0
1	42	6.90	8	52	4	66	7000	1	0	0	...	0	0	0	0	0	1	0
2	34	6.95	7	66	6	74	6100	1	0	0	...	0	0	0	0	0	0	0
3	32	6.90	6	52	7	71	6500	1	1	0	...	0	0	0	0	0	0	0
4	37	6.85	7	60	6	71	6500	1	0	0	...	0	0	0	0	0	0	0

5 rows × 45 columns

```
[ ] # Checking the basic information about the encoded features
    X_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 45 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Age                                15000 non-null  int64  
 1   Sleep Duration                     15000 non-null  float64
 2   Quality of Sleep                   15000 non-null  int64  
 3   Physical Activity Level            15000 non-null  int64  
 4   Stress Level                       15000 non-null  int64  
 5   Heart Rate                         15000 non-null  int64  
 6   Daily Steps                        15000 non-null  int64  
 7   Gender_Male                        15000 non-null  int64  
 8   Occupation_Doctor                  15000 non-null  int64  
 9   Occupation_Engineer                15000 non-null  int64  
 10  Occupation_Lawyer                  15000 non-null  int64  
 11  Occupation_Manager                 15000 non-null  int64  
 12  Occupation_Nurse                   15000 non-null  int64  
 13  Occupation_Sales Representative    15000 non-null  int64  
 14  Occupation_Salesperson             15000 non-null  int64  
 15  Occupation_Scientist               15000 non-null  int64  
 16  Occupation_Software Engineer       15000 non-null  int64  
 17  Occupation_Teacher                 15000 non-null  int64  
 18  BMI Category_Normal Weight         15000 non-null  int64  
 19  BMI Category_Obese                 15000 non-null  int64  
 20  BMI Category_Overweight            15000 non-null  int64  
 21  Blood Pressure_115/78              15000 non-null  int64  
 22  Blood Pressure_117/76              15000 non-null  int64  
 23  Blood Pressure_118/75              15000 non-null  int64  
 24  Blood Pressure_118/76              15000 non-null  int64  
 25  Blood Pressure_119/77              15000 non-null  int64  
 26  Blood Pressure_120/80              15000 non-null  int64  
 27  Blood Pressure_121/79              15000 non-null  int64  
 28  Blood Pressure_122/80              15000 non-null  int64  
 29  Blood Pressure_125/80              15000 non-null  int64  
 30  Blood Pressure_125/82              15000 non-null  int64  
 31  Blood Pressure_126/83              15000 non-null  int64  
 32  Blood Pressure_128/84              15000 non-null  int64  
 33  Blood Pressure_128/85              15000 non-null  int64  
 34  Blood Pressure_129/84              15000 non-null  int64  
 35  Blood Pressure_130/85              15000 non-null  int64  
 36  Blood Pressure_130/86              15000 non-null  int64  
 37  Blood Pressure_131/86              15000 non-null  int64  
 38  Blood Pressure_132/87              15000 non-null  int64  
 39  Blood Pressure_135/88              15000 non-null  int64  
 40  Blood Pressure_135/90              15000 non-null  int64  
 41  Blood Pressure_139/91              15000 non-null  int64  
 42  Blood Pressure_140/90              15000 non-null  int64  
 43  Blood Pressure_140/95              15000 non-null  int64  
 44  Blood Pressure_142/92              15000 non-null  int64  
dtypes: float64(1), int64(44)
memory usage: 5.1 MB
```

📌 After encoding, we have got 45 features. So we have to select the features with more importance to train the ML model to avoid over fitting or under fitting. The next step will be Feature selection.

Feature Selection

Feature selection is the process of choosing the most relevant features (independent variables) for a machine learning model. It helps improve model accuracy, reduce overfitting, and speed up training by eliminating irrelevant or redundant features.Mainly there are to methods for feature selection,

1. Select K Best using Cross-Validation (CV)
2. Feature Selection Using Random Forest

Here I am using "Feature Selection Using Random Forest"

Feature Selection Using Random Forest Random Forest is an ensemble learning method that provides built-in feature importance based on how much a feature contributes to reducing impurity (Gini or MSE).

✓ Handles non-linearity and complex feature interactions.

✓ Works well with high-dimensional data.

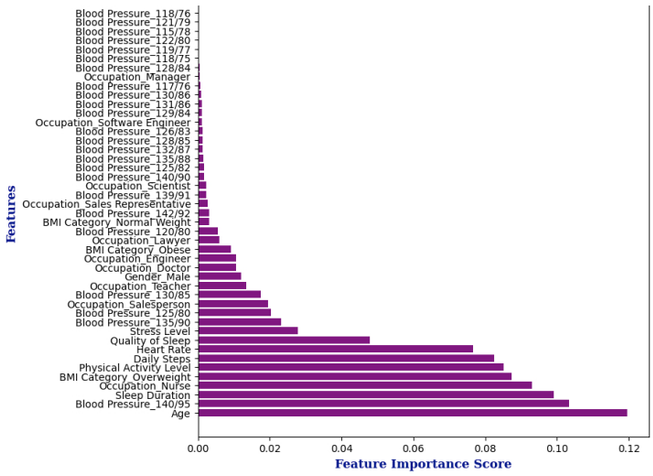
✓ Fast & scalable compared to wrapper methods.

✓ No need for additional feature selection techniques.

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier

    # Split data
    X_train,X_test,y_train,y_test=train_test_split(X_encoded,y,test_size=0.2,random_state=42)

    # Train Random Forest Model
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Get feature importance
    feature_importance=pd.Series(model.feature_importances_)

    # Create a DataFrame to display feature importances
    feature_importance = pd.DataFrame({ 'Feature': X_encoded.columns,
                                        'Importance': feature_importance}).sort_values(by='Importance', ascending=False)
    print(feature_importance)
```

```
                          Feature  Importance
0                             Age    0.119545
43           Blood Pressure_140/95    0.103430
1                   Sleep Duration    0.099177
12                Occupation_Nurse    0.092968
20         BMI Category_Overweight    0.087449
3          Physical Activity Level    0.085151
6                      Daily Steps    0.082499
5                       Heart Rate    0.076648
2                  Quality of Sleep    0.047854
4                     Stress Level    0.027894
40           Blood Pressure_135/90    0.023178
29           Blood Pressure_125/80    0.020354
14          Occupation_Salesperson    0.019566
35           Blood Pressure_130/85    0.017490
17              Occupation_Teacher    0.013383
7                      Gender_Male    0.011958
8                 Occupation_Doctor    0.010696
9               Occupation_Engineer    0.010648
19               BMI Category_Obese    0.009267
10                Occupation_Lawyer    0.005933
26           Blood Pressure_120/80    0.005591
18      BMI Category_Normal Weight    0.003214
44           Blood Pressure_142/92    0.003062
13  Occupation_Sales Representative    0.002751
41           Blood Pressure_139/91    0.002294
15             Occupation_Scientist    0.002233
42           Blood Pressure_140/90    0.001644
30           Blood Pressure_125/82    0.001631
39           Blood Pressure_135/88    0.001564
38           Blood Pressure_132/87    0.001395
33           Blood Pressure_128/85    0.001389
31           Blood Pressure_126/83    0.001230
16     Occupation_Software Engineer    0.001190
34           Blood Pressure_129/84    0.001131
37           Blood Pressure_131/86    0.001045
36           Blood Pressure_130/86    0.000923
22           Blood Pressure_117/76    0.000612
11              Occupation_Manager    0.000555
32           Blood Pressure_128/84    0.000523
23           Blood Pressure_118/75    0.000350
25           Blood Pressure_119/77    0.000302
28           Blood Pressure_122/80    0.000097
21           Blood Pressure_115/78    0.000096
27           Blood Pressure_121/79    0.000070
24           Blood Pressure_118/76    0.000019
```

```
[ ] # Create the plot with sorted feature importances
    plt.figure(figsize=(8,8))
    plt.barh(feature_importance["Feature"],feature_importance["Importance"], color='purple')

    # Add labels to your graph
    title_font={"family":"serif","color":"darkred","weight":"bold","size":15}
    axis_font={"family":"serif","color":"darkblue","weight":"bold","size":12}
    plt.xlabel('Feature Importance Score',fontdict=axis_font)
    plt.ylabel('Features',fontdict=axis_font)
    plt.title("Visualizing Important Features",fontdict=title_font)
    plt.show()
```



📌 The graph represents the feature importance of different features after encoding the data. When selecting the features using Random forest Classifier/Reggressor, we can set a threshold which helps in adjusting decision boundaries. Here I am setting a threshold of >0.02 and selecting the fatures with feature importance above the threshold.

```
[ ] # Setting the  threshold
    threshold = 0.02

    # Select features with importance above the threshold
    selected_features = feature_importance[feature_importance['Importance'] > threshold]

    print("Selected Features:")
    print(selected_features)
```

```
Selected Features:
                    Feature  Importance
0                       Age    0.119545
43    Blood Pressure_140/95    0.103430
1            Sleep Duration    0.099177
12           Occupation_Nurse    0.092968
20  BMI Category_Overweight    0.087449
3   Physical Activity Level    0.085151
6               Daily Steps    0.082499
5                Heart Rate    0.076648
2          Quality of Sleep    0.047854
4               Stress Level    0.027894
40    Blood Pressure_135/90    0.023178
29    Blood Pressure_125/80    0.020354
```

```
[ ] len(selected_features)
```

```
12
```

📌 We have got 12 selected features which has an importance above the threshold(0.02).

```
[ ] X_selected=X_encoded[selected_features['Feature']]
    X_selected.head()
```

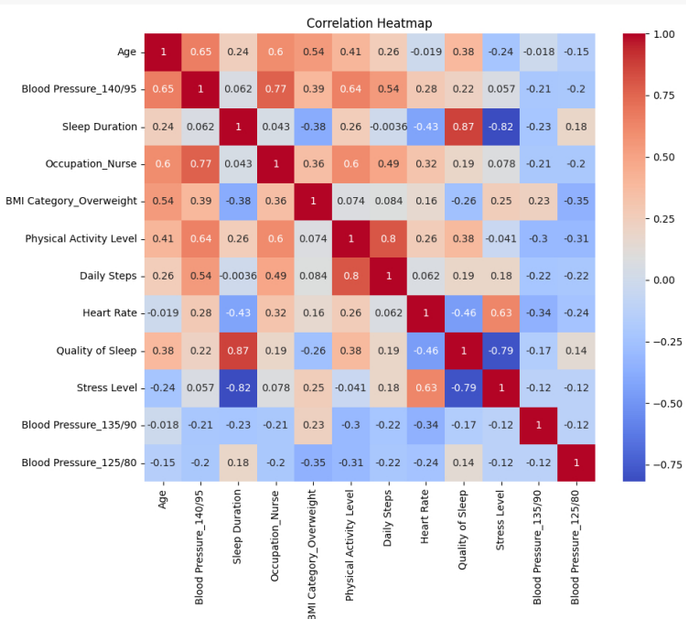
	Age	Blood Pressure_140/95	Sleep Duration	Occupation_Nurse	BMI Category_Overweight	Physical Activity Level	Daily Steps	Heart Rate	Quality of Sleep	Stress Level	Blood Pressure_135/90	Blood Pressure_125/80
0	35	0	6.65	0	0	50	7100	71	7	7	0	0
1	42	0	6.90	0	0	52	7000	66	8	4	1	0
2	34	0	6.95	0	1	66	6100	74	7	6	0	0
3	32	0	6.90	0	0	52	6500	71	6	7	0	0
4	37	0	6.85	0	0	60	6500	71	7	6	0	1

```
[ ] # checking the information about selected features(X_selected)
    X_selected.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       15000 non-null  int64
 1   Blood Pressure_140/95     15000 non-null  int64
 2   Sleep Duration            15000 non-null  float64
 3   Occupation_Nurse          15000 non-null  int64
 4   BMI Category_Overweight   15000 non-null  int64
 5   Physical Activity Level   15000 non-null  int64
 6   Daily Steps               15000 non-null  int64
 7   Heart Rate                15000 non-null  int64
 8   Quality of Sleep          15000 non-null  int64
 9   Stress Level              15000 non-null  int64
 10  Blood Pressure_135/90     15000 non-null  int64
 11  Blood Pressure_125/80     15000 non-null  int64
dtypes: float64(1), int64(11)
memory usage: 1.4 MB
```

📌 X_selected has 12 columns(features) and 15000 rows.

```
[ ] #Checking the correlation between selected features
    correlation=X_selected.corr()
    plt.figure(figsize=(10,8))
    sns.heatmap(correlation,annot=True,cmap="coolwarm")
    plt.title("Correlation Heatmap")
    plt.show()
```



📌 Insights

Strong Correlations (>|0.7|)

- Sleep Duration ↔ Quality of Sleep:+0.87
- Physical Activity Level ↔ Daily Steps:+0.80
- Stress Level ↔ Quality of Sleep:-0.79
- Sleep Duration ↔ Stress Level:-0.82
- Occupation_Nurse ↔ Blood Pressure_140/95:+0.77

📌 The correlation heat map is giving **Multicollinearity Alert** .So we need to apply PCA to this features.We need to do feature scaling before applying PCA.

Feature Scaling

Standardization:Standardization allows the model to learn better by equalizing feature ranges, avoiding biases that arise from unbalanced ranges.Standand Scaler and Min Max Scaler are the most commonly used scaling methods. Here I am going to use standard scaler method.

📌Usually we do not sacle binary columns.So before scaling, we have to identify binary columns and non binary columns.

```
[ ] # Identify Binary Columns (Columns with Only 2 Unique Values)
    binary_columns=[column for column in X_selected.columns if X_selected[column].nunique()==2]

    # Identify Non-Binary Numeric Columns
    numerical_columns=X_selected.select_dtypes(include=["float","int"]).columns
    non_binary_columns=[column for column in numerical_columns if column not in binary_columns]

    print("Binary columns:", binary_columns)
    print("Non Binary columns:",non_binary_columns)
```

```
Binary columns: ['Blood Pressure_140/95', 'Occupation_Nurse', 'BMI Category_Overweight', 'Blood Pressure_135/90', 'Blood Pressure_125/80']
Non Binary columns: ['Age', 'Sleep Duration', 'Physical Activity Level', 'Daily Steps', 'Heart Rate', 'Quality of Sleep', 'Stress Level']
```

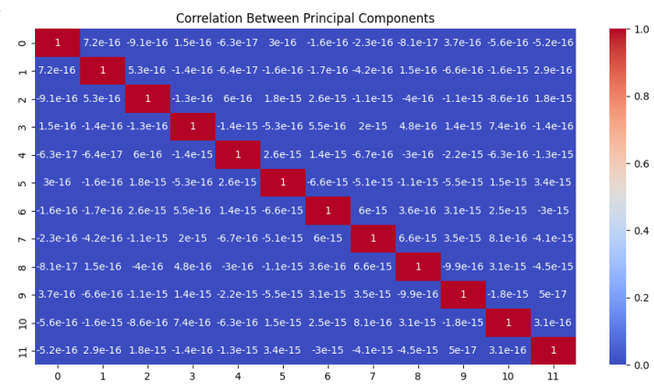
```
[ ] from sklearn.preprocessing import StandardScaler
    scaler=StandardScaler()

    X_selected_scaled = pd.DataFrame() # Initialize as an empty DataFrame

    # Apply StandardScaler only to Non-Binary Columns of X_selected
    X_selected_scaled[binary_columns]= X_selected[binary_columns]
    X_selected_scaled[non_binary_columns]=scaler.fit_transform(X_selected[non_binary_columns])
```

PCA (Dimensionality Reduction)

PCA (Principal Component Analysis) is a dimensionality reduction technique. It transforms your high-dimensional data (with possibly correlated features) into a new set of fewer, uncorrelated features called principal components, while keeping as much information (variance) as possible.

```
[ ] from sklearn.decomposition import PCA
    pca = PCA()
    X_pca = pca.fit_transform(X_selected_scaled)
```

```
[ ] #Checking correlation between features after PCA.
    X_pca_df = pd.DataFrame(X_pca)
    plt.figure(figsize=(12,6))
    sns.heatmap(X_pca_df.corr(), annot=True, cmap='coolwarm')
    plt.title("Correlation Between Principal Components")
    plt.show()
```



📌This heatmap confirms that our principal components are uncorrelated.

Split Data into Training and Testing Sets:

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)
    print("Training data feature shape(X_train):",X_train.shape)
    print("Training data target shape(y_train):",y_train.shape)
    print("Testing data feature shape(X_test):", X_test.shape)
    print("Testing data target shape(y_test):", y_test.shape)
```

```
Training data feature shape(X_train): (12000, 12)
Training data target shape(y_train): (12000,)
Testing data feature shape(X_test): (3000, 12)
Testing data target shape(y_test): (3000,)
```

Handling Imbalanced Data

```
[ ] # Calculate class Imbalance Ratio
    # target="y", we have already split the data.
    major_class=y.value_counts().max()
    minor_class=y.value_counts().min()
    imbalance_ratio=major_class/minor_class
    print("Imbalance Ratio:",imbalance_ratio)
```

```
Imbalance Ratio: 1.0
```

📌 We have got an imbalance ratio "1", which indicates the data is properly balanced. So we don't need to apply any techniques to balance it. We can move to model building process.

BUILD THE MACHINE LEARNING MODELS

📌 It is a clssification problems. The classiffication models are:

1. Support Vector Machine(SVC-Support Vector Classifier)
2. KNeighbors Classifier
3. Naive Bayes
4. Decision Tree Classifier
5. Random Forest Classifier
6. Gradient Boosting Classifier

```
[ ] #Importing required models
    from sklearn.svm import SVC
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.naive_bayes import GaussianNB
    from sklearn.metrics import accuracy_score
```

1.Support Vector Classifier-SVC

How it Works:

- SVC uses a set of labeled training examples to find a decision boundary that separates the data points into different classes.
- The decision boundary is represented as a linear function, and the goal is to find the boundary that maximizes the separation between the classes.

Why it's suitable:

- Works well when features are properly scaled.
- Performs well for high-dimensional data.
- SVC is commonly used in image recognition, text classification, and medical diagnosis.

```
svc=SVC()
svc.fit(X_train,y_train)
```

```
SVC()
```

```
#Creating a dictionary to store accuracy scores
Accuracy_scores={
    "Model":[],
    "Train Accuracy Score":[],
    "Test Accuracy Score":[]
}
```

```
svc_y_pred=svc.predict(X_test)
Train_accuracy=accuracy_score(y_train,svc.predict(X_train))
Test_accuracy=accuracy_score(y_test,svc_y_pred)
Accuracy_scores["Model"].append("Support Vector Classifier-SVC")
Accuracy_scores["Train Accuracy Score"].append(Train_accuracy)
Accuracy_scores["Test Accuracy Score"].append(Test_accuracy)
print("Train Accuracy-svc:",Train_accuracy)
print("Test Accuracy-svc:",Test_accuracy)
```

```
Train Accuracy-svc: 0.94875
Test Accuracy-svc: 0.9476666666666667
```

2.K-Nearest Neighbors Classifier

How it works:

- A non-parametric, instance-based algorithm that classifies a new point based on the majority class among its k closest neighbors.
- Uses distance metrics (e.g., Euclidean distance) to determine proximity.

Why it's suitable:

- It is Simple.
- Can handle non-linear decision boundaries.
- Sensitive to feature scaling and computationally expensive for large datasets.
- Works well if patterns exist in proximity.

```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier()
```

```
knn_y_pred=knn.predict(X_test)
Train_accuracy=accuracy_score(y_train,knn.predict(X_train))
Test_accuracy=accuracy_score(y_test,knn_y_pred)
Accuracy_scores["Model"].append("KNeighbors Classifier")
Accuracy_scores["Train Accuracy Score"].append(Train_accuracy)
Accuracy_scores["Test Accuracy Score"].append(Test_accuracy)
print("Train Accuracy-KNN:",Train_accuracy)
print("Test Accuracy-KNN:",Test_accuracy)
```

```
Train Accuracy-KNN: 0.9653333333333334
Test Accuracy-KNN: 0.958
```

3.Naive Bayes

How it works:

- A probabilistic model based on Bayes' theorem.
- Assumes that features are independent, which is often unrealistic but works well in many cases.

Why it's suitable:

- Good if Sleep Disorder has strong feature-dependent probabilities.
- Less effective if there are strong feature correlations.

```
gnb=GaussianNB()
gnb.fit(X_train,y_train)
```

```
GaussianNB()
```

```
gnb_y_pred=gnb.predict(X_test)
Train_accuracy=accuracy_score(y_train,gnb.predict(X_train))
Test_accuracy=accuracy_score(y_test,gnb_y_pred)
Accuracy_scores["Model"].append("Naive Bayes")
Accuracy_scores["Train Accuracy Score"].append(Train_accuracy)
Accuracy_scores["Test Accuracy Score"].append(Test_accuracy)
print("Train Accuracy-GNB:",Train_accuracy)
print("Test Accuracy-GNB:",Test_accuracy)
```

```
Train Accuracy-GNB: 0.8870833333333333
Test Accuracy-GNB: 0.8873333333333333
```

4.Decision Tree Classifier

How it Work :

- A Decision Tree is a supervised learning algorithm used for classification and regression tasks.
- The structure resembles a tree, with nodes representing decisions, branches representing choices, and leaves representing outcomes or predictions.

Why it's suitable:

- Easy to interpret and visualize, making it useful in medical applications.
- Handles non-linearly separable data.
- Can automatically capture interactions between features.
- Good for interpreting which factors influence sleep disorders.

```
dtcl=DecisionTreeClassifier()
dtcl.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```

```
dtcl_y_pred=dtcl.predict(X_test)
Train_accuracy=accuracy_score(y_train,dtcl.predict(X_train))
Test_accuracy=accuracy_score(y_test,dtcl_y_pred)
Accuracy_scores["Model"].append("Decision Tree Classifier")
Accuracy_scores["Train Accuracy Score"].append(Train_accuracy)
Accuracy_scores["Test Accuracy Score"].append(Test_accuracy)
print("Train Accuracy-DTCL:",Train_accuracy)
print("Test Accuracy-DTCL:",Test_accuracy)
```

```
Train Accuracy-DTCL: 0.9834166666666667
Test Accuracy-DTCL: 0.9626666666666667
```

5.Random Forest Classifier

How it Works:

- Random Forest is an ensemble learning method that builds multiple Decision Trees and averages their predictions to improve accuracy and reduce overfitting.
- Each tree is built on a random subset of data and features.

Why it's suitable:

- More accurate and robust than a single Decision Tree.
- Less prone to overfitting due to averaging across trees.
- Can handle large feature sets and complex relationships in data.

```
rfcl=RandomForestClassifier()
rfcl.fit(X_train,y_train)
```

```
RandomForestClassifier()
```

```
rfcl_y_pred=rfcl.predict(X_test)
Train_accuracy=accuracy_score(y_train,rfcl.predict(X_train))
Test_accuracy=accuracy_score(y_test,rfcl_y_pred)
Accuracy_scores["Model"].append("Random Forest Classifier")
Accuracy_scores["Train Accuracy Score"].append(Train_accuracy)
Accuracy_scores["Test Accuracy Score"].append(Test_accuracy)
print("Train Accuracy-RFCL:",Train_accuracy)
print("Test Accuracy-RFCL:",Test_accuracy)
```

```
Train Accuracy-RFCL: 0.9834166666666667
Test Accuracy-RFCL: 0.9676666666666667
```

6.Gradient Boosting Classifier

How it Works:

- Gradient Boosting Classifier is an ensemble learning method that builds multiple Trees.
- Builds trees sequentially, with each tree correcting the errors of the previous one.
- More powerful than Random Forest but computationally expensive.

Why it's suitable:

- More accurate and robust.
- If your dataset has complex relationships and requires high accuracy, Gradient Boosting is likely the best.
- Works well on structured data like this.

```
gbcl=GradientBoostingClassifier()
gbcl.fit(X_train,y_train)
```

```
GradientBoostingClassifier()
```

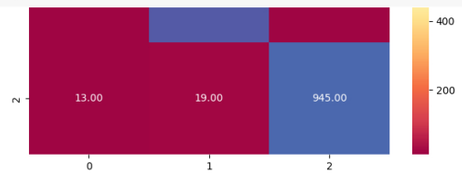
```
gbcl_y_pred=gbcl.predict(X_test)
Train_accuracy=accuracy_score(y_train,gbcl.predict(X_train))
Test_accuracy=accuracy_score(y_test,gbcl_y_pred)
Accuracy_scores["Model"].append("Gradient Boosting Classifier")
Accuracy_scores["Train Accuracy Score"].append(Train_accuracy)
Accuracy_scores["Test Accuracy Score"].append(Test_accuracy)
print("Train Accuracy-GBCL:",Train_accuracy)
print("Test Accuracy-GBCL:",Test_accuracy)
```

```
Train Accuracy-GBCL: 0.9641666666666666
Test Accuracy-GBCL: 0.9553333333333334
```

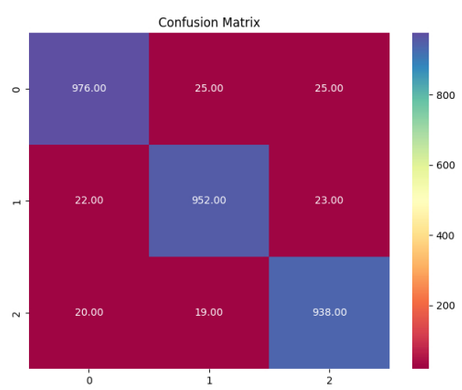
MODEL EVALUATION

```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,confusion_matrix,classification_report
```

```
# Define models
Models = {
    'Support Vector Classifier' : SVC(),
    'K-Nearest Neighbors' : KNeighborsClassifier(),
    'Naive Bayes' : GaussianNB(),
    'Decision Tree Classifier' : DecisionTreeClassifier(),
    'Random Forest Classifier' : RandomForestClassifier(),
    'Gradient Boosting Classifier' : GradientBoostingClassifier(),
}

# predictions
predictions= {
    "Support Vector Classifier":svc_y_pred,
    "K-Nearest Neighbors":knn_y_pred,
    "Naive Bayes":gnb_y_pred,
    "Decision Tree Classifier":dtcl_y_pred,
    "Random Forest Classifier":rfcl_y_pred,
    "Gradient Boosting Classifier":gbcl_y_pred
    }

# Analysing the metrics
for model_name,y_pred in predictions.items():

  print("Model:",model_name)
  print("Accuracy:", accuracy_score(y_test, y_pred))
  print("Classification Report :")
  print(classification_report(y_test, y_pred))

  Confusion_matrix = confusion_matrix(y_test, y_pred)
  plt.figure(figsize=(8, 6))
  sns.heatmap(Confusion_matrix, annot=True, fmt=".2f", cmap='Spectral')
  plt.title("Confusion Matrix")
  plt.show()
```




```
Model: Gradient Boosting Classifier
Accuracy: 0.9553333333333334
Classification Report :
              precision    recall  f1-score   support

     Healthy       0.96      0.95      0.95      1026
    Insomnia       0.96      0.95      0.96       997
 Sleep Apnea       0.95      0.96      0.96       977

    accuracy                           0.96      3000
   macro avg       0.96      0.96      0.96      3000
weighted avg       0.96      0.96      0.96      3000
```



Insights:

- **Random Forest Classifier** model shows the highest accuracy 96.7 % indicating strong predictive capabilities.
- **Decision Tree Classifier** & **K-Nearest Neighbors** models hold the second and third highest accuracies(96.2 % and 95.8%).
- **Gradient Boosting Classifier** is performing well with 95.5% accuracy.
- **Support Vector Classifier** is also performing well and it has an accuracy of 94.7%.
- **Gaussian Naive Bayes** is the least performing model compared to others. It shows an accuracy of 88.7% only.

Model Comparison

```
# predictions
predictions= {
    "Support Vector Classifier":svc_y_pred,
    "K-Nearest Neighbors":knn_y_pred,
    "Naive Bayes":gnb_y_pred,
    "Decision Tree Classifier":dtcl_y_pred,
    "Random Forest Classifier":rfcl_y_pred,
    "Gradient Boosting Classifier":gbcl_y_pred
    }
```

```
    }

# Store results
Results= {"Model":[],
          "Accuracy Score":[],
          "Precision Score":[],
          "Recall Score":[],
          "F1 Score":[]
          }

# Append metrics

for model_name,y_pred in predictions.items():

  Results["Model"].append(model_name)
  Results["Accuracy Score"].append(accuracy_score(y_test,y_pred))
  Results["Precision Score"].append(precision_score(y_test,y_pred,average="weighted"))
  Results["Recall Score"].append(recall_score(y_test,y_pred,average="weighted"))
  Results["F1 Score"].append(f1_score(y_test,y_pred,average="weighted"))

# Convert to DataFrame and sort results
Results=pd.DataFrame(Results)
Results.sort_values(by=["Accuracy Score"],ascending=False,inplace=True)
Results
```

	Model	Accuracy Score	Precision Score	Recall Score	F1 Score
4	Random Forest Classifier	0.967667	0.967679	0.967667	0.967665
3	Decision Tree Classifier	0.962667	0.962710	0.962667	0.962675
1	K-Nearest Neighbors	0.958000	0.958295	0.958000	0.957976
5	Gradient Boosting Classifier	0.955333	0.955355	0.955333	0.955332
0	Support Vector Classifier	0.947667	0.948005	0.947667	0.947664
2	Naive Bayes	0.887333	0.888247	0.887333	0.886825

Insights:

📌The table above presents the evaluation scores of different Machine Learning models. Based on the accuracy scores:

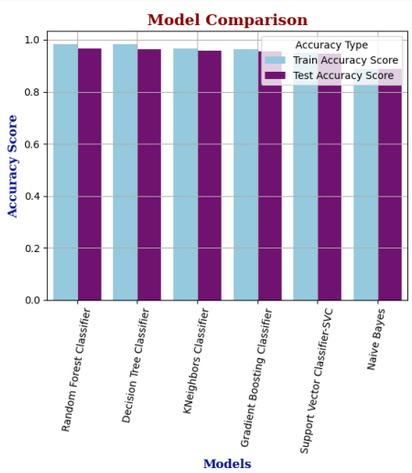
1. **Best Performing Model:**
   - 🥇 Random Forest Classifier achieves the highest accuracy of 96.7 %, making it the top-performing model.
2. **High-Performing Models:**
   - 🥈 Decision Tree Classifier ranks second with 96.2 % accuracy.
   - 🥉 K-Nearest Neighbors (KNN) follows closely in third place with 95.8% accuracy.
   - Gradient Boosting Classifier holds the fourth position with 95.5% accuracy.
   - Support Vector Classifier (SVC) secures the fifth position with 94.7% accuracy.
3. **Lowest Performing Model:**
   - Gaussian Naive Bayes performs the worst, with an accuracy of 88.7%, significantly lower than other models.
4. **Overall Observations:**
   - All models except Gaussian Naive Bayes achieve 90% or above accuracy, indicating strong performance across most models.

Analysing the models for over fitting

```
[ ] # Converting the "Accuracy_scores" that we already created to a DataFrame

    Accuracy_scores=pd.DataFrame(Accuracy_scores)
    Accuracy_scores.sort_values(by=["Test Accuracy Score"],ascending=False,inplace=True)
    Accuracy_scores
```

	Model	Train Accuracy Score	Test Accuracy Score
4	Random Forest Classifier	0.983417	0.967667
3	Decision Tree Classifier	0.983417	0.962667
1	KNeighbors Classifier	0.965333	0.958000
5	Gradient Boosting Classifier	0.964167	0.955333
0	Support Vector Classifier-SVC	0.948750	0.947667
2	Naive Bayes	0.887083	0.887333

```
[ ] # Creating a DataFrame to combine 'Train Accuracy Score' and 'Test Accuracy Score' into a single column
    Accuracy= pd.melt(Accuracy_scores, id_vars=['Model'], value_vars=['Train Accuracy Score', 'Test Accuracy Score'], var_name='Accuracy Type', value_name='Accuracy Score')

    # Now plot using the melted DataFrame
    sns.barplot(x="Model", y="Accuracy Score", hue="Accuracy Type", data=Accuracy, palette={"Train Accuracy Score": "skyblue", "Test Accuracy Score": "purple"})
    plt.xticks(rotation=80)
    plt.xlabel("Models",fontdict=axis_font)
    plt.ylabel("Accuracy Score",fontdict=axis_font)
    plt.title("Model Comparison",fontdict=title_font)
    plt.grid(True)
    plt.show()
```



Insights:

📌From the bar plot it is clear that, there is no significant difference between the Train accuracy and the Test accuracy of any of the models.It indicates there is no overfitting or underfitting.

Regularization

Regularization is a technique in machine learning used to prevent overfitting by adding a penalty to the loss function. This ensures that the model generalizes well to unseen data. Examples for common types of regularization: L1 (Lasso) and L2 (Ridge).

📌Since none of the models showing overfitting, we don't need to perform any regularization techniques.

HYPERPARAMETER TUNING

Hyperparameter tuning is the process of selecting the best set of hyperparameters to optimize a machine learning model's performance. Unlike model parameters (learned from data), hyperparameters are set manually before training.

- Improves Accuracy – Finds the best settings for better predictions.
- Prevents Overfitting/Underfitting – Balances model complexity.
- Enhances Generalization – Ensures the model works well on unseen data.
- GridSearchCV, RandomizedSearchCV, and Bayesian Optimization are some methods for hyperparameter tuning.

📌I am selecting first two best performing models and the worst performing model for hyperparameter tuning.

- Random Forest Classifier

- Decision Tree Classifier
- Gaussian Naïve Bayes

```
[ ] from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier
```

```
[ ] #Creating a dictionary to store new accuracy scores
    Performance={
        "Model":[],
        "Train Accuracy-Before Tuning":[],
        "Train Accuracy-After Tuning":[],
        "Test Accuracy-Before Tuning":[],
        "Test Accuracy-After Tuning":[],
    }

    #Appending the accuracy score before tuning
    # Random Forest Classifier
    Train_accuracy_before=accuracy_score(y_train,rfcl.predict(X_train))
    Test_accuracy_before=accuracy_score(y_test,rfcl_y_pred)
    Performance["Model"].append("Random Forest Classifier")
    Performance["Train Accuracy-Before Tuning"].append(Train_accuracy_before)
    Performance["Test Accuracy-Before Tuning"].append(Test_accuracy_before)

    #Decision Tree Classifier
    Train_accuracy_before=accuracy_score(y_train,dtcl.predict(X_train))
    Test_accuracy_before=accuracy_score(y_test,dtcl_y_pred)
    Performance["Model"].append("Decision Tree Classifier")
    Performance["Train Accuracy-Before Tuning"].append(Train_accuracy_before)
    Performance["Test Accuracy-Before Tuning"].append(Test_accuracy_before)

    #Gaussian Naive Bayes
    Train_accuracy_before=accuracy_score(y_train,gnb.predict(X_train))
    Test_accuracy_before=accuracy_score(y_test,gnb_y_pred)
    Performance["Model"].append(" Gaussian Naïve Bayes")
    Performance["Train Accuracy-Before Tuning"].append(Train_accuracy_before)
    Performance["Test Accuracy-Before Tuning"].append(Test_accuracy_before)
```

Random Forest Classifier

```
[ ]
    # Define hyperparameters and their possible values
    rf_params={
        'n_estimators': [10,20,50,100,200],
        'max_depth': [None, 5,10,15],
        'min_samples_split': [2, 5, 10],
    }

    #Define Random Forest model
    rf=RandomForestClassifier()
    random_search_rf=RandomizedSearchCV(rf,rf_params,n_iter=10,cv=5,random_state=42)
    random_search_rf.fit(X_train,y_train)

    print("Best Hyperparameters:",random_search_rf.best_params_)
```

```
Best Hyperparameters: {'n_estimators': 200, 'min_samples_split': 5, 'max_depth': 15}
```

```
[ ] #Training the New Random Forest model
    rf_new=RandomForestClassifier(n_estimators=200, min_samples_split=5, max_depth= 15)
    rf_new.fit(X_train,y_train)
```

```
RandomForestClassifier(max_depth=15, min_samples_split=5, n_estimators=200)
```

```
[ ] # Predictions with  the New Random Forest model
    rf_new_y_pred=rf_new.predict(X_test)
    Train_accuracy_after=accuracy_score(y_train,rf_new.predict(X_train))
    Test_accuracy_after=accuracy_score(y_test,rf_new_y_pred)
    Performance["Train Accuracy-After Tuning"].append(Train_accuracy_after)
    Performance["Test Accuracy-After Tuning"].append(Test_accuracy_after)
```

Decision Tree Classifier

```
[ ] # Define hyperparameters and their possible values
    dt_params={
        'criterion': ['gini', 'entropy'],
        'max_depth': [3, 5, 10, None],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }

    #Define Random Forest model
    dt=DecisionTreeClassifier()
    random_search_dt=RandomizedSearchCV(dt,dt_params,n_iter=10,cv=5,random_state=42)
    random_search_dt.fit(X_train,y_train)

    print("Best Hyperparameters:",random_search_dt.best_params_)
```

```
Best Hyperparameters: {'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': None, 'criterion': 'gini'}
```

```
[ ] #Traing the New Decision Tree model
    dt_new=DecisionTreeClassifier(min_samples_split=5, min_samples_leaf= 1, max_depth= None, criterion='gini')
    dt_new.fit(X_train,y_train)
```

```
DecisionTreeClassifier(min_samples_split=5)
```

```
[ ] # Predictions with the New Decision Tree model
    dt_new_y_pred=dt_new.predict(X_test)
    Train_accuracy_after=accuracy_score(y_train,dt_new.predict(X_train))
    Test_accuracy_after=accuracy_score(y_test,dt_new_y_pred)
    Performance["Train Accuracy-After Tuning"].append(Train_accuracy_after)
    Performance["Test Accuracy-After Tuning"].append(Test_accuracy_after)
```

Gaussian Naïve Bayes

```
[ ] # Define hyperparameters and their possible values
    gnb_params={
        'var_smoothing': np.logspace(-9, 0, 50)
    }

    #Define Gaussian Naïve Bayes
    gnb = GaussianNB()
    grid_search_gnb=GridSearchCV(gnb,gnb_params,cv=5, scoring='accuracy', n_jobs=-1)
    grid_search_gnb.fit(X_train,y_train)

    print("Best Hyperparameters:",grid_search_gnb.best_params_)
```

```
Best Hyperparameters: {'var_smoothing': np.float64(0.01456348477501245)}
```

```
[ ] gnb_new=GaussianNB(var_smoothing=np.float64(0.01456348477501245))
    gnb_new.fit(X_train,y_train)
```

```
GaussianNB(var_smoothing=np.float64(0.01456348477501245))
```

```
[ ] # Predictions with the New DGaussian Naïve Bayes model
    gnb_new_y_pred=gnb_new.predict(X_test)
    Train_accuracy_after=accuracy_score(y_train,gnb_new.predict(X_train))
    Test_accuracy_after=accuracy_score(y_test,gnb_new_y_pred)
    Performance["Train Accuracy-After Tuning"].append(Train_accuracy_after)
    Performance["Test Accuracy-After Tuning"].append(Test_accuracy_after)
```

```
[ ] Performance=pd.DataFrame(Performance)
    Performance
```

	Model	Train Accuracy-Before Tuning	Train Accuracy-After Tuning	Test Accuracy-Before Tuning	Test Accuracy-After Tuning
0	Random Forest Classifier	0.983417	0.982750	0.967667	0.970333
1	Decision Tree Classifier	0.983417	0.982333	0.962667	0.962667
2	Gaussian Naïve Bayes	0.887083	0.899167	0.887333	0.899333

📌The table shows the train and test accuracy scores before and after hyper parameter tuning.

- Overall, hyperparameter tuning did not result in significant accuracy improvements for most models.

Random Forest Classifier:

- The Test accuracy Random Forest Classifier is slightly increased from 96.7 to 97 % after hyperparameter tuning.

Decision Tree Classifier:

- The test accuracy of the Decision Tree classifier remained the same (96.2%) even after hyperparameter tuning.

Gaussian Naïve Bayes (GNB):

- Significant Improvement after hyperparameter tuning.
- Both Train accuracy and Test accuracy has increased significantly for Gaussian Naïve Bayes model.
- The Test accuracy increased from 88.7 to 89.9 %.

```
[ ] Best_Model=Performance.sort_values(by=["Test Accuracy-After Tuning"],ascending=False).head(1)
    Best_Model
```

	Model	Train Accuracy-Before Tuning	Train Accuracy-After Tuning	Test Accuracy-Before Tuning	Test Accuracy-After Tuning
0	Random Forest Classifier	0.983417	0.98275	0.967667	0.970333

📌Best performing Model:

- Best Model: Random Forest Classifier
- Best Hyperparameters: n_estimators= 200, min_samples_split= 5, max_depth= 15
- Best Accuracy: 97 %

SAVE THE MODEL

```
[ ] # Importing library
    import joblib

    # Saveing the trained model
    joblib.dump(rf_new, "The_Best_Model.joblib")
    print("Best Model:","Random Forest Classifier")
    print("Best Hyperparameters:","n_estimators= 200, min_samples_split= 5, max_depth= 15")
    print("Best Accuracy:",accuracy_score(y_test,rf_new_y_pred))
    print("Model saved successfully!")
```

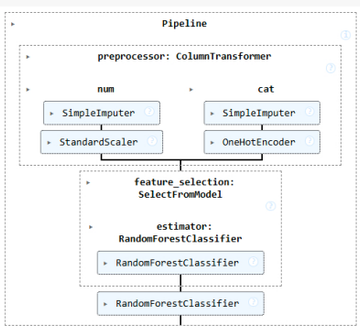
```
Best Model: Random Forest Classifier
Best Hyperparameters: n_estimators= 200, min_samples_split= 5, max_depth= 15
Best Accuracy: 0.9703333333333334
Model saved successfully!
```

Pipeline For ML Model

A machine learning pipeline was implemented to automate data preprocessing, feature selection, and model training, ensuring efficiency and reproducibility.

- Data Preprocessing: Missing values were handled using an imputer,fearure encoding is performed by OneHotEncoder and feature scaling was applied using StandardScaler
- Feature Selection: Feature Selection Using Random Forest was used to select the most relevant features.
- Model Training: Random Forest was chosen as the final model after hyperparameter tuning.

```
[ ] from sklearn.pipeline import Pipeline
    from sklearn.compose import ColumnTransformer
    from sklearn.impute import SimpleImputer
    from sklearn.preprocessing import OneHotEncoder, StandardScaler
    from sklearn.feature_selection import SelectFromModel
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import train_test_split
    from imblearn.pipeline import Pipeline as ImbPipeline
    import pandas as pd
    import joblib

    # Splitting the data to features and target. 'df' is the dataframe.
    X = df.drop(columns=["Sleep Disorder"])  # 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)

    # Identify numerical and categorical columns
    numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
    categorical_features = X.select_dtypes(include=['object', 'category']).columns

    # Define transformations
    num_transformer = Pipeline([
        ('imputer', SimpleImputer(strategy='mean')),  # Handle missing numerical values
        ('scaler', StandardScaler())  # Scale numerical features
    ])

    cat_transformer = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),  # Handle missing categorical values
        ('encoder', OneHotEncoder(handle_unknown="ignore", sparse_output=False))  # One-hot encoder for  categorical features
    ])

    # Combine transformations
    preprocessor = ColumnTransformer([
        ('num', num_transformer, numerical_features),
        ('cat', cat_transformer, categorical_features)
    ])

    # Define the pipeline
    pipeline = ImbPipeline([
        ('preprocessor', preprocessor),
        ('feature_selection', SelectFromModel(RandomForestClassifier(n_estimators=200, random_state=42), threshold=0.02)),
        ('classifier',joblib.load("The_Best_Model.joblib"))
    ])

    # Train the pipeline
    pipeline.fit(X_train, y_train)
```




```
[ ] # Saving the Pipeline
    joblib.dump(pipeline, 'Sleep_Health_Analysis_pipeline.joblib')
    print("Pipeline saved successfully!")
```

```
Pipeline saved successfully!
```

TEST WITH UNSEEN DATA

```
[ ] #Loading the unseen data
    unseen_data=pd.read_csv("/content/Unseen_Data.csv")
    unseen_data.head()
```

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate	Daily Steps
0	Female	44.0	Teacher	6.55	6	45	6	Overweight	135/90	68	6000
1	Female	32.0	Scientist	6.75	6	54	7	Normal	131/86	76	6600
2	Female	48.0	Teacher	7.40	8	38	4	Normal	135/90	65	5500
3	Male	35.0	Lawyer	6.65	7	45	6	Normal	130/85	70	6500
4	Female	44.0	Nurse	6.75	6	75	6	Obese	140/95	78	6650

```
[ ] unseen_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Gender                   20 non-null     object
 1   Age                      20 non-null     float64
 2   Occupation               20 non-null     object
 3   Sleep Duration           20 non-null     float64
 4   Quality of Sleep         20 non-null     int64
 5   Physical Activity Level  20 non-null     int64
 6   Stress Level             20 non-null     int64
```

```
 7   BMI Category             20 non-null     object
 8   Blood Pressure           20 non-null     object
 9   Heart Rate               20 non-null     int64
 10  Daily Steps              20 non-null     int64
dtypes: float64(2), int64(5), object(4)
memory usage: 1.8+ KB
```

```
[ ] #Testing with Unseen Data
    # Load the saved pipeline
    loaded_pipe = joblib.load('Sleep_Health_Analysis_pipeline.joblib')

    # Make predictions on the test data
    predictions = loaded_pipe.predict(unseen_data)

    # Print the predictions
    print(predictions)
```

```
['Insomnia' 'Sleep Apnea' 'Insomnia' 'Healthy' 'Sleep Apnea' 'Sleep Apnea'
 'Healthy' 'Healthy' 'Healthy' 'Healthy' 'Healthy' 'Healthy' 'Insomnia'
 'Sleep Apnea' 'Sleep Apnea' 'Sleep Apnea' 'Sleep Apnea' 'Insomnia'
 'Insomnia' 'Sleep Apnea']
```

Insights

Random Forest Classifier successfully predicted all three target classes. Out of 20 predictions;

- Healthy = 7
- Insomnia =5
- Sleep Apnea=8

Which indicates the Random Forest Classifier effectively identifies patterns and distinguishes between different sleep conditions.

CONCLUSION :

- The Random Forest Classifier achieved 97 % accuracy, the highest among all six models. This demonstrates its strong ability to predict Sleep Disorders based on various features.
- When tested with unseen data, the Random Forest Classifier successfully predicted all three target classes: "Healthy," "Insomnia," and "Sleep Apnea." This indicates that the model effectively identifies patterns and distinguishes between different sleep conditions.
- The dataset was well-balanced, with no missing values or duplicates. Various preprocessing techniques such as encoding, feature selection, and scaling contributed to all models achieving over 90% accuracy, except for the Gaussian Naïve Bayes model.
- Hyperparameter tuning had no significant impact on the performance of the Random Forest Classifier and Decision Tree Classifier. These models remained the top performers, each achieving over 96% accuracy.
- Gaussian Naïve Bayes showed a dramatic improvement after hyperparameter tuning, with accuracy increasing from 88.7% to 89.9%, highlighting the importance of tuning for this model.

FUTURE WORK:

- **Model Updates**: Periodically update the model with new data to ensure it adapts to changing trends and behaviors over time.
- **Imbalanced Data Handling:** Implement resampling techniques, like oversampling or undersampling, to address any class imbalances in the dataset, ensuring the model is not biased.
- **Feature Engineering:** Consider adding more features or engineering existing ones to capture more nuanced patterns that could enhance the model's predictive power.

Best Model:

The Random Forest Classifier model, with an accuracy of 97 %, has been saved as The_Best_Model.joblib for future use.

Colab paid products - Cancel contracts here