jupyter Heart Disease Predictor Project Last Checkpoint: Last Tuesday at 11:15 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3 (ipykernel)

Run Code

In [1]:
```
# Group Names
```

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

In [3]:
```
df = pd.read_csv("heart.csv")
```

In [4]:
```
df
```

Out[4]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

918 rows × 12 columns

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

Out[5]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

In [6]:
```
df.tail(5)
```

Out[6]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

In [7]:
```
df.shape
```

Out[7]: (918, 12)

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

In [10]:
```
df.describe()
```

Out[10]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

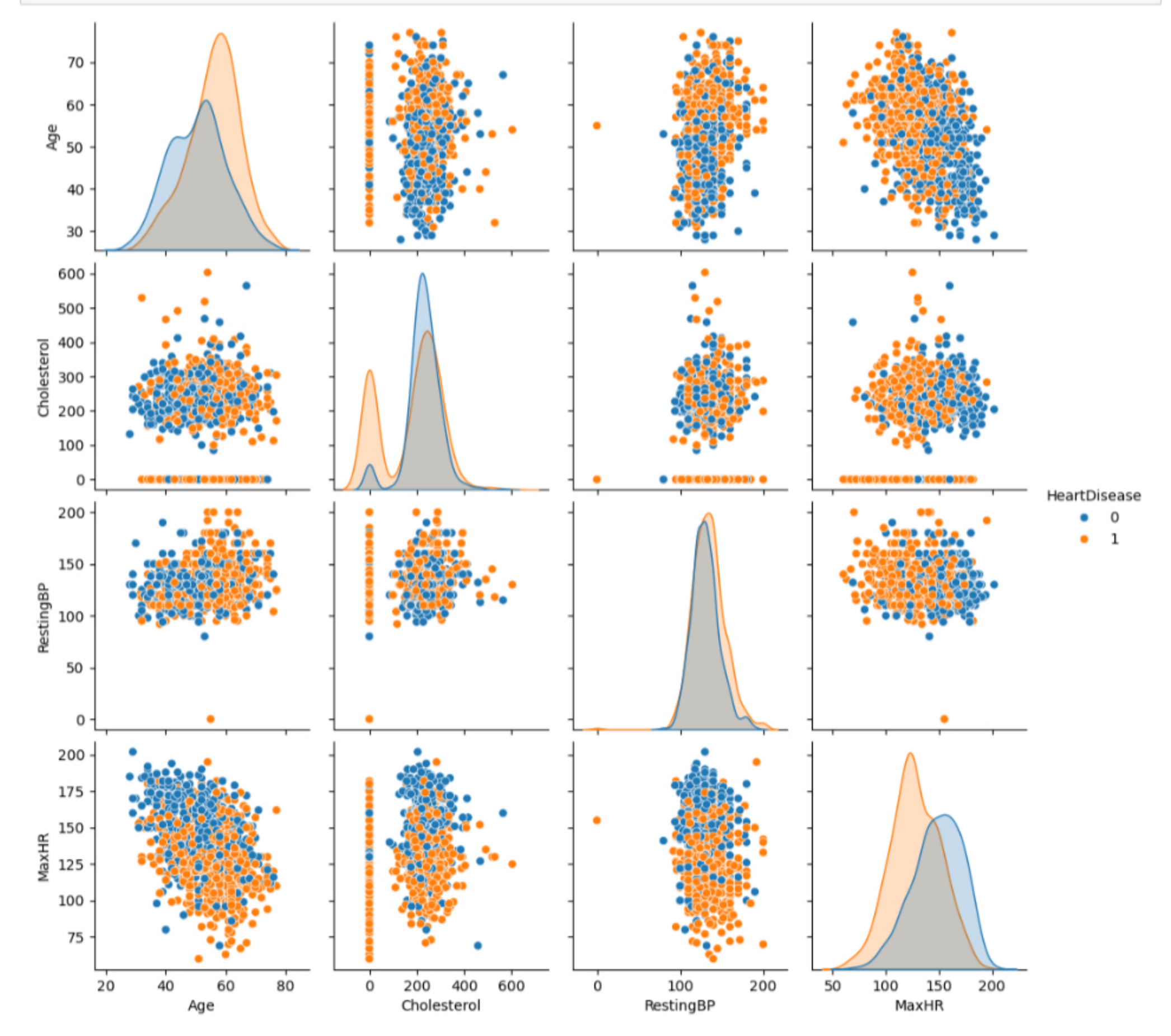
In [11]:
```
# Bar chart for Sex count
fig = px.bar(
    df['Sex'].value_counts().reset_index(),
    x='index',
    y='Sex',
    title="Count of Males vs. Females",
    labels={'index': 'Sex', 'Sex': 'Count'},
    color='index'
)
fig.show()
```

Based On this Data set, Males are 4 times more likely to have a Heart Disease over Females.

In [12]:
```
fig = px.bar(
    df['ChestPainType'].value_counts().reset_index(),
    x='index',
    y='ChestPainType',
    title="Frequency of Different Chest Pain Types",
    labels={'index': 'Chest Pain Type', 'ChestPainType': 'Count'},
    color='index'
)
fig.show()
```

ASY is the most common type of Chest Pain Type

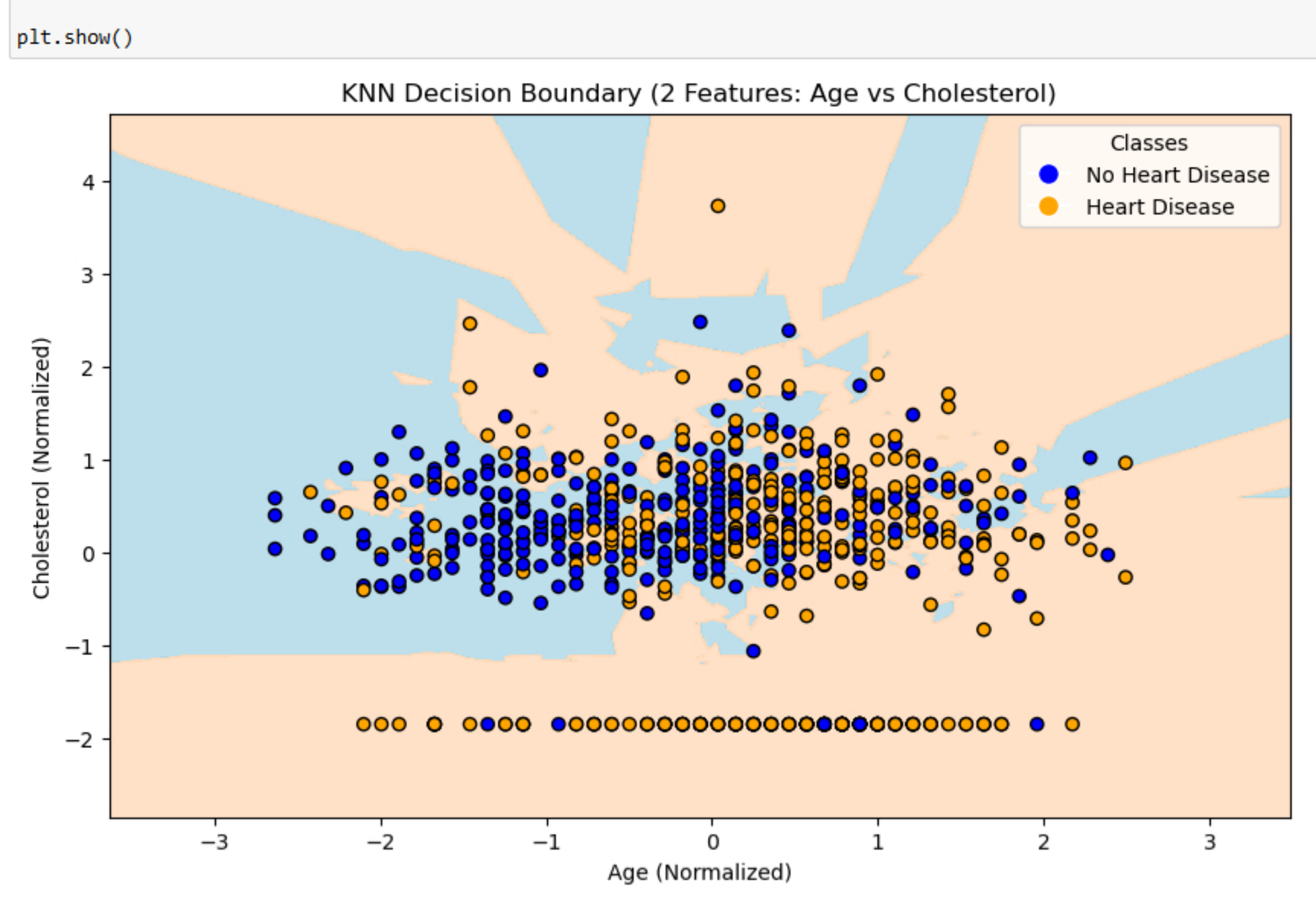
In [15]:
```
sns.pairplot(df, hue='HeartDisease', diag_kind='kde', vars=['Age', 'Cholesterol', 'RestingBP', 'MaxHR'])
plt.show()
```



In [13]:
```
# Example: Chest Pain Type vs. Heart Disease
fig = px.bar(
    df, x='ChestPainType', color='HeartDisease',
    title="Chest Pain Type vs. Heart Disease",
    barmode='group'
)
fig.show()
```

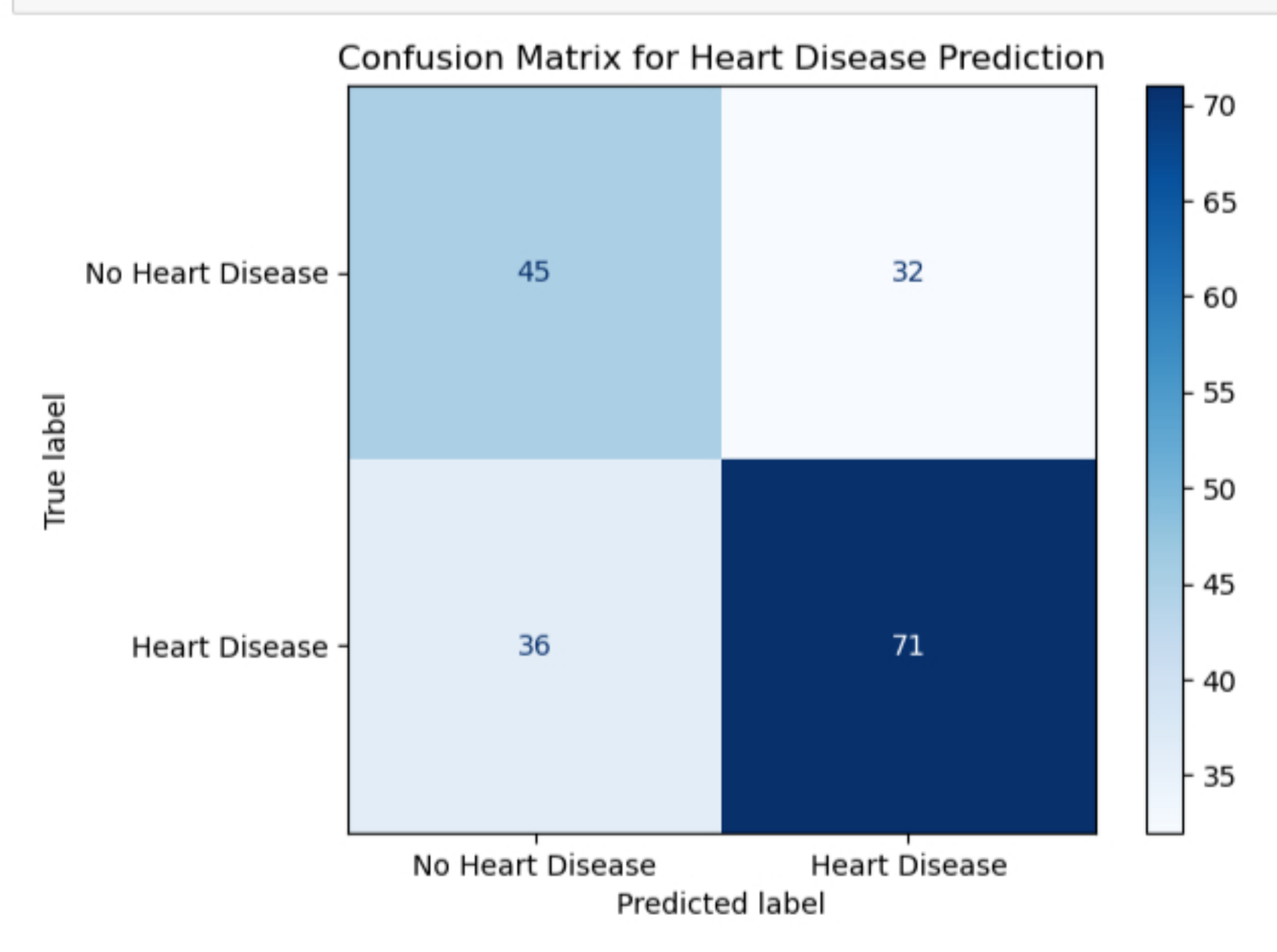
ASY is also the most ChestPain, People with Hear disease Experience ASY the most and ....

In [22]:
```
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

# Plot the decision boundary
plt.figure(figsize=(10, 6))
colors = ListedColormap(["lightblue", "peachpuff"])
scatter_colors = ListedColormap(["blue", "orange"])
plt.contourf(xx, yy, Z, alpha=0.8, cmap=colors)
scatter = plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolor='k', cmap=scatter_colors)
plt.title("KNN Decision Boundary (2 Features: Age vs Cholesterol)")
plt.xlabel("Age (Normalized)")
plt.ylabel("Cholesterol (Normalized)")

# Add legend
legend_labels = {0: "No Heart Disease", 1: "Heart Disease"}
handles = [plt.Line2D([0], [0], marker='o', color='w', label=legend_labels[key],
                      markerfacecolor=scatter_colors.colors[key], markersize=10)
           for key in legend_labels]
plt.legend(handles=handles, title="Classes")

plt.show()
```



Decision Tree

In [27]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier  # or use your decision tree model here
import pandas as pd


# Select relevant features for classification
X = df[['Age', 'Cholesterol']]  # Adjust columns based on your dataset
y = df['HeartDisease']  # Adjust target column

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

# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train KNN or Decision Tree
knn = KNeighborsClassifier(n_neighbors=5)  # Replace with decision tree model if needed
knn.fit(X_train, y_train)

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

# Generate confusion matrix
confusion_dt2 = confusion_matrix(y_test, y_pred)

# Display confusion matrix
disp2 = ConfusionMatrixDisplay(confusion_matrix=confusion_dt2, display_labels=["No Heart Disease", "Heart Disease"])
disp2.plot(cmap="Blues")
plt.title("Confusion Matrix for Heart Disease Prediction")
plt.show()
```



In [46]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression  # Import Logistic Regression
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming your dataframe 'df' is already loaded


# Convert categorical columns to numeric using LabelEncoder (if needed)
label_encoder = LabelEncoder()

# Assuming 'ExerciseAngina', 'FastingBS', 'ST_Slope' are categorical columns in your dataset
df['ExerciseAngina'] = label_encoder.fit_transform(df['ExerciseAngina'])
df['FastingBS'] = label_encoder.fit_transform(df['FastingBS'])
df['ST_Slope'] = label_encoder.fit_transform(df['ST_Slope'])


# Select relevant features and target variable
X = df[['Age', 'Cholesterol', 'RestingBP', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope', 'FastingBS']]
y = df['HeartDisease']  # Assuming 'HeartDisease' is the target column

# 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=42)

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

# Initialize the models
knn = KNeighborsClassifier(n_neighbors=5)
dt = DecisionTreeClassifier(random_state=42)
logreg = LogisticRegression(random_state=42)  # Initialize Logistic Regression model

# Train the models
knn.fit(X_train, y_train)
dt.fit(X_train, y_train)
logreg.fit(X_train, y_train)  # Train Logistic Regression

# Predict with KNN, Decision Tree, and Logistic Regression
y_pred_knn = knn.predict(X_test)
y_pred_dt = dt.predict(X_test)
y_pred_logreg = logreg.predict(X_test)  # Logistic Regression prediction

# Print accuracy for each model
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn):.2f}")
print(f"Decision Tree Accuracy: {accuracy_score(y_test, y_pred_dt):.2f}")
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_logreg):.2f}")

# To make predictions for a new person (example input):
new_person = pd.DataFrame({
    'Age': [50],           # Example: 50 years old
    'Cholesterol': [140],  # Example: 140 mg/dL cholesterol (healthy range)
    'RestingBP': [130],    # Example: 130 mmHg resting blood pressure (healthy range)
    'MaxHR': [140],        # Example: 140 beats per minute max heart rate (healthy range)
    'ExerciseAngina': [1], # 1 for 'Y' (Exercise induced angina: Yes)
    'Oldpeak': [1.5],      # Example: 1.5 depression induced by exercise (indicates mild symptoms)
    'ST_Slope': [1],       # 1, 2, or 3 depending on the slope (ST-segment slope)
    'FastingBS': [1]       # 1 for 'Y' (fasting blood sugar: Yes)
})

# Apply the same transformations
new_person_scaled = scaler.transform(new_person)

# Predict using all models
knn_prediction = knn.predict(new_person_scaled)
dt_prediction = dt.predict(new_person_scaled)
logreg_prediction = logreg.predict(new_person_scaled)  # Logistic Regression prediction

# Get the probabilities for each model
knn_probabilities = knn.predict_proba(new_person_scaled)
dt_probabilities = dt.predict_proba(new_person_scaled)
logreg_probabilities = logreg.predict_proba(new_person_scaled)  # Logistic Regression probabilities

# Print predictions and probabilities for KNN
print(f"KNN Prediction (0: No Heart Disease, 1: Heart Disease): {knn_prediction[0]}")
print(f"KNN Probability for Heart Disease: {knn_probabilities[0][0] * 100:.2f}%")
print(f"KNN Probability for No Heart Disease: {knn_probabilities[0][1] * 100:.2f}%")
# Print predictions and probabilities for Decision Tree
print(f"Decision Tree Prediction (0: No Heart Disease, 1: Heart Disease): {dt_prediction[0]}")
print(f"Decision Tree Probability for No Heart Disease: {dt_probabilities[0][0] * 100:.2f}%")
print(f"Decision Tree Probability for Heart Disease: {dt_probabilities[0][1] * 100:.2f}%")

# Print predictions and probabilities for Logistic Regression
print(f"Logistic Regression Prediction (0: No Heart Disease, 1: Heart Disease): {logreg_prediction[0]}")
print(f"Logistic Regression Probability for No Heart Disease: {logreg_probabilities[0][0] * 100:.2f}%")
print(f"Logistic Regression Probability for Heart Disease: {logreg_probabilities[0][1] * 100:.2f}%")
```

```
KNN Accuracy: 0.82
Decision Tree Accuracy: 0.81
Logistic Regression Accuracy: 0.79
KNN Prediction (0: No Heart Disease, 1: Heart Disease): 1
KNN Probability for Heart Disease: 0.00%
KNN Probability for No Heart Disease: 100.00%
Decision Tree Prediction (0: No Heart Disease, 1: Heart Disease): 1
Decision Tree Probability for No Heart Disease: 0.00%
Decision Tree Probability for Heart Disease: 100.00%
Logistic Regression Prediction (0: No Heart Disease, 1: Heart Disease): 1
Logistic Regression Probability for No Heart Disease: 4.16%
Logistic Regression Probability for Heart Disease: 95.84%
```

In [ ]: