Predictive Analytics for Patient Outcomes in Healthcare

This a prototype model of the predict analystics model, where we have used a sample dataset from kaggle, in order to understand the overall structure of the model which includes visualisation as well as model building

In [1]:
```
import pandas as pd

# Load the dataset
file_path = 'C:/Users/varshithbr/Desktop/case study final exam/finaldatasetnew2.csv
df = pd.read_csv(file_path)

# Display the first few rows of the dataset
df.head()
```

Out[1]:

	Name	Blood Type	age	Sex	ChestPain	BloodPressure	Cholesterol	BloodSugar	ECG	HeartRat
0	Bobby JacksOn	B-	63	1	3	145	233	1	0	15
1	LesLie TErRy	A+	37	1	2	130	250	0	1	18
2	DaNnY sMitH	A-	41	0	1	130	204	0	0	17
3	andrEw waTtS	O+	56	1	1	120	236	0	1	17
4	adrIENNE bEll	AB+	57	0	0	120	354	0	1	16

Data cleaning

This is the process where we refine the data take it further for the EDA

In [130…
```
# 1. Check for missing values
print("Missing Values:\n", df.isnull().sum())

# 2. Handle missing values by filling them with the mean (for numerical columns)
# df.fillna(df.mean(), inplace=True)

# 3. Check data types
print("\nData Types:\n", df.dtypes)
```

```
Missing Values:
 Name             0
Blood Type       0
age              0
Sex              0
ChestPain        0
BloodPressure    0
Cholesterol      0
BloodSugar       0
ECG              0
HeartRate        0
Angina           0
Depression       0
Slope            0
Vessels          0
Thalassemia      0
output           0
dtype: int64

Data Types:
 Name              object
Blood Type        object
age                int64
Sex                int64
ChestPain          int64
BloodPressure      int64
Cholesterol        int64
BloodSugar         int64
ECG                int64
HeartRate          int64
Angina             int64
Depression       float64
Slope              int64
Vessels            int64
Thalassemia        int64
output             int64
dtype: object
```

Exploratory data analysis

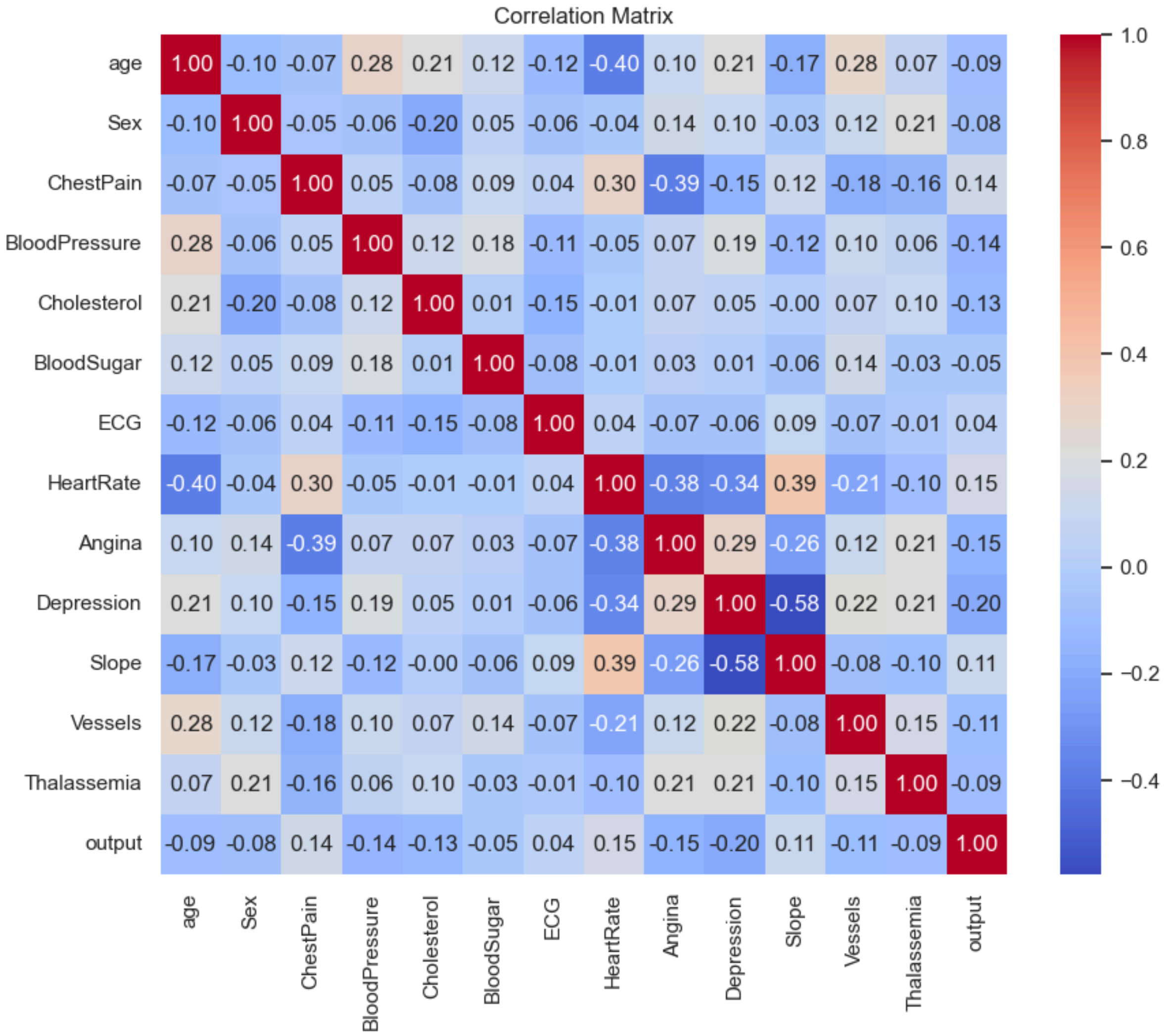
Here we get an idea of relationships,patterns and trends between the variables, so we are proceeding with the best insightful variables

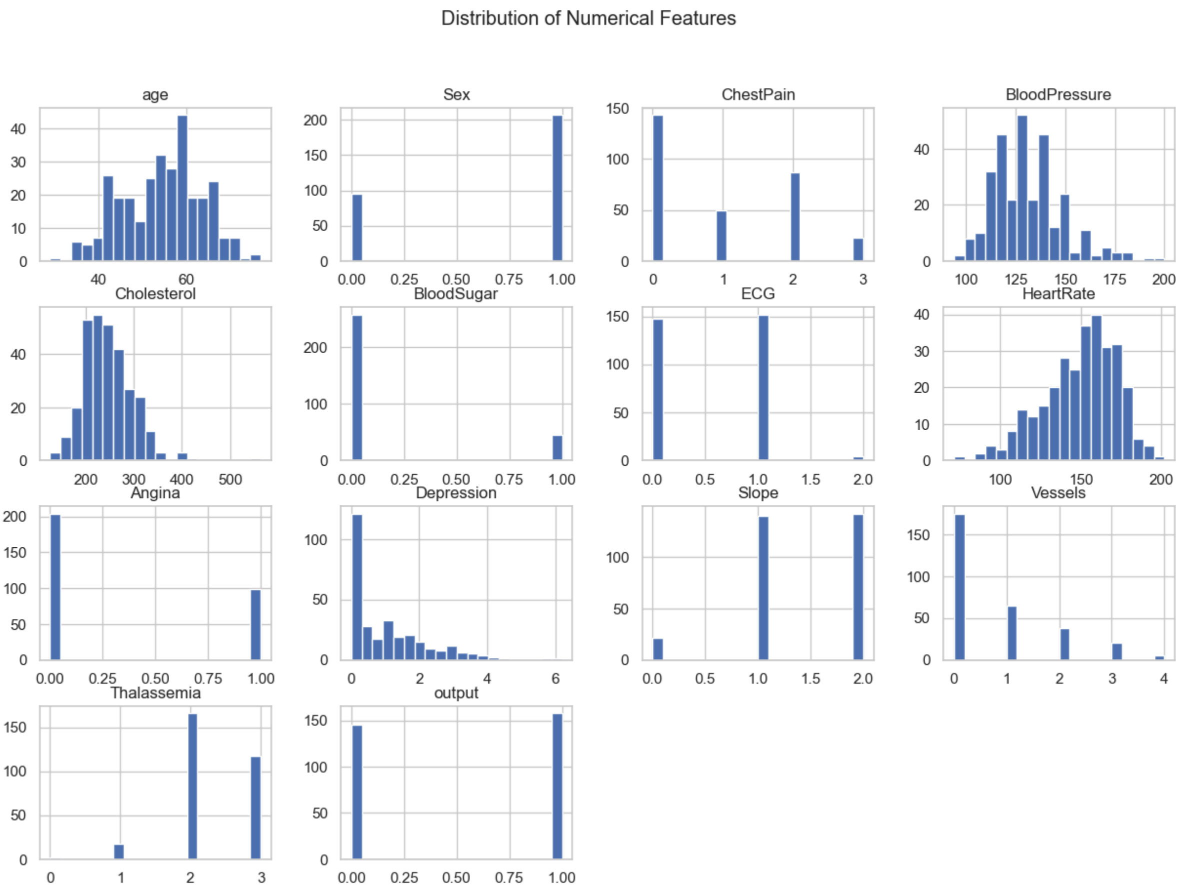
In [99]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Summary statistics
print(df.describe())


# removing stringbased columns in order to obtain a certain correlations between th
rem = [
     'Name', 'Blood Type']

# Drop the specified columns
df_clean = df.drop(columns=rem)

# Correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(df_clean.corr(), annot=True, cmap='coolwarm', fmt='.2f')
```

```python
plt.title("Correlation Matrix")
plt.show()

# Distribution plots for numerical features
df.hist(bins=20, figsize=(15, 10))
plt.suptitle("Distribution of Numerical Features")
plt.show()

# Pairplot to visualize relationships between features
sns.pairplot(df, diag_kind='kde')
plt.suptitle("Pairplot of Features", y=1.02)
plt.show()

# Countplot for categorical variables (if any)
# Example: sns.countplot(x='categorical_column', data=df)
```

```
              age         Sex   ChestPain  BloodPressure  Cholesterol  \
count  303.000000  303.000000  303.000000     303.000000   303.000000   
mean    54.366337    0.683168    0.966997     131.623762   246.264026   
std      9.082101    0.466011    1.032052      17.538143    51.830751   
min     29.000000    0.000000    0.000000      94.000000   126.000000   
25%     47.500000    0.000000    0.000000     120.000000   211.000000   
50%     55.000000    1.000000    1.000000     130.000000   240.000000   
75%     61.000000    1.000000    2.000000     140.000000   274.500000   
max     77.000000    1.000000    3.000000     200.000000   564.000000   

        BloodSugar         ECG   HeartRate      Angina  Depression       Slope  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000   
mean     0.148515    0.528053  149.646865    0.326733    1.039604    1.399340   
std      0.356198    0.525860   22.905161    0.469794    1.161075    0.616226   
min      0.000000    0.000000   71.000000    0.000000    0.000000    0.000000   
25%      0.000000    0.000000  133.500000    0.000000    0.000000    1.000000   
50%      0.000000    1.000000  153.000000    0.000000    0.800000    1.000000   
75%      0.000000    1.000000  166.000000    1.000000    1.600000    2.000000   
max      1.000000    2.000000  202.000000    1.000000    6.200000    2.000000   

          Vessels  Thalassemia      output  
count  303.000000   303.000000  303.000000  
mean     0.729373     2.313531    0.521452  
std      1.022606     0.612277    0.500366  
min      0.000000     0.000000    0.000000  
25%      0.000000     2.000000    0.000000  
50%      0.000000     2.000000    1.000000  
75%      1.000000     3.000000    1.000000  
max      4.000000     3.000000    1.000000  
```





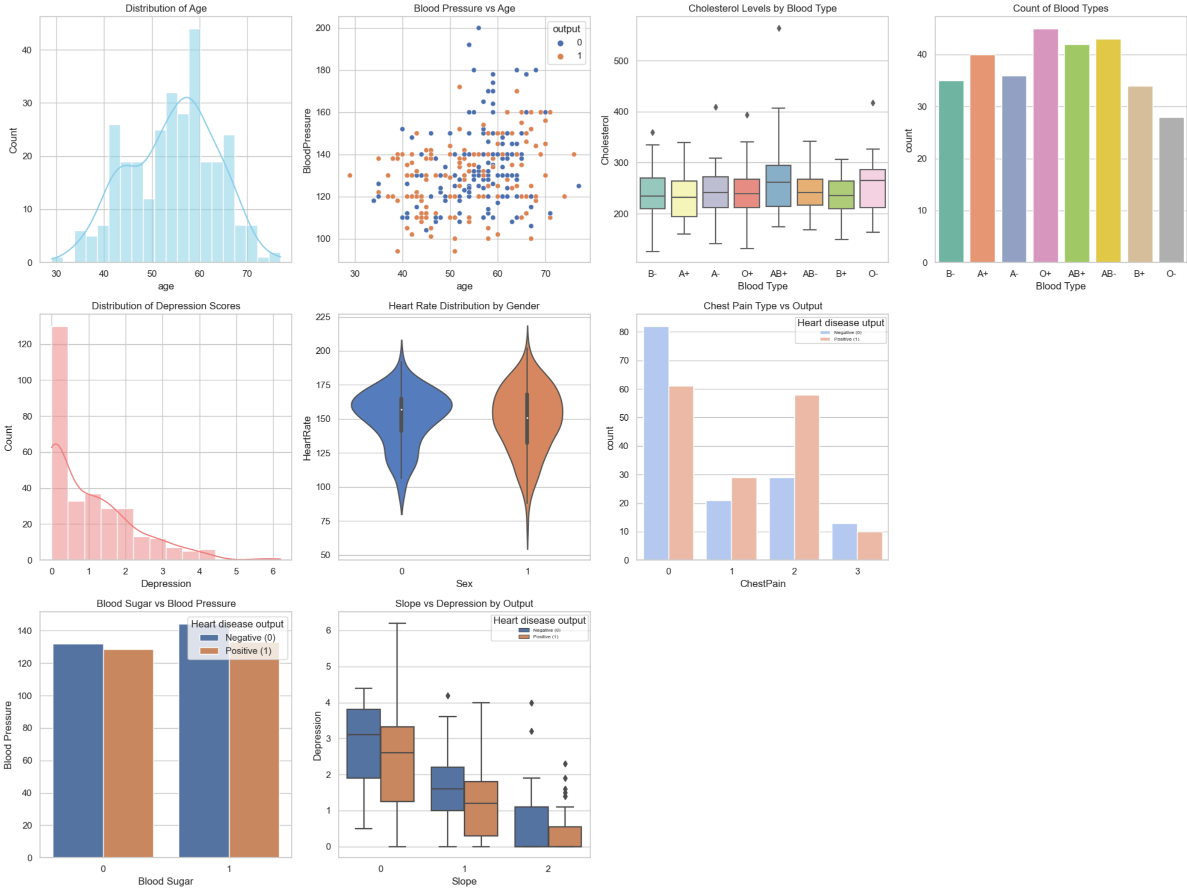

```
C:\Users\varshithbr\AppData\Local\Programs\Python\Python311\Lib\site-packages\seab
orn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

In [183…

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the CSV file
rem = ['Name']

# Drop the specified columns
df_clean2 = df.drop(columns=rem)

df = df_clean2
# Set up the plotting environment
sns.set(style="whitegrid")
plt.figure(figsize=(20, 15))

# 1. Distribution of Age
plt.subplot(3, 4, 1)
sns.histplot(df['age'], kde=True, bins=20, color='skyblue')
plt.title('Distribution of Age')

# 2. Blood Pressure vs Age
plt.subplot(3, 4, 2)
sns.scatterplot(x='age', y='BloodPressure', hue='output', data=df)
plt.title('Blood Pressure vs Age')
```

```
# 3. Cholesterol Levels by Blood Type
plt.subplot(3, 4, 3)
sns.boxplot(x='Blood Type', y='Cholesterol', data=df, palette="Set3")
plt.title('Cholesterol Levels by Blood Type')

# 4. Count of Blood Types
plt.subplot(3, 4, 4)
sns.countplot(x='Blood Type', data=df, palette="Set2")
plt.title('Count of Blood Types')

# 5. Distribution of Depression Scores
plt.subplot(3, 4, 5)
sns.histplot(df['Depression'], kde=True, color='lightcoral')
plt.title('Distribution of Depression Scores')

# 6. Heart Rate Distribution by Gender (Sex)
plt.subplot(3, 4, 6)
sns.violinplot(x='Sex', y='HeartRate', data=df, palette="muted")
plt.title('Heart Rate Distribution by Gender')

# 7. Chest Pain Type vs Output
plt.subplot(3, 4, 7)
sns.countplot(x='ChestPain', hue='output', data=df, palette="coolwarm")
plt.title('Chest Pain Type vs Output')
count_plot = sns.countplot(x='ChestPain', hue='output', data=df, palette="coolwarm'
handles, labels = count_plot.get_legend_handles_labels()
labels = ['Negative (0)', 'Positive (1)']
count_plot.legend(handles, labels, title='Heart disease utput', prop={'size': 6}, l


# 9. Blood Sugar vs Blood Pressure

plt.subplot(3, 4, 9)
sns.barplot(x='BloodSugar', y='BloodPressure', hue='output', data=df, ci=None)

plt.legend(title='Heart disease output', labels=['Negative (0)', 'Positive (1)'])

plt.title('Blood Sugar vs Blood Pressure')
plt.xlabel('Blood Sugar')
plt.ylabel('Blood Pressure')


# 10. Slope vs Depression by Output
plt.subplot(3, 4, 10)
sns.boxplot(x='Slope', y='Depression', hue='output', data=df)
plt.title('Slope vs Depression by Output')
count_plot = sns.boxplot(x='Slope', y='Depression', hue='output', data=df)
handles, labels = count_plot.get_legend_handles_labels()
labels = ['Negative (0)', 'Positive (1)']
count_plot.legend(handles, labels, title='Heart disease output', prop={'size': 6},


# Adjust layout
plt.tight_layout()
plt.show()
```

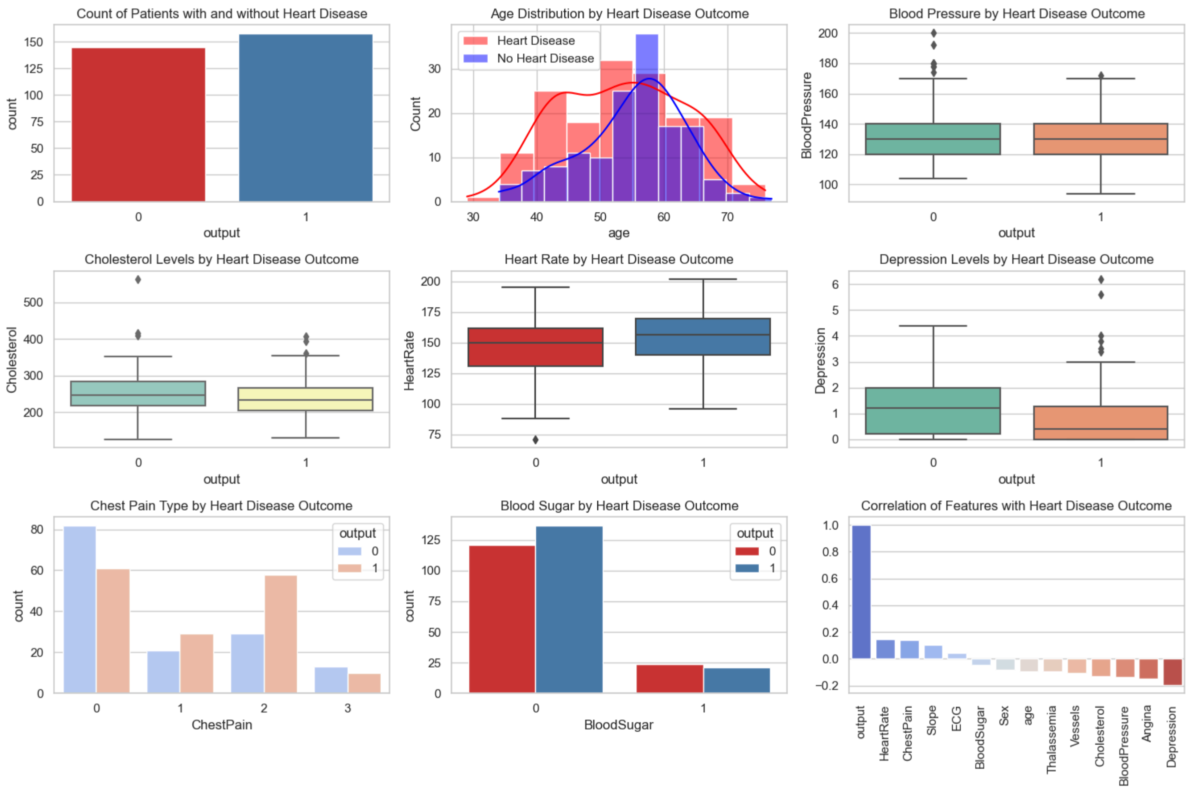
```
C:\Users\varshithbr\AppData\Local\Temp\ipykernel_23636\3036792497.py:60: FutureWar
ning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='BloodSugar', y='BloodPressure', hue='output', data=df, ci=None)
```



In [95]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = df_clean
# Set up the plotting environment
sns.set(style="whitegrid")
plt.figure(figsize=(15, 10))

# 1. Count of Patients with and without Heart Disease
plt.subplot(3, 3, 1)
sns.countplot(x='output', data=df, palette="Set1")
plt.title('Count of Patients with and without Heart Disease')

# 2. Age Distribution by Heart Disease Outcome
plt.subplot(3, 3, 2)
sns.histplot(df[df['output'] == 1]['age'], kde=True, color='red', label='Heart Dise
sns.histplot(df[df['output'] == 0]['age'], kde=True, color='blue', label='No Heart 
plt.legend()
plt.title('Age Distribution by Heart Disease Outcome')

# 3. Blood Pressure by Heart Disease Outcome
plt.subplot(3, 3, 3)
sns.boxplot(x='output', y='BloodPressure', data=df, palette="Set2")
plt.title('Blood Pressure by Heart Disease Outcome')

# 4. Cholesterol Levels by Heart Disease Outcome
plt.subplot(3, 3, 4)
sns.boxplot(x='output', y='Cholesterol', data=df, palette="Set3")
plt.title('Cholesterol Levels by Heart Disease Outcome')

# 5. Heart Rate by Heart Disease Outcome
plt.subplot(3, 3, 5)
sns.boxplot(x='output', y='HeartRate', data=df, palette="Set1")
plt.title('Heart Rate by Heart Disease Outcome')
```

```
# 6. Depression Levels by Heart Disease Outcome
plt.subplot(3, 3, 6)
sns.boxplot(x='output', y='Depression', data=df, palette="Set2")
plt.title('Depression Levels by Heart Disease Outcome')

# 7. Chest Pain Type by Heart Disease Outcome
plt.subplot(3, 3, 7)
sns.countplot(x='ChestPain', hue='output', data=df, palette="coolwarm")
plt.title('Chest Pain Type by Heart Disease Outcome')

# 8. Blood Sugar by Heart Disease Outcome
plt.subplot(3, 3, 8)
sns.countplot(x='BloodSugar', hue='output', data=df, palette="Set1")
plt.title('Blood Sugar by Heart Disease Outcome')

# 9. Correlation of Features with Heart Disease Outcome
plt.subplot(3, 3, 9)
correlation = df.corr()['output'].sort_values(ascending=False)
sns.barplot(x=correlation.index, y=correlation.values, palette="coolwarm")
plt.xticks(rotation=90)
plt.title('Correlation of Features with Heart Disease Outcome')

# Adjust layout
plt.tight_layout()
plt.show()
```



Model buidling

We have choose classification model and here we are going to be buiding several classifcation models in order to choose the best model for our project.

In [52]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

df=df_clean
# Define the target and features
X = df[['Depression','HeartRate','Angina','Slope']]  # Replace 'target_column' with
y = df['output']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Standardize the features (if necessary)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[25 16]
 [15 35]]

Classification Report:
               precision    recall  f1-score   support

           0       0.62      0.61      0.62        41
           1       0.69      0.70      0.69        50

    accuracy                           0.66        91
   macro avg       0.66      0.65      0.66        91
weighted avg       0.66      0.66      0.66        91

Accuracy: 0.6593406593406593
```

In [53]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = df_clean
# Define the target and features
X = df[['Depression','HeartRate','Angina','Slope']]
y = df['output']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Standardize the features (if necessary)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[26 15]
 [25 25]]

Classification Report:
               precision    recall  f1-score   support

           0       0.51      0.63      0.57        41
           1       0.62      0.50      0.56        50

    accuracy                           0.56        91
   macro avg       0.57      0.57      0.56        91
weighted avg       0.57      0.56      0.56        91

Accuracy: 0.5604395604395604
```

In [54]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = df_clean
# Define the target and features
X = df[['Depression','HeartRate','Angina','Slope']]
y = df['output']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Standardize the features (if necessary)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the Gradient Boosting model
model = GradientBoostingClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[26 15]
 [22 28]]

Classification Report:
               precision    recall  f1-score   support

           0       0.54      0.63      0.58        41
           1       0.65      0.56      0.60        50

    accuracy                           0.59        91
   macro avg       0.60      0.60      0.59        91
weighted avg       0.60      0.59      0.59        91

Accuracy: 0.5934065934065934
```

In [55]:

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = df_clean
# Define the target and features
X = df[['Depression','HeartRate','Angina','Slope']]
y = df['output']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Standardize the features (if necessary)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the SVM model with RBF kernel
model = SVC(kernel='rbf', C=1, gamma='scale', random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[25 16]
 [19 31]]

Classification Report:
               precision    recall  f1-score   support

           0       0.57      0.61      0.59        41
           1       0.66      0.62      0.64        50

    accuracy                           0.62        91
   macro avg       0.61      0.61      0.61        91
weighted avg       0.62      0.62      0.62        91

Accuracy: 0.6153846153846154
```

In [56]:

```
models = {
    'Logistic Regression': LogisticRegression(),
```

```
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, random_state=
    'SVM (RBF Kernel)': SVC(kernel='rbf', C=1, gamma='scale', random_state=42)
}


accuracies = {}

# Train and evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracies[name] = accuracy_score(y_test, y_pred)

# Plotting the accuracies
plt.figure(figsize=(10, 6))
plt.bar(accuracies.keys(), accuracies.values(), color=['blue', 'green', 'red', 'pur
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.ylim([0, 1])
plt.show()
```