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

In [2]:
```
data = pd.read_csv('Heart Disease data.csv')
```

In [3]:
```
data
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

In [4]:
```
data.shape
```

Out[4]:
```
(1025, 14)
```

In [5]:
```
data.head()
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

In [7]:
```
data.describe()
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	0.754146	2.323902	0.513171
std	9.072290	0.460273	1.029641	17.516718	51.59251	0.356357	0.527878	23.005724	0.472772	1.175053	0.617755	1.030798	0.620660	0.500070
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.000000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

In [8]:
```
missing_values_count = data.isnull().sum()

# Print the count of missing values in each column
print("Missing Values in Each Column:")
print(missing_values_count)
```

```
Missing Values in Each Column:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [9]:
```
data.columns
```

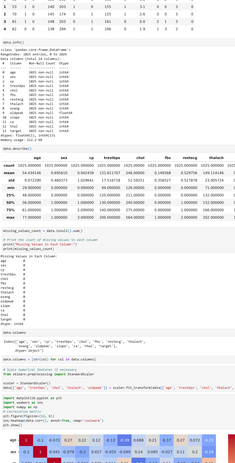
Out[9]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

In [10]:
```
data.columns = [str(col) for col in data.columns]
```

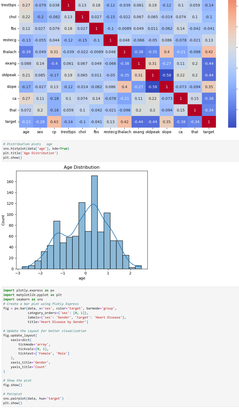
In [11]:
```
# Data numerics features if necessary
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']] = scaler.fit_transform(data[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']])
```

In [12]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Correlation matrix
corr = data.corr()
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.show()
```



In [13]:
```
# Distribution plots - age
sns.histplot(data['age'], kde=True)
plt.title('Age Distribution')
plt.show()
```

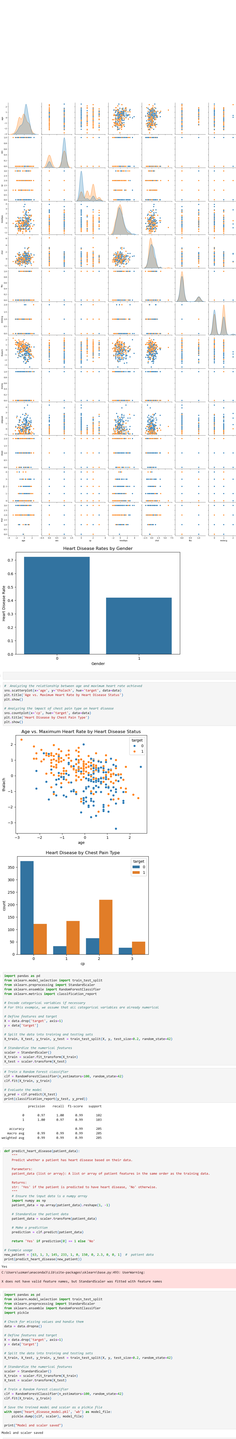


In [14]:
```
import plotly.express as px
import plotly.figure_factory as ff
import numpy as np
# Create a bar plot using Plotly Express
fig = px.bar(data, x='sex', y='target', barmode='group',
             category_orders={'sex': [0, 1]},
             labels={'sex': 'Gender', 'target': 'Heart Disease'},
             title='Heart Disease by Gender')

# Update the layout for better visualization
fig.update_layout(
    xaxis=dict(
        tickmode='array',
        tickvals=[0, 1],
        ticktext=['Female', 'Male']
    ),
    xaxis_title='Gender',
    yaxis_title='Count'
)

# Show the plot
fig.show()

# Pairplot
sns.pairplot(data, hue='target')
plt.show()
```



In [15]:
```
# Analyzing the relationship between age and maximum heart rate achieved
sns.scatterplot(x='age', y='thalach', hue='target', data=data)
plt.title('Age vs. Maximum Heart Rate by Heart Disease Status')
plt.show()

# Analyzing the impact of chest pain type on heart disease
sns.countplot(x='cp', hue='target', data=data)
plt.title('Heart Disease by Chest Pain Type')
plt.show()
```


In [16]:
```
# Analyzing the resting blood pressure
sns.boxplot(x='target', y='trestbps', data=data)
plt.title('Resting Blood Pressure by Heart Disease Status')
plt.show()

# Analyzing the number of major vessels colored by fluoroscopy
sns.countplot(x='ca', hue='target', data=data)
plt.title('Number of Major Vessels Colored by Fluoroscopy')
plt.show()
```


In [17]:
```
# Analyzing the relationship between cholesterol levels and heart disease
sns.boxplot(x='target', y='chol', data=data)
plt.title('Cholesterol Levels by Heart Disease Status')
plt.show()

# Analyzing the distribution of ST depression induced by exercise relative to rest
sns.histplot(data=data, x='oldpeak', hue='target', kde=True)
plt.title('ST Depression Induced by Exercise Relative to Rest')
plt.show()
```


In [18]:
```
# Analyzing the slope of the peak exercise ST segment
sns.countplot(x='slope', hue='target', data=data)
plt.title('Slope of the Peak Exercise ST Segment')
plt.show()

# Analyzing the number of major vessels colored by fluoroscopy
sns.countplot(x='ca', hue='target', data=data)
plt.title('Number of Major Vessels Colored by Fluoroscopy')
plt.show()
```


In [19]:
```
# Analyzing the thalassemia
sns.countplot(x='thal', hue='target', data=data)
plt.title('Thalassemia')
plt.show()

# Analyzing the target variable
sns.countplot(x='target', data=data)
plt.title('Target Variable')
plt.show()
```


In [20]:
```
# Analyzing the relationship between age and maximum heart rate achieved
sns.scatterplot(x='age', y='thalach', hue='target', data=data)
plt.title('Age vs. Maximum Heart Rate by Heart Disease Status')
plt.show()

# Analyzing the impact of chest pain type on heart disease
sns.countplot(x='cp', hue='target', data=data)
plt.title('Heart Disease by Chest Pain Type')
plt.show()
```


In [21]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Encode categorical variables if necessary
# For this example, we assume that all categorical variables are already numerical

# Define features and target
X = data.drop('target', axis=1)
y = data['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Train a Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Evaluate the model
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       102
           1       1.00      0.97      0.99       103

    accuracy                           0.99       205
   macro avg       0.99      0.99      0.99       205
weighted avg       0.99      0.99      0.99       205
```

In [22]:
```
def predict_heart_disease(patient_data):
    """
    Predict whether a patient has heart disease based on their data.

    Parameters:
    patient_data (list or array): A list or array of patient features in the same order as the training data.

    Returns:
    str: 'Yes' if the patient is predicted to have heart disease, 'No' otherwise.
    """
    # Ensure the input data is a numpy array
    import numpy as np
    patient_data = np.array(patient_data).reshape(1, -1)

    # Standardize the patient data
    patient_data = scaler.transform(patient_data)

    # Make a prediction
    prediction = clf.predict(patient_data)

    return 'Yes' if prediction[0] == 1 else 'No'

# Example usage
new_patient = [62, 1, 3, 145, 233, 1, 0, 150, 0, 2.3, 0, 0, 1]  # patient data
print("Predicted Heart Disease:", predict_heart_disease(new_patient))
```

```
Yes
```

```
C:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\base.py:493: UserWarning:

X does not have valid feature names, but StandardScaler was fitted with feature names
```

In [23]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
import pickle

# Check for missing values and handle them
data = data.dropna()

# Define features and target
X = data.drop('target', axis=1)
y = data['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Train a Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Save the trained model and scaler to a pickle file
with open('heart_disease_model.pkl', 'wb') as model_file:
    pickle.dump(clf, model_file)

print("Model and scaler saved")
```

```
Model and scaler saved
```

In [ ]:
```

```