Sirui Tang

stang22@u.rochester.edu

University of Rochester

CSC 240

Professor Pawlicki

Exploring the factors that define the sleep disorder

A Multiclass Classification Approach Using Sleeping Data

I. Abstract

In this project, I will focus on exploring factors that define sleeping disorders using the Sleep Health and Lifestyle Dataset. The main goal of this study is to utilize a range of sleep and health indices to look at the kind of sleep symptoms that the person might be suffering from. The study focuses on data gathered from individuals, analyzing attributes such as sleep duration, efficiency, and heart rate, alongside lifestyle factors like occupation, exercise, and stress level. After preprocessing the dataset, I applied naïve Bayes classification, decision tree classifier, and support vector machine to the dataset in order to train the model. Then I will use the K-Fold validation, ROC curve, and AUC score to assess the accuracy of a classification model. Hopefully, this model will help people self-test their sleep health in the future.

II. Introduction

Because of the fast pace of modern life, people seldom bother to consciously develop healthy sleeping habits. Young people are more accustomed to making up for their mental needs by sacrificing sleep, and the older generation may also suffer from various types of sleep disorders due to physical reasons. Therefore, timely sleep diagnosis can help people identify the causes and treat them to a certain extent. In this project, I will explore the Sleep Health and Lifestyle Dataset to investigate the factors that lead to sleep disorders and build a model to predict what kind of sleep disorder the person may be diagnosed with.

In this report, I will focus on training to build a model and testing the different sleep disorders from the dataset using the naive Bayes, decision tree, and support vector machine algorithms. The fundamental concept behind the Bayesian classifier involves applying Bayes' theorem to determine the likelihood of each class based on the training data. On the other hand, the Decision Tree classifier employs a non-parametric method that repeatedly divides the input space into distinct regions corresponding to various classes. This classifier is straightforward to interpret and accommodates both continuous and categorical features. Additionally, the Support Vector Machine (SVM) identifies the optimal hyperplane (decision boundary) that distinguishes between different data point classes. This method offers several benefits, such as high precision, strong generalization capabilities, and the capacity to manage data with many dimensions. I will finally compare the performance of each model, and examine the accuracy of each model to predict classes that represent sleep disorders with influential features, by using cross-validation to evaluate the performance of each classifier and compare their Receiver Operating Characteristic (ROC) Area Under the Curve (AUC) scores.

III. Methods

a. Data Collection

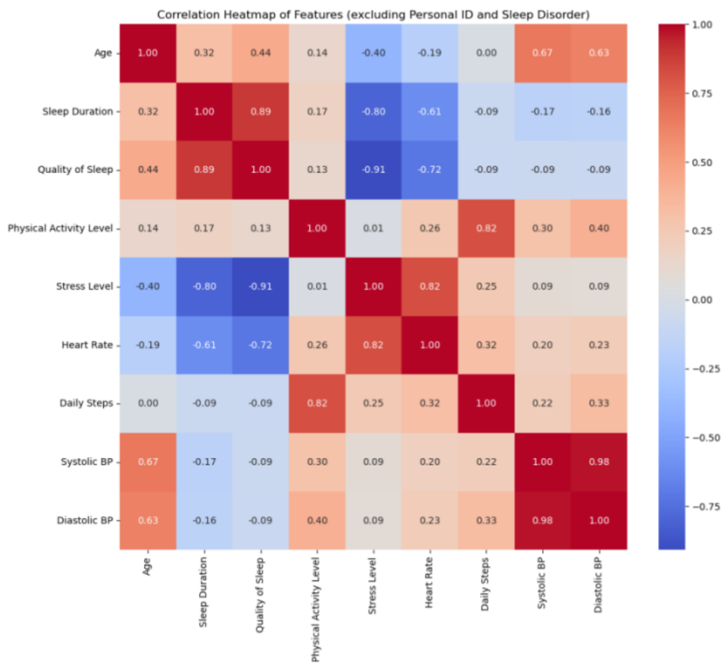
The dataset is from Kaggle and maintains 374 respondents with 3 types of sleep disorders, including None, Insomnia, and Sleep Apnea[1]. I used Python programming language and various libraries such as pandas, NumPy, and Scikit-learn to preprocess the data, extract relevant features, and build the classification models.

b. Data Processing

When I first looked at the data and checked whether there were some missing values, I found out that this dataset is quite clean. However, since the dataset is quite small, extreme outliers may influence the dataset. In order to make a better prediction and build a model with good performance, I decided to remove the outliers from the dataset first. I go through all of the features and remove the entire rows that contain at least one outlier. Finally, the dataset ended up with 359 respondents with 3 types of sleep disorders.

Also, I noticed that the feature called Blood Pressure contains two different pieces of information, Systolic BP and Diastolic BP. It is reasonable for us to keep them together as one feature; however, from the article, “The relationship between systolic and diastolic blood pressure: a clinically meaningful slope?” [2], shows that there is a linear relationship between those two blood pressures, which means that those two features may have a strong correlation with each other. Therefore, I preferred to split the Blood Pressure column into two columns, one is for Systolic BP and the other is for Diastolic BP.

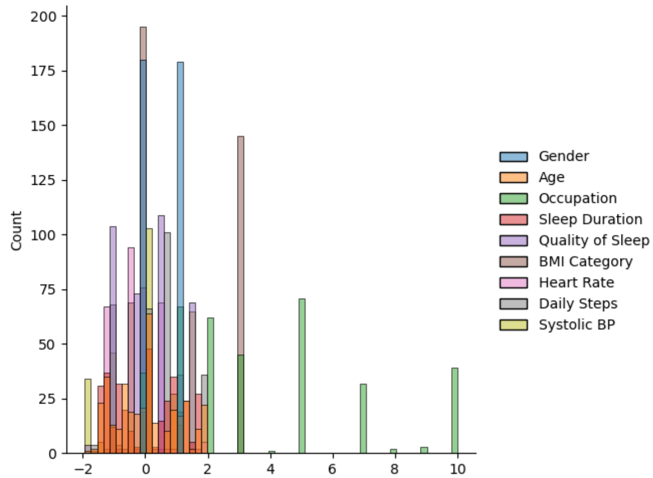
After collecting and cleaning the data, I proceeded to check the correlation between each of the variables in the dataset. I generated a correlation table, which revealed the relationships between the different attributes. Upon examining the correlation table, I observed that there were strong correlations between some of the variables, like Sleep Duration and Quality of Sleep, Systolic BP and Diastolic BP, Daily steps and Physical activity level, and heart rate and stress level. All of those correlations are higher than 0.85 in the heatmap and indicate that I need to remove any of the variables from my analysis. However, as the research “Shorter sleep duration and better sleep quality are associated with greater tissue density in the brain” shows, it is possible to have a better sleep quality with a shorter sleep duration[3]. Therefore, I will only remove three features, Physical Activity Level, Stress Level, and Diastolic BP. The reason why I try to not delete all the highly correlated features, is I want to use most features I can obtain to gain a more comprehensive understanding of the data. In the end, the absence of strong correlations between variables suggests that there is no multicollinearity issue, which could have affected the accuracy of my model. Overall, this finding reinforces the validity and reliability of my data analysis and allows me to proceed with confidence in our further analysis and modeling.



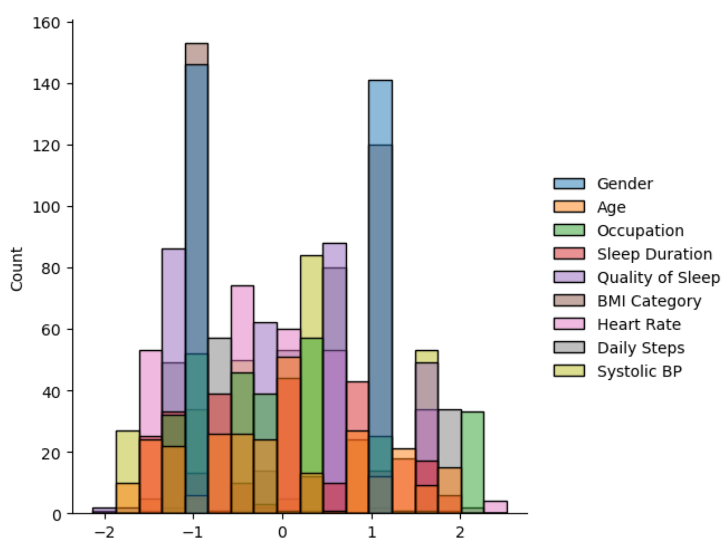
Then I encoded categorical variables, and normalized the numerical features; detailed code is attached:

```python
# Encoding categorical variables
categorical_vars = ['Gender', 'Occupation', 'BMI Category', 'Sleep Disorder']
label_encoders = {}  # Storing label encoders for potential inverse transformation

for var in categorical_vars:
    le = LabelEncoder()
    data[var] = le.fit_transform(data[var])
    label_encoders[var] = le

# Normalizing numerical features excluding 'Person ID' as it's just an identifier
numerical_vars = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Systolic BP'
                  , 'Heart Rate', 'Daily Steps']
data[numerical_vars] = StandardScaler().fit_transform(data[numerical_vars])
```

Then I check the normality of the dataset, and I graph the distribution of the data:



I observed that values in data columns have varying magnitudes and ranges, meaning that each column of data is not comparable to the others. In order to address this issue and make the data suitable for analysis, I have decided to apply a technique called Standard Scaler (*Standardscaler()*) to the feature data variables. The StandardScaler is a feature scaling technique that is part of the scikit-learn machine learning library in Python. It is used to standardize the features of a dataset around the mean with a unit standard deviation. This scaling process is important because many machine learning algorithms perform better when numerical input variables are on a similar scale. is particularly useful when we know that the data distribution for features is normal or when we want to enforce this assumption for the algorithms that expect standardized inputs. And here is the data distribution after the scaling:



My dataset contains 359 data points, which is a sufficiently large sample size (greater than 30). Additionally, the variables in our dataset are independent and randomly collected, further supporting our assumption of normal distribution.

We declared the target variable and made some modifications to the dataset. Then we randomly split the dataset into training and test sets using a 7:3 ratio, with 70% of the data used for training and 30% used for testing; detailed code is attached:

```
# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data_drop_all, classes, test_size = 0.2, random_state = 42)
```

c. Decision Tree Classifier

Following the dataset's preprocessing, which entailed rectifying missing data, transforming categorical variables into numerical equivalents, and applying normalization to the numerical features, I embarked on constructing a predictive model for the target outcome. My initial strategy was to apply a Decision Tree Classifier to the prepared training data. For guidance on utilizing the `sklearn.tree` package[4], I consulted resources available on the Kaggle platform, which presented an instructive and succinct overview. This resource was instrumental in enabling the swift creation of the classifier object and its subsequent fitting to the training dataset.

Once the model was trained, I evaluated its efficacy by predicting outcomes for both the training and test datasets. The resulting accuracy was 93.03% for the training set and slightly higher at 93.06% for the test set, indicating that the model achieved similar performance on both sets. Nevertheless, it is critical to consider that this represents our preliminary model version, and there could be opportunities to enhance precision through further refining our preprocessing tactics, feature selection, or the adoption of alternative classifiers. Enclosed is the code utilized for the training and assessment of the Decision Tree Classifier on our data:

```
from sklearn.tree import DecisionTreeClassifier
# generate decision tree classifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
# print the scores on training and test set
print('Training set score: {:.4f}'.format(dt.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(dt.score(X_test, y_test)))
```

```
Training set score: 0.9303
Test set score: 0.9306
```

I then proceeded to chart the receiver operating characteristic (ROC) curve, incorporating the area under the curve (AUC) computation[5]. The process began with converting the labels into a binary format to facilitate the ROC and AUC calculations, appropriate for a binary classification context where the focus is on the proportions of true positives and false positives. This binary transformation of the labels is crucial as it delineates whether a sample is part of the positive or negative class, which is a fundamental step in this analysis. Here's the method we employed:

```
# Instantiate the label binarizer and fit it to the true class labels
label_binarizer = LabelBinarizer().fit(y_train)
# Binarize the labels for computing ROC curve and AUC score
y_test_bin = label_binarizer.transform(y_test)
y_pred_prob = dt.predict_proba(X_test)  # Get the predicted probabilities
```

Next, within a loop, I calculated the ROC curve and AUC score for every class, recording the true positive rate, false positive rate, and the AUC metrics using the code below:

```
# Compute ROC curve and AUC score for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

Once those metrics were captured, I used a for loop to plot the ROC curve. The specific code for this task can be found in the following section, and a discussion of the ROC and AUC results will be presented subsequently. Detailed codes are shown:

```
# Plot ROC curves
plt.figure()
lw = 2
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=lw, label='Class %0.0f ROC (AUC = %0.2f)' % (i, roc_auc[i]))
plt.plot([0, 1], [0, 1], color='gray', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

I also checked the Cross validated ROC AUC scores:

```
from sklearn.model_selection import cross_val_score

# Calculate the ROC-AUC score
Cross_Val_Score = cross_val_score(dt, X_train, y_train, cv=5, scoring='roc_auc_ovr').mean() # Use 'micro' or 'weight

print(f"Cross validated ROC AUC: {Cross_Val_Score}")
```

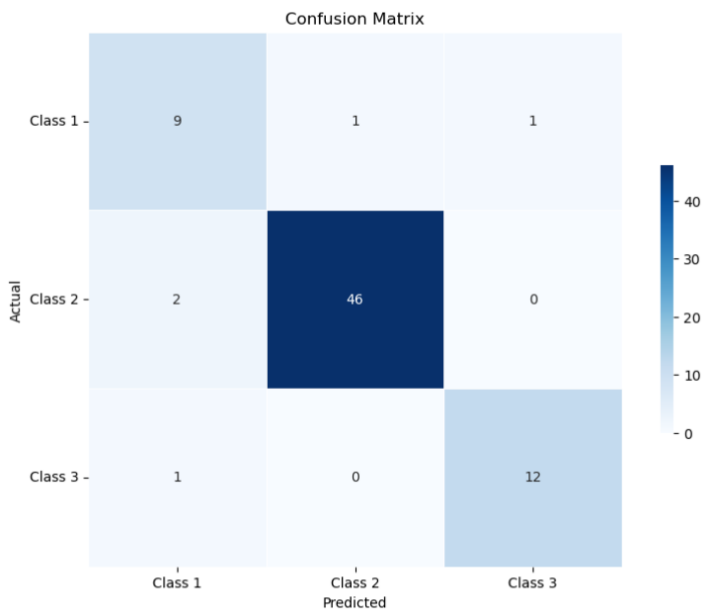
```
Cross validated ROC AUC: 0.9003757203021682
```

Furthermore, I employed 10-Fold Cross Validation[6] on the dataset to determine the cross-validation scores. The average accuracy from this cross-validation, denoted by its mean, was calculated to be 90.82%:

```
from sklearn.model_selection import cross_val_score, KFold

# Create k-fold cross-validation object
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Perform k-fold cross-validation
scores = cross_val_score(dt, data_drop_all, classes, cv=kfold)

# Print accuracy scores for each fold and the mean score
print("Accuracy scores:", scores)
print("Mean accuracy:", scores.mean())
```

```
Accuracy scores: [0.94444444 0.91666667 0.91666667 0.77777778 0.97222222 0.97222222
 0.88888889 0.94444444 0.80555556 0.94285714]
Mean accuracy: 0.9081746031746032
```

In the end, I also create the confusion matrix based on the tutorial on Kaggle[6]. The detailed confusion matrix in this model is shown below:



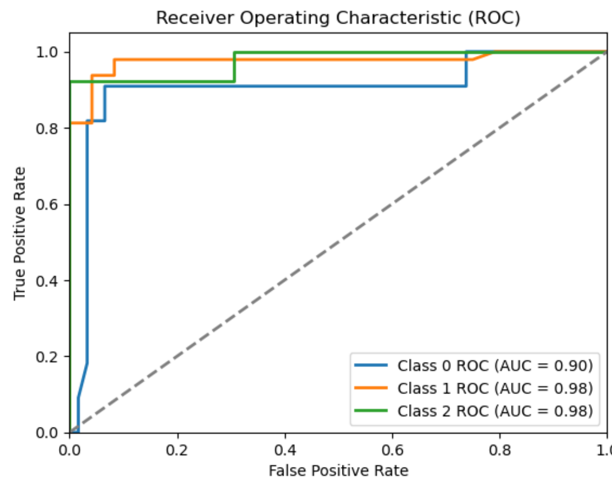
d. Bayesian Classifier

Additionally, I employed a Gaussian Naive Bayes Classifier, utilizing identical training and testing datasets for the inputs (X_train, X_test) and outputs (y_train, y_test). This model was constructed using the Gaussian Naive Bayes module from the sklearn.naive_bayes library[7]. The resulting accuracy was 89.55% for the training dataset, while the model achieved a slightly higher accuracy of 91.67% on the testing dataset. It seems counterintuitive, but this situation may be due to the size and distribution of the dataset, randomness, and so on. The Naive Bayes algorithm is based on the assumption that the features are independent given the class label. If this assumption happens to hold reasonably true for the testing set but not for the training set, the model might perform better on the testing set.

```python
# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB
# instantiate the model
gnb = GaussianNB()
# fit the model
gnb.fit(X_train, y_train)
y_pred_gnb = gnb.predict(X_test)
from sklearn.metrics import accuracy_score
y_pred_train_gnb = gnb.predict(X_train)
# print the scores on training and test set
print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))
```

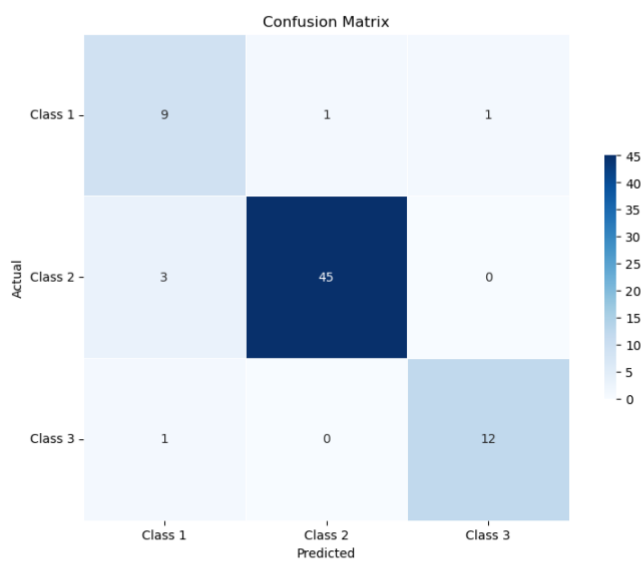
```
Training set score: 0.8955
Test set score: 0.9167
```

Then I go through a similar process of plotting ROC/AUC, confusion matrix, and implementing k-fold as we did in the Decision Tree Classifier. For k-hold, I got a mean accuracy of 92.29%.




```python
from sklearn.model_selection import cross_val_score

# Calculate the ROC-AUC score
Cross_Val_Score = cross_val_score(gnb, X_train, y_train, cv=5, scoring='roc_auc_ovr').mean()
# Use 'micro' or 'weighted' for alternative aggregation methods

print(f"Cross validated ROC AUC: {Cross_Val_Score}")
```

```
Cross validated ROC AUC: 0.9228978483004349
```




```python
from sklearn.model_selection import cross_val_score, KFold

# Create k-fold cross-validation object
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Perform k-fold cross-validation
scores = cross_val_score(gnb, data_drop_all, classes, cv=kfold)

# Print accuracy scores for each fold and the mean score
print("Accuracy scores:", scores)
print("Mean accuracy:", scores.mean())
```

```
Accuracy scores: [0.94444444 0.88888889 0.86111111 0.88888889 0.97222222 0.94444444
 0.88888889 0.86111111 0.80555556 0.91428571]
Mean accuracy: 0.896984126984127
```

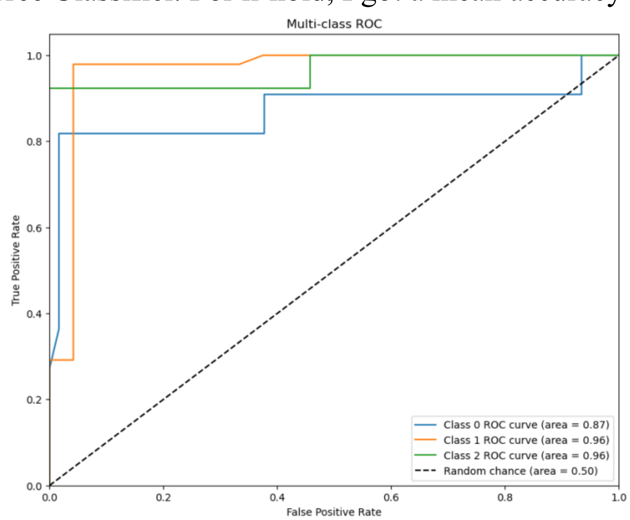
e. Support Vector Machine

For the Support Vector Machine (SVM) Classifier, I selected the linear kernel for its effectiveness and computational efficiency with data that is linearly separable[8]. I fixed the random state at 42 to guarantee consistent results across different runs. Post-training the SVM model with the training data, its performance was assessed on the test data. The model demonstrated comparable results on both datasets, with an accuracy of 91.99% on the training set and 94.44% on the testing set. It is still counterintuitive, but it makes sense that an SVM model can show comparable or even slightly better results on the testing set compared to the training set. This could be due to several reasons, including effective model generalization, the representativeness of the test data, or potentially a simpler model that doesn't overfit.

```python
from sklearn import svm #Import svm model
from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.metrics import confusion_matrix

# Instantiate the SVM classifier and fit it to the training data
# Use CalibratedClassifierCV to get the probability-like outputs from decision_function
svm = SVC(kernel='linear', probability=True, random_state=42)  # Linear kernel used as an example
clf = CalibratedClassifierCV(svm)  # Calibrate the SVM
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Training set score: 0.9199
Test set score: 0.9444
```

Then I go through a similar process of plotting ROC/AUC and implementing k-fold as we did in the Decision Tree Classifier. For k-hold, I got a mean accuracy of 92.21%.



```python
from sklearn.model_selection import cross_val_score

# Calculate the ROC-AUC score
Cross_Val_Score = cross_val_score(svm, X_train, y_train, cv=5, scoring='roc_auc_ovr').mean() # Use 'micro' or 'weigh

print(f"Cross validated ROC AUC: {Cross_Val_Score}")
```

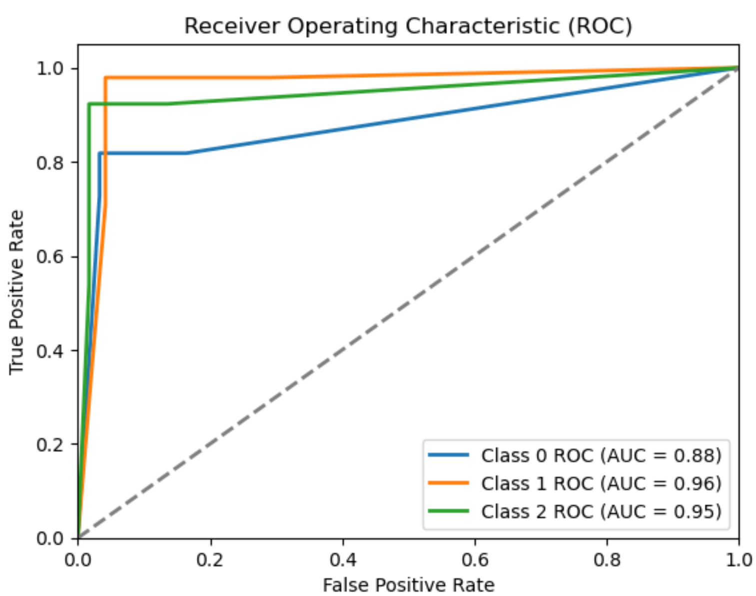
```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/validation.py:1688: FutureWarning: Feature names only supp
ort names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  warnings.warn(

Cross validated ROC AUC: 0.9090830828978603
```

```python
from sklearn.model_selection import cross_val_score, KFold

# Create k-fold cross-validation object
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
#scores = cross_val_score(clf, spotify_drop_all, classes, cv=5, scoring='roc_auc_ovr').mean()
#print(scores)
# Perform k-fold cross-validation
scores = cross_val_score(clf, data_drop_all, classes, cv=kfold)
Cross_validated_ROC_AUC = cross_val_score(svm, X_train, y_train, cv=5, scoring='roc_auc_ovr').mean()

# Print accuracy scores for each fold and the mean score
print("Accuracy scores:", scores)
print("Mean accuracy:", scores.mean())
```

```
Accuracy scores: [0.94444444 0.94444444 0.88888889 0.88888889 1.         0.97222222
 0.88888889 0.94444444 0.77777778 0.97142857]
Mean accuracy: 0.9221428571428572
```

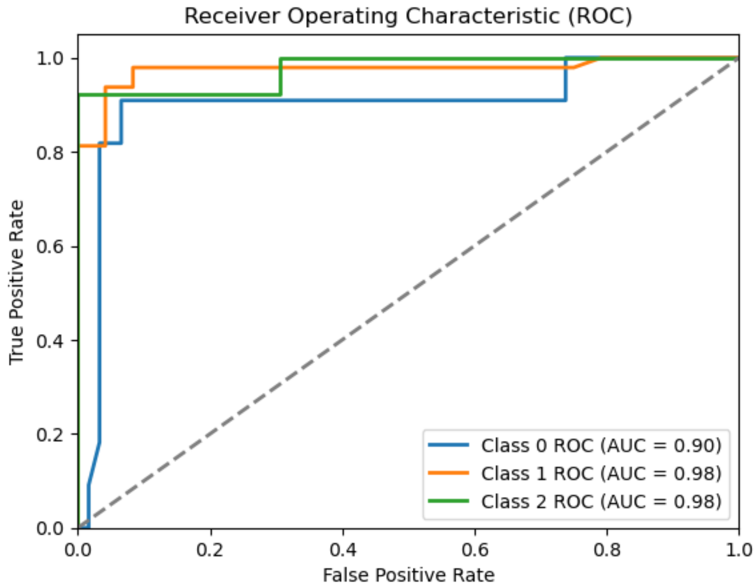
IV. Results

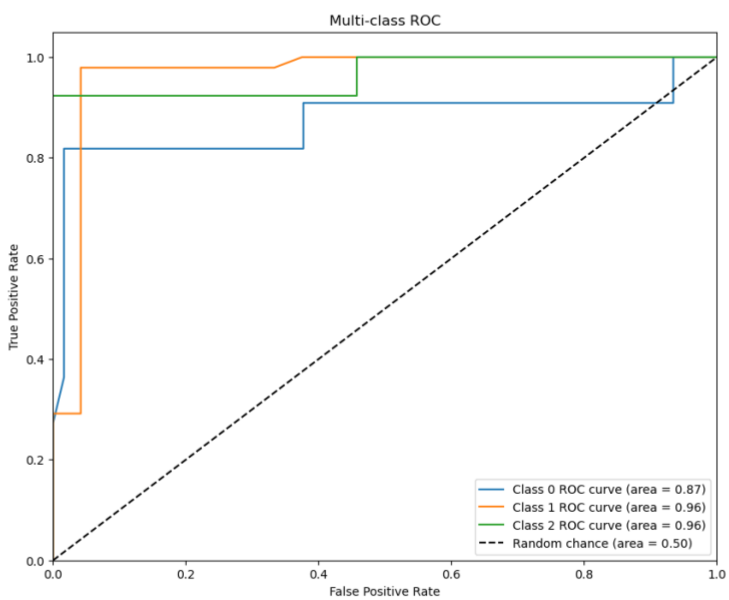
Here's a breakdown of the information for each model:

1. Decision Tree Classifier:
   a. Training Accuracy: 93.03%
   b. Test Accuracy: 93.06%
   c. Mean Cross-Validated Accuracy: 90.82%
   d. Cross-Validated ROC AUC: 0.9004
   e. 

2. Gaussian Naive Bayes Classifier:
   a. Training Accuracy: 89.55%
   b. Test Accuracy: 91.67%

c. Mean Cross-Validated Accuracy: 92.29%
d. Cross-Validated ROC AUC: 0.9229

e. 

3. SVM:
   a. Training Accuracy: 91.99%
   b. Test Accuracy: 94.44%
   c. Mean Cross-Validated Accuracy: 92.21%
   d. Cross-Validated ROC AUC: 0.9091
   e. 

Based on the provided metrics, here are some considerations. The Gaussian Naive Bayes model has the highest cross-validated ROC AUC score, suggesting good model discrimination capacity. The SVM has the highest test set accuracy and a fairly good AUC score, which might indicate better generalization from the training set to unseen data. The Decision Tree has the lowest cross-validated accuracy and AUC, which might suggest it is less reliable under cross-validation compared to the other models.

If prediction reliability on unseen data is most critical, the SVM might be preferable due to its higher test accuracy and strong generalization as indicated by its AUC score. If the interpretability of the model is a priority, a Decision Tree might be favored despite slightly lower performance metrics. If the balance between false positives and false negatives is crucial, the model with the highest ROC AUC score (the Gaussian Naive Bayes) could be the best choice. However, The mean accuracy of the decision tree classifier is lower than the accuracy for the test set and train set, indicating that overfitting might exist in this model. Likewise, SVM encounters similar problems.

Given that the Gaussian Bayes Classifier has the highest mean accuracy and AUC, it may be the best overall model among the three, especially if the goal is to maximize both accuracy and the AUC score. However, if interpretability is crucial, the Decision Tree could still be a strong contender despite slightly lower performance metrics.

V. Conclusions

The result of this study suggests that features like sleep quality, occupations, and daily steps, play significant roles in differentiating between sleep disorders. This study could be taken to people for sleep self-testing. Knowing your condition before going to the hospital can also avoid wasting medical resources accordingly. In addition, if the model can be upgraded in a more comprehensive way, it can also be used as a testing tool in hospitals to assist doctors in understanding patients and prescribing medication. In addition, for pharmacies, customers can also find their own corresponding sleep disorders through this model to purchase medicines. It is even possible to set up a separate self-assessment website and combine medical resources to help people analyze their sleep conditions and provide appropriate treatment plans; for populous countries such as China, this can also further reduce the occupation of medical resources. In addition to its application in the medical field, the model can also be used in human resource management in companies, where people are offered vacations based on their current state of health and sleep.

However, it is worth noting that this study was limited by the size of the data. Consequently, additional exploration is required to evaluate the model's efficacy with more extensive datasets and its capacity to classify a broader spectrum of sleep disorders. Moreover, subsequent research should contemplate the integration of more variables, like brain tissue density and the length of time spent in a sedentary state, to potentially improve the precision of the classification models.

VI. Reference

[1] Sleep Health and Lifestyle Dataset: 2023. *https://www.kaggle.com/datasets/uom190346a/sleep-health-and-lifestyle-dataset*.

[2]Schillaci, G. and Pucci, G. 2011. The relationship between systolic and diastolic blood pressure: a clinically meaningful slope? *Hypertension Research*. 34, 11 (Sep. 2011), 1175–1178. DOI:https://doi.org/10.1038/hr.2011.161.

[3] Takeuchi, H. et al. 2018. Shorter sleep duration and better sleep quality are associated with greater tissue density in the brain. *Scientific Reports*. 8, 1 (Apr. 2018). DOI:https://doi.org/10.1038/s41598-018-24226-0.

[4] Decision-Tree Classifier tutorial: 2020. *https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial#17.-Results-and-conclusion-*.

[5] Naive Bayes classifier in Python: 2020. *https://www.kaggle.com/code/prashant111/naive-bayes-classifier-in-python#18.-ROC---AUC-*.

[6] Tutorial: K Fold Cross Validation: 2020. *https://www.kaggle.com/code/satishgunjal/tutorial-k-fold-cross-validation*.

[7] Naive Bayes classifier in Python: 2020. *https://www.kaggle.com/code/prashant111/naive-bayes-classifier-in-python#15.-Confusion-matrix-*.

[8] Support vector machines : Predicting heart disease: 2019. *https://www.kaggle.com/code/adepvenugopal/support-vector-machines-predicting-heart-disease*.

VII. Appendix

Data Processing

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import plotly.express as px
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
import warnings

# Filter out future warnings from scikit-learn
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [3]:
```
file_path = 'Sleep_health_and_lifestyle_dataset.csv'
data = pd.read_csv(file_path)
```

In [4]:
```
# Split the 'Blood Pressure' column into two columns
data[['Systolic BP', 'Diastolic BP']] = data['Blood Pressure'].str.split('/', expand=True)

# Convert the new columns to numeric type
data[['Systolic BP', 'Diastolic BP']] = data[['Systolic BP', 'Diastolic BP']].apply(pd.to_numeric)

# Drop the original 'Blood Pressure' column
data = data.drop(['Physical Activity Level','Blood Pressure', 'Stress Level', 'Diastolic BP'], axis=1)
```

In [5]:
```python
# Display the first few rows of the dataframe and its summary information
data.head(), data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Person ID         374 non-null    int64
 1   Gender            374 non-null    object
 2   Age               374 non-null    int64
 3   Occupation        374 non-null    object
 4   Sleep Duration    374 non-null    float64
 5   Quality of Sleep  374 non-null    int64
 6   BMI Category      374 non-null    object
 7   Heart Rate        374 non-null    int64
 8   Daily Steps       374 non-null    int64
 9   Sleep Disorder    374 non-null    object
 10  Systolic BP       374 non-null    int64
dtypes: float64(1), int64(6), object(4)
memory usage: 32.3+ KB
```

Out[5]:
```
(   Person ID Gender  Age            Occupation  Sleep Duration  \
 0          1   Male   27     Software Engineer             6.1
 1          2   Male   28                Doctor             6.2
 2          3   Male   28                Doctor             6.2
 3          4   Male   28  Sales Representative             5.9
 4          5   Male   28  Sales Representative             5.9

    Quality of Sleep BMI Category  Heart Rate  Daily Steps Sleep Disorder  \
 0                 6   Overweight          77         4200           None
 1                 6       Normal          75        10000           None
 2                 6       Normal          75        10000           None
 3                 4        Obese          85         3000    Sleep Apnea
 4                 4        Obese          85         3000    Sleep Apnea

    Systolic BP
 0          126
 1          125
 2          125
 3          140
 4          140  ,
 None)
```

In [6]:
```python
# Encoding categorical variables
categorical_vars = ['Gender', 'Occupation', 'BMI Category', 'Sleep Disorder']
label_encoders = {}  # Storing label encoders for potential inverse transformation

for var in categorical_vars:
    le = LabelEncoder()
    data[var] = le.fit_transform(data[var])
    label_encoders[var] = le

# Normalizing numerical features excluding 'Person ID' as it's just an identifier
numerical_vars = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Systolic BP'
                  , 'Heart Rate', 'Daily Steps']
data[numerical_vars] = StandardScaler().fit_transform(data[numerical_vars])

# Check the dataset after encoding and normalization
data
```

Out[6]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	BMI Category	Heart Rate	Daily Steps	Sleep Disorder	Systolic BP
0	1	1	-1.753096	9	-1.298887	-1.098280	3	1.654719	-1.619584	1	-0.330002
1	2	1	-1.637643	1	-1.173036	-1.098280	0	1.170474	1.970077	1	-0.459239
2	3	1	-1.637643	1	-1.173036	-1.098280	0	1.170474	1.970077	1	-0.459239
3	4	1	-1.637643	6	-1.550588	-2.771424	2	3.591698	-2.362273	2	1.479309
4	5	1	-1.637643	6	-1.550588	-2.771424	2	3.591698	-2.362273	2	1.479309
...	...	...	...	...	...	...	...	...	...	...	...
369	370	0	1.941401	5	1.218127	1.411435	3	-0.524383	0.113356	2	1.479309
370	371	0	1.941401	5	1.092276	1.411435	3	-0.524383	0.113356	2	1.479309
371	372	0	1.941401	5	1.218127	1.411435	3	-0.524383	0.113356	2	1.479309
372	373	0	1.941401	5	1.218127	1.411435	3	-0.524383	0.113356	2	1.479309
373	374	0	1.941401	5	1.218127	1.411435	3	-0.524383	0.113356	2	1.479309

374 rows × 11 columns

In [7]:
```
data.head(), data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Person ID         374 non-null    int64  
 1   Gender            374 non-null    int64  
 2   Age               374 non-null    float64
 3   Occupation        374 non-null    int64  
 4   Sleep Duration    374 non-null    float64
 5   Quality of Sleep  374 non-null    float64
 6   BMI Category      374 non-null    int64  
 7   Heart Rate        374 non-null    float64
 8   Daily Steps       374 non-null    float64
 9   Sleep Disorder    374 non-null    int64  
 10  Systolic BP       374 non-null    float64
dtypes: float64(6), int64(5)
memory usage: 32.3 KB
```

Out[7]:
```
(   Person ID  Gender       Age  Occupation  Sleep Duration  Quality of Sleep  \
 0          1       1 -1.753096           9       -1.298887         -1.098280   
 1          2       1 -1.637643           1       -1.173036         -1.098280   
 2          3       1 -1.637643           1       -1.173036         -1.098280   
 3          4       1 -1.637643           6       -1.550588         -2.771424   
 4          5       1 -1.637643           6       -1.550588         -2.771424   

    BMI Category  Heart Rate  Daily Steps  Sleep Disorder  Systolic BP  
 0             3    1.654719    -1.619584               1    -0.330002  
 1             0    1.170474     1.970077               1    -0.459239  
 2             0    1.170474     1.970077               1    -0.459239  
 3             2    3.591698    -2.362273               2     1.479309  
 4             2    3.591698    -2.362273               2     1.479309  ,
 None)
```

In [8]:
```python
# Define a function to remove outliers based on IQR
def remove_outliers(data, column_names):
    clean_df = data.copy()
    for column in column_names:
        Q1 = clean_df[column].quantile(0.25)
        Q3 = clean_df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Filter out the outliers
        clean_df = clean_df[(clean_df[column] >= lower_bound) & (clean_df[column] <= upper_bound)]
    return clean_df

# Numerical columns to check for outliers
numerical_columns = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Systolic BP', 'Heart Rate', 'Daily Steps']

# Remove outliers
cleaned_data = remove_outliers(data, numerical_columns)

# Display the shape of the data before and after removing outliers
original_shape = data.shape
cleaned_shape = cleaned_data.shape

original_shape, cleaned_shape

cleaned_data
```

Out[8]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	BMI Category	Heart Rate	Daily Steps	Sleep Disorder	Systolic BP
0	1	1	-1.753096	9	-1.298887	-1.098280	3	1.654719	-1.619584	1	-0.330002
1	2	1	-1.637643	1	-1.173036	-1.098280	0	1.170474	1.970077	1	-0.459239
2	3	1	-1.637643	1	-1.173036	-1.098280	0	1.170474	1.970077	1	-0.459239
7	8	1	-1.522190	1	0.840575	-0.261708	0	-0.040138	0.732263	1	-1.105421
8	9	1	-1.522190	1	0.840575	-0.261708	0	-0.040138	0.732263	1	-1.105421
...	...	...	...	...	...	...	...	...	...	...	...
369	370	0	1.941401	5	1.218127	1.411435	3	-0.524383	0.113356	2	1.479309
370	371	0	1.941401	5	1.092276	1.411435	3	-0.524383	0.113356	2	1.479309
371	372	0	1.941401	5	1.218127	1.411435	3	-0.524383	0.113356	2	1.479309
372	373	0	1.941401	5	1.218127	1.411435	3	-0.524383	0.113356	2	1.479309
373	374	0	1.941401	5	1.218127	1.411435	3	-0.524383	0.113356	2	1.479309

359 rows × 11 columns

EDA

In [9]:
```
file_path = 'Sleep_health_and_lifestyle_dataset.csv'
data = pd.read_csv(file_path)
```

In [10]:
```
# Split the 'Blood Pressure' column into two columns
data[['Systolic BP', 'Diastolic BP']] = data['Blood Pressure'].str.split('/', expand=True)

# Convert the new columns to numeric type
data[['Systolic BP', 'Diastolic BP']] = data[['Systolic BP', 'Diastolic BP']].apply(pd.to_numeric)

# Drop the original 'Blood Pressure' column
data = data.drop('Blood Pressure', axis=1)
```

In [11]:
```
# Display the first few rows of the dataframe and its summary information
data.head(), data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Person ID                374 non-null    int64  
 1   Gender                   374 non-null    object 
 2   Age                      374 non-null    int64  
 3   Occupation               374 non-null    object 
 4   Sleep Duration           374 non-null    float64
 5   Quality of Sleep         374 non-null    int64  
 6   Physical Activity Level  374 non-null    int64  
 7   Stress Level             374 non-null    int64  
 8   BMI Category             374 non-null    object 
 9   Heart Rate               374 non-null    int64  
 10  Daily Steps              374 non-null    int64  
 11  Sleep Disorder           374 non-null    object 
 12  Systolic BP              374 non-null    int64  
 13  Diastolic BP             374 non-null    int64  
dtypes: float64(1), int64(9), object(4)
memory usage: 41.0+ KB
```

Out[11]:
```
(   Person ID Gender  Age            Occupation  Sleep Duration  \
 0          1   Male   27     Software Engineer             6.1   
 1          2   Male   28                Doctor             6.2   
 2          3   Male   28                Doctor             6.2   
 3          4   Male   28  Sales Representative             5.9   
 4          5   Male   28  Sales Representative             5.9   

    Quality of Sleep  Physical Activity Level  Stress Level BMI Category  \
 0                 6                       42             6   Overweight   
 1                 6                       60             8       Normal   
 2                 6                       60             8       Normal   
 3                 4                       30             8        Obese   
 4                 4                       30             8        Obese   

    Heart Rate  Daily Steps Sleep Disorder  Systolic BP  Diastolic BP  
 0          77         4200           None          126            83  
 1          75        10000           None          125            80  
 2          75        10000           None          125            80  
 3          85         3000    Sleep Apnea          140            90  
 4          85         3000    Sleep Apnea          140            90  ,
 None)
```

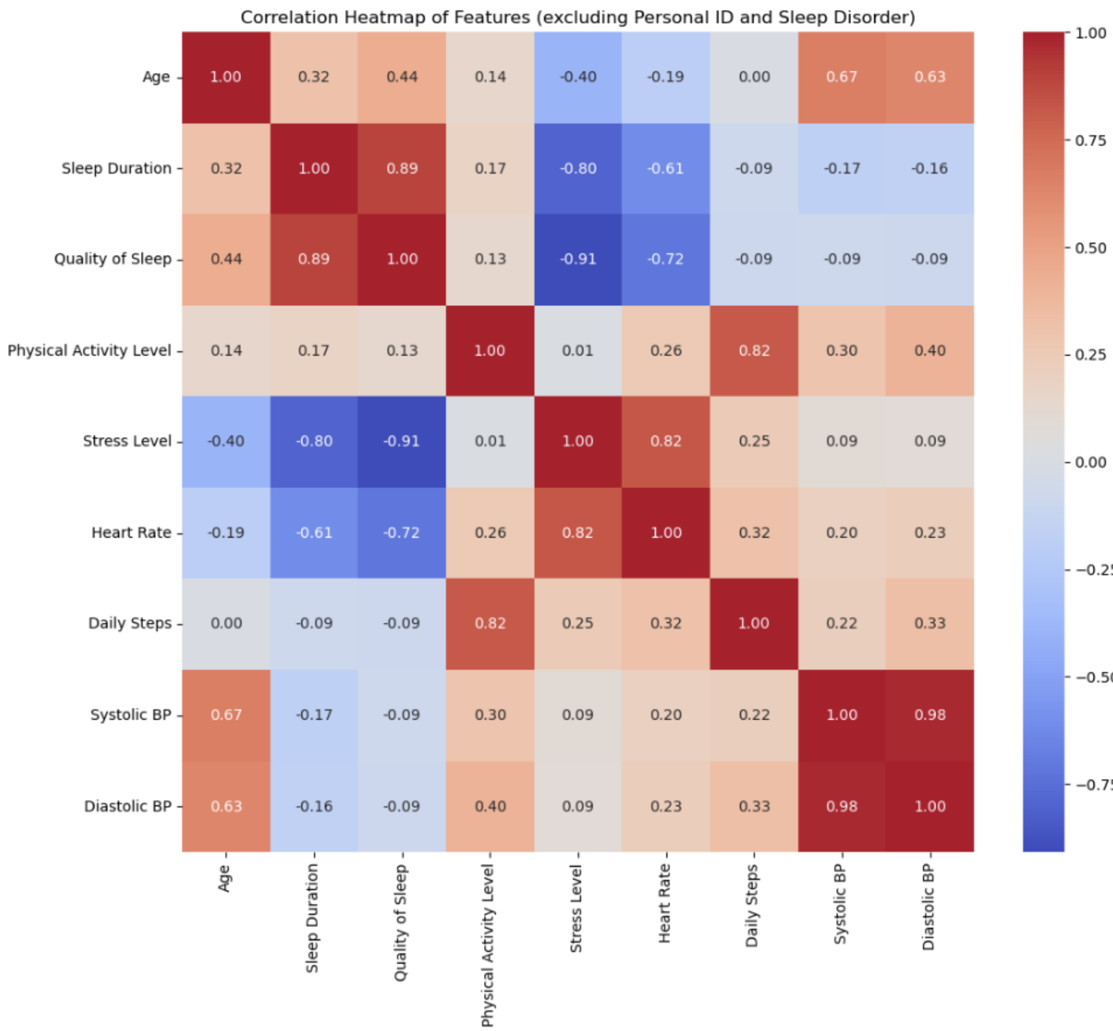
In [12]:
```python
# Define a function to remove outliers based on IQR
def remove_outliers(data, column_names):
    clean_df = data.copy()
    for column in column_names:
        Q1 = clean_df[column].quantile(0.25)
        Q3 = clean_df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Filter out the outliers
        clean_df = clean_df[(clean_df[column] >= lower_bound) & (clean_df[column] <= upper_bound)]
    return clean_df

# Numerical columns to check for outliers
numerical_columns = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level', 'Stress Level', 'Systol

# Remove outliers
cleaned_data_for_EDA = remove_outliers(data, numerical_columns)

# Display the shape of the data before and after removing outliers
original_shape = data.shape
cleaned_shape = cleaned_data.shape

original_shape, cleaned_shape

cleaned_data_for_EDA
```

Out[12]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Heart Rate	Daily Steps	Sleep Disorder	Systolic BP	Diastolic BP
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	77	4200	None	126	83
1	2	Male	28	Doctor	6.2	6	60	8	Normal	75	10000	None	125	80
2	3	Male	28	Doctor	6.2	6	60	8	Normal	75	10000	None	125	80
7	8	Male	29	Doctor	7.8	7	75	6	Normal	70	8000	None	120	80
8	9	Male	29	Doctor	7.8	7	75	6	Normal	70	8000	None	120	80
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
369	370	Female	59	Nurse	8.1	9	75	3	Overweight	68	7000	Sleep Apnea	140	95
370	371	Female	59	Nurse	8.0	9	75	3	Overweight	68	7000	Sleep Apnea	140	95
371	372	Female	59	Nurse	8.1	9	75	3	Overweight	68	7000	Sleep Apnea	140	95
372	373	Female	59	Nurse	8.1	9	75	3	Overweight	68	7000	Sleep Apnea	140	95
373	374	Female	59	Nurse	8.1	9	75	3	Overweight	68	7000	Sleep Apnea	140	95

359 rows × 14 columns

In [17]:
```python
# Dropping 'Person ID' and the target variable 'Sleep Disorder' for the correlation heatmap
data_for_correlation = cleaned_data_for_EDA.drop(['Person ID', 'Sleep Disorder'], axis=1)

# Calculating the correlation matrix
corr_matrix = data_for_correlation.corr()

# Generating a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap of Features (excluding Personal ID and Sleep Disorder)')
plt.show()
```



Preprocessing

In [19]:
```
#Declare target variable
classes = cleaned_data['Sleep Disorder']
#Drop personal_Id and stleep_disorder
data_drop_all = cleaned_data.drop(['Person ID', 'Sleep Disorder'], axis=1)

data_drop_all.head()
```

Out[19]:

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	BMI Category	Heart Rate	Daily Steps	Systolic BP
0	1	-1.753096	9	-1.298887	-1.098280	3	1.654719	-1.619584	-0.330002
1	1	-1.637643	1	-1.173036	-1.098280	0	1.170474	1.970077	-0.459239
2	1	-1.637643	1	-1.173036	-1.098280	0	1.170474	1.970077	-0.459239
7	1	-1.522190	1	0.840575	-0.261708	0	-0.040138	0.732263	-1.105421
8	1	-1.522190	1	0.840575	-0.261708	0	-0.040138	0.732263	-1.105421

In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data_drop_all, classes, test_size = 0.2, random_state = 42)
# check the shape of X_train and X_test
X_train.shape, X_test.shape
```

Out[21]:
```
((287, 9), (72, 9))
```

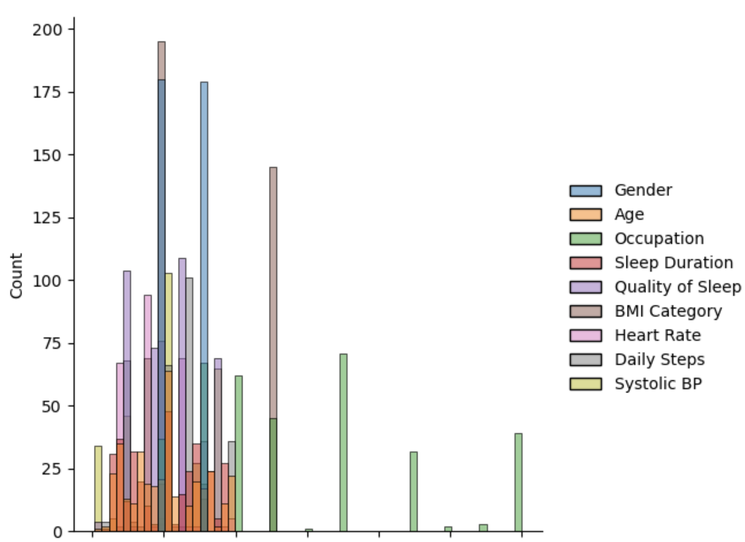
In [22]:
```
# check data types in X_train
X_train.dtypes
```

Out[22]:
```
Gender                int64
Age                 float64
Occupation            int64
Sleep Duration      float64
Quality of Sleep    float64
BMI Category          int64
Heart Rate          float64
Daily Steps         float64
Systolic BP         float64
dtype: object
```

In [23]:
```
#Feature Scaling
cols = X_train.columns
cols
```

Out[23]:
```
Index(['Gender', 'Age', 'Occupation', 'Sleep Duration', 'Quality of Sleep',
       'BMI Category', 'Heart Rate', 'Daily Steps', 'Systolic BP'],
      dtype='object')
```

In [24]:
```
sns.displot(data_drop_all)
plt.show()
```



In [25]:

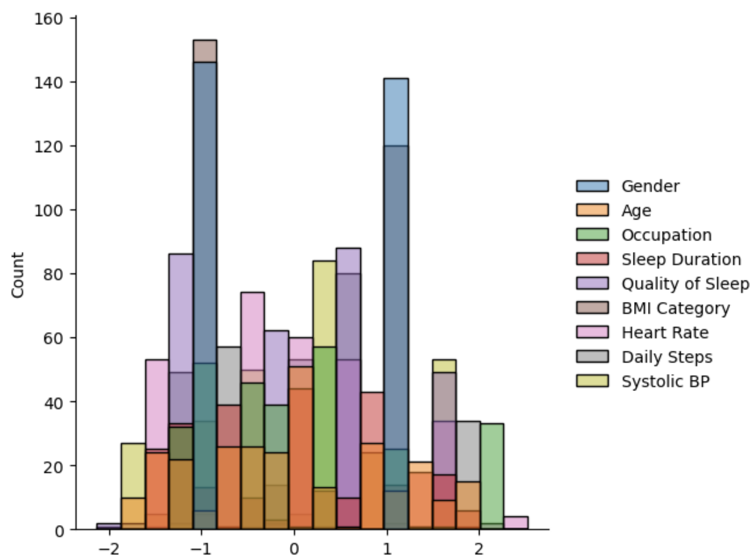
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
#X_train = np.log(X_train)
X_test = scaler.transform(X_test)
#X_test = np.log(X_test)
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
X_train.head(10)
X_train.describe()
```

Out[25]:

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	BMI Category	Heart Rate	Daily Steps	Systolic BP
count	2.870000e+02	2.870000e+02	2.870000e+02	2.870000e+02	2.870000e+02	2.870000e+02	2.870000e+02	2.870000e+02	2.870000e+02
mean	-1.547349e-18	-1.353931e-17	5.792888e-17	3.984424e-17	-4.371261e-17	5.531773e-17	6.421499e-17	1.083144e-17	1.769781e-17
std	1.001747e+00	1.001747e+00	1.001747e+00	1.001747e+00	1.001747e+00	1.001747e+00	1.001747e+00	1.001747e+00	1.001747e+00
min	-9.827275e-01	-1.852758e+00	-1.219172e+00	-1.559073e+00	-2.126502e+00	-8.963337e-01	-1.437537e+00	-1.904373e+00	-1.743078e+00
25%	-9.827275e-01	-6.597100e-01	-8.927709e-01	-7.882652e-01	-1.215597e+00	-8.963337e-01	-5.240908e-01	-6.695874e-01	-4.443672e-01
50%	-9.827275e-01	5.611899e-02	-2.399677e-01	1.110107e-01	-3.046928e-01	-8.963337e-01	8.487301e-02	-1.970041e-02	2.049881e-01
75%	1.017576e+00	8.912528e-01	4.128355e-01	8.818185e-01	6.062118e-01	1.167151e+00	6.938368e-01	6.301866e-01	8.543434e-01
max	1.017576e+00	1.964996e+00	2.044843e+00	1.781094e+00	1.517116e+00	1.167151e+00	2.520728e+00	1.929960e+00	1.503699e+00

In [26]:

```
sns.displot(X_train)
plt.show()
```



Decision Tree

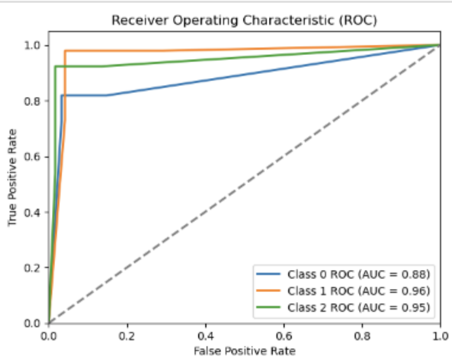
In [27]:
```python
from sklearn.tree import DecisionTreeClassifier
# generate decision tree classifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
# print the scores on training and test set
print('Training set score: {:.4f}'.format(dt.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(dt.score(X_test, y_test)))
```

```
Training set score: 0.9303
Test set score: 0.9444
```

In [28]:
```python
from sklearn.model_selection import cross_val_score

# Calculate the ROC-AUC score
Cross_Val_Score = cross_val_score(dt, X_train, y_train, cv=5, scoring='roc_auc_ovr').mean() # Use 'micro' or 'weight

print(f"Cross validated ROC AUC: {Cross_Val_Score}")
```

```
Cross validated ROC AUC: 0.8968181963556843
```

In [29]:
```python
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Instantiate the label binarizer and fit it to the true class labels
label_binarizer = LabelBinarizer().fit(y_train)
# Binarize the labels for computing ROC curve and AUC score
y_test_bin = label_binarizer.transform(y_test)
y_pred_prob = dt.predict_proba(X_test)  # Get the predicted probabilities

# Get the number of classes from the label binarizer
n_classes = label_binarizer.classes_.size

# Compute ROC curve and AUC score for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves
plt.figure()
lw = 2
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=lw, label='Class %0.0f ROC (AUC = %0.2f)' % (i, roc_auc[i]))
plt.plot([0, 1], [0, 1], color='gray', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

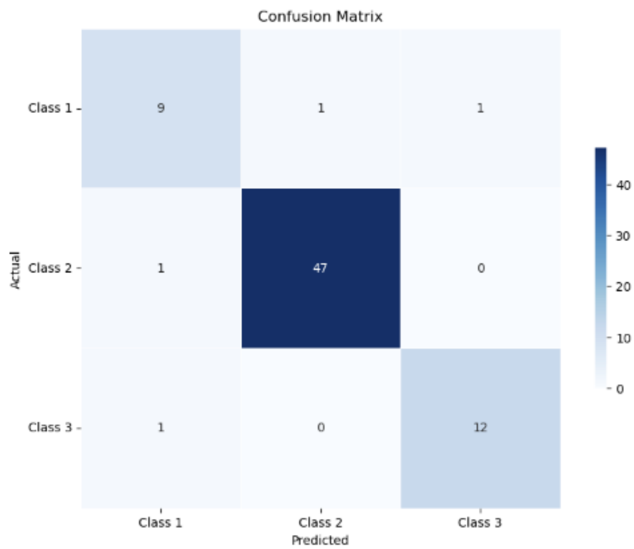


In [30]:
```python
# Print the Confusion Matrix and slice it into four pieces
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

# Print the Confusion Matrix
print('Confusion matrix:\n', cm)

# If you want to print the confusion matrix for each class
for i in range(len(cm)):
    print(f"\nConfusion matrix for class {i+1}:")
    print(f"True Positives (TP) for class {i+1}: {cm[i, i]}")
    FP = cm[:, i].sum() - cm[i, i]
    print(f"False Positives (FP) for class {i+1}: {FP}")
    FN = cm[i, :].sum() - cm[i, i]
    print(f"False Negatives (FN) for class {i+1}: {FN}")
    TN = cm.sum() - (FP + FN + cm[i, i])
    print(f"True Negatives (TN) for class {i+1}: {TN}")
```

```
Confusion matrix:
 [[ 9  1  1]
 [ 1 47  0]
 [ 1  0 12]]

Confusion matrix for class 1:
True Positives (TP) for class 1: 9
False Positives (FP) for class 1: 2
False Negatives (FN) for class 1: 2
True Negatives (TN) for class 1: 59

Confusion matrix for class 2:
True Positives (TP) for class 2: 47
False Positives (FP) for class 2: 1
False Negatives (FN) for class 2: 1
True Negatives (TN) for class 2: 23

Confusion matrix for class 3:
True Positives (TP) for class 3: 12
False Positives (FP) for class 3: 1
```

True Negatives (TN) for Class 3: 58

In [31]:
```
# Set up the matplotlib figure
plt.figure(figsize=(10, 7))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', square=True, linewidths=.5,
            cbar_kws={"shrink": .5})

# Add labels to the heatmap
class_names = ['Class 1', 'Class 2', 'Class 3'] # Adjust these to your actual class names
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.xticks(ticks=np.arange(len(class_names)) + 0.5, labels=class_names)
plt.yticks(ticks=np.arange(len(class_names)) + 0.5, labels=class_names, rotation=0)

# Show the plot
plt.show()
```



In [32]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        11
           1       0.98      0.98      0.98        48
           2       0.92      0.92      0.92        13

    accuracy                           0.94        72
   macro avg       0.91      0.91      0.91        72
weighted avg       0.94      0.94      0.94        72
```

In [33]:
```
from sklearn.model_selection import cross_val_score, KFold

# Create k-fold cross-validation object
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Perform k-fold cross-validation
scores = cross_val_score(dt, data_drop_all, classes, cv=kfold)

# Print accuracy scores for each fold and the mean score
print("Accuracy scores:", scores)
print("Mean accuracy:", scores.mean())
```

```
Accuracy scores: [0.94444444 0.91666667 0.91666667 0.77777778 0.97222222 0.97222222
 0.88888889 0.94444444 0.75       0.94285714]
Mean accuracy: 0.9026190476190475
```

Naive Bayes Classifier

In [34]:
```
# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB
# instantiate the model
gnb = GaussianNB()
# fit the model
gnb.fit(X_train, y_train)
y_pred_gnb = gnb.predict(X_test)
from sklearn.metrics import accuracy_score
y_pred_train_gnb = gnb.predict(X_train)
# print the scores on training and test set
print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))
```

```
Training set score: 0.8955
Test set score: 0.9167
```

In [35]:
```
print(len(X_train))
print(len(X_test))
```

```
287
72
```

In [36]:
```
# check class distribution in test set
y_test.value_counts()
```

Out[36]:
```
1    48
2    13
0    11
Name: Sleep Disorder, dtype: int64
```

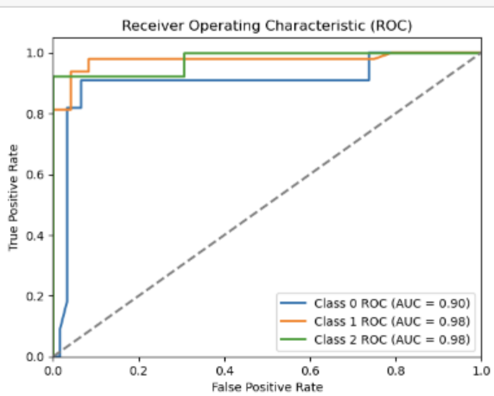
In [37]:
```
# check null accuracy score

null_accuracy = (71/(71+20+17))

print('Null accuracy score: {:.4f}'.format(null_accuracy))
```

```
Null accuracy score: 0.6574
```

In [38]:
```
from sklearn.model_selection import cross_val_score

# Calculate the ROC-AUC score
Cross_Val_Score = cross_val_score(gnb, X_train, y_train, cv=5, scoring='roc_auc_ovr').mean()
# Use 'micro' or 'weighted' for alternative aggregation methods

print(f"Cross validated ROC AUC: {Cross_Val_Score}")
```

```
Cross validated ROC AUC: 0.9228978483004349
```

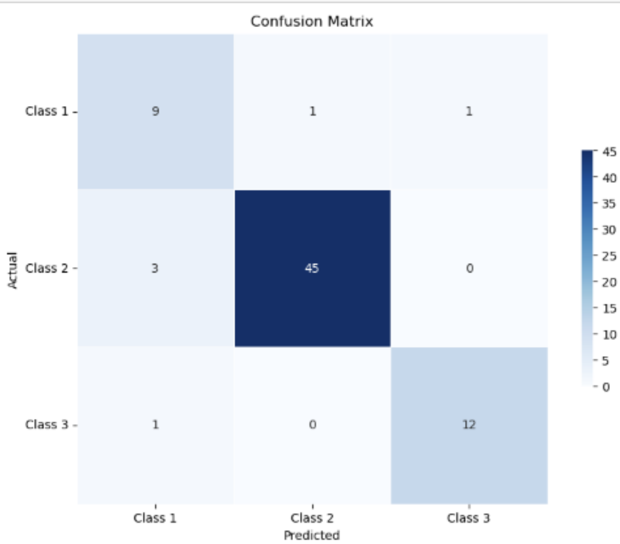
In [39]:
```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Instantiate the label binarizer and fit it to the true class labels
label_binarizer = LabelBinarizer().fit(y_train)
# Binarize the labels for computing ROC curve and AUC score
y_test_bin = label_binarizer.transform(y_test)
y_pred_prob = gnb.predict_proba(X_test)  # Get the predicted probabilities

# Get the number of classes from the label binarizer
n_classes = label_binarizer.classes_.size

# Compute ROC curve and AUC score for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves
plt.figure()
lw = 2
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=lw, label='Class %0.0f ROC (AUC = %0.2f)' % (i, roc_auc[i]))
plt.plot([0, 1], [0, 1], color='gray', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```



In [40]:
```python
# Print the Confusion Matrix and slice it into four pieces
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_gnb)

# Print the Confusion Matrix
print('Confusion matrix:\n', cm)

# If you want to print the confusion matrix for each class
for i in range(len(cm)):
    print(f"\nConfusion matrix for class {i+1}:")
    print(f"True Positives (TP) for class {i+1}: {cm[i, i]}")
    FP = cm[:, i].sum() - cm[i, i]
    print(f"False Positives (FP) for class {i+1}: {FP}")
    FN = cm[i, :].sum() - cm[i, i]
    print(f"False Negatives (FN) for class {i+1}: {FN}")
    TN = cm.sum() - (FP + FN + cm[i, i])
    print(f"True Negatives (TN) for class {i+1}: {TN}")
```

```
Confusion matrix:
 [[ 9  1  1]
 [ 3 45  0]
 [ 1  0 12]]

Confusion matrix for class 1:
True Positives (TP) for class 1: 9
False Positives (FP) for class 1: 4
False Negatives (FN) for class 1: 2
True Negatives (TN) for class 1: 57

Confusion matrix for class 2:
True Positives (TP) for class 2: 45
False Positives (FP) for class 2: 1
False Negatives (FN) for class 2: 3
True Negatives (TN) for class 2: 23

Confusion matrix for class 3:
True Positives (TP) for class 3: 12
False Positives (FP) for class 3: 1
False Negatives (FN) for class 3: 1
True Negatives (TN) for class 3: 58
```

In [41]:
```python
# Set up the matplotlib figure
plt.figure(figsize=(10, 7))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', square=True, linewidths=.5,
            cbar_kws={"shrink": .5})

# Add labels to the heatmap
class_names = ['Class 1', 'Class 2', 'Class 3'] # Adjust these to your actual class names
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.xticks(ticks=np.arange(len(class_names)) + 0.5, labels=class_names)
plt.yticks(ticks=np.arange(len(class_names)) + 0.5, labels=class_names, rotation=0)

# Show the plot
plt.show()
```



In [42]:
```python
from sklearn.model_selection import cross_val_score, KFold

# Create k-fold cross-validation object
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Perform k-fold cross-validation
scores = cross_val_score(gnb, data_drop_all, classes, cv=kfold)

# Print accuracy scores for each fold and the mean score
print("Accuracy scores:", scores)
print("Mean accuracy:", scores.mean())
```

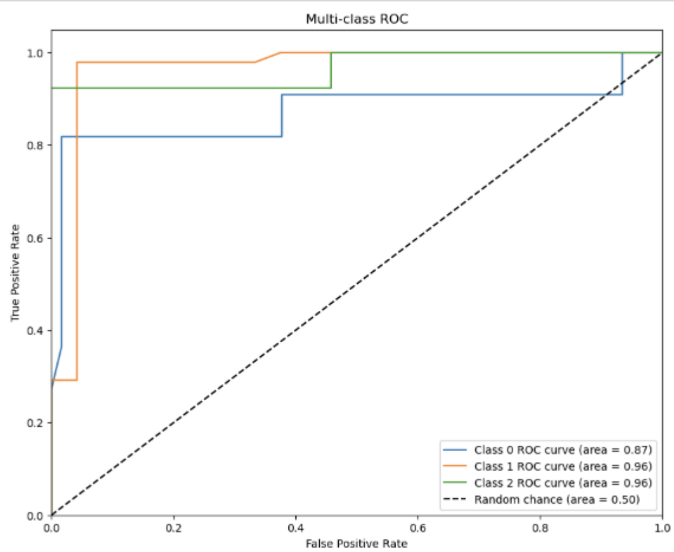
```
Accuracy scores: [0.94444444 0.88888889 0.86111111 0.88888889 0.97222222 0.94444444
 0.88888889 0.86111111 0.80555556 0.91428571]
Mean accuracy: 0.896984126984127
```

Support Vector Machines

In [43]:
```
from sklearn import svm #Import svm model
from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.metrics import confusion_matrix

# Instantiate the SVM classifier and fit it to the training data
# Use CalibratedClassifierCV to get the probability-like outputs from decision_function
svm = SVC(kernel='linear', probability=True, random_state=42)  # Linear kernel used as an example
clf = CalibratedClassifierCV(svm)  # Calibrate the SVM
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Training set score: 0.9199
Test set score: 0.9444
```

In [44]:
```
from sklearn.svm import SVC
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# Get the predicted probabilities for the test set
y_pred_prob = clf.predict_proba(X_test)

# Binarize the true class labels for the test set
y_test_bin = label_binarize(y_test, classes=np.unique(classes))

# Calculate the ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
n_classes = y_test_bin.shape[1]
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(10, 8))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label='Class {0} ROC curve (area = {1:0.2f})'
                                   ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', label='Random chance (area = 0.50)')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multi-class ROC')
plt.legend(loc="lower right")
plt.show()
```



In [45]:
```
from sklearn.model_selection import cross_val_score

# Calculate the ROC-AUC score
Cross_Val_Score = cross_val_score(svm, X_train, y_train, cv=5, scoring='roc_auc_ovr').mean() # Use 'micro' or 'weigh

print(f"Cross validated ROC AUC: {Cross_Val_Score}")
```

```
Cross validated ROC AUC: 0.9090830828978603
```

In [46]:
```
from sklearn.model_selection import cross_val_score, KFold

# Create k-fold cross-validation object
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
#scores = cross_val_score(clf, spotify_drop_all, classes, cv=5, scoring='roc_auc_ovr').mean()
#print(scores)
# Perform k-fold cross-validation
scores = cross_val_score(clf, data_drop_all, classes, cv=kfold)
Cross_validated_ROC_AUC = cross_val_score(svm, X_train, y_train, cv=5, scoring='roc_auc_ovr').mean()

# Print accuracy scores for each fold and the mean score
print("Accuracy scores:", scores)
print("Mean accuracy:", scores.mean())
```

```
Accuracy scores: [0.94444444 0.94444444 0.88888889 0.88888889 1.         0.97222222
 0.88888889 0.94444444 0.77777778 0.97142857]
Mean accuracy: 0.9221428571428572
```