Optimizing Sleep Efficiency : Harnessing Machine Learning for Enhanced Restorative Rest

Project Description:

The project "Optimizing Sleep Efficacy: Harnessing Machine Learning for Enhanced Restorative Rest" aims to leverage machine learning algorithms to improve the quality and efficiency of sleep for individuals. By analyzing various factors such as sleep patterns, environmental conditions, and lifestyle habits, the system will provide personalized recommendations and interventions to optimize sleep quality and duration. The goal is to enhance restorative rest, promote overall well-being, and mitigate sleep-related issues such as insomnia, sleep apnea and disrupted sleep cycles.

Project Scenario:

Scenario 1: Personalized Sleep Schedule Adjustment

The system collects data on an individual's sleep patterns, daily routines, and lifestyle factors. Using machine learning algorithms, it identifies optimal sleep schedules tailored to the person's circadian rhythm and lifestyle preferences. For example, it may recommend adjusting bedtime or wake-up times to align with natural sleep cycles, resulting in more restorative sleep .

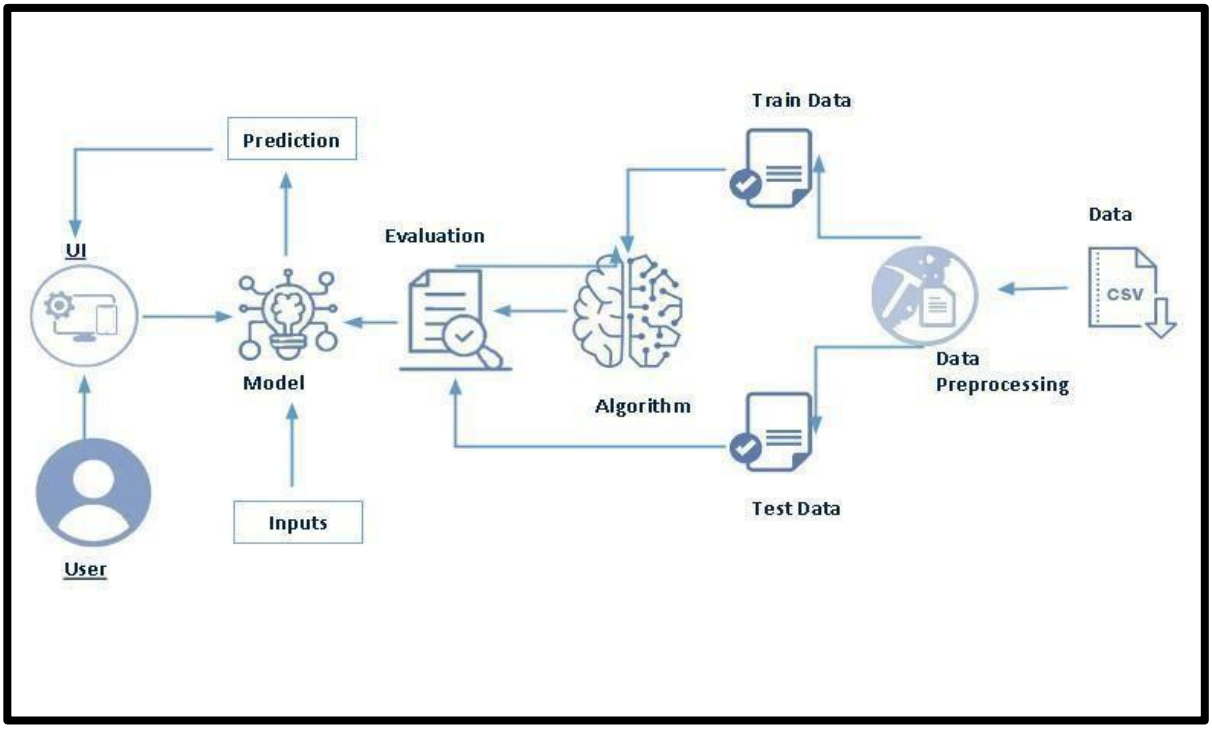
Scenario 2: Environmental Optimization for Better Sleep

By integrating data from smart home devices such as temperature sensors, light controls, and noise monitors, the system creates an ideal sleep environment. Machine learning algorithms analyze this data to suggest adjustments like adjusting room temperature, dimming lights, or playing soothing sounds to promote better sleep quality.

Scenario 3: Sleep Quality Monitoring and Intervention

The system continuously monitors sleep quality metrics such as REM sleep duration, deep sleep stages, and interruptions. If it detects patterns indicative of poor sleep quality, it provides personalized interventions such as relaxation techniques, sleep hygiene tips, or suggesting adjustments to bedtime routines to improve overall sleep efficacy.

Technical Architecture:



Project Flow:

- The user interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once the model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- DataCollection & Preparation
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis
- Model Building
  - Training the model in multiple algorithms
  - Testing the model
- Performance Testing & Hyperparameter Tuning
  - A testing model with multiple evaluation metrics
  - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
  - Save the best model
  - Integrate with Web Framework

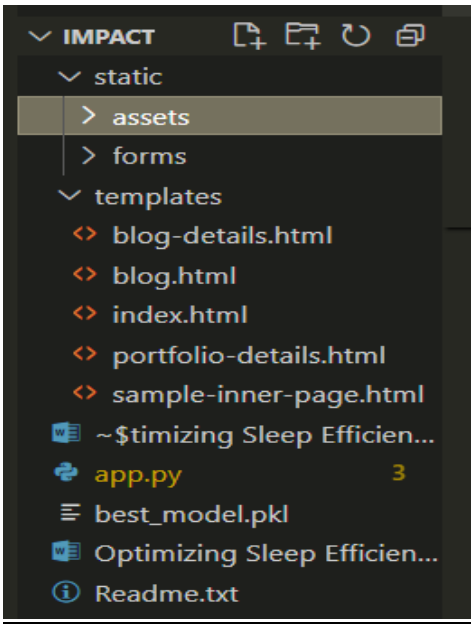
Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- ML Concepts
- Supervised learning: https://www.javatpoint.com/supervised-machine-learning
- Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
- Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
- Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
- Hyper parameter Tuning : https://www.geeksforgeeks.org/hyperparameter-tuning/
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



- We are building a Flask application that needs HTML pages stored in the templates folder and a Python script app.py for scripting.
- best_model.pkl is our saved model. Further, we will use this model for flask integration.
- App.ipynb contains a model training file.

Milestone 1: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project, we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https: https://www.kaggle.com/datasets/equilibriumm/sleep-efficiency.

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 2: Importing the libraries

Import the necessary libraries as shown in the image.

```
Out[30]: 'C:\\Users\\HP'
```

Importing Libraries

```
In [31]: import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as plt
```

```
90]: from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import MinMaxScaler
```

```
(135,)
```

```
In [93]: from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error, r2_score
```

```
R-squared Score: 0.7147227776503846
```

```
In [99]: from sklearn.tree import DecisionTreeRegressor
         from sklearn.metrics import accuracy_score
```

```
105]: from sklearn.ensemble import RandomForestRegressor
```

```
106]: RF = RandomForestRegressor()
```

```
In [118]: import pickle
```

```
In [119]: pickle.dump(RF,open("RF.pkl","wb"))
```

```
In [120]: from sklearn.model_selection import train_test_split, GridSearchCV
          from sklearn.ensemble import RandomForestRegressor
          from sklearn.metrics import mean_squared_error
          from sklearn.datasets import make_regression
```

Activity 3: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas, we have a function called read_csv() to read the dataset. As a parameter, we have to give the directory of the CSV file.

```
32]: df = pd.read_csv("E:/smart bridge/Sleep_Efficiency.csv")
```

Activity 4: Data Preparation

As we have understood how the data is, let's pre-process the collected data. The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 5: Handling missing values

Let's find the shape of our dataset first. To find the shape of our data, the data. shape method is used. To find the data type, the data.info() function is used.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452 entries, 0 to 451
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   ID                      452 non-null    int64  
 1   Age                     452 non-null    int64  
 2   Gender                  452 non-null    object 
 3   Bedtime                 452 non-null    object 
 4   Wakeup time             452 non-null    object 
 5   Sleep duration          452 non-null    float64
 6   Sleep efficiency        452 non-null    float64
 7   REM sleep percentage    452 non-null    int64  
 8   Deep sleep percentage   452 non-null    int64  
 9   Light sleep percentage  452 non-null    int64  
 10  Awakenings              432 non-null    float64
 11  Caffeine consumption    427 non-null    float64
 12  Alcohol consumption     438 non-null    float64
 13  Smoking status          452 non-null    object 
 14  Exercise frequency      446 non-null    float64
dtypes: float64(6), int64(5), object(4)
memory usage: 53.1+ KB
```

For checking the null values, data.isnull() function is used. To sum those null values we use .sum() function. From the below image, we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
df.isnull().sum()
```

```
ID                      0
Age                     0
Gender                  0
Bedtime                 0
Wakeup time             0
Sleep duration          0
Sleep efficiency        0
REM sleep percentage    0
Deep sleep percentage   0
Light sleep percentage  0
Awakenings             20
Caffeine consumption   25
Alcohol consumption    14
Smoking status          0
Exercise frequency      6
dtype: int64
```

As we can see our dataset has missing values.

As we have fewer missing values so, we now drop the missing values by using the below code.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
column_to_impute = df['Exercise frequency'].values.reshape(-1, 1)
imputer.fit(column_to_impute)
df['Exercise frequency'] = imputer.transform(column_to_impute)
```

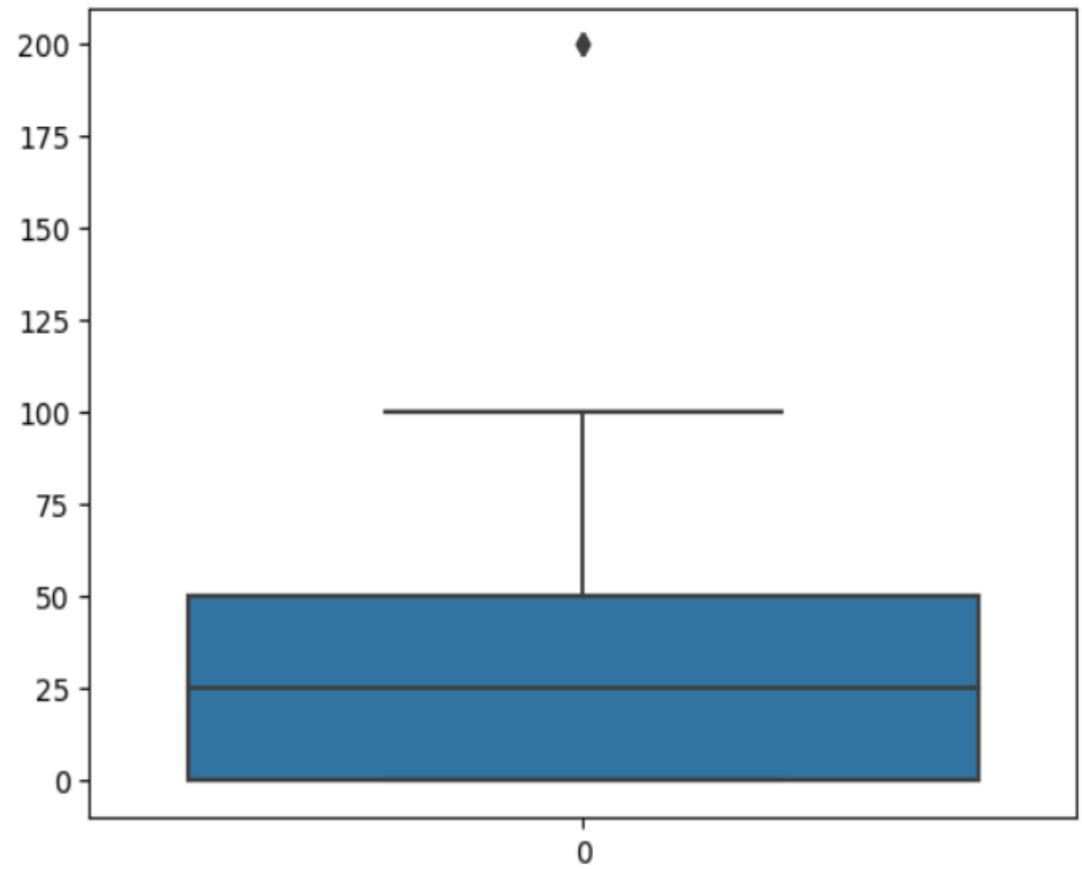
Activity 6: Handling Imbalance Data

With the help of a boxplot, outliers are visualized. Here we are going to find the upper bound and lower bound of every feature with some mathematical formula.

From the below diagram, we can visualize that every feature has outliers. Boxplot from Seaborn Library is used here.

```
sns.boxplot(df['Caffeine consumption'])
```

```
<Axes: >
```



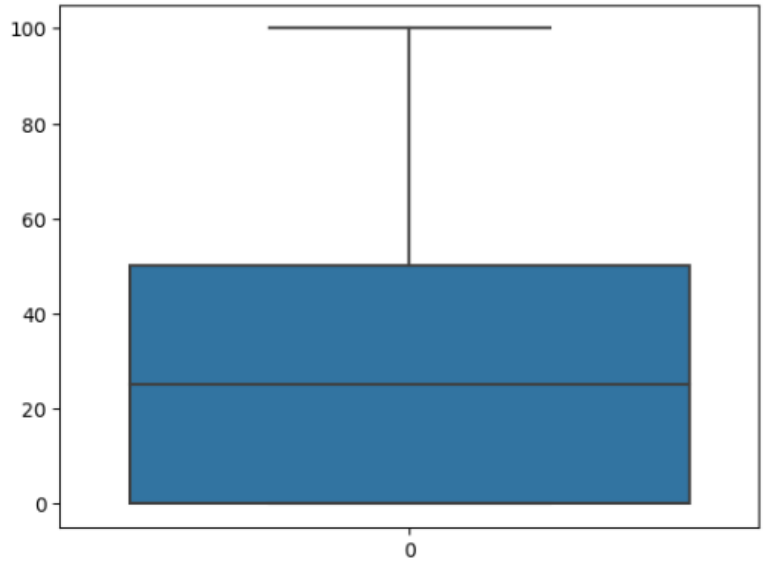
To find the upper bound we have to multiply IQR (Interquartile range) by 1.5 and add it to 3rd quantile. To find a lower bound instead of adding, subtract it with 1st quantile. Take the image attached below as your reference.

```
q1 = np.percentile(df['Caffeine consumption'], 25)
q3 = np.percentile(df['Caffeine consumption'], 75)
q1
q3
IQR = q3-q1
upper_limit = q3+1.5*IQR
lower_limit = q1 - 1.5*IQR
```

```
df = df[(df["Caffeine consumption"]<=upper_limit)  & (df["Caffeine consumption"]>=lower_limit)]
```

```
sns.boxplot(df["Caffeine consumption"])
```

```
<Axes: >
```



Milestone 2: Exploratory Data Analysis

Activity 1: Descriptive statistics

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas have a worthy function called describe. With this described function we can understand the unique, top, and frequent values of categorical features. And we can find mean, std, min, max, and percentile values of continuous features.

```
df.describe()
```

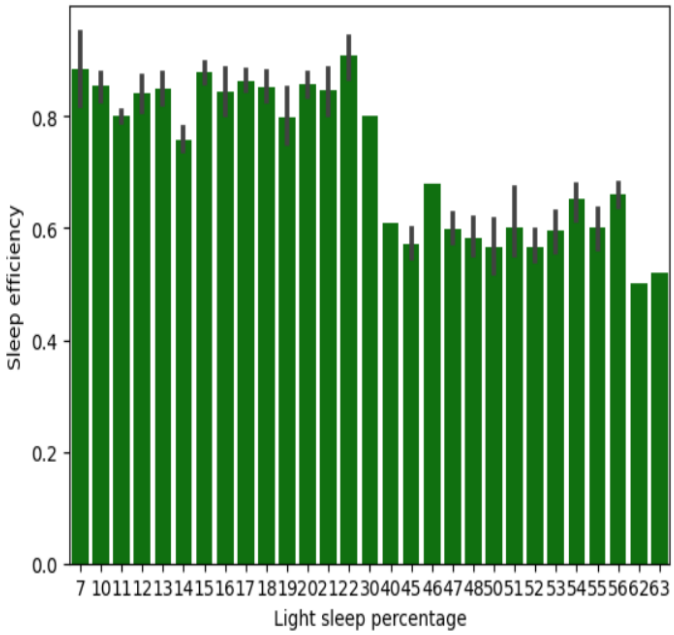
	ID	Age	Sleep duration	Sleep efficiency	REM sleep percentage	Deep sleep percentage	Light sleep percentage	Awakenings	Caffeine consumption	Alcohol consumption	Exercise frequency
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	432.000000	427.000000	438.000000	446.000000
mean	226.500000	40.285398	7.465708	0.788916	22.615044	52.823009	24.561947	1.641204	23.653396	1.173516	1.791480
std	130.625419	13.172250	0.866625	0.135237	3.525963	15.654235	15.313665	1.356762	30.202785	1.621377	1.428134
min	1.000000	9.000000	5.000000	0.500000	15.000000	18.000000	7.000000	0.000000	0.000000	0.000000	0.000000
25%	113.750000	29.000000	7.000000	0.697500	20.000000	48.250000	15.000000	1.000000	0.000000	0.000000	0.000000
50%	226.500000	40.000000	7.500000	0.820000	22.000000	58.000000	18.000000	1.000000	25.000000	0.000000	2.000000
75%	339.250000	52.000000	8.000000	0.900000	25.000000	63.000000	32.500000	3.000000	50.000000	2.000000	3.000000
max	452.000000	69.000000	10.000000	0.990000	30.000000	75.000000	63.000000	4.000000	200.000000	5.000000	5.000000

Activity 2: Visual analysis

Visual analysis is using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

```
sns.barplot( x = "Light sleep percentage" , y = "Sleep efficiency" ,data =df, color = "green")
```

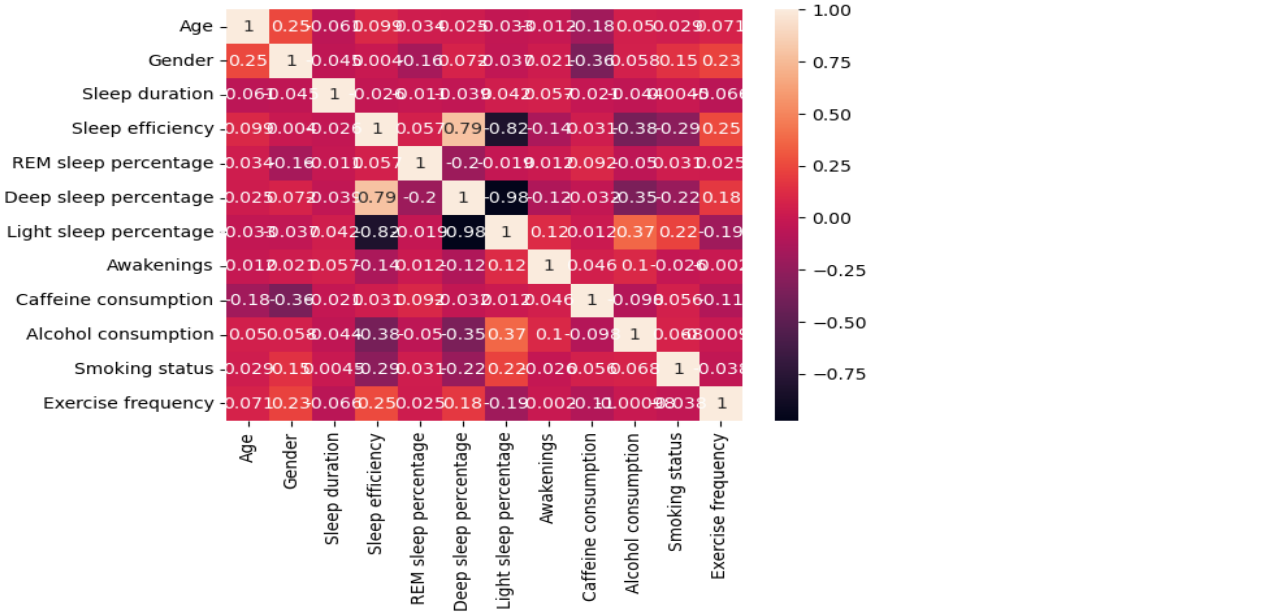
```
<Axes: xlabel='Light sleep percentage', ylabel='Sleep efficiency'>
```



The above code is the count plot where it describes the Quality count and gives the title Distribution of quality.

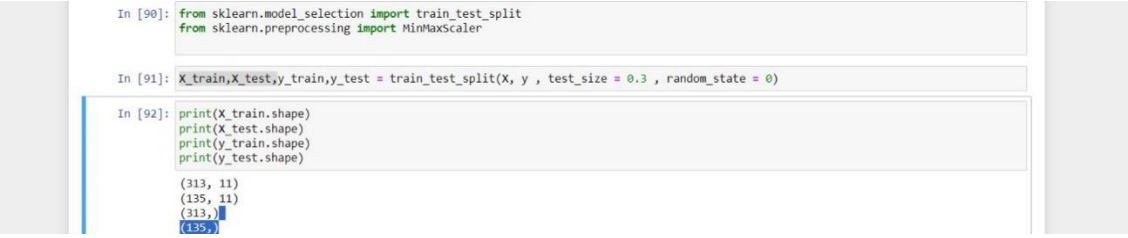
```
sns.heatmap(df.corr(), annot = True)
```

```
<Axes: >
```



Activity 3: Splitting data into train and test

Now let's split the Dataset into train and test sets. First, split the dataset into x and y and then split the data set, here x and y variables are created. On the x variable, data is passed by dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.



```
In [90]: from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import MinMaxScaler

In [91]: X_train,X_test,y_train,y_test = train_test_split(X, y , test_size = 0.3 , random_state = 0)

In [92]: print(X_train.shape)
         print(X_test.shape)
         print(y_train.shape)
         print(y_test.shape)

         (313, 11)
         (135, 11)
         (313,)
         (135,)
```

The code imports the MinMaxScaler module from sci-kit-learn for feature scaling. It then scales the features in the input data 'x' using standardization and splits the dataset into training and testing sets with a test size of 30% and a random state of 0.

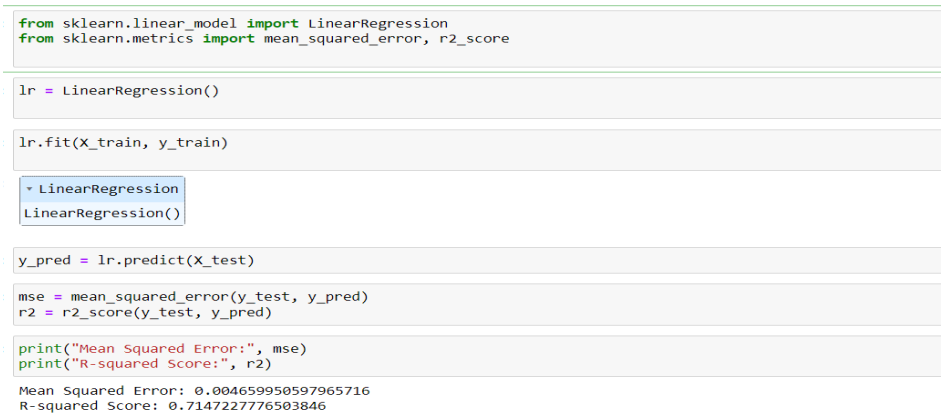
Milestone 3: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project, we are applying four regression algorithms. The best model is saved based on its performance.

Model 1 :

Linear Regression



```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lr = LinearRegression()

lr.fit(X_train, y_train)

LinearRegression
LinearRegression()

y_pred = lr.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared Score:", r2)

Mean Squared Error: 0.004659950597965716
R-squared Score: 0.7147227776503846
```

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. It aims to find the best-fitting line through the data points, minimizing the sum of the squared differences between the observed and predicted values. This technique is commonly employed for prediction and forecasting tasks in various fields such as economics, finance, and machine learning.

Model 2:

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score
```

```
dt = DecisionTreeRegressor(random_state=42)
```

```
dt.fit(X_train, y_train)
```

```
DecisionTreeRegressor(random_state=42)
```

```
y_pred = dt.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

```
Mean Squared Error: 0.00424962962962963
R-squared Score: 0.739842191184473
```

Decision trees are a machine learning algorithm used for both classification and regression tasks. They work by recursively partitioning the data into subsets based on the values of input features. At each step, the algorithm selects the feature that best splits the data, aiming to minimize impurity or maximize information gain. Decision trees are interpretable, versatile, and can handle both numerical and categorical data, making them popular in various domains such as finance, healthcare, and marketing.

Model 3:
Random Forest

```
from sklearn.ensemble import RandomForestRegressor
```

```
RF = RandomForestRegressor()
```

```
RF.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor()
```

```
RF.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor()
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

```
Mean Squared Error: 0.00424962962962963
R-squared Score: 0.739842191184473
```

Random Forest is an ensemble learning method that constructs multiple decision trees during training and combines their predictions to improve accuracy and reduce overfitting. Each tree is trained on a random subset of the training data and a random subset of features. The final prediction is determined by averaging or taking a majority vote of the predictions made by individual trees. Random Forest is widely used for classification and regression tasks and is known for its robustness and high performance on various types of datasets.

Model 4:

Hyper Paramter tuning:

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.datasets import make_regression
```

```
X, y = make_regression(n_samples=1000, n_features=10, noise=0.1, random_state=42)
```

```
model = RandomForestRegressor()
```

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
```

```
grid_search.fit(X_train, y_train)
```

GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor

```
print("Best hyperparameters:", grid_search.best_params_)
```

```
Best hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
```

```
best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error on Test Set:", mse)
```

```
Mean Squared Error on Test Set: 0.002291220873435892
```

```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error on Test Set:", mse)
```

```
Mean Squared Error on Test Set: 0.002291220873435892
```

```
print("Mean Squared Error on Test Set:", mse)
```

```
Mean Squared Error on Test Set: 0.002291220873435892
```

```
Prediction = best_model.predict([[18,1,7.5,22,23,55,1.0,50.0,0.0,0,1.0]])
```

```
D:\machinelearning\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but RandomF
ressor was fitted with feature names
  warnings.warn(
```

```
Prediction
```

```
array([0.66041037])
```

```
Prediction = best_model.predict([[65,0,6.0,18,70,12,0.0,0.0,0.0,0.0,1,3.0]])
```

```
D:\machinelearning\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but RandomF
ressor was fitted with feature names
  warnings.warn(
```

```
Prediction
```

```
array([0.88837728])
```

```
r2 = r2_score(y_test, y_pred)
```

```
r2
```

```
0.8597338935634657
```

Hyperparameter tuning involves optimizing the settings of a machine learning model that are not learned during training, such as learning rates or regularization parameters. It aims to improve the model's performance by selecting the best combination of hyperparameters through techniques like grid search, random search, or Bayesian optimization. Hyperparameter tuning is crucial for maximizing model accuracy and generalization on unseen data, but it can be computationally intensive and requires careful validation to avoid overfitting.

Prediction :

```
Prediction = best_model.predict([[65,0,6.0,18,70,12,0.0,0.0,0.0,0.0,1,3.0]])
```

```
D:\machinelearning\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but RandomForest
ressor was fitted with feature names
  warnings.warn(
```

```
Prediction
```

```
array([0.88837728])
```

Milestone 5: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
import pickle
```

```
pickle.dump(best_model,open("best_model.pkl","wb"))
```

This code exports the trained machine learning model 'model1' using pickle serialization and saves it as a file named "best_model. pkl". This allows for the model to be easily stored, shared, and loaded for future use without needing to retrain it.

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 3: Building Html Pages:

For this project create two HTML files namely

- index.html
- portfolio-details.html
- blog.html

and save them in the templates folder.

Activity 4: Build Python code:

Import the libraries

```
import numpy as np
import pickle
import pandas as pd
import os
from flask import Flask, request, render_template
app = Flask(__name__)
```

Load the saved model. Importing the Flask module in the project is mandatory. An object of the Flask class is our WSGI application. The Flask constructor takes the name of the current module (__name__) as argument.

Here we will be using a declared constructor to route to the HTML page that we have created earlier.

In the below example, the '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.

Whenever you enter the values from the html page the values can be retrieved using POST Method.

- Retrieves the value from UI:

```
app = Flask(__name__)
model = pickle.load(open('best_model.pkl', 'rb'))

@app.route('/') # rendering the html template
def index():
    return render_template('index.html')
@app.route('/predict') # rendering the html template
def innerpage():
    return render_template('portfolio-details.html')
```

- Here we are routing our app to predict the () function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model. predict() function. This function returns the prediction. This prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

```python
@app.route('/submit', methods=["POST"])
def submit():


    Age=int(request.form["Age"])
    Gender = int(request.form["Gender"])
    Sleep_Duration = int(request.form["Sleep Duration"])
    Rem_sleep_percentage = int(request.form["REM sleep percentage"])
    Deep_sleep_percentage = int(request.form["Deep sleep percentage"])
    Light_sleep_percentage = int(request.form["Light sleep percentage"])
    Awakenings = float(request.form["Awakenings"])
    Caffeine_consumption = float(request.form["Caffeine consumption"])
    Alcohol_consumption = float(request.form["Alcohol consumption"])
    Smoking_status = float(request.form["Smoking status"])
    Exercise_frequency = float(request.form["Exercise frequency"])
```

Main Function:

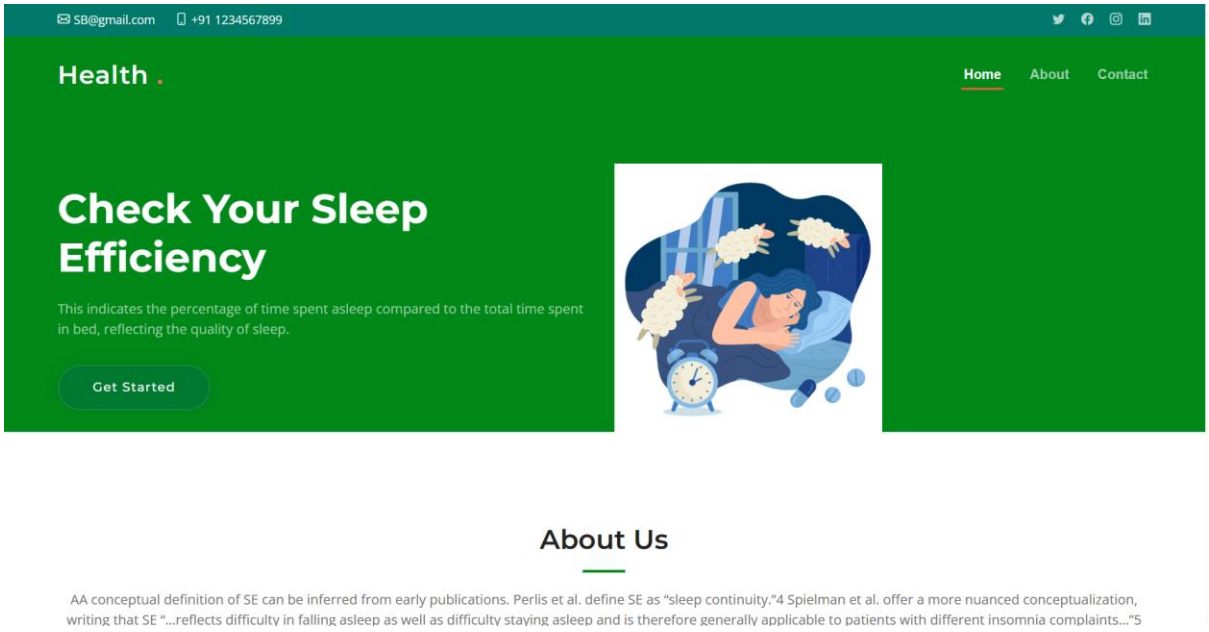
```python
if __name__=="__main__":
    app.run( debug=True, port = 5000)
```

Activity 5: Run the web application

- Open the anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type the “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

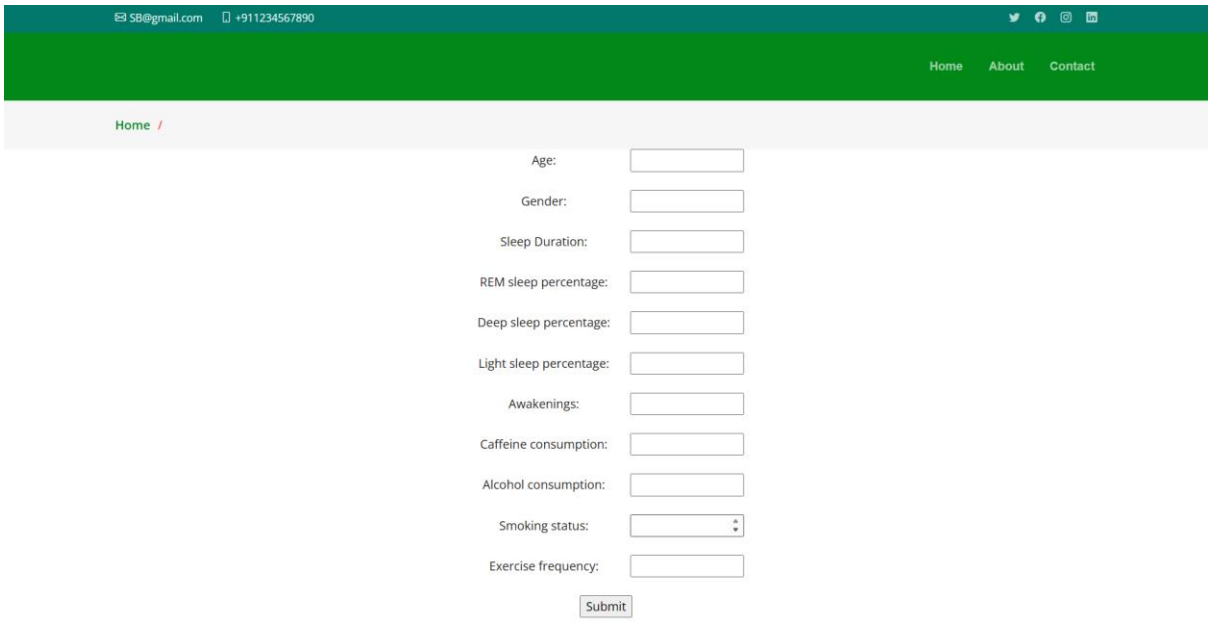
```
 * Debug mode: on
WARNING: This is a development server. D
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
```

Now, Go to the web browser and write the localhost URL (http://127.0.0.1:5000) to get the below result



The above UI page is the index page. In the index page, it contains home, about, and contact.

On the page we have Get Started the button click on it. So, that it will be redirected to the inner page.



In this page gives the respected values for the respected fields .

Output :

In this final page the output that is sleep efficiency will be shown.

Home / submit

Your sleep efficiency is 88.72%