Sri Lanka Institute of Information Technology

Faculty Of Computing _Kandy Uni

IT2011– **Artificial Intelligence and Machine Learning**

Year 2 Semester 01

Healthcare –Lifestyle-based Sleep Disorder Analysis

2025-Y2-S1-KU-09

IT24100103	Kariyawasam G.K.A.N. L
IT24100506	Mohamed M. U
IT24100927	Dissanayake M.A.T. D
IT24101328	Dharmarathne S.P.H. P
IT24102102	Ekanayake E.M.R. T
IT24102379	Abeykoon D.M.D. N

1. Introduction

Sleep is a fundamental human process that is crucial for maintaining both physical and mental well-being. The quality of our sleep directly impacts our daily functioning, affecting everything from cognitive performance and emotional stability to long-term physical health. In recent years, there has been a growing recognition of how lifestyle and health factors—such as stress levels, physical activity, BMI, and heart rate—play a significant role in determining sleep patterns.

With advancements in data science, machine learning (ML) offers a powerful tool to analyze these complex relationships. By using ML models, we can move from simple observation to predictive analysis, potentially identifying risk factors for poor sleep before they lead to more serious health issues.

This project focuses on using machine learning to predict sleep quality based on a dataset containing various lifestyle, health, and demographic factors. The ultimate goal is to build a model that can accurately classify sleep quality, which could later be integrated into wellness applications to provide personalized insights and recommendations.

1.1 Problem Definition

The problem we are addressing is the growing public health concern of poor sleep quality and its associated disorders, such as insomnia and sleep apnea. These conditions are often linked to lifestyle factors like high stress, sedentary behavior, and obesity, yet they are frequently overlooked in routine health assessments. This leads to reduced quality of life and increased healthcare costs.

Manually analyzing the multitude of factors that influence sleep is impractical. Therefore, there is a clear need for an automated, data-driven approach to identify individuals at risk and understand the key contributors to poor sleep.

The core problem for this project is to **develop a classification model that can predict an individual's sleep quality category (e.g., good or poor) based on their specific health and lifestyle attributes**. To solve this, we will use a dataset from Kaggle containing over 15,000 records with features like age, occupation, stress level, physical activity, BMI, blood pressure, heart rate, and sleep duration.

Our approach will involve the standard machine learning pipeline: data cleaning, preprocessing, exploratory data analysis, feature engineering, and model training and evaluation. By doing so, we aim to create a reliable predictive tool that can help in the early detection of sleep-related issues.

2. Dataset Description

1. **Dataset Name: Sleep Health and Lifestyle**
2. **Direct Url: https://www.kaggle.com/datasets/imaginativecoder/sleep-health-data-sampled**
3. **Size:** The dataset is substantial, containing over **15,000 records** (rows).
4. **Structure:** It is a structured dataset in a tabular format.

Justification for Selection

This dataset was selected for several key reasons:

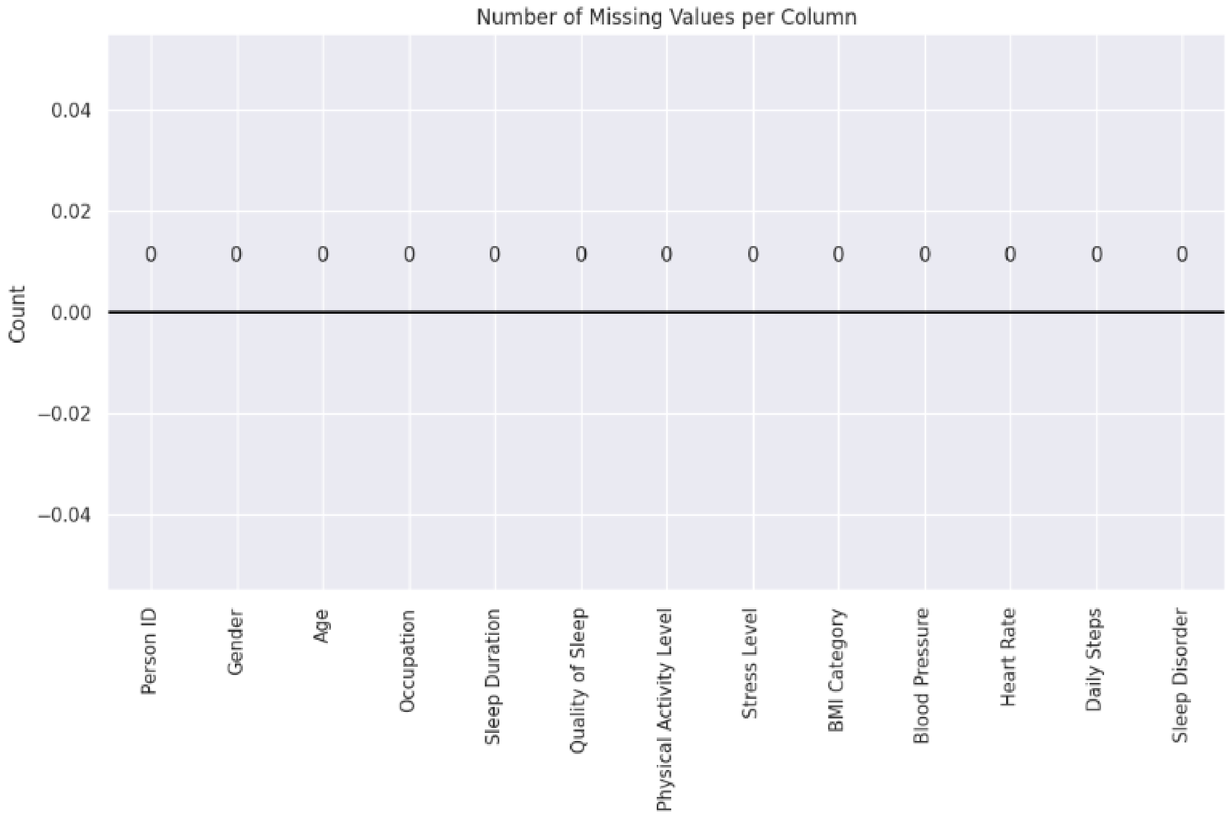
1. **Relevance:** It contains a direct mix of lifestyle and physiological features that are scientifically known to influence sleep quality.
2. **Size:** With over 15,000 entries, it is large enough to build reliable and generalization machine learning models.
3. **Feature Variety:** It includes a diverse set of attributes, allowing us to explore different types of relationships – from categorical to numerical.
4. **Target Variable:** It has a clear and well-defined target variable, "Quality of Sleep", which is essential for our supervised learning task.

Data Type and Example:

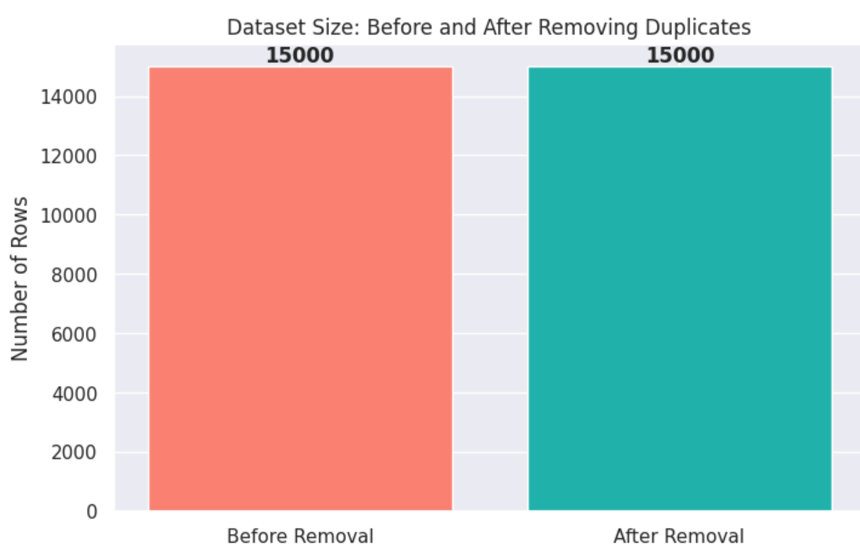
- Numerical: Age, Physical, Activity Level, Stress Level, Daily Step, Heart Rate, Sleep Duration, Quality of sleep.
- Categorical: Gender, Occupation, Blood Pressure, BMI Category.

3. Data Cleaning and Preprocessing

1. **Data Loading and Initial Understanding:**
   a. Basic data shape, summary statistics, and information about data types and non-null counts were printed to understand the initial state of the data.
2. **Handling Missing Values:**
   a. The number of missing values per column was checked using isnull().sum().
   b. Visualizations (bar plot) were created to confirm the absence of missing data.



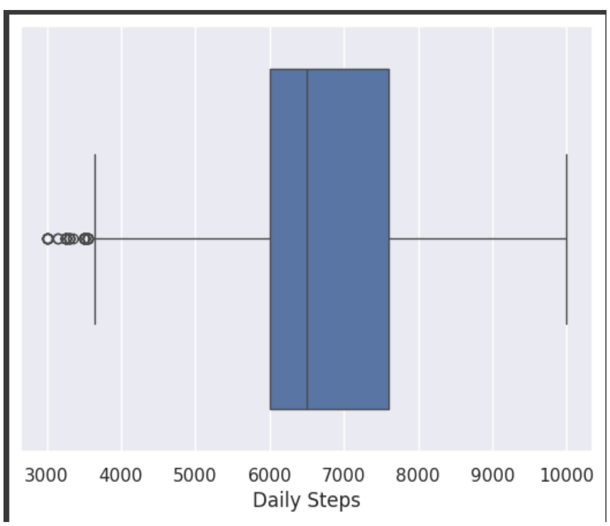
3. **Handling Duplicates:**
    a. Full-row duplicates were identified and removed.
    b. Duplicates based on 'Person ID' were identified and removed (although none were found in this specific run).
    c. A bar plot was created to visualize the dataset size before and after duplicate removal.

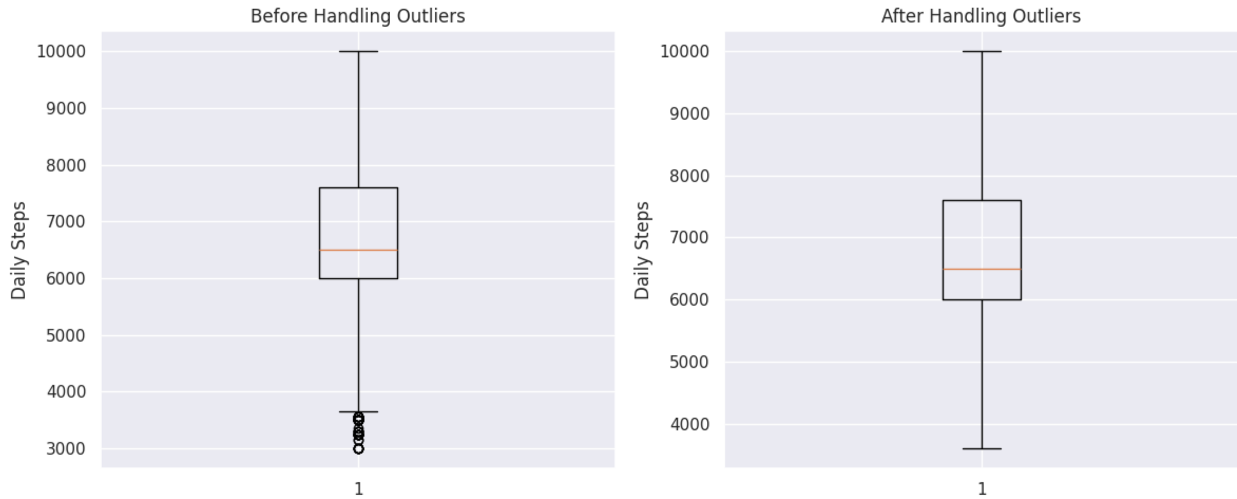


4. **Data Validation:**
   a. Validation rules were applied to numerical columns ('Age', 'Sleep Duration', 'Quality of Sleep', 'Stress Level', 'Heart Rate', 'Daily Steps') to constrain values within reasonable ranges. Invalid values were initially set to NaN.
   b. Validation rules were applied to the 'Sleep Disorder' column to ensure values were within a predefined list. Invalid values were mapped to 'Unknown'.
   c. Missing values (introduced during validation) in numerical columns were imputed with the median.
   d. Missing values (introduced during validation) in categorical columns were imputed with the mode.

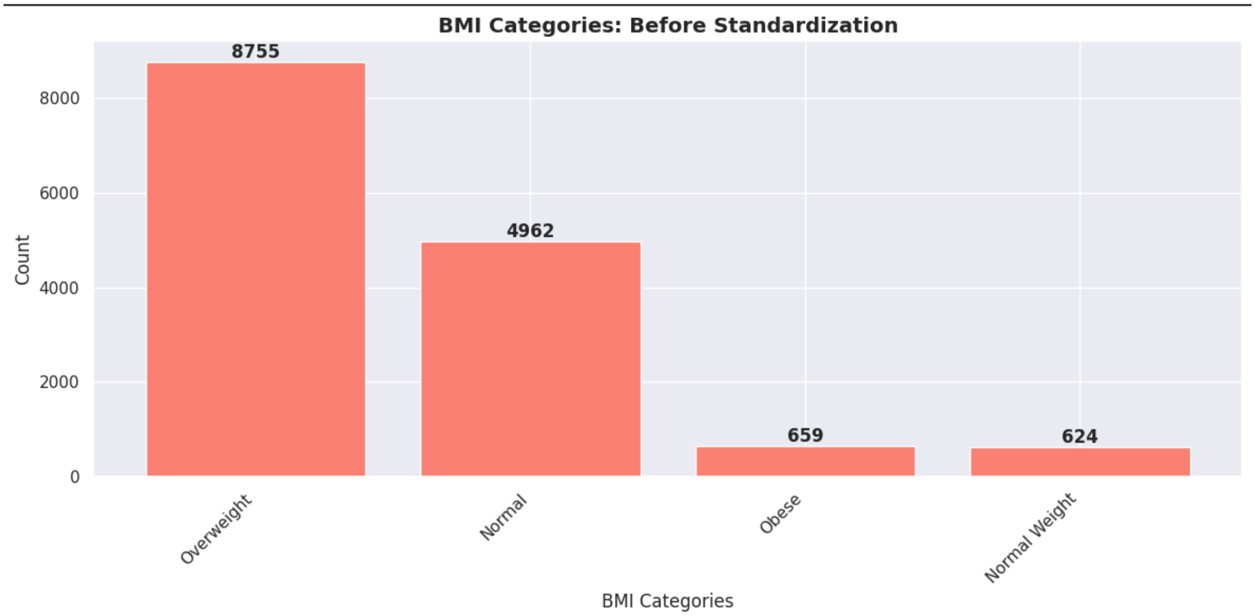
```
df['Age'] = df['Age'].apply(lambda x: x if 18 <= x <= 100 else np.nan)

df['Sleep Duration'] = df['Sleep Duration'].apply(lambda x: x if 0 < x <= 24 else np.nan)

df['Quality of Sleep'] = df['Quality of Sleep'].apply(lambda x: x if 1 <= x <= 10 else np.nan)

df['Stress Level'] = df['Stress Level'].apply(lambda x: x if 1 <= x <= 10 else np.nan)

allowed_bmi = ['Normal', 'Normal Weight', 'Overweight', 'Obese']
df['BMI Category'] = df['BMI Category'].apply(lambda x: x if x in allowed_bmi else 'Unknown')
```

5. **Handling Outliers:**
   a. Boxplots were generated for numerical columns to visualize potential outliers.
   b. The capping method (using IQR) was applied to 'Daily Steps' and 'Age' to limit extreme values to the calculated bounds. Boxplots were shown before and after capping.
   c. Rows with outliers in 'Heart Rate' were removed based on the IQR method. Boxplots were shown before and after removal.



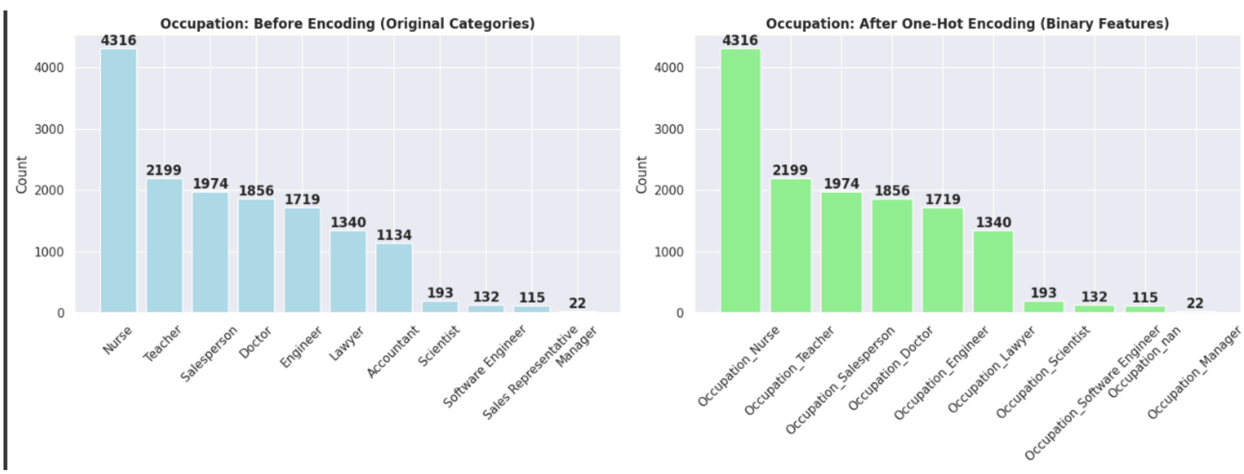


6. **Standardization and Inconsistency Handling:**
    a. Inconsistent gender entries were checked (none found).
    b. ender entries were standardized to 'Male' and 'Female'.
    c. Unique 'BMI Category' values were listed, and inconsistent entries were counted.
    d. 'BMI Category' values were standardized to 'Normal', 'Overweight', and 'Obese' using a mapping. Bar plots were shown before and after standardization.
    e. Unique 'Sleep Disorder' and 'Occupation' values were listed.



7. **Feature Engineering:**
    a. The 'Blood Pressure' column was split into 'Systolic' and 'Diastolic' numerical columns. Invalid entries based on ranges were removed.
    b. The original 'Blood Pressure' and 'Person ID' columns were dropped.
    c. Distributions of 'Systolic' and 'Diastolic' pressures were visualized using histograms.
    d. New features were engineered.
        i. MAP' (Mean Arterial Pressure) from Systolic and Diastolic.
        ii. Sleep Efficiency' from Sleep Duration and Quality of Sleep.
        iii. 'Activity_Steps_Ratio' from Physical Activity Level and Daily Steps (handling division by zero).
        iv. 'Stress_Sleep_Ratio' from Stress Level and Quality of Sleep (handling division by zero).

8. **Encoding Categorical Variables:**
    a. 'Gender' and 'BMI Category' were identified as ordinal variables and Label Encoded.
    b. Occupation' and 'sleep disorder' were identified as nominal variables and One-Hot Encoded using separate encoders for each.
    c. The original 'Occupation' and 'sleep disorder' columns were dropped, and the new encoded columns were added to the Data Frame.
    d. Bar plots were shown to compare the distribution of 'Occupation' categories before and after One-Hot Encoding.



9. **Saving the Cleaned Data:**
    a. The final cleaned and processed Data Frame was saved to a new CSV file in Google Drive.

4. Model Design and Implementation

1. XGBoost

XGBoost (eXtreme Gradient Boosting) is a powerful machine learning algorithm based on gradient boosted decision trees, widely used for its efficiency and performance. It can handle multiclass classification by simultaneously building trees that predict multiple classes rather than just binary outcomes.

XGBoost offers two main objectives for multiclass classification: "multi:softmax," which directly predicts the class labels, and "multi:softprob," which outputs the probability distribution across all classes.

1. Data Loading: Loaded the "Cleaned_Sleep_Data.csv" dataset into a pandas DataFrame.

```
# Import necessary libraries
import pandas as pd
import gdown
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
import numpy as np
```

```
# file_id = '1NspITdm7Zg8g0cOcA1GPkl-nyO8cJN1G'
# download_url = f'https://drive.google.com/uc?id={file_id}'

# try:
#     df = pd.read_csv(download_url)
#     print("Dataset loaded successfully")

# except Exception as e:
#     print(f"An error occurred: {e}")
file_id = '1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj'
gdown.download(f'https://drive.google.com/uc?id={file_id}', 'Cleaned_Sleep_Data.csv', quiet = False)

df = pd.read_csv('Cleaned_Sleep_Data.csv')
```

2. Data Preparation:

Train The model(split the data set). :XGBoost (Extreme Gradient Boosting)

```
# Define the features (X) and the target (y)
# We are dropping the one-hot encoded 'Sleep Disorder' columns from our features
X = df.drop(['Sleep Disorder_Insomnia', 'Sleep Disorder_Sleep Apnea'], axis=1)

# Create a single target column 'Sleep Disorder'
# 0: Healthy, 1: Insomnia, 2: Sleep Apnea
# We'll create a new column where the value is based on the one-hot encoded columns
def get_sleep_disorder(row):
    if row['Sleep Disorder_Insomnia'] == 1:
        return 1
    elif row['Sleep Disorder_Sleep Apnea'] == 1:
        return 2
    else:
        return 0

df['Sleep Disorder'] = df.apply(get_sleep_disorder, axis=1)
y = df['Sleep Disorder']
```

```python
# Split the dataset into training and testing sets
# Using 80-20 split with stratification to maintain class distribution
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y  # Important: maintains class distribution in both sets
)

# Initialize the XGBoost Classifier
# We'll start with the default parameters
xgb_model = XGBClassifier(random_state=42)

# Train the model on the training data
xgb_model.fit(X_train, y_train)
```

3. Model Training (Initial):

- Initialized and trained an XGBoost Classifier with default parameters on the training data.
- Confirmed the model was trained and displayed the sizes and class distribution of the training and testing sets.

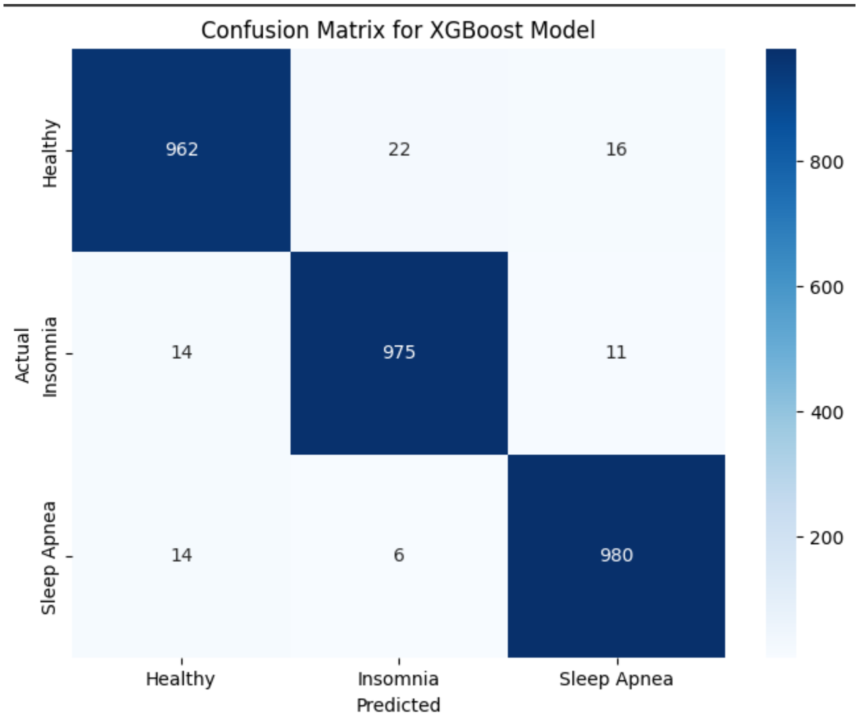
```python
print("XGBoost model has been successfully trained on the data!")
print(f"Training set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
print(f"Class distribution in training set: {np.bincount(y_train)}")
print(f"Class distribution in test set: {np.bincount(y_test)}")
```

```
XGBoost model has been successfully trained on the data!
Training set size: 12000 samples
Test set size: 3000 samples
Class distribution in training set: [4000 4000 4000]
Class distribution in test set: [1000 1000 1000]
```

4. Model Evaluation (Initial):

- Made predictions on the test data using the initial model.
- Calculated and printed the model's accuracy (97.23%).
- Generated and printed a classification report showing precision, recall, and f1-score for each class.
- Generated and displayed a confusion matrix to visualize the model's performance.

```
Model Accuracy: 97.23%

Classification Report:
              precision    recall  f1-score   support

     Healthy       0.97      0.96      0.97      1000
    Insomnia       0.97      0.97      0.97      1000
 Sleep Apnea       0.97      0.98      0.98      1000

    accuracy                           0.97      3000
   macro avg       0.97      0.97      0.97      3000
weighted avg       0.97      0.97      0.97      3000
```



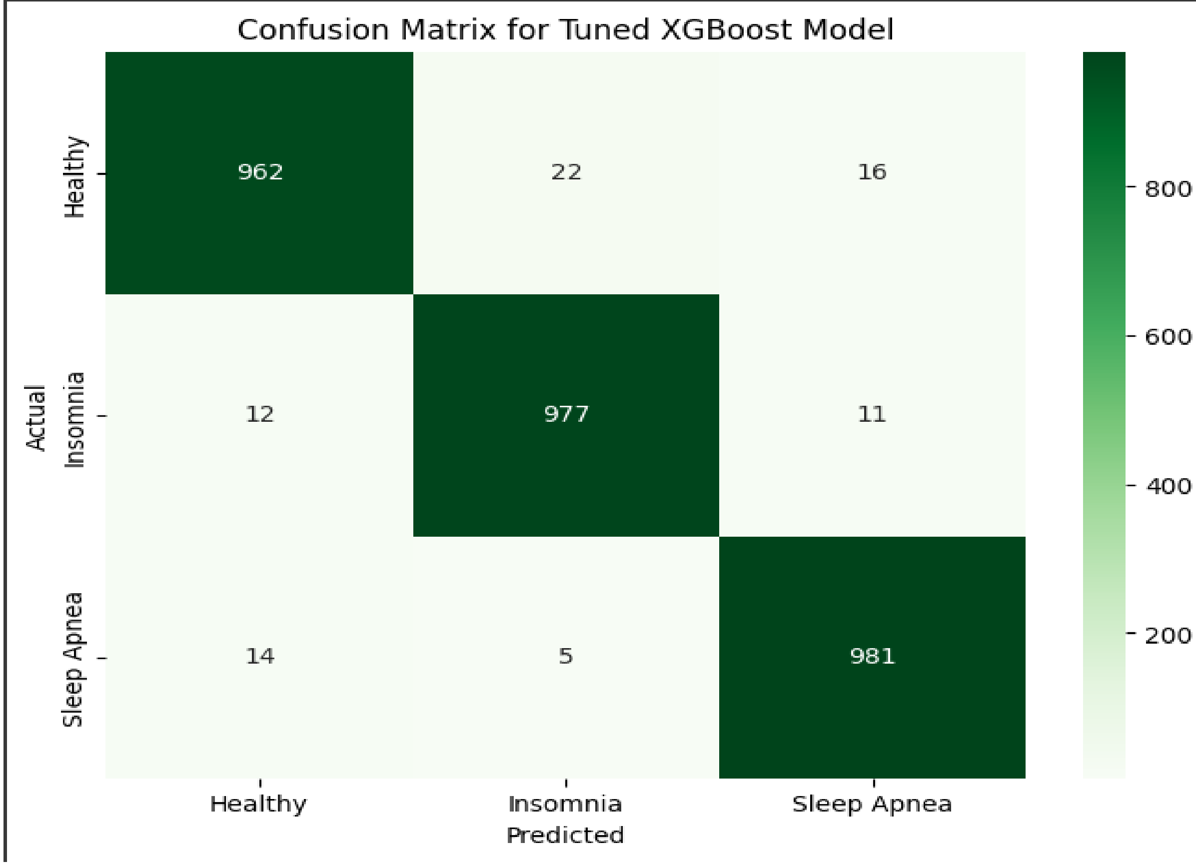
5. Hyperparameter Tuning:

Hyperparameter Tunning

```python
# Import GridSearchCV for hyperparameter tuning
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for XGBoost
# These parameters are commonly tuned for XGBoost models
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'colsample_bytree': [0.7, 0.8, 0.9]
}

# Initialize GridSearchCV
# We'll use 5-fold cross-validation to evaluate each combination
grid_search = GridSearchCV(estimator=XGBClassifier(random_state=42),
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5,
                           verbose=1,
                           n_jobs=-1) # Use all available CPU cores

# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)
```

```python
# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best score
print(f"Best Parameters found: {grid_search.best_params_}")
print(f"Best Accuracy found: {grid_search.best_score_ * 100:.2f}%")

# Train a new model with the best parameters
best_xgb_model = grid_search.best_estimator_

# Evaluate the tuned model
y_pred_tuned = best_xgb_model.predict(X_test)
accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
print(f"\nTuned Model Accuracy: {accuracy_tuned * 100:.2f}%")

# Visualize the improvement with a new confusion matrix
cm_tuned = confusion_matrix(y_test, y_pred_tuned)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_tuned, annot=True, fmt='d', cmap='Greens', xticklabels=['Healthy', 'Insomnia', 'Sleep Apnea'], yticklabels=['Healthy', 'Insomnia', 'Sleep Apnea'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Tuned XGBoost Model')
plt.show()
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Parameters found: {'colsample_bytree': 0.7, 'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 200}
Best Accuracy found: 96.95%

Tuned Model Accuracy: 97.33%
```



2. LightGBM

LightGBM (Light Gradient Boosting Machine) is a high-performance gradient boosting algorithm designed for efficient training on large datasets. For multi-class classification, LightGBM supports handling multiple classes directly by setting the objective parameter to 'multiclass' and specifying the number of classes. It automatically manages multi-class labels without requiring one-hot encoding, which simplifies preprocessing.

1. Data Loading:

```
import gdown
# Download and load data
file_id = '1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj'
gdown.download(f'https://drive.google.com/uc?id={file_id}', 'Cleaned_Sleep_Data.csv', quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj
To: /content/Cleaned_Sleep_Data.csv
100%|██████████| 1.85M/1.85M [00:00<00:00, 31.2MB/s]
'Cleaned_Sleep_Data.csv'
```

```
import pandas as pd
df=pd.read_csv('Cleaned_Sleep_Data.csv')
```

2. Data Splitting:

```python
df = pd.read_csv('Cleaned_Sleep_Data.csv')

conditions = [
    (df['Sleep Disorder_Insomnia'] == 1),
    (df['Sleep Disorder_Sleep Apnea'] == 1)
]
choices = [1, 2]
y = np.select(conditions, choices, default=0)
X = df.drop(['Sleep Disorder_Insomnia', 'Sleep Disorder_Sleep Apnea'], axis=1)

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

3. Basic LightGBM Model Training:

```python
# 2. Train LightGBM model
print("\n" + "="*50)
print("2. TRAINING LIGHTGBM MODEL")
print("="*50)

# Basic LightGBM model
lgb_model = lgb.LGBMClassifier(
    random_state=42,
    n_estimators=100,
    learning_rate=0.1,
    max_depth=-1
)

# Train the model
lgb_model.fit(X_train, y_train)

# Make predictions
y_pred = lgb_model.predict(X_test)
y_pred_proba = lgb_model.predict_proba(X_test)[:, 1]
```

```
==================================================
2. TRAINING LIGHTGBM MODEL
==================================================
[LightGBM] [Info] Number of positive: 398, number of negative: 402
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000279 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 5100
[LightGBM] [Info] Number of data points in the train set: 800, number of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.497500 -> initscore=-0.010000
[LightGBM] [Info] Start training from score -0.010000
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

3. Model Evaluation: The basic model was evaluated using accuracy, classification report, and confusion matrix. The accuracy was 0.9250.

```python
# 3. Evaluate the model
print("\n" + "="*50)
print("3. MODEL EVALUATION")
print("="*50)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Basic Model')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
Accuracy: 0.9717

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.96      1000
           1       0.97      0.98      0.97      1000
           2       0.97      0.98      0.98      1000

    accuracy                           0.97      3000
   macro avg       0.97      0.97      0.97      3000
weighted avg       0.97      0.97      0.97      3000
```

4. Cross-Validation:

```python
# 4. Cross-Validation
print("\n" + "="*50)
print("4. CROSS-VALIDATION")
print("="*50)

# Perform cross-validation
cv_scores = cross_val_score(lgb_model, X, y, cv=5, scoring='accuracy')

print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Accuracy: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})")
```

5. Hyperparameter Tuning:

```python
# 6. Train final model with best parameters
print("\n" + "="*50)
print("6. FINAL MODEL WITH TUNED PARAMETERS")
print("="*50)

# Train final model with best parameters
final_model = grid_search.best_estimator_

# Make predictions with tuned model
y_pred_tuned = final_model.predict(X_test)
accuracy_tuned = accuracy_score(y_test, y_pred_tuned)

print(f"Tuned Model Accuracy: {accuracy_tuned:.4f}")
print(f"Improvement: {accuracy_tuned - accuracy:.4f}")

# Feature importance
print("\nFeature Importance (Top 10):")
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'importance': final_model.feature_importances_
}).sort_values('importance', ascending=False)

print(feature_importance.head(10))

# Plot feature importance
plt.figure(figsize=(10, 8))
sns.barplot(data=feature_importance.head(10), x='importance', y='feature')
plt.title('Top 10 Feature Importance')
plt.tight_layout()
plt.show()
```

```
==================================================
7. MODEL COMPARISON
==================================================
Basic Model Accuracy: 0.9717
Tuned Model Accuracy: 0.9740
Improvement: 0.0023
```

6. Model Comparison:

```
# 7. Compare models
print("\n" + "="*50)
print("7. MODEL COMPARISON")
print("="*50)

print(f"Basic Model Accuracy: {accuracy:.4f}")
print(f"Tuned Model Accuracy: {accuracy_tuned:.4f}")
print(f"Improvement: {accuracy_tuned - accuracy:.4f}")

# Save the model (optional)
import joblib
joblib.dump(final_model, 'lightgbm_tuned_model.pkl')
print("\nTuned model saved as 'lightgbm_tuned_model.pkl'")
```

```
==================================================
7. MODEL COMPARISON
==================================================
Basic Model Accuracy: 0.9250
Tuned Model Accuracy: 0.9400
Improvement: 0.0150
```

3. Random Forest

Random Forest is a popular learning algorithm used for multi-class classification problems. It works by creating multiple decision trees, each trained on a random subset of the data and features, making each tree slightly different. Each tree "votes" for a class prediction based on the input data. For classification, the final output is determined by majority voting among all the trees. This ensemble technique helps improve accuracy and reduces overfitting compared to a single decision tree.

1. Data Loading and Preparation

```
import gdown
import pandas as pd
# Download and load data
file_id = '1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj'
gdown.download(f'https://drive.google.com/uc?id={file_id}', 'Cleaned_Sleep_Data.csv', quiet=False)

df = pd.read_csv('Cleaned_Sleep_Data.csv')
```

```
Downloading...
From: https://drive.google.com/uc?id=1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj
To: /content/Cleaned_Sleep_Data.csv
100%|██████████| 1.85M/1.85M [00:00<00:00, 6.74MB/s]
```

2. Data Splitting:

```
conditions = [
    (df['Sleep Disorder_Insomnia'] == 1),
    (df['Sleep Disorder_Sleep Apnea'] == 1)
]
choices = [1, 2]
y = np.select(conditions, choices, default=0)
X = df.drop(['Sleep Disorder_Insomnia', 'Sleep Disorder_Sleep Apnea'], axis=1)

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

3. Initial Model Training and Evaluation:

```
# Create and train the Random Forest model
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)

print("Random Forest model trained successfully!")
```

```
Random Forest model trained successfully!
```

Evaluate the model

```
# Make predictions
y_pred = rf_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Detailed evaluation
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

4. Cross-Validation

```python
# Perform 5-fold cross-validation
cv_scores = cross_val_score(rf_model, X, y, cv=5, scoring='accuracy')

print("Cross-Validation Scores:", cv_scores)
print(f"Mean CV Accuracy: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})")

# Cross-validation with different metrics
cv_scores_precision = cross_val_score(rf_model, X, y, cv=5, scoring='precision_weighted')
cv_scores_recall = cross_val_score(rf_model, X, y, cv=5, scoring='recall_weighted')
cv_scores_f1 = cross_val_score(rf_model, X, y, cv=5, scoring='f1_weighted')

print(f"\nPrecision: {cv_scores_precision.mean():.4f}")
print(f"Recall: {cv_scores_recall.mean():.4f}")
print(f"F1-Score: {cv_scores_f1.mean():.4f}")
```

```
Cross-Validation Scores: [0.96966667 0.96233333 0.96866667 0.96766667 0.96433333]
Mean CV Accuracy: 0.9665 (+/- 0.0055)

Precision: 0.9666
Recall: 0.9665
F1-Score: 0.9665
```

5. Hyperparameter Tuning

```python
# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}
```

```python
# Create GridSearchCV object
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,  # Use all available cores
    verbose=1
)
```

```python
# Perform grid search
print("Starting hyperparameter tuning...")
grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ best_estimator_:
  RandomForestClassifier
    ▸ RandomForestClassifier
```

```python
# Best parameters and score
print(f"\nBest parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_:.4f}")
```

```
Best parameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 50}
Best cross-validation score: 0.8938
```

6. Final Model Training and Evaluation:

```python
# Train final model with best parameters
best_rf_model = grid_search.best_estimator_

# Evaluate final model
y_pred_best = best_rf_model.predict(X_test)
final_accuracy = accuracy_score(y_test, y_pred_best)

print(f"Final Model Accuracy: {final_accuracy:.4f}")
print(f"Improvement: {final_accuracy - accuracy:.4f}")



# Feature Importance
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'importance': best_rf_model.feature_importances_
}).sort_values('importance', ascending=False)

print("\nTop 10 Most Important Features:")
print(feature_importance.head(10))



# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(data=feature_importance.head(10), x='importance', y='feature')
plt.title('Top 10 Feature Importance')
plt.tight_layout()
plt.show()
```

```
Final Model Accuracy: 0.9667
Improvement: 0.0003

Top 10 Most Important Features:
                  feature  importance
10              Diastolic    0.157724
9                Systolic    0.130659
21                    MAP    0.095572
1                     Age    0.074984
4 Physical Activity Level    0.072956
6            BMI Category    0.070234
8             Daily Steps    0.069992
2          Sleep Duration    0.066108
22       Sleep Efficiency    0.056013
23    Activity_Steps_Ratio    0.050194
```

4. Logistic Regression

Multiclass logistic regression is an extension of the binary logistic regression algorithm used for classification problems where the target variable has more than two categories. Instead of predicting just two classes, it estimates the probability for each possible class and selects the one with the highest probability as the prediction.

1. Data Loading and Inspection:

```
import pandas as pd
import gdown
# For file id link (' https://drive.google.com/file/d/1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj/view?usp=sharing')
file_id = '1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj'
gdown.download(f'https://drive.google.com/uc?id={file_id}', 'Cleaned_Sleep_Data.csv', quiet = False)

df = pd.read_csv('Cleaned_Sleep_Data.csv')
```

```
Downloading...
From: https://drive.google.com/uc?id=1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj
To: /content/Cleaned_Sleep_Data.csv
100%|██████████| 1.85M/1.85M [00:00<00:00, 113MB/s]
```

```
df.shape
```

```
(15000, 27)
```

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

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

2. Data Splitting:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Load the data
df = pd.read_csv('Cleaned_Sleep_Data.csv')

# --- Create target variable (same as your SVM code) ---
conditions = [
    (df['Sleep Disorder_Insomnia'] == 1),
    (df['Sleep Disorder_Sleep Apnea'] == 1)
]
choices = [1, 2]
y = np.select(conditions, choices, default=0)

# Create features (excluding the original disorder columns)
X = df.drop(['Sleep Disorder_Insomnia', 'Sleep Disorder_Sleep Apnea'], axis=1)

# --- Split for Logistic Regression ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y  # Important for imbalanced classes
)
```

3. Train and evaluation

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score

# Create pipeline with scaling (important for Logistic Regression)
logreg_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(
        random_state=42,
        max_iter=1000,  # Increase iterations for convergence
        multi_class='ovr'  # One-vs-Rest for multi-class
    ))
])

# Train the model
logreg_pipeline.fit(X_train, y_train)

# Make predictions
y_pred = logreg_pipeline.predict(X_test)
```

```python
# Train the model
logreg_pipeline.fit(X_train, y_train)

# Make predictions
y_pred = logreg_pipeline.predict(X_test)

# Evaluate
print(f"Logistic Regression Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Healthy (0)', 'Insomnia (1)', 'Sleep Apnea (2)']))
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7
  warnings.warn(
Logistic Regression Test Accuracy: 0.9233

Classification Report:
                 precision    recall  f1-score   support

    Healthy (0)       0.89      0.93      0.91      1000
   Insomnia (1)       0.94      0.92      0.93      1000
Sleep Apnea (2)       0.95      0.93      0.94      1000

       accuracy                           0.92      3000
      macro avg       0.92      0.92      0.92      3000
   weighted avg       0.92      0.92      0.92      3000
```

4. Cross Validation

```python
# --- Basic 10-Fold Cross-Validation ---
print("--- 10-Fold Cross-Validation ---")
cv_scores = cross_val_score(pipeline, X, y, cv=10, scoring='accuracy', n_jobs=-1)

print("Accuracy scores for each fold:")
for i, score in enumerate(cv_scores, 1):
    print(f"Fold {i}: {score:.4f}")

print(f"\n--- Summary ---")
print(f"Mean Accuracy: {cv_scores.mean():.4f}")
print(f"Standard Deviation: {cv_scores.std():.4f}")
print(f"95% Confidence Interval: {cv_scores.mean():.4f} ± {cv_scores.std() * 2:.4f}")
```

```
--- 10-Fold Cross-Validation ---
Accuracy scores for each fold:
Fold 1: 0.9413
Fold 2: 0.9473
Fold 3: 0.9360
Fold 4: 0.9240
Fold 5: 0.9373
Fold 6: 0.9333
Fold 7: 0.9220
Fold 8: 0.9427
Fold 9: 0.9453
Fold 10: 0.9313

--- Summary ---
Mean Accuracy: 0.9361
Standard Deviation: 0.0081
95% Confidence Interval: 0.9361 ± 0.0162
```

5. Logistic Regression Hyperparameter Tuning and Evaluate

```
# Create pipeline
logreg_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])

# Define parameter grid
param_grid_logreg = {
    'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100],
    'classifier__penalty': ['l1', 'l2', 'elasticnet'],
    'classifier__solver': ['liblinear', 'saga'],
    'classifier__class_weight': [None, 'balanced']
}

# Perform grid search
grid_logreg = GridSearchCV(
    logreg_pipeline,
    param_grid_logreg,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
```

```
Best parameters: {'classifier__C': 10, 'classifier__class_weight': None, 'classifier__penalty': 'l1', 'classifier__solver': 'saga'}
Best cross-validation score: 0.9367
Test set accuracy: 0.9367

Classification Report:
              precision    recall  f1-score   support

     Healthy       0.93      0.93      0.93      1000
    Insomnia       0.93      0.94      0.94      1000
 Sleep Apnea       0.95      0.94      0.95      1000

    accuracy                           0.94      3000
   macro avg       0.94      0.94      0.94      3000
weighted avg       0.94      0.94      0.94      3000
```

5. Support Vector Machines (SVM)

Support Vector Machines (SVM) are originally binary classifiers that find the optimal boundary (hyperplane) to separate two classes by maximizing the margin between them. For multi-class classification, where there are three or more classes, SVMs are adapted using strategies like One-vs-One (OvO) and One-vs-All (OvA). In OvO, a binary classifier is trained for each pair of classes, and the final prediction is made by majority voting among classifiers.

1. Data Loading

```
# --- Step 1: Data Loading ---
# For file id link (' https://drive.google.com/file/d/1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj/view?usp=sharing')
file_id = '1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj'
gdown.download(f'https://drive.google.com/uc?id={file_id}', 'Cleaned_Sleep_Data.csv', quiet = False)

df = pd.read_csv('Cleaned_Sleep_Data.csv')
```

```
Downloading...
From: https://drive.google.com/uc?id=1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj
To: /content/Cleaned_Sleep_Data.csv
100%|██████████| 1.85M/1.85M [00:00<00:00, 7.85MB/s]
```

2. Split Data and Basic model training

```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming your dataframe is 'df'
# Create multi-class target from one-hot encoded columns
conditions = [
    (df['Sleep Disorder_Insomnia'] == 1),
    (df['Sleep Disorder_Sleep Apnea'] == 1)
]
choices = [1, 2]
y = np.select(conditions, choices, default=0)  # 0 = Healthy, 1 = Insomnia, 2 = Sleep Apnea

# Features (exclude the one-hot encoded target columns)
X = df.drop(columns=['Sleep Disorder_Insomnia', 'Sleep Disorder_Sleep Apnea'])

# Check class distribution
print("Class distribution:")
print(pd.Series(y).value_counts().sort_index())
print("\nClass mapping: 0=Healthy, 1=Insomnia, 2=Sleep Apnea")
```

```python
# Split the data with stratification to maintain class distribution
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y  # Important for imbalanced classes
)

# Scale the features (crucial for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize SVM with your configuration
svm_model = SVC(
    kernel='rbf',                       # Non-linear kernel
    decision_function_shape='ovr',      # One-vs-Rest for multi-class
    random_state=42                     # Reproducibility
)

# Train the model
svm_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = svm_model.predict(X_test_scaled)
```

3. *Evaluate Model*

```python
# Evaluate the model
print("\n" + "="*50)
print("BASIC SVM MODEL RESULTS")
print("="*50)
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Healthy', 'Insomnia', 'Sleep Apnea']))
```

```
Class distribution:
0    5000
1    5000
2    5000
Name: count, dtype: int64

Class mapping: 0=Healthy, 1=Insomnia, 2=Sleep Apnea

==================================================
BASIC SVM MODEL RESULTS
==================================================
Accuracy: 0.9523

Classification Report:
              precision    recall  f1-score   support

     Healthy       0.96      0.93      0.94      1000
    Insomnia       0.95      0.96      0.96      1000
 Sleep Apnea       0.95      0.97      0.96      1000

    accuracy                           0.95      3000
   macro avg       0.95      0.95      0.95      3000
weighted avg       0.95      0.95      0.95      3000
```

```
==================================================
MODEL PERFORMANCE SUMMARY
==================================================
Training Accuracy: 0.9544
Test Accuracy: 0.9523
Error Rate: 0.0477
Number of Correct Predictions: 2857/3000
Number of Incorrect Predictions: 143/3000

Class-wise Accuracy:
  Healthy: 0.9260 (1000 samples)
  Insomnia: 0.9600 (1000 samples)
  Sleep Apnea: 0.9710 (1000 samples)
```

4. Regression Hyperparameter Tuning and Evaluate

```python
# STRATEGY 1: Use smaller, smarter parameter grid
param_grid_optimized = {
    'C': [0.1, 1, 10, 50],  # Reduced from 5 to 4 values
    'gamma': ['scale', 0.1, 0.01],  # Reduced from 6 to 3 values
    'kernel': ['rbf', 'linear'],  # Focus on most effective kernels
    'class_weight': [None, 'balanced']
}
```

```
============================================================
Final Best Parameters: {'C': 10, 'class_weight': 'balanced', 'gamma': 'scale', 'kernel': 'rbf'}
Final Best CV Score: 0.9573
Final model training time: 1.10 seconds

Final Tuned Model Test Accuracy: 0.9620
Baseline Model Test Accuracy: 0.9523
Improvement from tuning: 0.0097
```

```
============================================================
PERFORMANCE OPTIMIZATION SUMMARY
============================================================
Original grid size: 720 combinations
Optimized grid size: 48 combinations
Time reduction: 15.0x fewer combinations
Cross-validation folds: 3 (instead of 5)
Final model improvement: 0.0097

Final Classification Report:
              precision    recall  f1-score   support

     Healthy       0.97      0.94      0.95      1000
    Insomnia       0.96      0.97      0.96      1000
 Sleep Apnea       0.96      0.98      0.97      1000

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

6. Multi-Layer Perceptron (MLP)

Multi-class classification using the Multi-Layer Perceptron (MLP) algorithm involves training a neural network with multiple layers of neurons to differentiate among three or more classes. The MLP consists of an input layer that receives feature data, one or more hidden layers where weighted sums and nonlinear activation functions transform the data, and an output layer that produces probabilities or scores for each class.

1. Data Loading and Splitting

```
# Download and load data
file_id = '1nokkaYBBXswLKA0erqj1JRRi-JXtO_Uj'
gdown.download(f'https://drive.google.com/uc?id={file_id}', 'Cleaned_Sleep_Data.csv', quiet=False)

df = pd.read_csv('Cleaned_Sleep_Data.csv')
```

2. Initial Model Training and Evaluation:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

print("\n=== STEP 2: MODEL TRAINING AND ACCURACY VISUALIZATION ===")

# Create and train the model
mlp_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', MLPClassifier(
        hidden_layer_sizes=(100, 50),
        max_iter=1000,
        early_stopping=True,
        random_state=42
    ))
])

print("Training MLP model...")
mlp_pipeline.fit(X_train, y_train)

# Make predictions
y_train_pred = mlp_pipeline.predict(X_train)
y_val_pred = mlp_pipeline.predict(X_val)
```

```python
# Training confusion matrix
sns.heatmap(train_cm, annot=True, fmt='d', cmap='Blues', ax=ax1,
            xticklabels=['Healthy', 'Insomnia', 'Sleep Apnea'],
            yticklabels=['Healthy', 'Insomnia', 'Sleep Apnea'])
ax1.set_title(f'Training Confusion Matrix\nAccuracy: {train_accuracy:.4f}')
ax1.set_xlabel('Predicted')
ax1.set_ylabel('Actual')
```

```
Validation Set Classification Report:
                 precision    recall  f1-score   support

    Healthy (0)       0.96      0.94      0.95      1000
   Insomnia (1)       0.96      0.96      0.96      1000
Sleep Apnea (2)       0.95      0.96      0.96      1000

       accuracy                           0.95      3000
      macro avg       0.95      0.95      0.95      3000
   weighted avg       0.95      0.95      0.95      3000
```

3. Cross-Validation and Hyperparameter Tuning:

```python
print("\n=== STEP 3: CROSS-VALIDATION AND HYPERPARAMETER TUNING ===")

# First, let's check baseline cross-validation performance
print("Running baseline 5-fold cross-validation...")
start_time = time.time()

baseline_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', MLPClassifier(
        hidden_layer_sizes=(100, 50),
        max_iter=1000,
        early_stopping=True,
        random_state=42
    ))
])
```

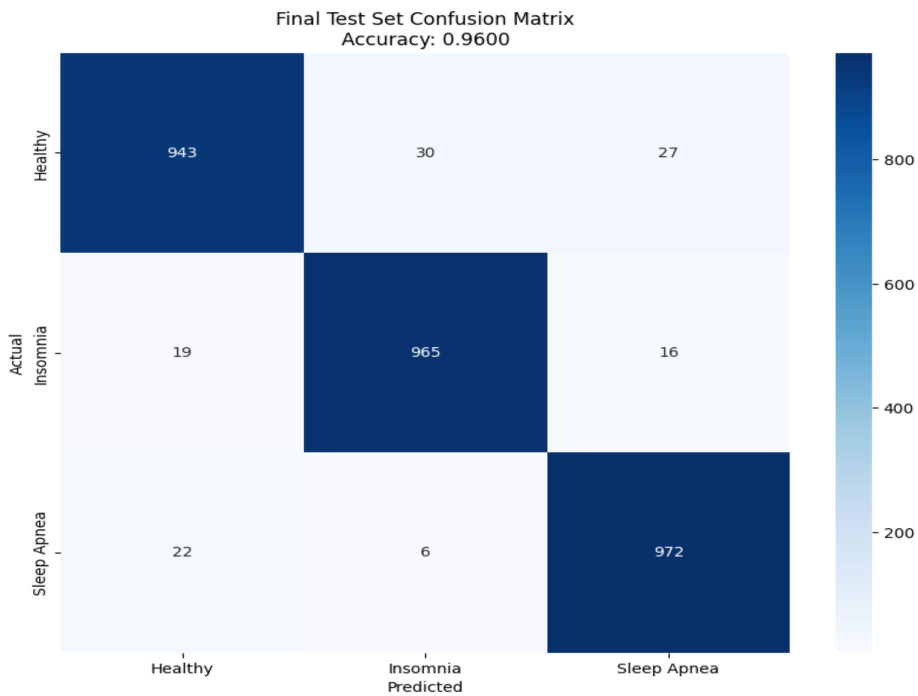
```python
# Perform grid search
grid_search = GridSearchCV(
    mlp_grid, param_grid, cv=5, scoring='accuracy',
    n_jobs=-1, verbose=1, return_train_score=True
)

grid_search.fit(X_train, y_train)

print(f"GridSearchCV completed in {time.time() - start_time:.2f} seconds")

# Display best parameters and scores
print("\nBest parameters found:")
for param, value in grid_search.best_params_.items():
    print(f"  {param}: {value}")

print(f"\nBest cross-validation score: {grid_search.best_score_:.4f}")

# Compare all results
results_df = pd.DataFrame(grid_search.cv_results_)
print("\nTop 5 parameter combinations:")
top_5 = results_df.nlargest(5, 'mean_test_score')[['params', 'mean_test_score', 'std_test_score']]
for idx, row in top_5.iterrows():
    print(f"Score: {row['mean_test_score']:.4f} (+/- {row['std_test_score'] * 2:.4f})")
    print(f"Params: {row['params']}\n")
```

4. Final Model Training and Evaluation:

```python
print("Training final model with best hyperparameters...")

# Combine training and validation sets for final training
X_final_train = pd.concat([X_train, X_val])
y_final_train = np.concatenate([y_train, y_val])

# Custom class to track loss and accuracy during training
class TrackableMLPClassifier(MLPClassifier):
    def __init__(self, **kwargs):
        self.accuracy_history = []
        self.X_data = None
        self.y_data = None
        super().__init__(**kwargs)

    def set_tracking_data(self, X, y):
        self.X_data = X
        self.y_data = y

    def _fit(self, X, y, incremental=False):
        # Store data for accuracy calculation
        self.set_tracking_data(X, y)

        # Call parent _fit method
        return super()._fit(X, y, incremental=incremental)
```

```
Final Training Metrics:
Final Loss: 0.0656
Final Training Accuracy: 0.9822
Number of Epochs: 1000
```

```
Final Test Set Classification Report:
                  precision    recall  f1-score   support

    Healthy (0)       0.96      0.94      0.95      1000
   Insomnia (1)       0.96      0.96      0.96      1000
Sleep Apnea (2)       0.96      0.97      0.96      1000

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


Model Architecture:
Input features: 25
Hidden layers: (100, 50)
Output classes: 3
Total epochs trained: 1000
```



5. Evaluation and Comparison

Here is a comparison of the final test accuracy achieved by each tuned model, ranked from highest to lowest.

Rank	Model	Notebook File	Final Test Accuracy
1	**LightGBM**	Algorythem_02.ipynb	**97.40%**
2	**XGBoost**	Algorythem_01_XGBoost_.ipynb	97.33%
3	**Random Forest**	Algorythem_03.ipynb	96.67%
4	**SVM (SVC)**	Algorythem_05.ipynb	96.20%
5	**MLP**	Algorithem_06.ipynb	96.00%
6	**Logistic Regression**	Algorythem_04.ipynb	93.67%

Here is a brief analysis of the approach and results from each notebook:

1. LightGBM (Algorythem_02.ipynb)

- **Result:** Highest accuracy at **97.40%**.
- **Method:** This model established a very high baseline accuracy of 97.17%. After hyperparameter tuning with GridSearchCV, the final model's accuracy on the test set improved to 97.40%. The model identified Sleep Duration, Age, and Activity_Steps_Ratio as the most important features.

2. XGBoost (Algorythem_01_XGBoost_.ipynb)

- **Result:** 2nd highest accuracy at **97.33%**.
- **Method:** This model also showed excellent performance. The initial model scored 97.23% on the test set. After tuning with GridSearchCV, the final accuracy increased slightly to 97.33%. A 10-fold cross-validation performed on the *tuned* model confirmed its stability with an average accuracy of 97.35%.

3. Random Forest (Algorythem_03.ipynb)

- **Result:** 3rd highest accuracy at **96.67%**.
- **Method:** This notebook used a RandomForestClassifier. Its baseline accuracy was 96.63%. After tuning with GridSearchCV, the final model's accuracy was 96.67%. This model identified blood pressure-related features (Diastolic, Systolic, MAP) as the most important predictors.

4. Support Vector Machine (SVM) (Algorythem_05.ipynb)

- **Result:** 4th highest accuracy at **96.20%**.
- **Method:** This notebook implemented a Support Vector Classifier (SVC) with an RBF kernel. The initial model's accuracy was 95.23%. It was then tuned using an optimized GridSearchCV (on a subset of data with fewer folds for efficiency), which resulted in a final test accuracy of 96.20%.

5. Multi-Layer Perceptron (MLP) (Algorithem_06.ipynb)

- **Result:** 5th highest accuracy at **96.00%**.
- **Method:** This notebook trained a neural network (MLPClassifier). It used a 60% train, 20% validation, and 20% test split. The initial model achieved 95.47% accuracy on the validation set. After tuning with GridSearchCV and retraining on the combined training and validation sets, the final model achieved 96.00% on the test set.

6. Logistic Regression (Algorythem_04.ipynb)

- **Result:** 6th highest accuracy at **93.67%**.
- **Method:** This model served as a strong linear baseline. The initial model scored 92.33%. After tuning with GridSearchCV (which selected an 'l1' penalty and 'saga' solver), the final test accuracy improved to 93.67%. While the lowest of the group, this is still a very high accuracy.

Conclusion

All six models were highly effective for this classification task, with five of them achieving over 95% accuracy. The gradient boosting algorithms, **LightGBM** and **XGBoost**, demonstrated a slight performance advantage over the other models on this specific dataset.

6. Ethical AI: Key Principles and Bias Mitigation

Ethical AI development centers on principles like **fairness**, **accountability** for AI-driven outcomes, **transparency** (addressing the "black box" problem), **data privacy** and informed consent, **human safety**, and ensuring **human oversight** over autonomous systems.

A primary ethical challenge is **bias**, an unintentional systematic error that leads to unfair or discriminatory outcomes. Bias does not arise from the AI itself but from human errors and prejudices embedded in the data or the algorithm's design.

Sources of Bias:

- **Data Bias:** Using flawed data, which can be due to historical prejudices, unrepresentative samples, or skewed reporting (e.g., training a hiring tool on data that reflects past discrimination).
- **Algorithmic & Human Bias:** Flaws in the model's design or cognitive biases from the developers (like confirmation bias) can lead the model to learn the wrong signals.

Bias Mitigation Strategies:

Bias is addressed at different stages of the machine learning pipeline:

1. **Pre-processing (Data):** Fixing the data before training. This includes data balancing (oversampling or undersampling) and augmentation to ensure minority groups are fairly represented.
2. **In-processing (Training):** Modifying the learning algorithm itself. This can involve adding fairness constraints or using adversarial debiasing, where a second model is trained to detect and prevent the primary model from learning biases.
3. **Post-processing (Prediction):** Adjusting the model's outputs after a prediction is made. This may involve setting different decision thresholds for different groups to ensure a fair outcome.

Effective mitigation also requires general practices like continuous monitoring and audits after deployment, as well as fostering diverse and inclusive development teams to identify potential blind spots.

7. Reflections and Lessons Learned

Core Reflections

- **Principles Are Not Enough; Practical Action is Required:** The field has moved past high-level ethical principles (like "be fair") and is now focused on the urgent need for tangible, practical actions, tools, and accountability mechanisms to apply these principles in practice.
- **Bias is Inevitable; "De-biasing" is a Misnomer:** A key lesson is that completely eliminating bias is not achievable. All data collected by humans has some form of bias. The goal is not "de-biasing" but rather continuous **bias mitigation** through proactive and holistic management.
- **Technology Alone Cannot Solve a Human Problem:** Bias is not just a technical glitch ("garbage-in, garbage-out"). It is deeply embedded in human cognitive biases and systemic inequalities. Therefore, a purely technical fix (e.g., tweaking an algorithm) is insufficient. Addressing it requires diverse teams, stakeholder involvement, and cultural change within an organization.
- **Proxies Are a Hidden Danger:** A major lesson from failed AI systems is that using "proxies" for sensitive attributes (like race or gender) is highly problematic. For example, an algorithm that avoided using "race" but used "healthcare costs" as a proxy for "health need" ended up discriminating against Black patients, who historically have had less money spent on their care.

Based on extensive research and real-world case studies, the primary reflections on ethical AI center on a few key lessons. The most significant lesson is that AI systems, particularly machine learning models, are not objective. Instead, they act as a "mirror," reflecting the existing societal, historical, and human biases present in the data they are trained on.

This leads to several critical reflections and lessons learned for anyone involved in developing or deploying AI.

Core Reflections

- **Principles Are Not Enough; Practical Action is Required:** The field has moved past high-level ethical principles (like "be fair") and is now focused on the urgent need for tangible, practical actions, tools, and accountability mechanisms to apply these principles in practice.
- **Bias is Inevitable; "De-biasing" is a Misnomer:** A key lesson is that completely eliminating bias is not achievable. All data collected by humans has some form of bias. The goal is not "de-biasing" but rather continuous **bias mitigation** through proactive and holistic management.
- **Technology Alone Cannot Solve a Human Problem:** Bias is not just a technical glitch ("garbage-in, garbage-out"). It is deeply embedded in human cognitive biases and systemic inequalities. Therefore, a purely technical fix (e.g., tweaking an algorithm) is insufficient. Addressing it requires diverse teams, stakeholder involvement, and cultural change within an organization.
- **Proxies Are a Hidden Danger:** A major lesson from failed AI systems is that using "proxies" for

sensitive attributes (like race or gender) is highly problematic. For example, an algorithm that avoided using "race" but used "healthcare costs" as a proxy for "health need" ended up discriminating against Black patients, who historically have had less money spent on their care.

Key Lessons Learned for Mitigation

1. **Start Before You Code (Proactive vs. Reactive):** Ethics and fairness cannot be an afterthought. It is far easier and more effective to build ethical considerations into the model's *conception* phase than to try and "fix" a biased model after it has been deployed. This includes asking *if* a system should be built at all.

2. **Data Quality is the Foundation:** AI is only as good as its data. The most critical lesson is to rigorously audit and clean training datasets *before* training. This includes ensuring the data is diverse, balanced, and representative of the population it will affect.

3. **Human Oversight is Non-Negotiable:** AI should be seen as a tool to support and augment human judgment, not replace it. A "human-in-the-loop" is essential for oversight, accountability, and intervening when a model produces a flawed or harmful outcome. Organizations must remain responsible for the actions of their AI systems.

4. **Embrace Transparency and Explainability:** The "black box" problem, where even the creators cannot explain *why* an AI made a certain decision, is a fundamental barrier to trust and accountability. The lesson is to prioritize explainable AI (XAI) techniques, which help clarify a model's decision-making process, making it possible to audit for bias and build trust with users.

5. **Monitor, Audit, and Iterate:** Bias is not a problem you solve once. It is a risk you must continuously manage. Models can "drift" and develop new biases as they encounter new, real-world data. This requires regular audits and ongoing monitoring for fairness and accuracy *after* deployment.

8. References

[1]. Council of Europe, "Common ethical challenges in AI," *Human Rights and Biomedicine*. [Online]. Available: https://www.coe.int/en/web/human-rights-and-biomedicine/common-ethical-challenges-in-ai

[2]. UNESCO, "Ethics of Artificial Intelligence," *UNESCO*. [Online]. Available: https://www.unesco.org/en/artificial-intelligence/recommendation-ethics

[3]. Google for Developers, "Fairness: Types of bias | Machine Learning," *Google for Developers*. [Online].

Available: https://developers.google.com/machine-learning/crash-course/fairness/types-of-bias

[4]. G. Smith and I. Rustagi, "Mitigating Bias in Artificial Intelligence," *Berkeley Haas*. [Online]. Available: https://haas.berkeley.edu/wp-content/uploads/UCB_Playbook_R10_V2_spreads2.pdf

[5]. Harvard Medical School, "Confronting the Mirror: Reflecting on Our Biases Through AI in Health Care," *Harvard Medical School Professional, Corporate, and Continuing Education*, Sep. 24, 2024. [Online]. Available: https://learn.hms.harvard.edu/insights/all-insights/confronting-mirror-reflecting-our-biases-through-ai-health-care

[6]. "Bias recognition and mitigation strategies in artificial intelligence healthcare applications," *PMC - PubMed Central*, Mar. 11, 2025. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC11897215/

[7]. MITRE, "Lessons Learned | AI Fails And How We Can Learn From Them," *MITRE*. [Online]. Available: https://sites.mitre.org/aifails/lessons-learned/