In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(color_codes=True)
```

In [2]:
```
df = pd.read_csv('Sleep_health_and_lifestyle_dataset.csv')
df.head()
```

Out[2]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate	Daily Steps	Sleep Disorder
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77	4200	None
1	2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None
2	3	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85	3000	Sleep Apnea
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85	3000	Sleep Apnea

Data Preprocessing Part 1

In [3]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[3]:
```
Gender             2
Occupation        11
BMI Category       4
Blood Pressure    25
Sleep Disorder     3
dtype: int64
```

In [4]:
```
# Drop identifier column like 'Person ID'
df.drop(columns='Person ID', inplace=True)
df.head()
```

Out[4]:

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate	Daily Steps	Sleep Disorder
0	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77	4200	None
1	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None
2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None
3	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85	3000	Sleep Apnea
4	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85	3000	Sleep Apnea

Split Blood Pressure Column into 2 column

In [5]:
```
# Split the 'Blood Pressure' column into two separate columns
df[['Blood Pressure 1', 'Blood Pressure 2']] = df['Blood Pressure'].str.split('/', expand=True).astype(int)

df.head()
```

Out[5]:

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate	Daily Steps	Sleep Disorder	Blood Pressure 1	Blood Pressure 2
0	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77	4200	None	126	83
1	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None	125	80
2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None	125	80
3	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85	3000	Sleep Apnea	140	90
4	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85	3000	Sleep Apnea	140	90

In [6]:
```
# Drop Blood Pressure Column
df.drop(columns='Blood Pressure', inplace=True)
df.head()
```

Out[6]:

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Heart Rate	Daily Steps	Sleep Disorder	Blood Pressure 1	Blood Pressure 2
0	Male	27	Software Engineer	6.1	6	42	6	Overweight	77	4200	None	126	83
1	Male	28	Doctor	6.2	6	60	8	Normal	75	10000	None	125	80
2	Male	28	Doctor	6.2	6	60	8	Normal	75	10000	None	125	80
3	Male	28	Sales Representative	5.9	4	30	8	Obese	85	3000	Sleep Apnea	140	90
4	Male	28	Sales Representative	5.9	4	30	8	Obese	85	3000	Sleep Apnea	140	90

Check BMI Category Unique Value

In [8]:
```
df['BMI Category'].unique()
```

Out[8]:
```
array(['Overweight', 'Normal', 'Obese', 'Normal Weight'], dtype=object)
```

In [9]:
```
# Replace 'Normal Weight' with 'Normal' in 'BMI Category' column
df['BMI Category'] = df['BMI Category'].replace('Normal Weight', 'Normal')
df['BMI Category'].unique()
```

Out[9]:
```
array(['Overweight', 'Normal', 'Obese'], dtype=object)
```

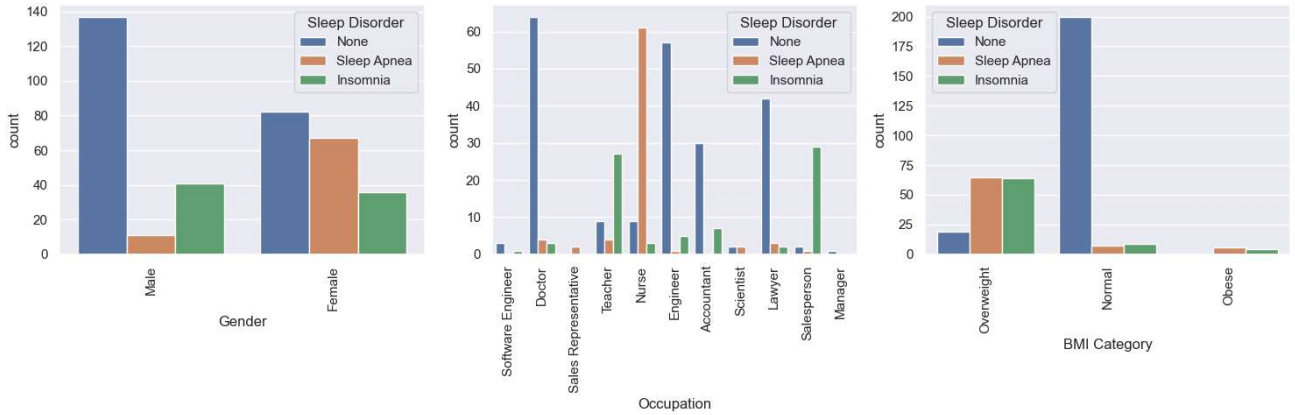
Check Occupation Unique Value

In [10]:
```
df['Occupation'].unique()
```

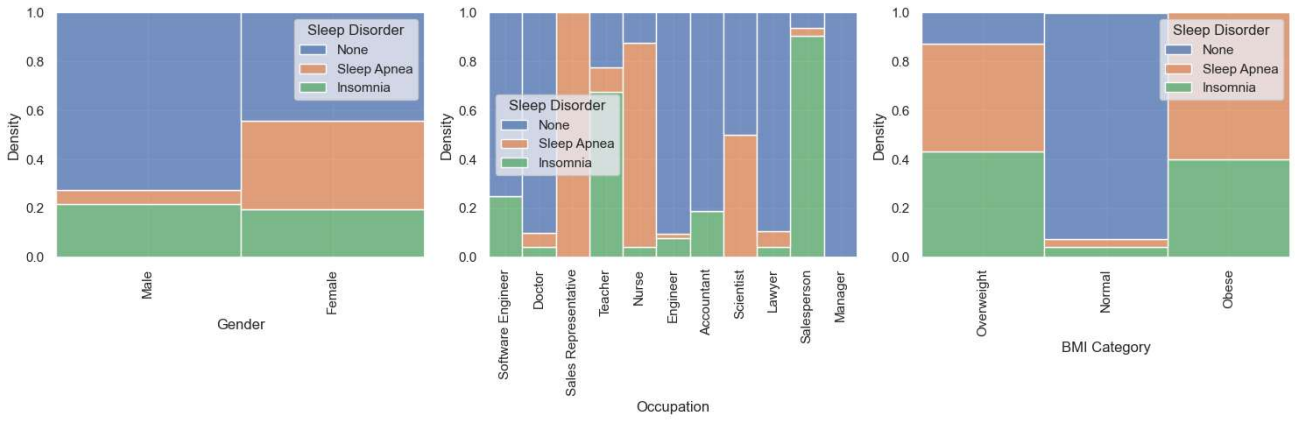
Out[10]:
```
array(['Software Engineer', 'Doctor', 'Sales Representative', 'Teacher',
       'Nurse', 'Engineer', 'Accountant', 'Scientist', 'Lawyer',
       'Salesperson', 'Manager'], dtype=object)
```

Exploratory Data Analysis

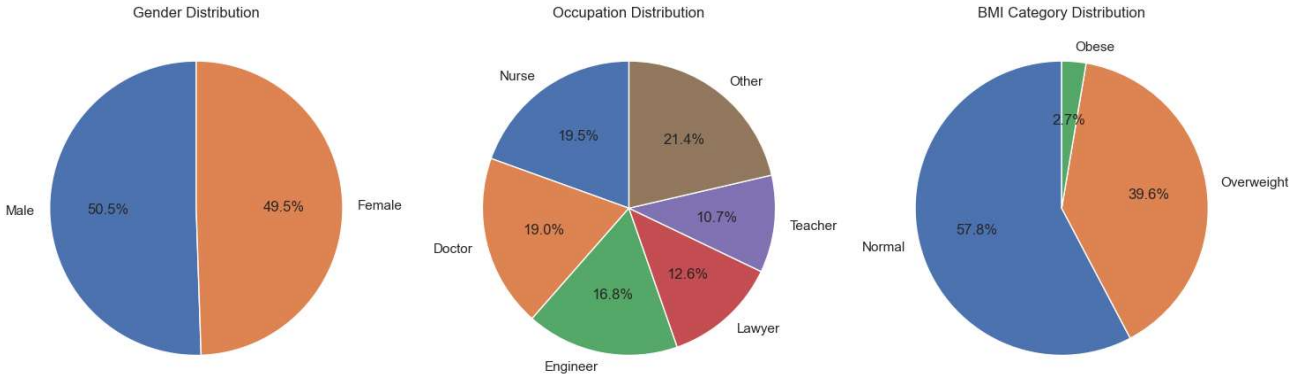
In [12]:
```
# list of categorical variables to plot
cat_vars = ['Gender', 'Occupation', 'BMI Category']

# create figure with subplots
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='Sleep Disorder', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



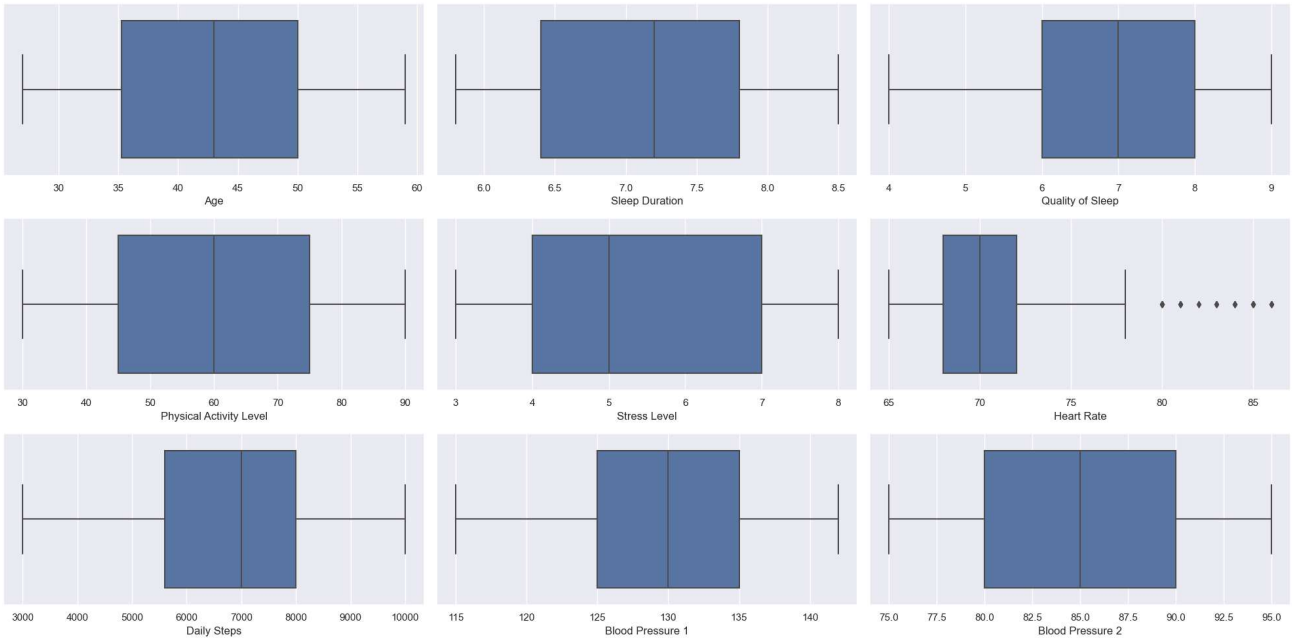
In [13]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Gender', 'Occupation', 'BMI Category']

# create figure with subplots
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
axs = axs.flatten()

# create histplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='Sleep Disorder', data=df, ax=axs[i], multiple="fill", kde=False, element="bars", fill=Tr
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



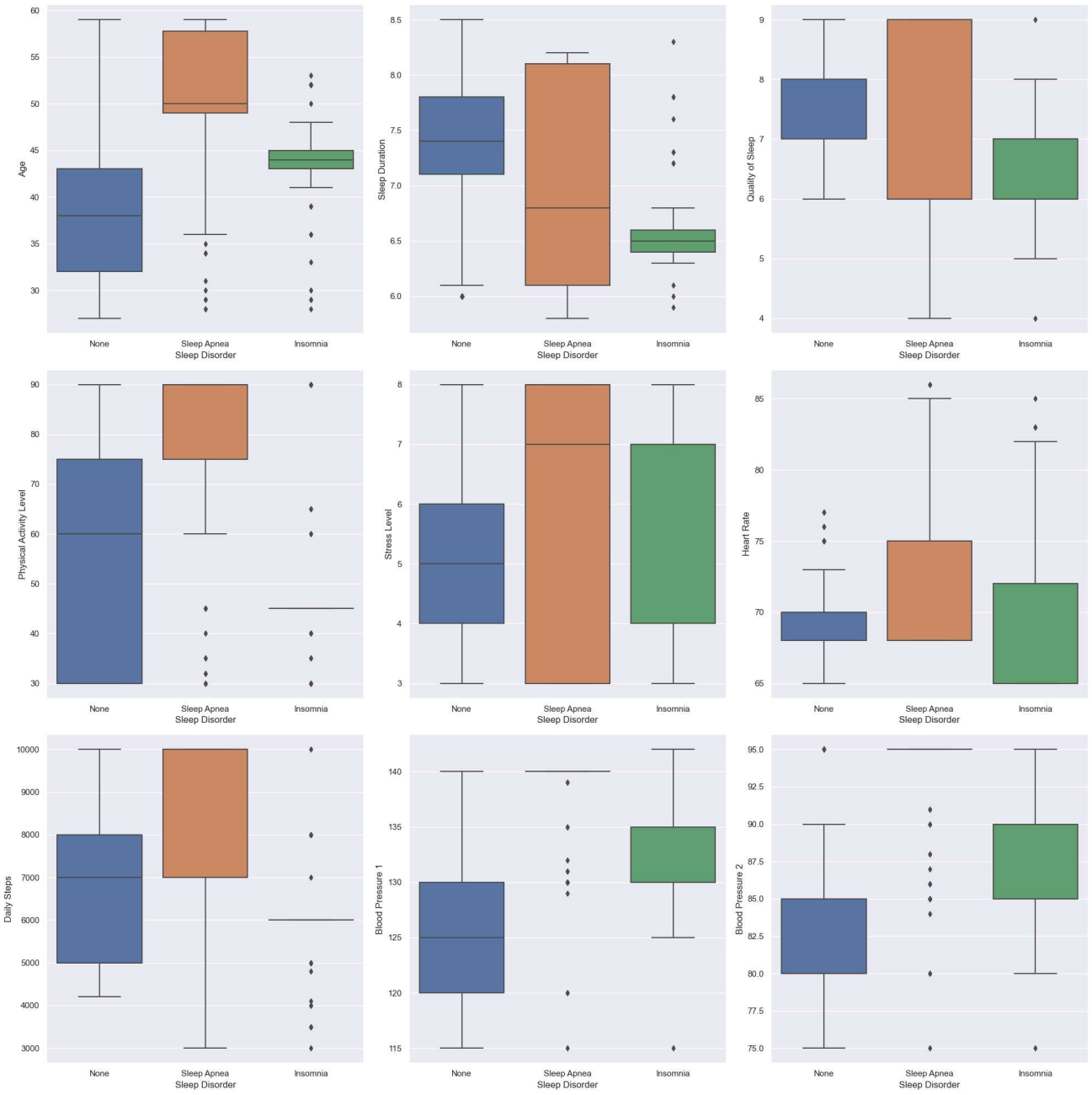
In [15]:
```
# Specify the maximum number of categories to show individually
max_categories = 5

cat_vars = ['Gender', 'Occupation', 'BMI Category']

# Create a figure and axes
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 15))

# Create a pie chart for each categorical variable
for i, var in enumerate(cat_vars):
    if i < len(axs.flat):
        # Count the number of occurrences for each category
        cat_counts = df[var].value_counts()

        # Group categories beyond the top max_categories as 'Other'
        if len(cat_counts) > max_categories:
            cat_counts_top = cat_counts[:max_categories]
            cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), index=['Other'])
            cat_counts = cat_counts_top.append(cat_counts_other)

        # Create a pie chart
        axs.flat[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)

        # Set a title for each subplot
        axs.flat[i].set_title(f'{var} Distribution')

# Adjust spacing between subplots
fig.tight_layout()

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



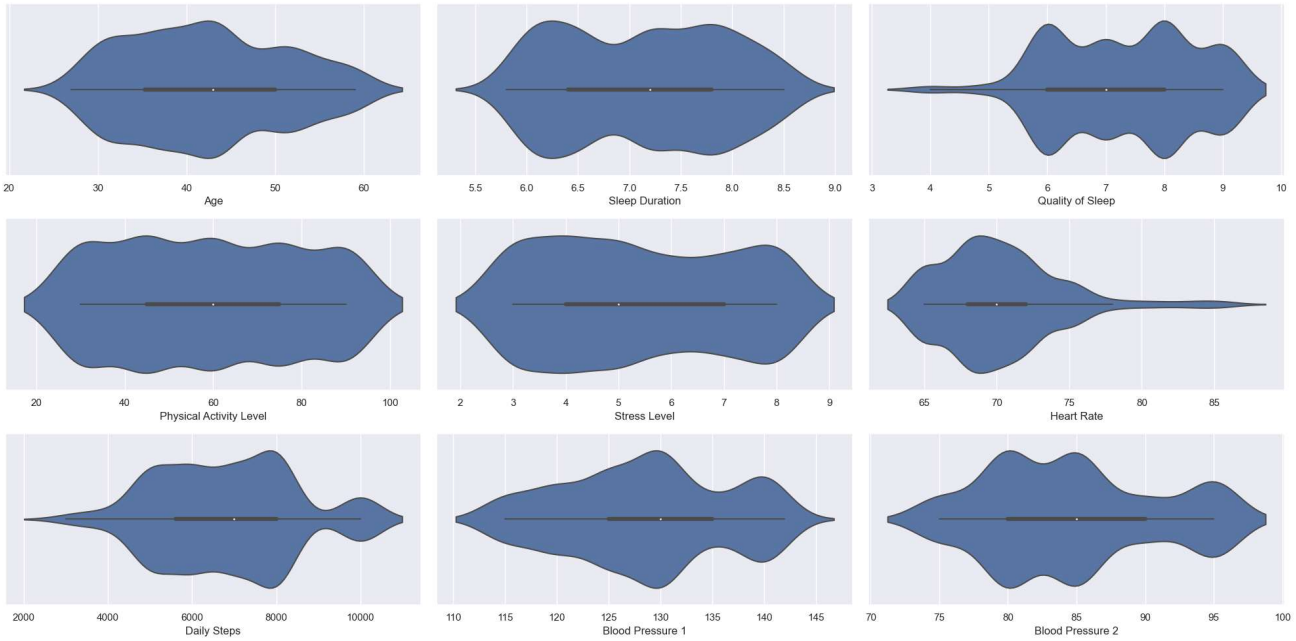
In [16]:
```
num_vars = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level',
            'Stress Level','Heart Rate', 'Daily Steps', 'Blood Pressure 1', 'Blood Pressure 2']

fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



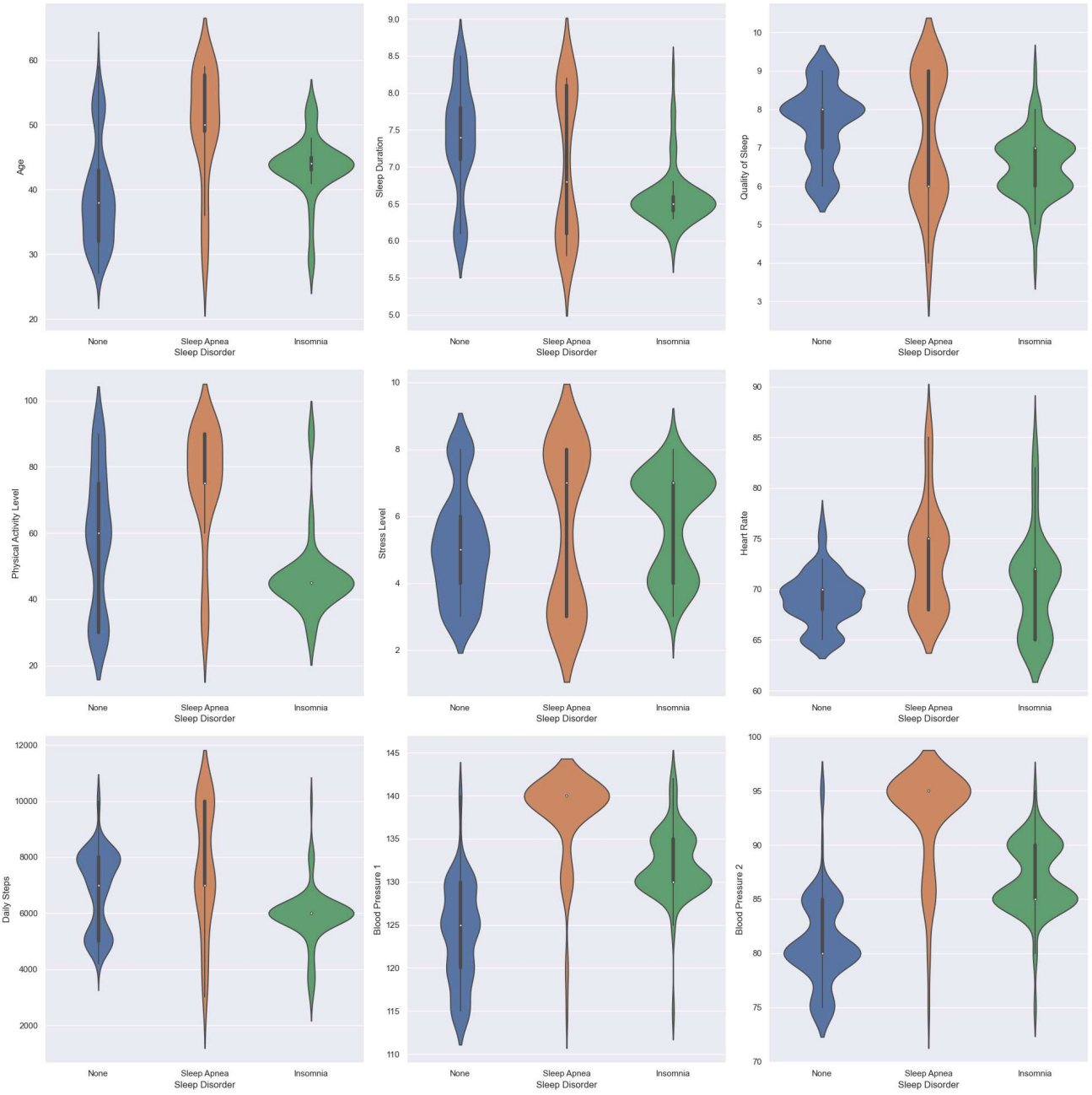
In [17]:
```
num_vars = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level',
            'Stress Level','Heart Rate', 'Daily Steps', 'Blood Pressure 1', 'Blood Pressure 2']

fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(20, 20))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(y=var, x='Sleep Disorder', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [18]:
```
num_vars = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level',
            'Stress Level','Heart Rate', 'Daily Steps', 'Blood Pressure 1', 'Blood Pressure 2']

fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [19]:
```
num_vars = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level',
            'Stress Level','Heart Rate', 'Daily Steps', 'Blood Pressure 1', 'Blood Pressure 2']

fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(20, 20))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(y=var, data=df, x='Sleep Disorder', ax=axs[i])

fig.tight_layout()

plt.show()
```



Data Preprocessing Part 2

In [20]:
```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[20]:
```
Series([], dtype: float64)
```

Label Encoding for each Object datatype

In [21]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
Gender: ['Male' 'Female']
Occupation: ['Software Engineer' 'Doctor' 'Sales Representative' 'Teacher' 'Nurse'
 'Engineer' 'Accountant' 'Scientist' 'Lawyer' 'Salesperson' 'Manager']
BMI Category: ['Overweight' 'Normal' 'Obese']
Sleep Disorder: ['None' 'Sleep Apnea' 'Insomnia']
```

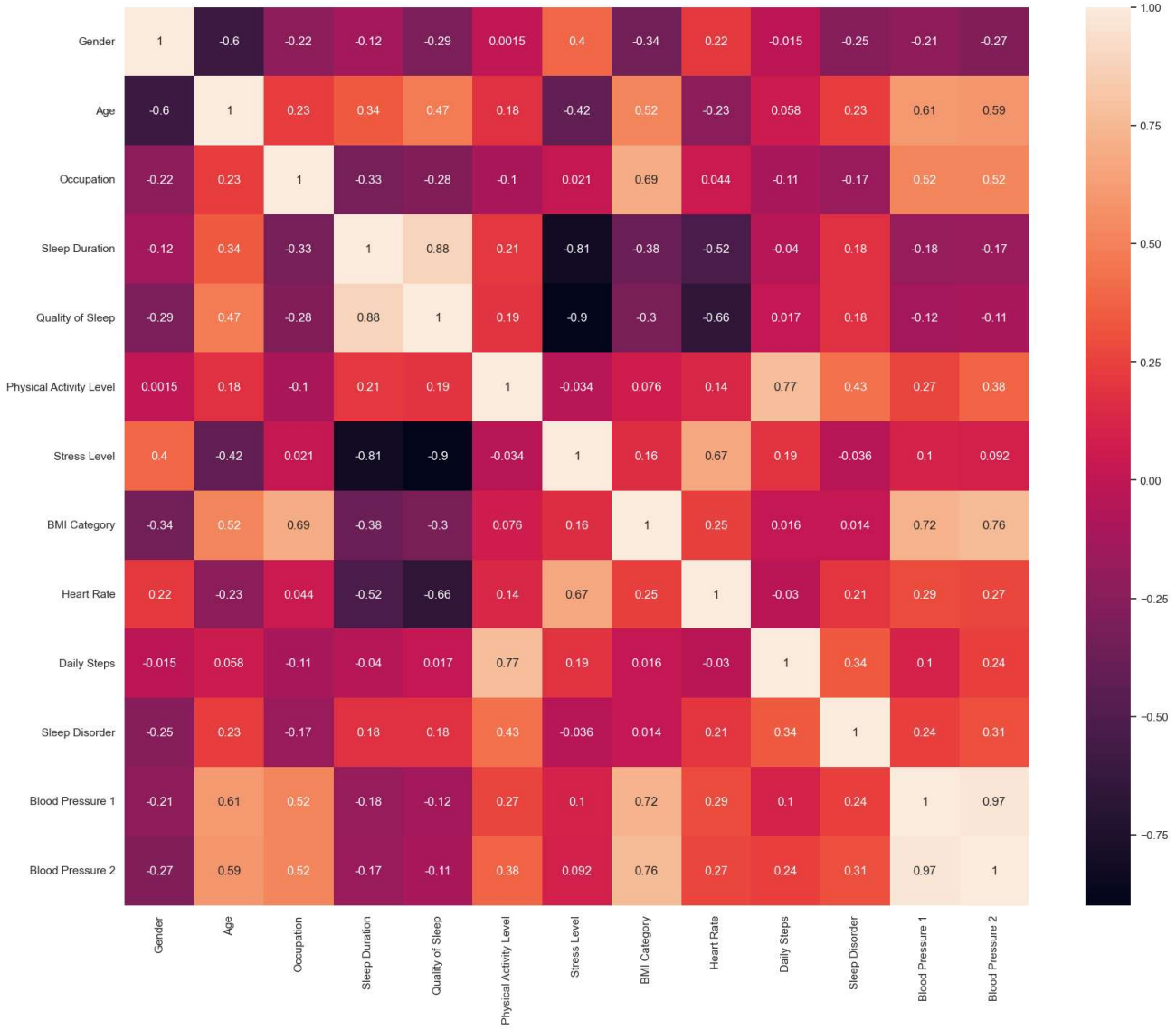
In [22]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Gender: [1 0]
Occupation: [ 9  1  6 10  5  2  0  8  3  7  4]
BMI Category: [2 0 1]
Sleep Disorder: [1 2 0]
```

Correlation Heatmap

In [23]:
```
#Correlation Heatmap (print the correlation score each variables)
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[23]: <AxesSubplot:>



Train Test Split

In [24]:
```
from sklearn.model_selection import train_test_split
# Select the features (X) and the target variable (y)
X = df.drop('Sleep Disorder', axis=1)
y = df['Sleep Disorder']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Remove the Outlier from train data using Z-Score

In [25]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['Heart Rate']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

Decision Tree

In [26]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = DecisionTreeClassifier(class_weight='balanced')
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(dtree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 0}
```

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=4, min_samples_leaf=1, min_samples_split=2, class_weight='ba
dtree.fit(X_train, y_train)
```

Out[27]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=4, random_state=0)
```

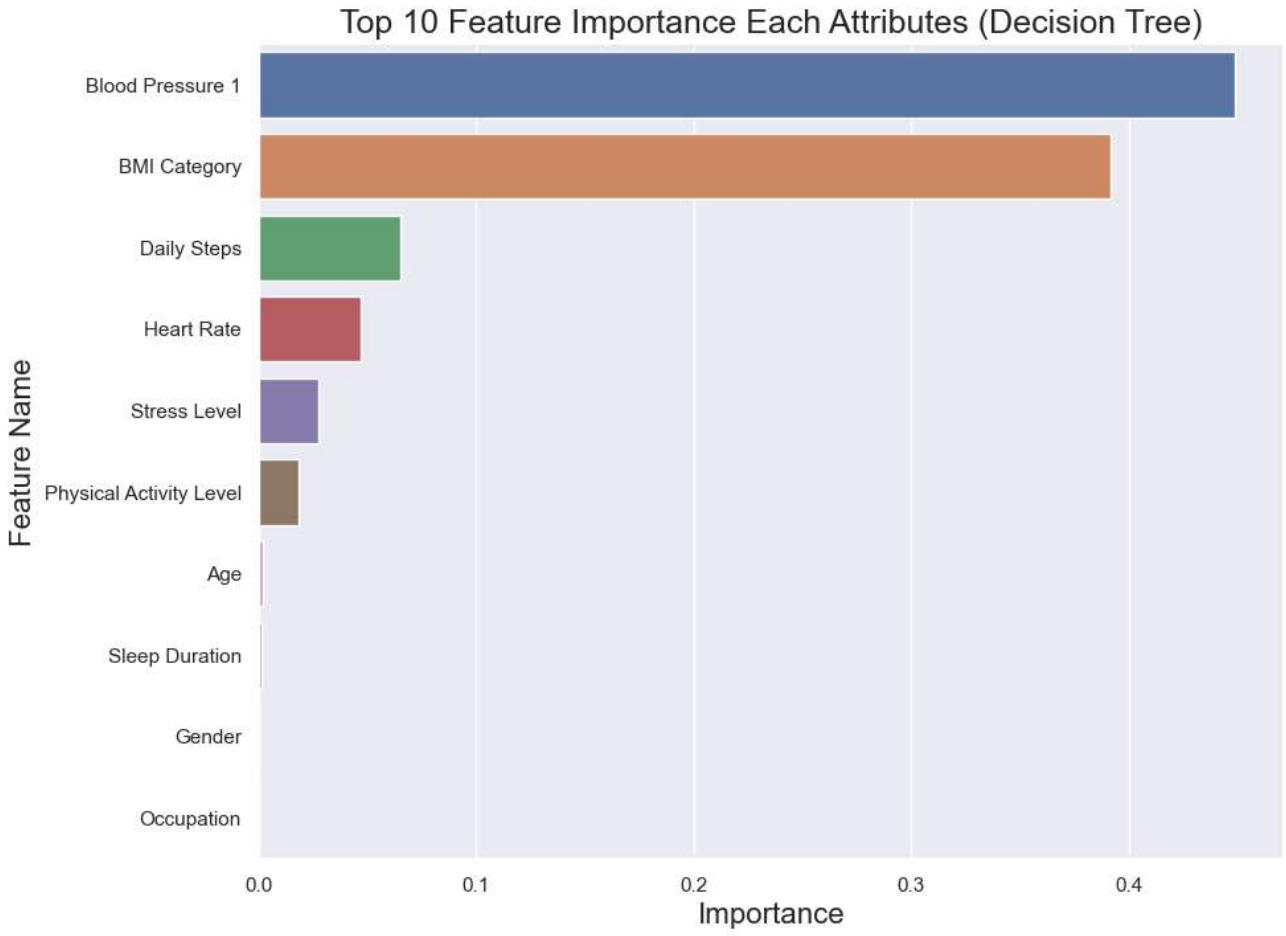
In [28]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 88.0 %
```

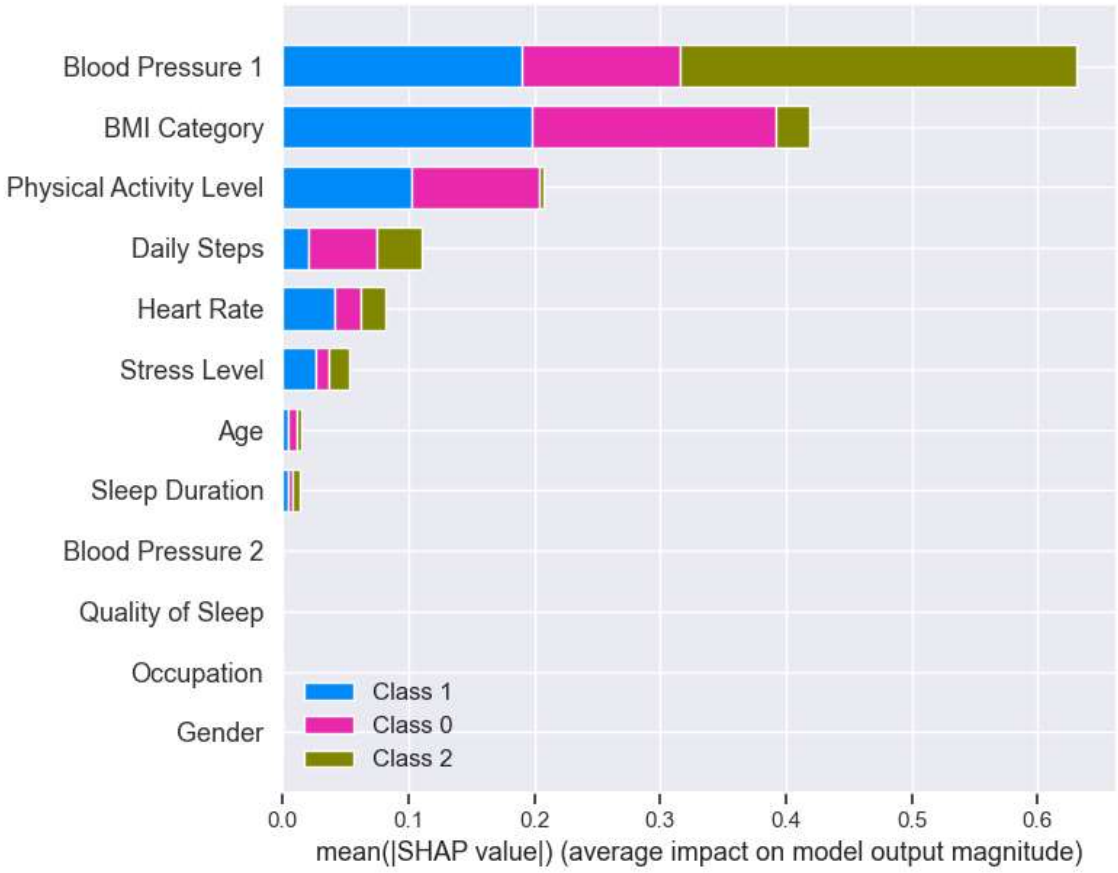
In [30]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
```

```
F-1 Score :  0.88
Precision Score :  0.88
Recall Score :  0.88
Jaccard Score :  0.7857142857142857
```

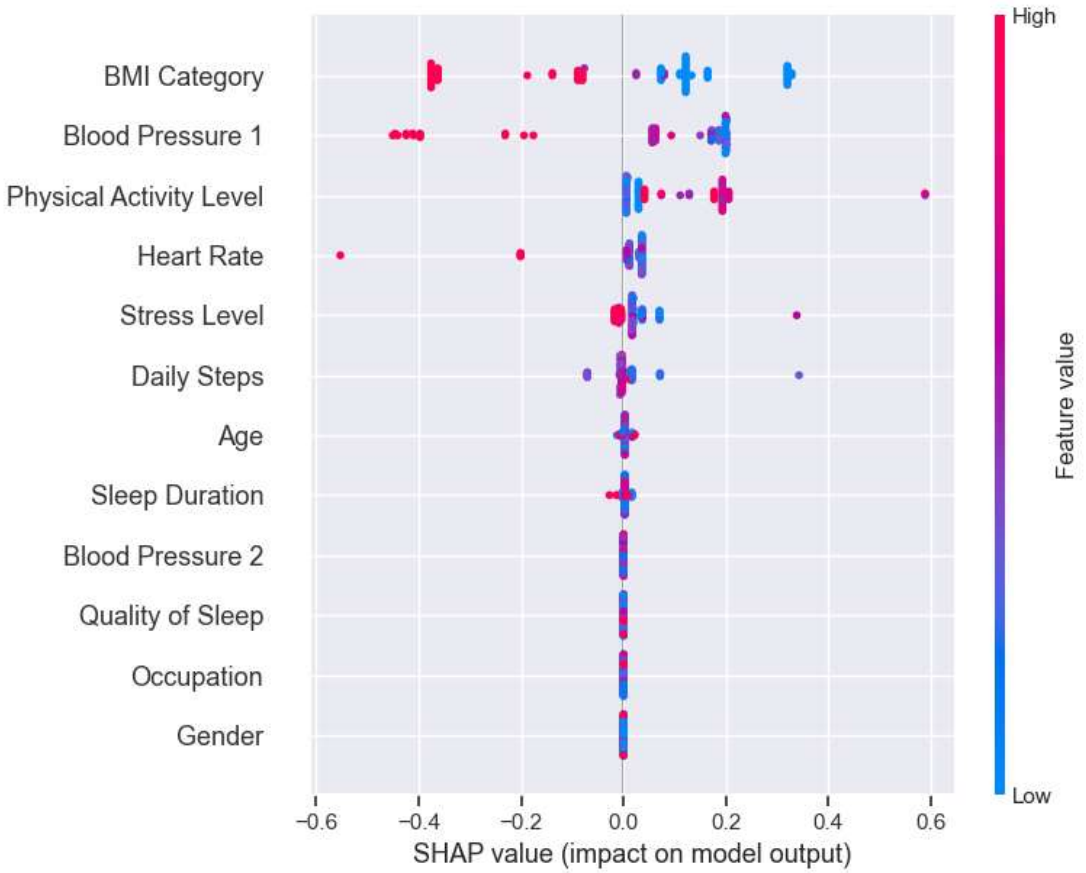
In [31]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



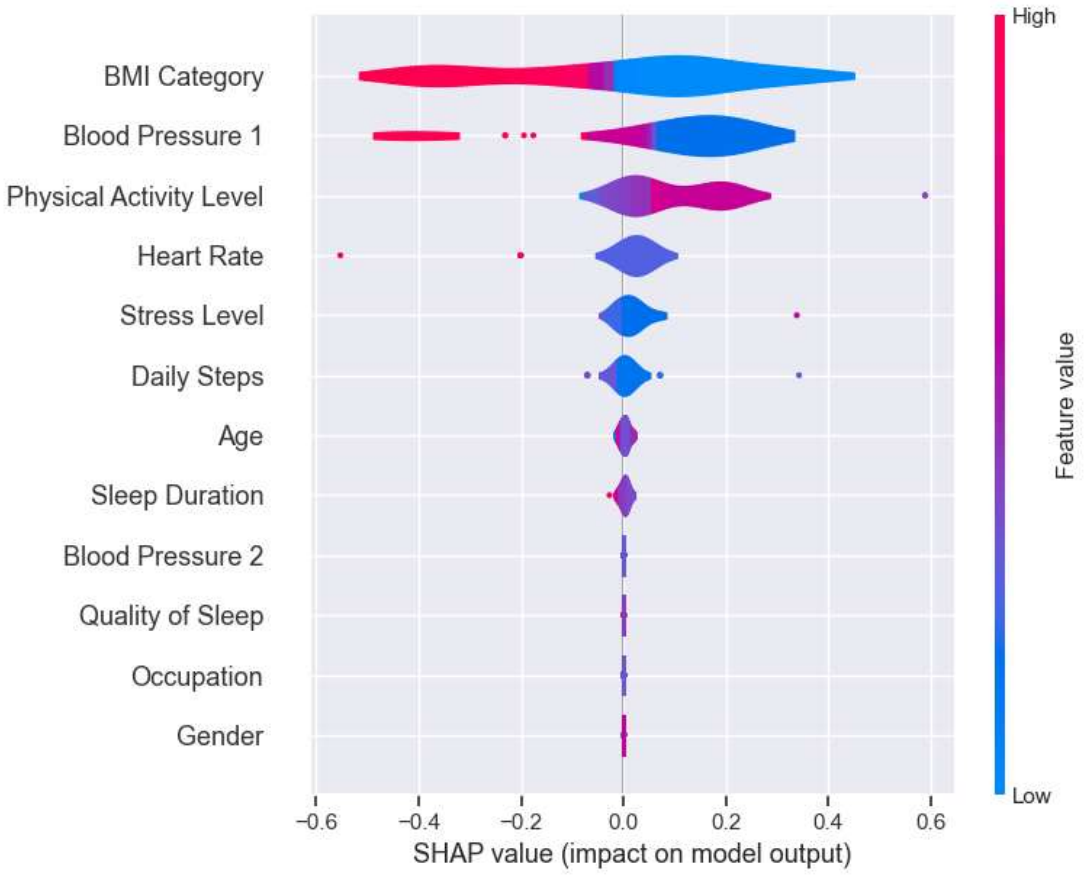
In [32]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [33]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```

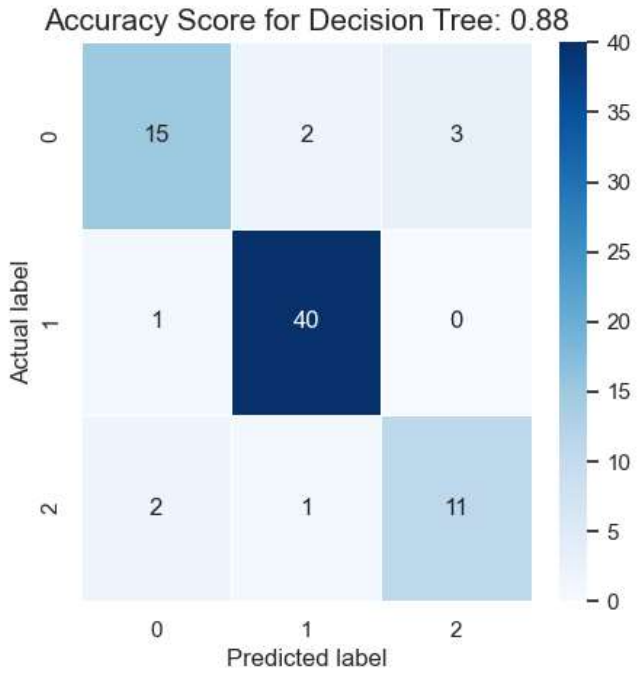


In [34]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type="violin")
```



In [35]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[35]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.88')
```



Random Forest

In [36]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(class_weight='balanced')
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2', None],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rfc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 5, 'max_features': 'sqrt', 'n_estimators': 100, 'random_state': 0}
```

In [37]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_features='sqrt', n_estimators=100, max_depth=5, class_weight='balanc
rfc.fit(X_train, y_train)
```

Out[37]:
```
RandomForestClassifier(class_weight='balanced', max_depth=5,
                       max_features='sqrt', random_state=0)
```

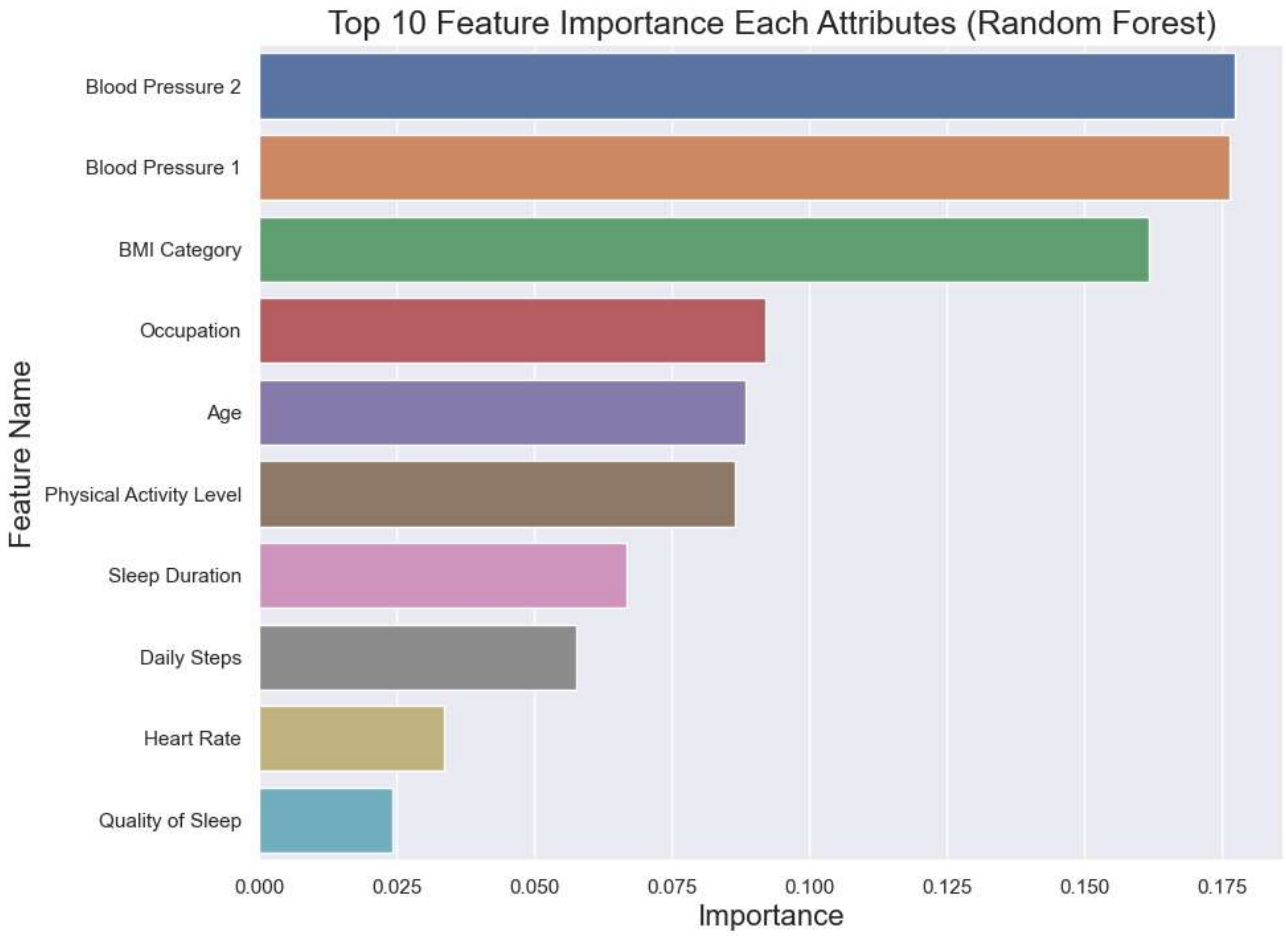
In [38]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 88.0 %
```

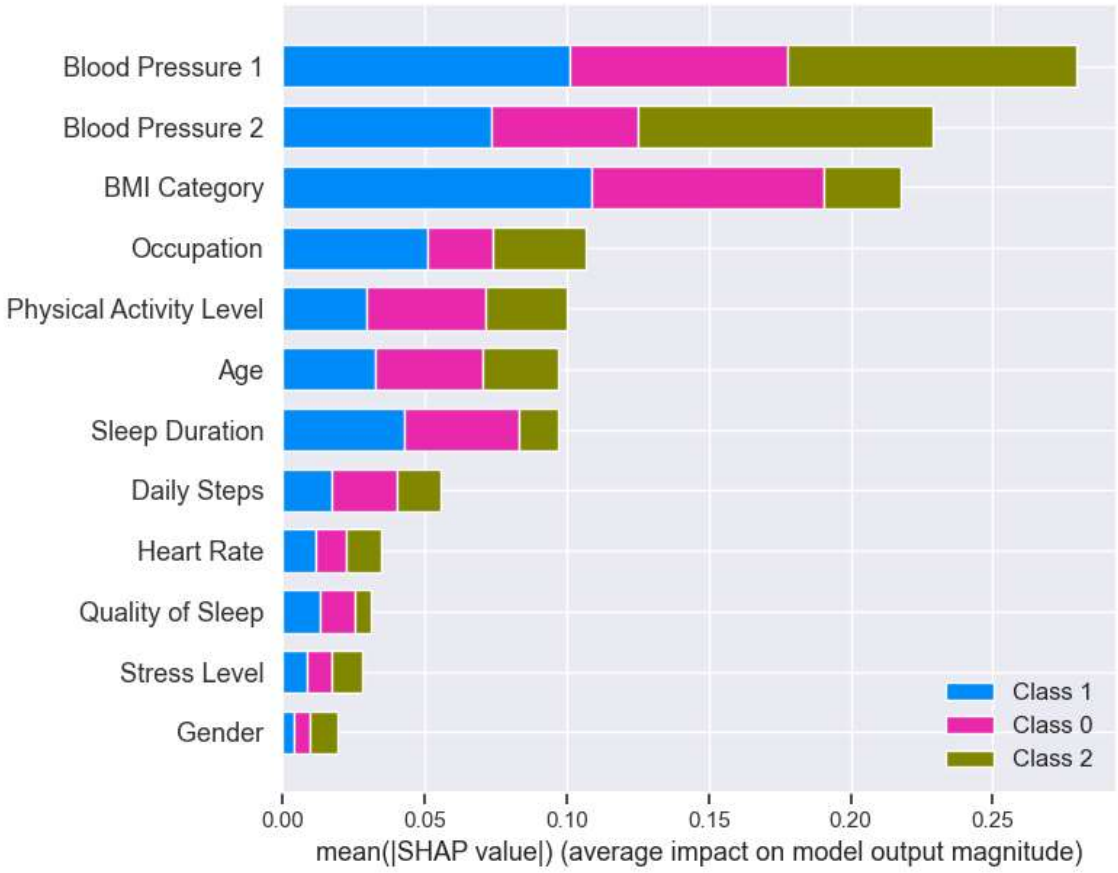
In [39]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
```

```
F-1 Score :  0.88
Precision Score :  0.88
Recall Score :  0.88
Jaccard Score :  0.7857142857142857
```

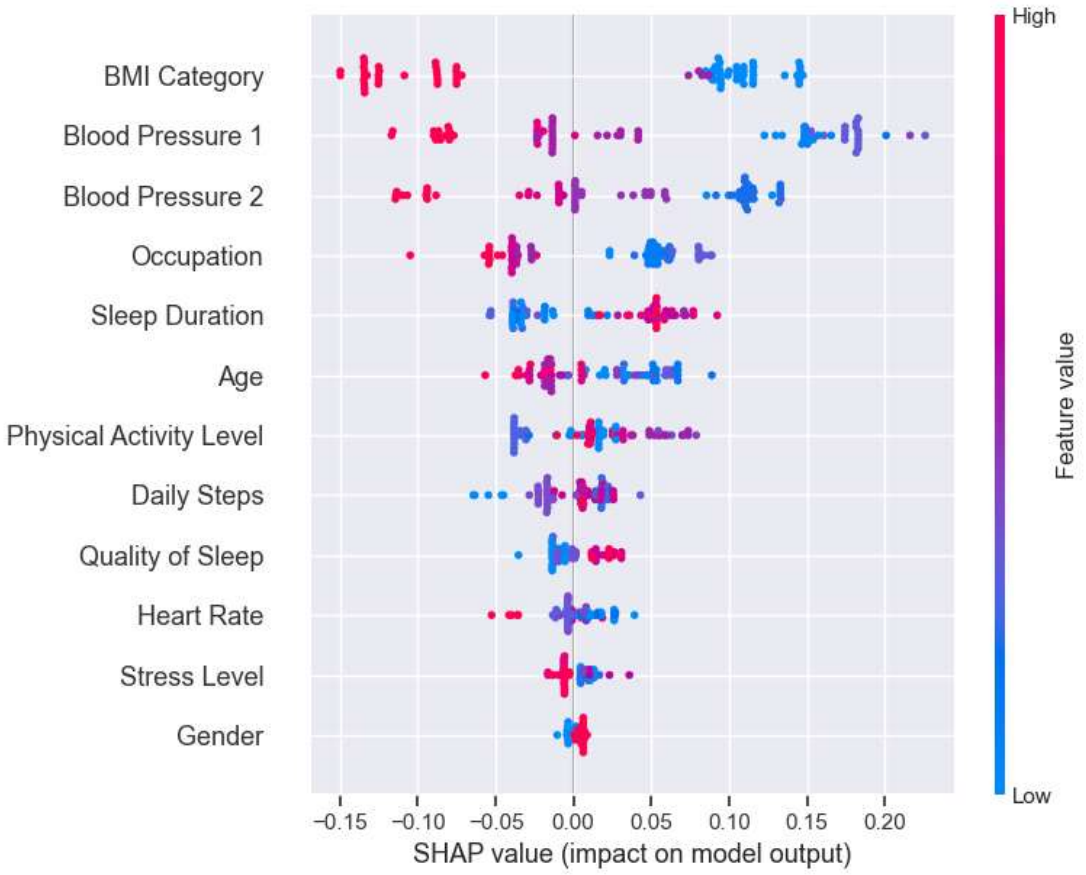
In [40]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



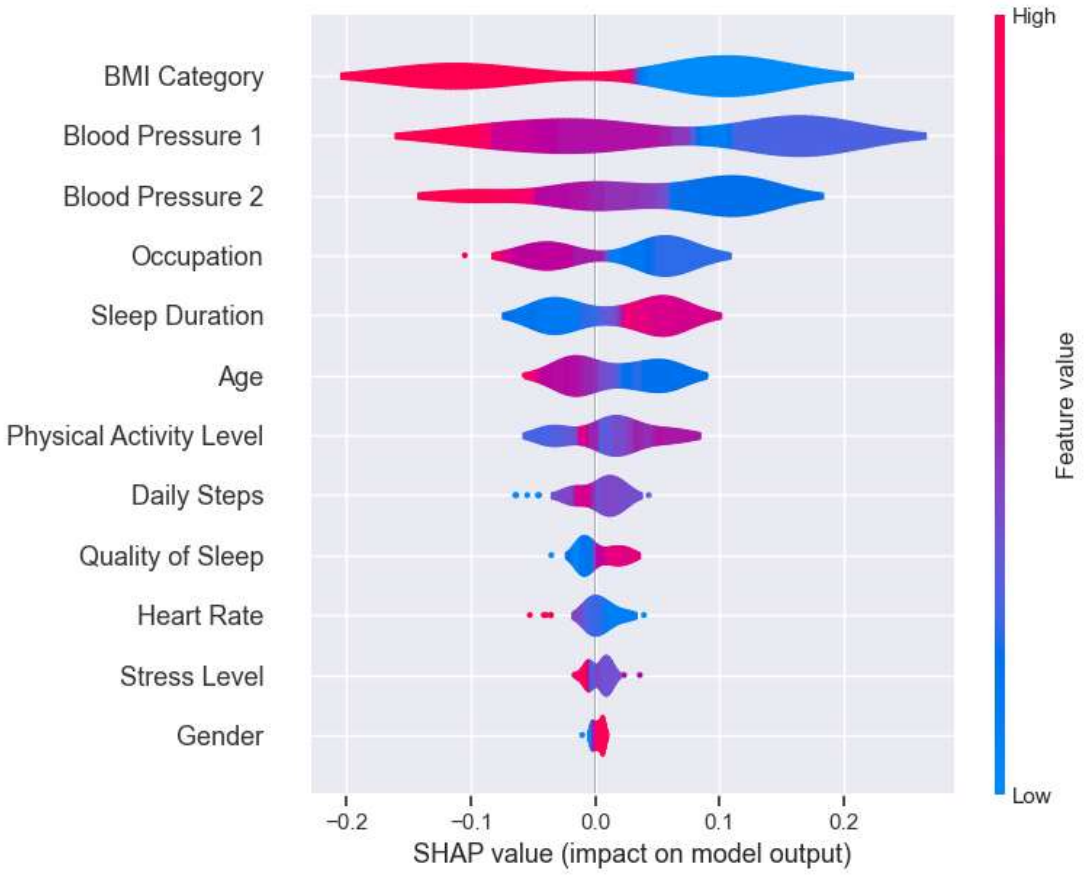
In [41]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [42]:
```
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



In [43]:
```
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type="violin")
```



In [44]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[44]: Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.88')