Heart Disease Predictor

Eddie Artis

8/27/2024

This is a machine learning application designed to predict the likelihood of heart disease in patients based on various health indicators such as age, blood pressure, cholesterol levels, and more.

In [1]:
```
# Import the necessary libraries

import pandas as pd
# pandas is used for data manipulation and analysis

import matplotlib.pyplot as plt
# matplotlib.pyplot is used for creating static, interactive, and animated visualizati

import seaborn as sns
# seaborn is a data visualization library based on matplotlib, providing a high-level

from sklearn.model_selection import train_test_split
# train_test_split is used to split the dataset into training and testing sets

from sklearn.preprocessing import LabelEncoder, label_binarize
# LabelEncoder is used to convert categorical labels into numerical form for machine l

from sklearn.ensemble import RandomForestClassifier
# RandomForestClassifier is an ensemble learning method for classification, regression

from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, conf
# Metrics to evaluate the performance of a classification model

from scipy.stats import chi2_contingency
# chi2_contingency is used to perform a Chi-Square test of independence for categorica

import re
# re is a library for regular expression operations, used here to parse and format tex
```

In [5]:
```
# Load the dataset
# Replace the file path location with the appropriate file path on your system or clou
file_path = 'C:/Users/Artis/Downloads/archive.zip'
data = pd.read_csv(file_path)
```

In [6]:
```
# Display basic information about the dataset and the first few rows
data.info()  # Shows data types and non-null counts
data.head()  # Displays the first 5 rows of the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   Gender                 1000 non-null   object
 1   Age                    1000 non-null   int64 
 2   Blood Pressure (mmHg)  1000 non-null   int64 
 3   Cholesterol (mg/dL)    1000 non-null   int64 
 4   Has Diabetes           1000 non-null   object
 5   Smoking Status         1000 non-null   object
 6   Chest Pain Type        1000 non-null   object
 7   Treatment              1000 non-null   object
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

Out[6]:

	Gender	Age	Blood Pressure (mmHg)	Cholesterol (mg/dL)	Has Diabetes	Smoking Status	Chest Pain Type	Treatment
0	Male	70	181	262	No	Never	Typical Angina	Lifestyle Changes
1	Female	55	103	253	Yes	Never	Atypical Angina	Angioplasty
2	Male	42	95	295	Yes	Current	Typical Angina	Angioplasty
3	Male	84	106	270	No	Never	Atypical Angina	Coronary Artery Bypass Graft (CABG)
4	Male	86	187	296	Yes	Current	Non-anginal Pain	Medication

In [7]:
```
# Generate summary statistics for numerical columns
numerical_summary = data.describe()  # Gives statistical information like mean, std, e
```

In [8]:
```
# Analyze distribution of categorical variables (like gender, smoking status, etc.)
categorical_distribution = {
    column: data[column].value_counts() for column in data.select_dtypes(include=['obj
}
```

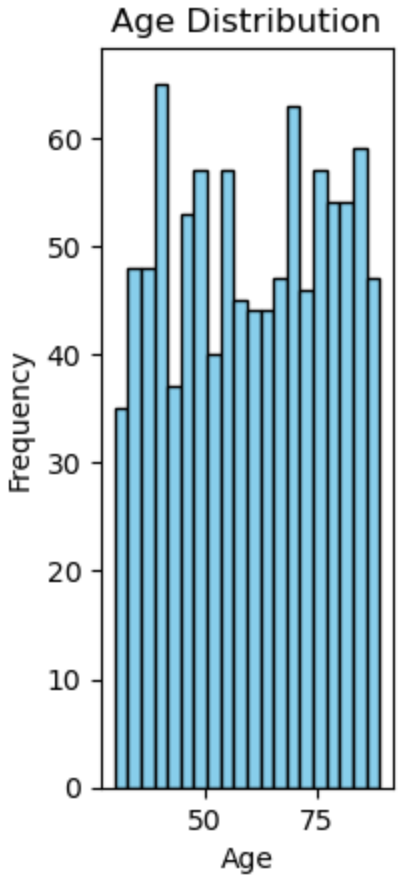
In [9]:
```
# Plot histograms for numerical variables like Age, Blood Pressure, Cholesterol
plt.figure(figsize=(15, 5))
```

Out[9]:
```
<Figure size 1500x500 with 0 Axes>
```


```
<Figure size 1500x500 with 0 Axes>
```

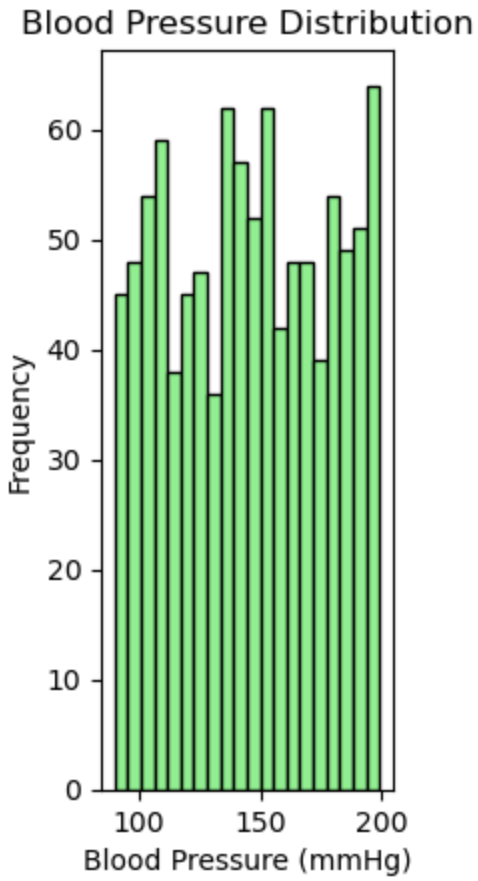

In [10]:
```
# Age Distribution
plt.subplot(1, 3, 1)
plt.hist(data['Age'], bins=20, color='skyblue', edgecolor='black')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
```

Out[10]:
```
Text(0, 0.5, 'Frequency')
```

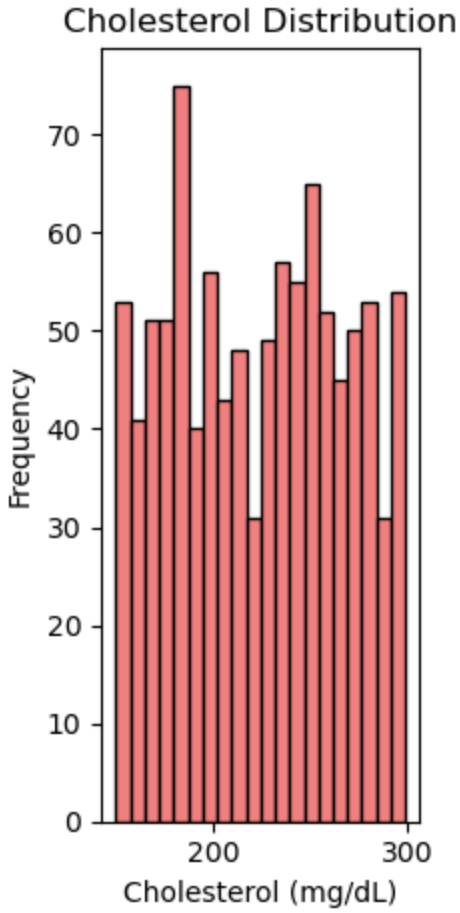


In [11]:
```
# Blood Pressure Distribution
plt.subplot(1, 3, 2)
plt.hist(data['Blood Pressure (mmHg)'], bins=20, color='lightgreen', edgecolor='black'
plt.title('Blood Pressure Distribution')
plt.xlabel('Blood Pressure (mmHg)')
plt.ylabel('Frequency')
```

Out[11]:
```
Text(0, 0.5, 'Frequency')
```



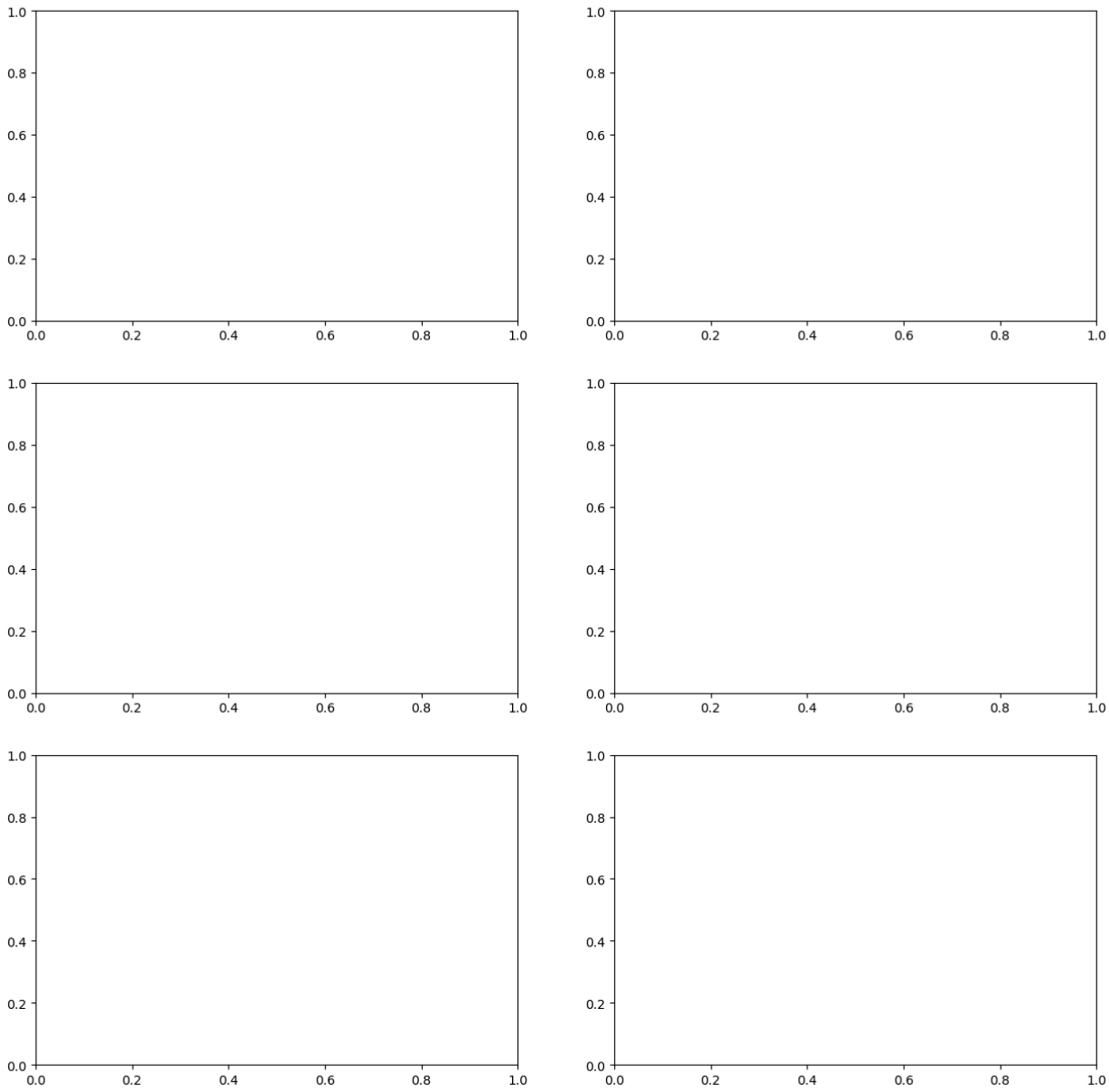
In [12]:
```
# Cholesterol Distribution
plt.subplot(1, 3, 3)
plt.hist(data['Cholesterol (mg/dL)'], bins=20, color='lightcoral', edgecolor='black')
plt.title('Cholesterol Distribution')
plt.xlabel('Cholesterol (mg/dL)')
plt.ylabel('Frequency')

plt.tight_layout()  # Adjusts subplot parameters to give specified padding
plt.show()
```



In [13]:
```
# Plot bar graphs for categorical variables
fig, axes = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle('Categorical Variable Distributions')
```

Out[13]:
```
Text(0.5, 0.98, 'Categorical Variable Distributions')
```

Categorical Variable Distributions

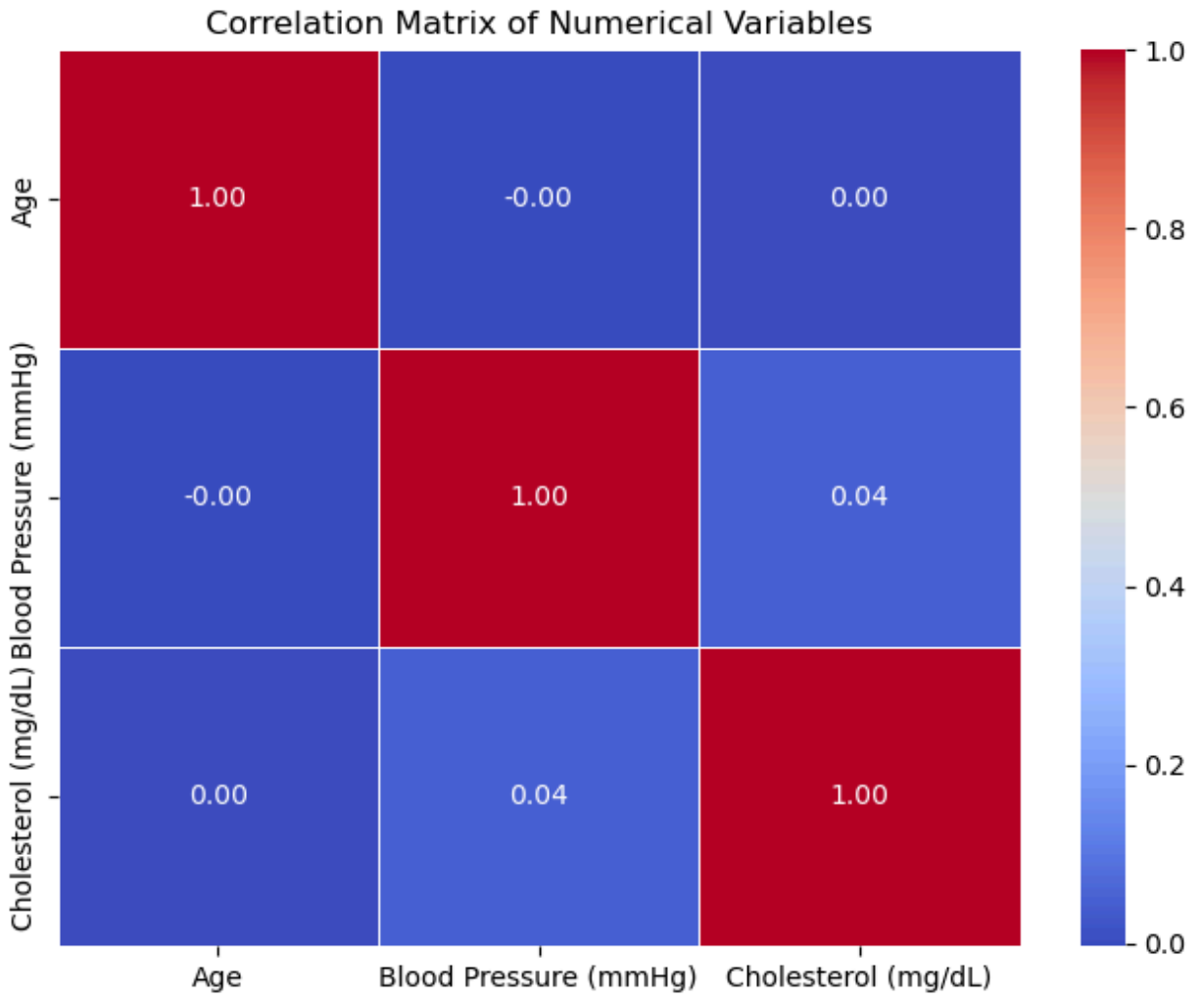


In [14]:
```
# Loop through each categorical column and create a bar plot
for ax, (column, values) in zip(axes.flatten(), categorical_distribution.items()):
    values.plot(kind='bar', ax=ax, color='mediumpurple')
    ax.set_title(f'Distribution of {column}')
    ax.set_xlabel(column)
    ax.set_ylabel('Frequency')

plt.tight_layout(rect=[0, 0, 1, 0.96])  # Adjust layout with extra space at the top fo
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

In [15]:
```
# Analyze correlation between numerical variables
correlation_matrix = data[['Age', 'Blood Pressure (mmHg)', 'Cholesterol (mg/dL)']].cor
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5
```

```
plt.title('Correlation Matrix of Numerical Variables')
plt.show()
```



In [17]:
```
# Perform Chi-Square Test for Independence to check relationships between categorical
contingency_tables = {
    'Smoking Status vs Treatment': pd.crosstab(data['Smoking Status'], data['Treatment
    'Has Diabetes vs Treatment': pd.crosstab(data['Has Diabetes'], data['Treatment']),
    'Gender vs Treatment': pd.crosstab(data['Gender'], data['Treatment']),
}

chi_square_results = {}
for test_name, table in contingency_tables.items():
    chi2, p, dof, expected = chi2_contingency(table)
    chi_square_results[test_name] = {'Chi2 Statistic': chi2, 'p-value': p, 'Degrees of
```

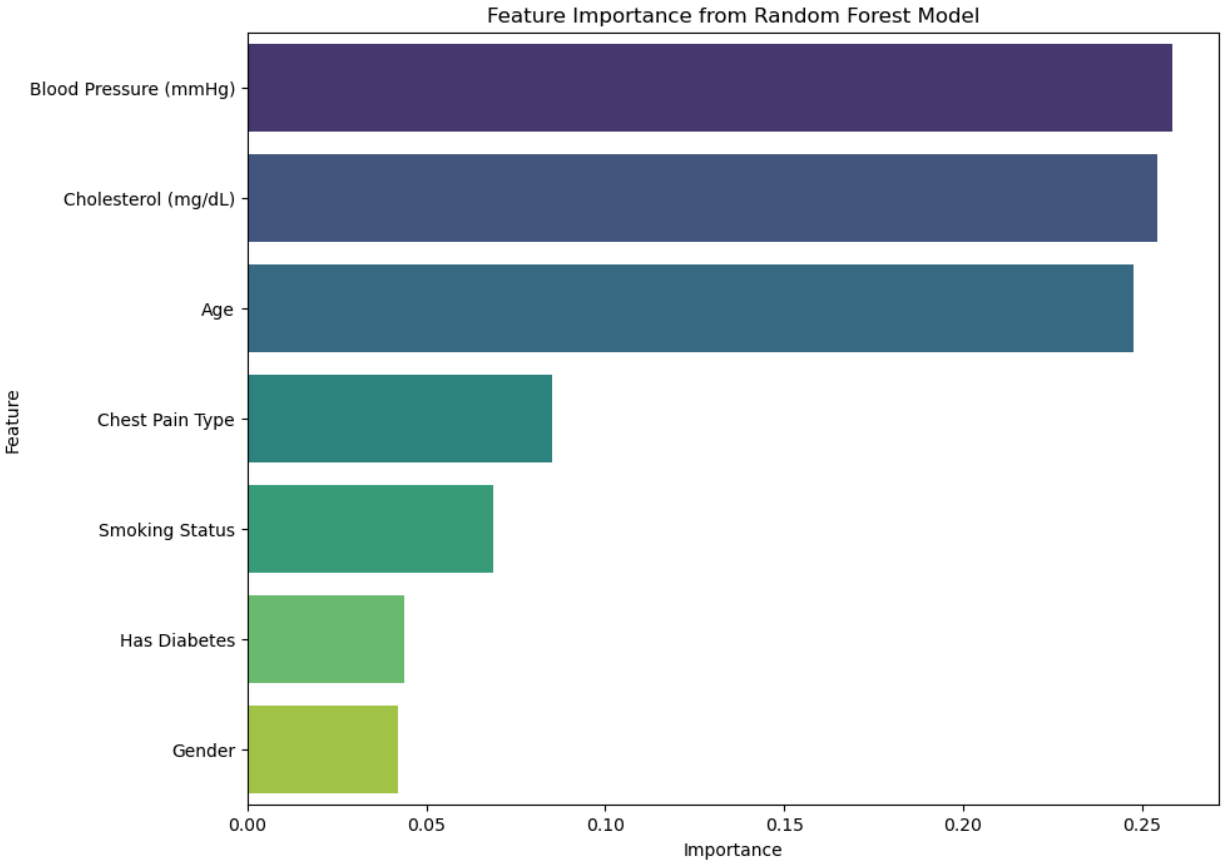
In [18]:
```
# Encode categorical variables to numbers for machine learning algorithms
encoded_data = data.copy()  # Create a copy of the dataset
label_encoders = {}  # Dictionary to store label encoders for each column

# Encode each categorical column using LabelEncoder
for column in encoded_data.select_dtypes(include=['object']).columns:
    label_encoders[column] = LabelEncoder()
    encoded_data[column] = label_encoders[column].fit_transform(encoded_data[column])
```

In [19]:
```
# Split the dataset into features (X) and the target variable (y)
X = encoded_data.drop(columns=['Treatment'])
```

```
y = encoded_data['Treatment']
```

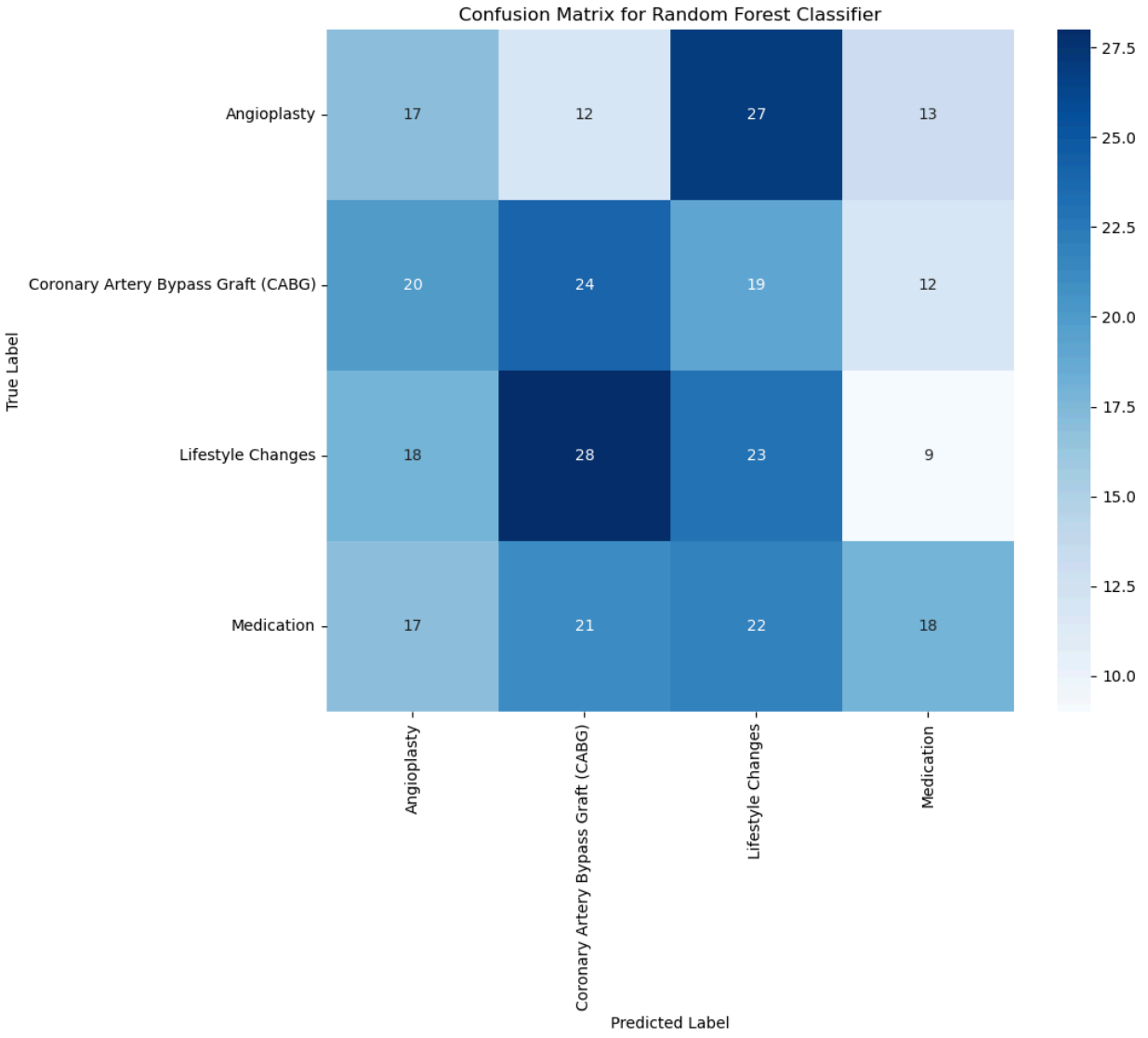
In [21]:
```
# Split the data into training and testing sets (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=

# Initialize and train the Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)  # Train the model using the training data

# Make predictions on the test data
y_pred = rf_model.predict(X_test)

# Evaluate the model using accuracy, classification report, and ROC-AUC score
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred, target_names=label_encoders
roc_auc = roc_auc_score(y_test, rf_model.predict_proba(X_test), multi_class='ovr')
```

In [22]:
```
# Display feature importance from the Random Forest model
importances = rf_model.feature_importances_  # Importance scores for each feature
features = X.columns  # Feature names
feature_importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df, palette='viridis'
plt.title('Feature Importance from Random Forest Model')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



In [23]:
```
# Confusion Matrix for the Random Forest predictions
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoders
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Random Forest Classifier')
plt.show()
```

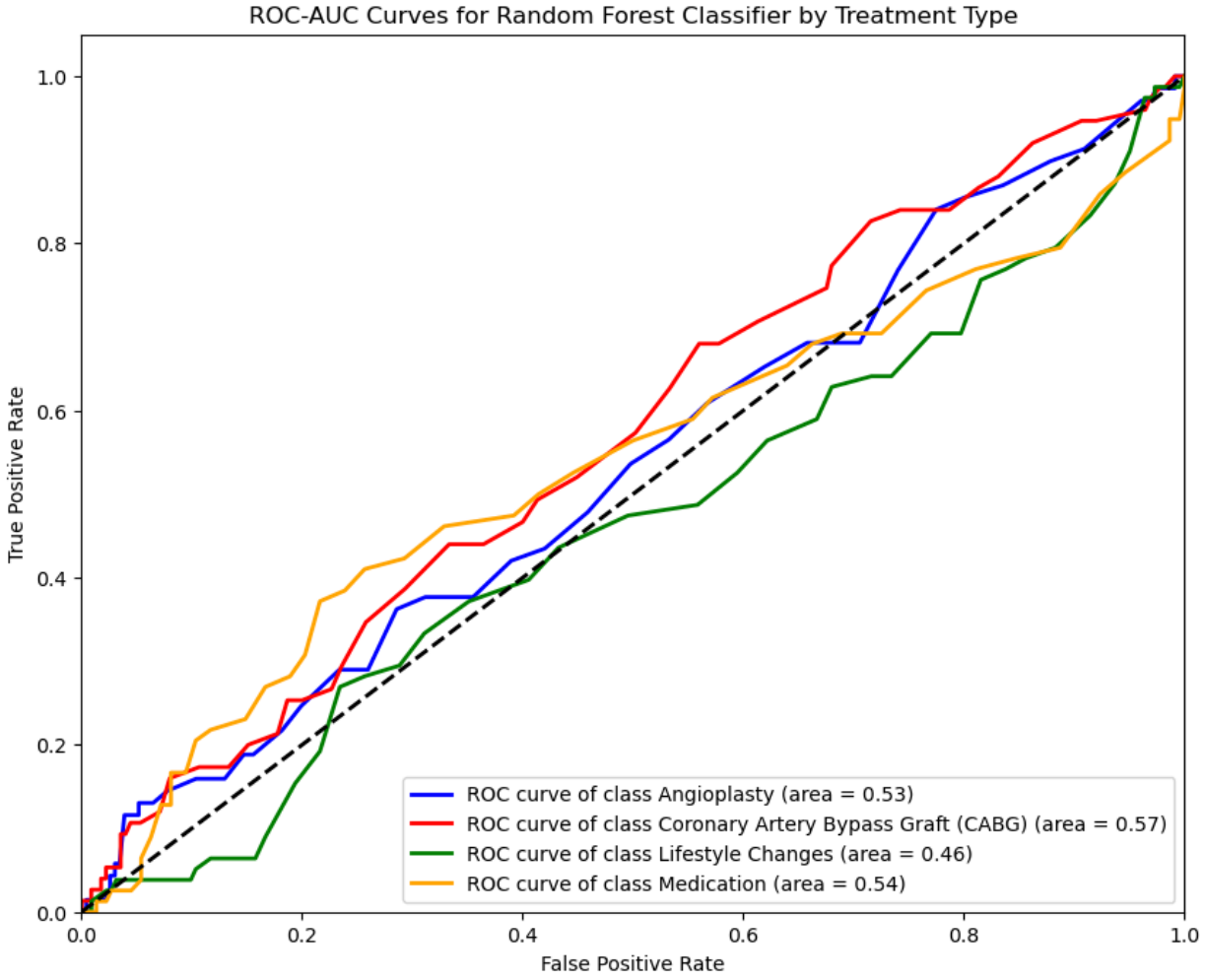


In [24]:
```
# ROC-AUC Curve for the Random Forest model
y_test_binarized = label_binarize(y_test, classes=[0, 1, 2, 3])  # Binarize the output
n_classes = y_test_binarized.shape[1]  # Number of classes
```

In [25]:
```
# Calculate ROC-AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], rf_model.predict_proba(X_tes
    roc_auc[i] = auc(fpr[i], tpr[i])
```

In [26]:
```
# Plot ROC-AUC Curves
plt.figure(figsize=(10, 8))
colors = ['blue', 'red', 'green', 'orange']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'ROC curve of class {label_encc
```

```
plt.plot([0, 1], [0, 1], 'k--', lw=2)  # Diagonal line for random chance
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curves for Random Forest Classifier by Treatment Type')
plt.legend(loc="lower right")
plt.show()
```



In [27]:
```
# Create summary tables for Chi-Square Test results, Summary Statistics, and Classific
chi_square_summary = pd.DataFrame({
    'Test': list(chi_square_results.keys()),
    'Chi2 Statistic': [result['Chi2 Statistic'] for result in chi_square_results.value
    'p-value': [result['p-value'] for result in chi_square_results.values()],
    'Degrees of Freedom': [result['Degrees of Freedom'] for result in chi_square_resul
})
```

In [28]:
```
# Convert numerical summary statistics into a table format
numerical_summary_table = numerical_summary.T.reset_index()
numerical_summary_table.columns = ['Feature', 'Count', 'Mean', 'Std Dev', 'Min', '25%'
```

In [29]:
```
# Extract classification report into a more readable format (table)
rf_classification_report_lines = re.findall(r'^\s*(\S.*?\S)\s+(\d\.\d{2})\s+(\d\.\d{2}
rf_summary_df = pd.DataFrame(rf_classification_report_lines, columns=['Treatment Type'
```