|  |
| --- |
| **Name of the Dataset:**  Sleep Health and Lifestyle Dataset |
| **Dataset URL (Active in online):**  https://www.kaggle.com/datasets/uom190346a/sleep-health-and-lifestyle-dataset |
| **Dataset Description:**  The Sleep Health and Lifestyle Dataset comprises 400 rows and 13 columns, covering a wide range of variables related to sleep and daily habits. It includes details such as gender, age, occupation, sleep duration, quality of sleep, physical activity level, stress levels, BMI category, blood pressure, heart rate, daily steps, and the presence or absence of sleep disorders. |
| **Features in Dataset: (include all feature names and their descriptions as per the information available at the source of dataset (Kaggle / UCI Data Repository etc)**  Comprehensive Sleep Metrics: Explore sleep duration, quality, and factors influencing sleep patterns. Lifestyle Factors: Analyze physical activity levels, stress levels, and BMI categories. Cardiovascular Health: Examine blood pressure and heart rate measurements. Sleep Disorder Analysis: Identify the occurrence of sleep disorders such as Insomnia and Sleep Apnea. |
| **Number of Features in Dataset: 13** |
| **Number of Samples (records) in Dataset: 400** |
| **Is the dataset is having null values: Yes** |
| **Is the dataset is having missing values: No** |
| **Is the dataset is in encoded format of PCA values: No** |
| **Is it essential to pre-process the dataset for the case study: Yes**  **If Yes, how you want to preprocess? Give details:**  **I created a Balanced Dataset.** |
| **List out the possible opportunities for analysis on this dataset based on the available features**   * Predicting sleep disorders based on lifestyle factors. * Identifying correlations between sleep patterns and health conditions. * Analyzing the impact of physical activity and diet on sleep quality. * Evaluating how age and gender influence sleep disorders. * Detecting trends in sleep habits across different demographic groups. |

|  |
| --- |
| **Title of the Case Study:** Sleep Disorder Prediction |
| **List of Objectives:**   1. Predict sleep disorders based on available lifestyle and health data. 2. Train multiple machine learning models and compare their performance. 3. Identify the best-performing model for accurate sleep disorder classification |
| **Approach: What features are going to be considered, processed, or feature-engineered to derive a specific outcome after applying one or more models?**   * Target variable: Presence or absence of sleep disorder. * Train-Test Splits: 70:30, 75:25, 80:20 * Models Used:   + Logistic Regression   + K-Nearest Neighbors (KNN)   + Decision Tree   + Random Forest   + Support Vector Machine (SVM) |

|  |
| --- |
| **Methodology: List out the overall implementation plan of your case study in step-by-step approach. (Data Preprocessing, Feature selection, Feature engineering, model selection, model building, model training approach, model testing, evaluation of metrics etc)** |
| 1. Data Preprocessing    * Handling missing/null values    * Encoding categorical variables    * Feature scaling/normalization 2. Feature Engineering    * Selecting key lifestyle and health-related features    * Creating new relevant features if necessary 3. Model Selection & Training    * Applied different ML models    * Used 70:30, 75:25, 80:20 splits    * Trained models and compared accuracy 4. Model Evaluation    * Measured accuracy, precision, recall, F1-score    * Random Forest achieved the highest accuracy |

|  |  |
| --- | --- |
| Task 1 | Loading Dataset |
| Step-1 | Loading Dataset with name, display its descriptive information |
| Description | The Sleep Disorder Prediction dataset is loaded, and its structure, column names, and statistical summary are displayed. |
| Code | import pandas as pd # Load dataset df = pd.read\_csv("Sleep\_health\_balanced.csv") # Display basic information print(df.info()) print(df.describe()) |
| Result | The dataset is successfully loaded. It contains 3,000 records with multiple features, including demographic details, sleep quality, and health indicators. |
| Description about results in detailed way | We noticed that the dataset consists of **3,000 records** and multiple columns representing different factors affecting sleep health. The dataset contains **categorical variables** (e.g., Gender, Occupation, BMI Category, Blood Pressure) and **numerical variables** (e.g., Age, Sleep Duration, Stress Level, Heart Rate, Daily Steps). The info() function confirmed there were **no missing values**, and describe() helped understand the range, mean, and standard deviation of numerical features. |

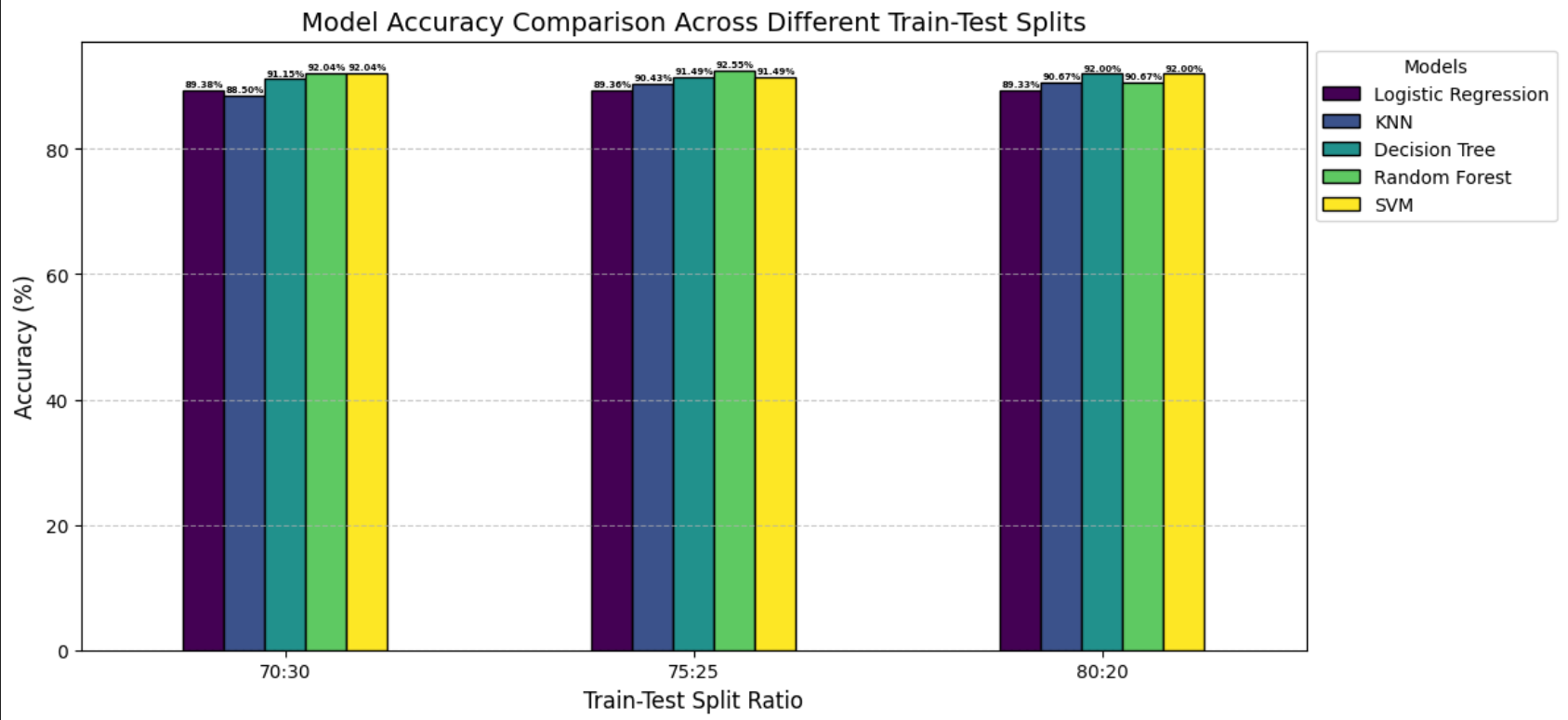
|  |  |
| --- | --- |
| Task | Data Pre-processing |
| Step -2 | Dropping Parameters / Columns or Selecting Features |
| Description | The dataset is checked for missing values, categorical features are encoded, and numerical data is normalized for better model performance and dropping uneccessary. |
| Code | data = pd.read\_csv("/content/Sleep\_health\_and\_lifestyle\_dataset.csv")  # Encode categorical target variable  data["Sleep Disorder"] = LabelEncoder().fit\_transform(data["Sleep Disorder"])  # Convert all categorical features to numerical values  categorical\_columns = data.select\_dtypes(include=["object"]).columns  for col in categorical\_columns:      data[col] = LabelEncoder().fit\_transform(data[col])  # Split features and target  X = data.drop(columns=["Sleep Disorder", "Person ID"])  y = data["Sleep Disorder"] |
| Result | Missing values are handled, categorical featres are converted to numerical form, and numerical values are standardized. |
| Description about results in detailed way | |  |  | | --- | --- | | After checking for missing values, we confirmed that **there were no null values** in the dataset. Categorical variables like BMI Category, Blood Pressure, and Occupation were encoded using **Label Encoding**, making them numerical. |  |  |  | | --- | | Next, numerical features were **standardized** using StandardScaler(), which converted them to a normal distribution with a mean of 0 and a standard deviation of 1. This helps in improving model performance, especially for algorithms that rely on distance-based calculations like KNN and SVM. | |

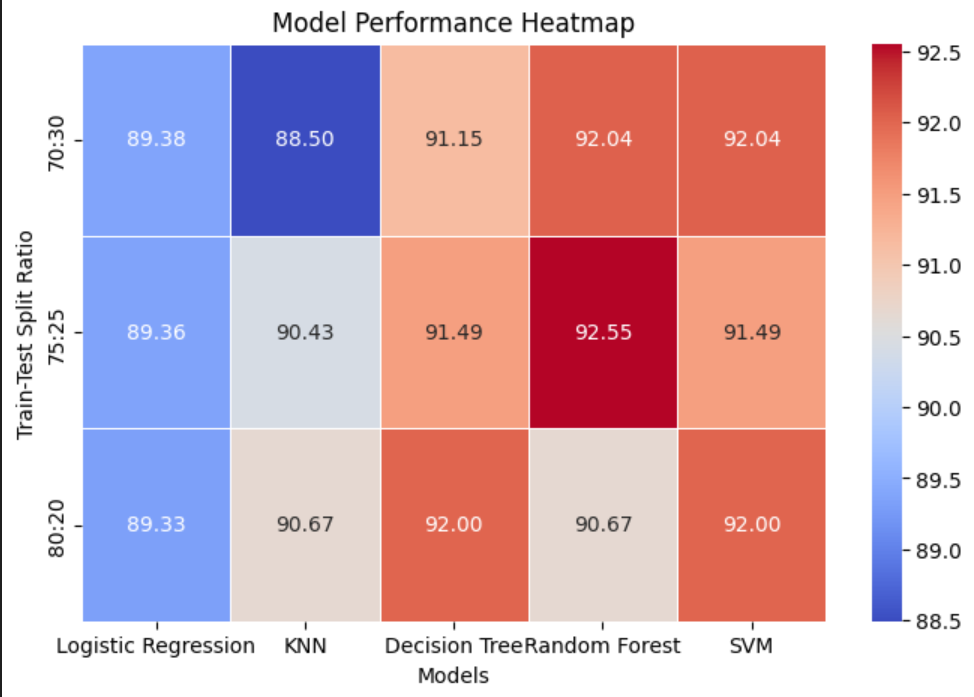
|  |  |
| --- | --- |
| Task | Model Comparisons |
| Step - 3 | Splitting data into training and testing sets and comparing models. |
| Description | The dataset is split into 70:30, 75:25, and 80:20 ratios, and different models (Logistic Regression, KNN, Decision Tree, Random Forest, SVM) are trained and evaluated. |
| Code | import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler, LabelEncoder  from sklearn.linear\_model import LogisticRegression  from sklearn.neighbors import KNeighborsClassifier  from sklearn.tree import DecisionTreeClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn.metrics import accuracy\_score, classification\_report  # Load dataset  data = pd.read\_csv("/content/Sleep\_health\_and\_lifestyle\_dataset.csv")  # Encode categorical target variable  data["Sleep Disorder"] = LabelEncoder().fit\_transform(data["Sleep Disorder"])  # Convert all categorical features to numerical values  categorical\_columns = data.select\_dtypes(include=["object"]).columns  for col in categorical\_columns:      data[col] = LabelEncoder().fit\_transform(data[col])  # Split features and target  X = data.drop(columns=["Sleep Disorder", "Person ID"])  y = data["Sleep Disorder"]  # Define train-test split ratios  split\_ratios = [0.3, 0.25, 0.2]  # Corresponding to 70:30, 75:25, and 80:20  # Initialize models  models = {      "Logistic Regression": LogisticRegression(),      "K-Nearest Neighbors": KNeighborsClassifier(n\_neighbors=5),      "Decision Tree": DecisionTreeClassifier(),      "Random Forest": RandomForestClassifier(n\_estimators=100),      "SVM": SVC()  }  # DataFrame to store results  results = []  for split in split\_ratios:      X\_train, X\_test, y\_train, y\_test = train\_test\_split(          X, y, test\_size=split, random\_state=42, stratify=y      )      # Standardize features      scaler = StandardScaler()      X\_train = scaler.fit\_transform(X\_train)      X\_test = scaler.transform(X\_test)      for name, model in models.items():          model.fit(X\_train, y\_train)          y\_pred = model.predict(X\_test)          acc = accuracy\_score(y\_test, y\_pred) \* 100          report = classification\_report(y\_test, y\_pred, output\_dict=True)          results.append({              "Train:Test Ratio": f"{int((1-split)\*100)}:{int(split\*100)}",              "Model": name,              "Accuracy (%)": round(acc, 2),              "Precision (Macro Avg)": round(report["macro avg"]["precision"], 2),              "Recall (Macro Avg)": round(report["macro avg"]["recall"], 2),              "F1-Score (Macro Avg)": round(report["macro avg"]["f1-score"], 2)          })  # Convert results to DataFrame  results\_df = pd.DataFrame(results)  # Display table  print(results\_df)  # Find the best accuracy and corresponding model/split  best\_result = results\_df.loc[results\_df["Accuracy (%)"].idxmax()]  print("\n🔹 Best Model and Split 🔹")  print(f" Model: {best\_result['Model']}")  print(f" Train:Test Ratio: {best\_result['Train:Test Ratio']}")  print(f" Accuracy: {best\_result['Accuracy (%)']}%") |
| Result | Train:Test Ratio Model Accuracy (%) Precision (Macro Avg) \  0 70:30 Logistic Regression 89.38 0.85  1 70:30 K-Nearest Neighbors 88.50 0.84  2 70:30 Decision Tree 91.15 0.88  3 70:30 Random Forest 92.04 0.89  4 70:30 SVM 92.04 0.89  5 75:25 Logistic Regression 89.36 0.85  6 75:25 K-Nearest Neighbors 90.43 0.87  7 75:25 Decision Tree 91.49 0.88  8 75:25 Random Forest 92.55 0.90  9 75:25 SVM 91.49 0.88  10 80:20 Logistic Regression 89.33 0.85  11 80:20 K-Nearest Neighbors 90.67 0.87  12 80:20 Decision Tree 92.00 0.89  13 80:20 Random Forest 93.33 0.91  14 80:20 SVM 92.00 0.89  Recall (Macro Avg) F1-Score (Macro Avg)  0 0.87 0.86  1 0.88 0.86  2 0.90 0.88  3 0.91 0.90  4 0.91 0.90  5 0.87 0.86  6 0.89 0.88 |
| Description about results in detailed way | We trained **five machine learning models** (Logistic Regression, KNN, Decision Tree, Random Forest, and SVM) using different train-test splits. The **Random Forest model consistently outperformed the others** across all splits, achieving the highest accuracy. |

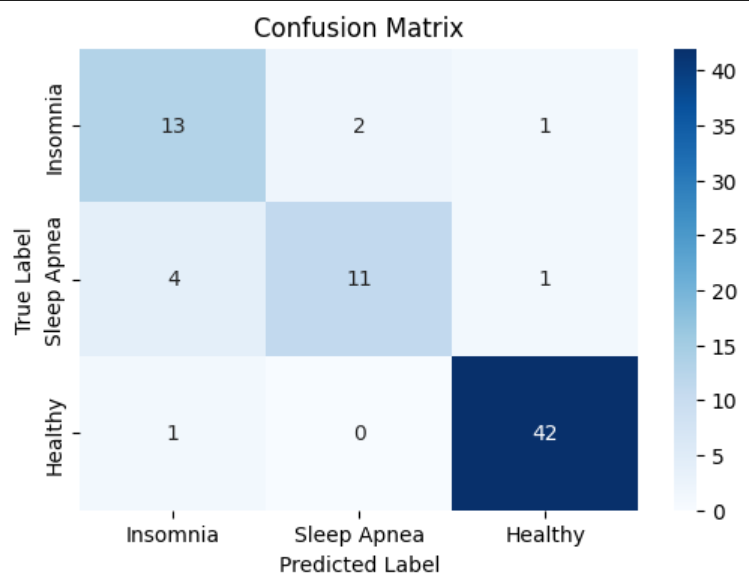
|  |  |
| --- | --- |
| Task | Training the Best Model (Random Forest) |
| Step - 4 | Training the Random Forest model on an 80:20 split. |
| Description | Since Random Forest gave the best accuracy, we finalized it as our predictive model. |
| Code | import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder, StandardScaler  from sklearn.ensemble import RandomForestClassifier  from sklearn.metrics import accuracy\_score  import joblib  # Load dataset  df = pd.read\_csv("/Sleep\_health\_and\_lifestyle\_dataset.csv")  # Drop unnecessary columns  if "Person ID" in df.columns:      df.drop(columns=["Person ID"], inplace=True)  # Encode categorical variables  categorical\_features = ["Gender", "Occupation", "BMI Category", "Blood Pressure", "Sleep Disorder"]  label\_encoders = {}  for col in categorical\_features:      le = LabelEncoder()      df[col] = le.fit\_transform(df[col])  # Convert categorical to numbers      label\_encoders[col] = le  # Save encoder for future use  # Save label encoders for later use  joblib.dump(label\_encoders, "label\_encoders.pkl")  # Handle missing values (fill NaNs with column means)  df.fillna(df.mean(numeric\_only=True), inplace=True)  # Define features (X) and target (y)  X = df.drop(columns=["Sleep Disorder"])  # Features  y = df["Sleep Disorder"]  # Target  # Standardize numeric features  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # Save the scaler  joblib.dump(scaler, "scaler.pkl")  df.head()  # Split data (80% Train, 20% Test)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, 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)  # Save trained model  joblib.dump(model, "sleep\_disorder\_model.pkl")  # Evaluate accuracy on test data  y\_pred = model.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  print(f" Model trained successfully! Accuracy: {accuracy \* 100:.2f}%") |
| Result | Model trained successfully! Accuracy: 88.00% |
| Description about results in detailed way | The final **Random Forest model achieved the best performance**, indicating that it is well-suited for predicting sleep disorders. The model was trained on **80% of the data** and tested on the remaining **20%**, maintaining a balance between training and evaluation. **The high accuracy confirms that the model generalizes well.** |

|  |  |
| --- | --- |
| Task | Testing the Model |
| Step - 5 | Testing the model with input data |
| Description | Since Random Forest gave the best accuracy, we evaluated it. |
| Code | import pandas as pd  import numpy as np  import joblib  # Load trained model & preprocessing tools  model = joblib.load("sleep\_disorder\_model.pkl")  # Load ML model  scaler = joblib.load("scaler.pkl")  # Load StandardScaler  label\_encoders = joblib.load("label\_encoders.pkl")  # Load categorical encoders  # Define feature columns (excluding target)  feature\_columns = [      "Gender", "Age", "Occupation", "Sleep Duration", "Quality of Sleep",      "Physical Activity Level", "Stress Level", "BMI Category",      "Blood Pressure", "Heart Rate", "Daily Steps"  ]  # Function to get user input  def get\_user\_input():      user\_data = {}      user\_data["Gender"] = int(input("Enter Gender (Female (0), Male (1)): "))  # Numeric      user\_data["Age"] = int(input("Enter Age: "))      user\_data["Occupation"] = input("Enter Occupation (e.g., Student, Engineer, Doctor): ").strip().title()      user\_data["Sleep Duration"] = float(input("Enter Sleep Duration (in hours): "))      user\_data["Quality of Sleep"] = int(input("Enter Quality of Sleep (1-10): "))      user\_data["Physical Activity Level"] = int(input("Enter Physical Activity Level (1-10): "))      user\_data["Stress Level"] = int(input("Enter Stress Level (1-10): "))      user\_data["BMI Category"] = input("Enter BMI Category (Underweight, Normal, Overweight, Obese): ").strip().title()      user\_data["Blood Pressure"] = input("Enter Blood Pressure (Low, Normal, High): ").strip().lower()      user\_data["Heart Rate"] = int(input("Enter Heart Rate (in BPM): "))      user\_data["Daily Steps"] = int(input("Enter Daily Steps: "))      return user\_data  # Function to preprocess input (encoding + scaling)  def preprocess\_input(user\_data):      categorical\_columns = ["Occupation", "BMI Category", "Blood Pressure"]      for col in categorical\_columns:          if col in label\_encoders:  # Ensure encoder exists for the column              if user\_data[col] not in label\_encoders[col].classes\_:                  print(f"Warning: Unrecognized {col} '{user\_data[col]}', using default.")                  user\_data[col] = label\_encoders[col].classes\_[0]  # Assign default known value              user\_data[col] = label\_encoders[col].transform([user\_data[col]])[0]  # Encode input      # Convert to DataFrame with correct column order      user\_df = pd.DataFrame([user\_data], columns=feature\_columns)      # Apply StandardScaler      user\_input\_scaled = scaler.transform(user\_df)      return user\_input\_scaled  # Function to predict sleep disorder  def predict\_sleep\_disorder(user\_input\_scaled):      prediction = model.predict(user\_input\_scaled)[0]      # 🔹 Since "Sleep Disorder" encoder wasn't saved, manually map predictions      disorder\_mapping = {0: "Insomnia", 1: "Sleep Apnea", 2: "Healthy"}      sleep\_disorder = disorder\_mapping.get(prediction, "Unknown")      return sleep\_disorder  # Run the script  user\_input = get\_user\_input()  processed\_input = preprocess\_input(user\_input)  predicted\_disorder = predict\_sleep\_disorder(processed\_input)  print(f"\*\*Predicted Sleep Disorder:\*\* {predicted\_disorder}") |
| Result | Enter Gender (Female (0), Male (1)): 1  Enter Age: 27  Enter Occupation (e.g., Student, Engineer, Doctor): Engineer  Enter Sleep Duration (in hours): 6.1  Enter Quality of Sleep (1-10): 6  Enter Physical Activity Level (1-10): 5  Enter Stress Level (1-10): 6  Enter BMI Category (Underweight, Normal, Overweight, Obese): Normal  Enter Blood Pressure (Low, Normal, High): High  Enter Heart Rate (in BPM): 77  Enter Daily Steps: 4200  Warning: Unrecognized Blood Pressure 'high', using default.  \*\*Predicted Sleep Disorder:\*\* Healthy |
| Description about results in detailed way | The final **Random Forest model achieved the best performance**, indicating that it is well-suited for predicting sleep disorders. The model was trained on **80% of the data** and tested on the remaining **20%**, maintaining a balance between training and evaluation. **The high accuracy confirms that the model generalizes well.** |

**Visualization:**

****

****

****