In [325…
```
## Importing Libraries

import numpy as np  # Linear algebra operations
import pandas as pd  # Data processing and analysis
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC

import warnings
warnings.filterwarnings("ignore")
```

In [32]:
```
## Upload dataset

df = pd.read_csv('/Users/serenaygoler/heart disease.csv')

df.head() # Displays the first 5 rows.
```

Out[32]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR
0	40	M	ATA	140	289	0	Normal	172
1	49	F	NAP	160	180	0	Normal	156
2	37	M	ATA	130	283	0	ST	98
3	48	F	ASY	138	214	0	Normal	108
4	54	M	NAP	150	195	0	Normal	122

In [34]:
```
df.tail() # Display the last 5 rows.
```

Out[34]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxH
913	45	M	TA	110	264	0	Normal	13
914	68	M	ASY	144	193	1	Normal	14
915	57	M	ASY	130	131	0	Normal	1
916	57	F	ATA	130	236	0	LVH	17
917	38	M	NAP	138	175	0	Normal	17

In [36]:
```
df.info() # Prints name and type of variables, number of observations, and c
```

```
<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 [38]:
```
df.shape # Displays the number of rows and columns in the dataset.
```

Out[38]:
```
(918, 12)
```

In [40]:
```
df.isna().sum() # Counts missing values in each column.
```

Out[40]:
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

In [42]:
```
df.duplicated().sum() # Counts the number of duplicate rows.
```

Out[42]:
```
0
```

In [46]:
```
## Provides summary statistics for numeric columns, rounded to 2 decimals ar
df.describe().round(2).T
```

Out[46]:

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.51	9.43	28.0	47.00	54.0	60.0	77.0
RestingBP	918.0	132.40	18.51	0.0	120.00	130.0	140.0	200.0
Cholesterol	918.0	198.80	109.38	0.0	173.25	223.0	267.0	603.0
FastingBS	918.0	0.23	0.42	0.0	0.00	0.0	0.0	1.0
MaxHR	918.0	136.81	25.46	60.0	120.00	138.0	156.0	202.0
Oldpeak	918.0	0.89	1.07	-2.6	0.00	0.6	1.5	6.2
HeartDisease	918.0	0.55	0.50	0.0	0.00	1.0	1.0	1.0

In [48]:
```
# Count how many Cholesterol values are zero
chol_zero_count = (df["Cholesterol"] == 0).sum()

# Count how many RestingBP values are zero
bp_zero_count = (df["RestingBP"] == 0).sum()

print(f"Number of Cholesterol values equal to 0: {chol_zero_count}")
print(f"Number of RestingBP values equal to 0: {bp_zero_count}")
```

```
Number of Cholesterol values equal to 0: 172
Number of RestingBP values equal to 0: 1
```

In [50]:
```
# Cross-tabulate Cholesterol = 0 with HeartDisease status
import pandas as pd

zero_chol = df[df["Cholesterol"] == 0]
ct = pd.crosstab(zero_chol["HeartDisease"], zero_chol["Cholesterol"])
print(ct)
```

```
Cholesterol     0
HeartDisease
0              20
1             152
```

In [54]:
```
## Filters out rows where Cholesterol equals zero and returns summary statis

print(df[df["Cholesterol"] != 0]["Cholesterol"].describe())
```

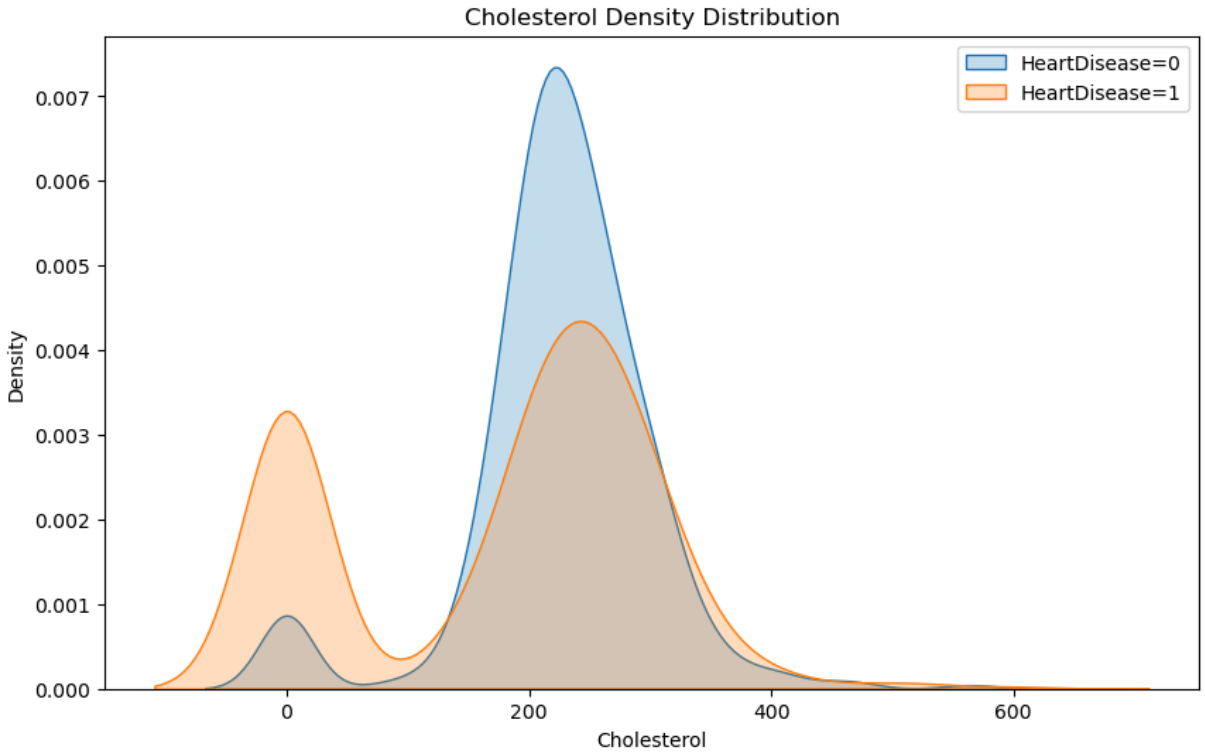
```
count    746.000000
mean     244.635389
std       59.153524
min       85.000000
25%      207.250000
50%      237.000000
75%      275.000000
max      603.000000
Name: Cholesterol, dtype: float64
```

In [56]:
```
# With zeros included
print("=== With Zero values Included ===")
print(df.groupby("HeartDisease")["Cholesterol"].describe())
```

```
# Zeros removed
print("\n===  With zero values removed ===")
print(df[df["Cholesterol"] != 0].groupby("HeartDisease")["Cholesterol"].desc
```

```
=== With Zero values Included ===
               count        mean         std  min     25%    50%     75%    m
ax
HeartDisease
0              410.0  227.121951   74.634659  0.0  197.25  227.0  266.75  56
4.0
1              508.0  175.940945  126.391398  0.0    0.00  217.0  267.00  60
3.0

===  With zero values removed ===
               count        mean         std    min    25%    50%     75%    m
ax
HeartDisease
0              390.0  238.769231   55.394617   85.0  203.0  231.5  269.00  56
4.0
1              356.0  251.061798   62.462713  100.0  212.0  246.0  283.25  60
3.0
```

In [58]:
```
# Plot the cholesterol distribution for HeartDisease=0 and HeartDisease=1 us
plt.figure(figsize=(10,6))
sns.kdeplot(df[df["HeartDisease"]==0]["Cholesterol"], label="HeartDisease=0"
sns.kdeplot(df[df["HeartDisease"]==1]["Cholesterol"], label="HeartDisease=1"
plt.legend()
plt.title("Cholesterol Density Distribution")
plt.show()
```

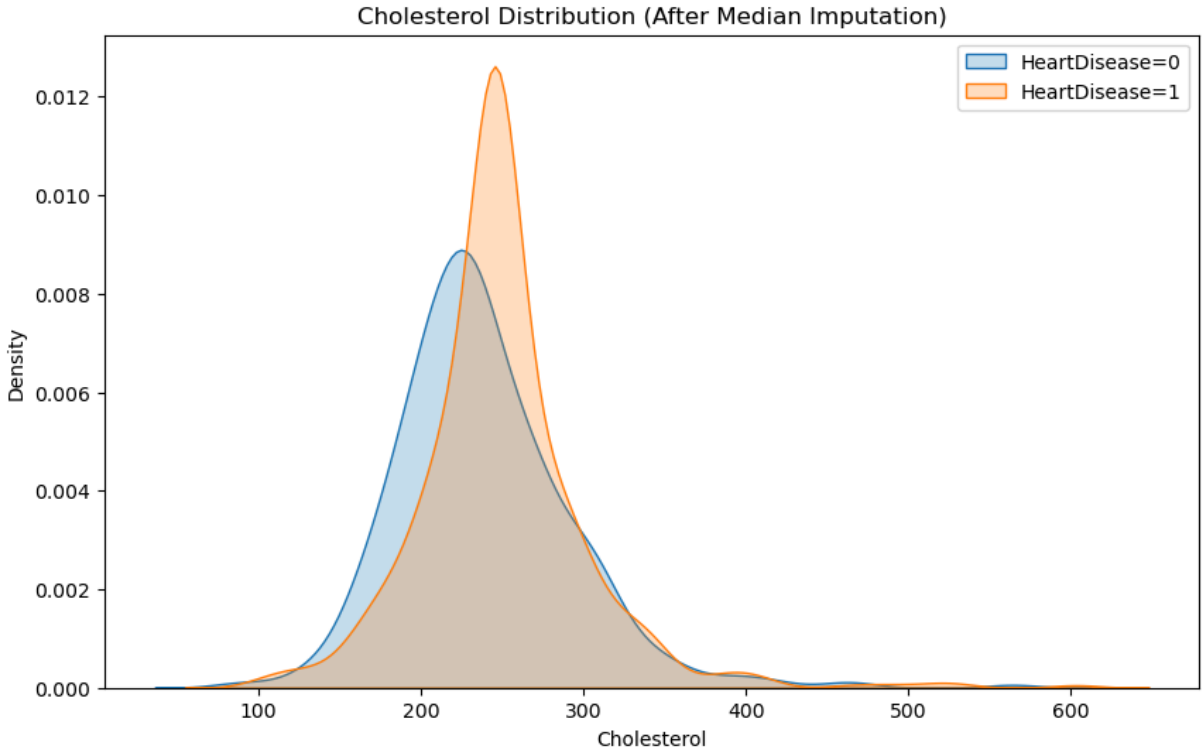


In [60]:
```
# This block cleans the dataset by:
# 1. Removing rows where RestingBP = 0 (unrealistic values).
# 2. Calculating group-wise medians of Cholesterol (by HeartDisease) excludi
```

```
# 3. Replacing Cholesterol values of zero with the corresponding group media
# 4. Checking that no zero values remain.
# 5. Displaying summary statistics of Cholesterol by HeartDisease after clea

df_clean = df.copy()
df_clean = df_clean[df_clean["RestingBP"] != 0].copy()

medians = (
    df_clean[df_clean["Cholesterol"] != 0]
    .groupby("HeartDisease")["Cholesterol"]
    .median()
)

mask_zero = df_clean["Cholesterol"] == 0
df_clean["Cholesterol"] = df_clean["Cholesterol"].astype(float)
df_clean.loc[mask_zero, "Cholesterol"] = (
    df_clean.loc[mask_zero, "HeartDisease"].map(medians)
)

print("Remaining zeros:", (df_clean["Cholesterol"] == 0).sum())
print(df_clean.groupby("HeartDisease")["Cholesterol"].describe())
```

```
Remaining zeros: 0
              count        mean        std    min    25%    50%     75%    m
ax
HeartDisease
0             410.0  238.414634  54.045994   85.0  204.0  231.5  266.75  56
4.0
1             507.0  249.554241  52.370323  100.0  225.0  246.0  267.00  60
3.0
```

In [62]:
```
# KDE plot – distribution comparison after median imputation
plt.figure(figsize=(10,6))
sns.kdeplot(df_clean[df_clean["HeartDisease"]==0]["Cholesterol"], label="Hea
sns.kdeplot(df_clean[df_clean["HeartDisease"]==1]["Cholesterol"], label="Hea
plt.title("Cholesterol Distribution (After Median Imputation)")
plt.xlabel("Cholesterol")
plt.ylabel("Density")
plt.legend()
plt.show()
```



In [64]:
```
## Provides summary statistics for numeric columns for clean data, rounded t
df_clean.describe().round(2).T
```

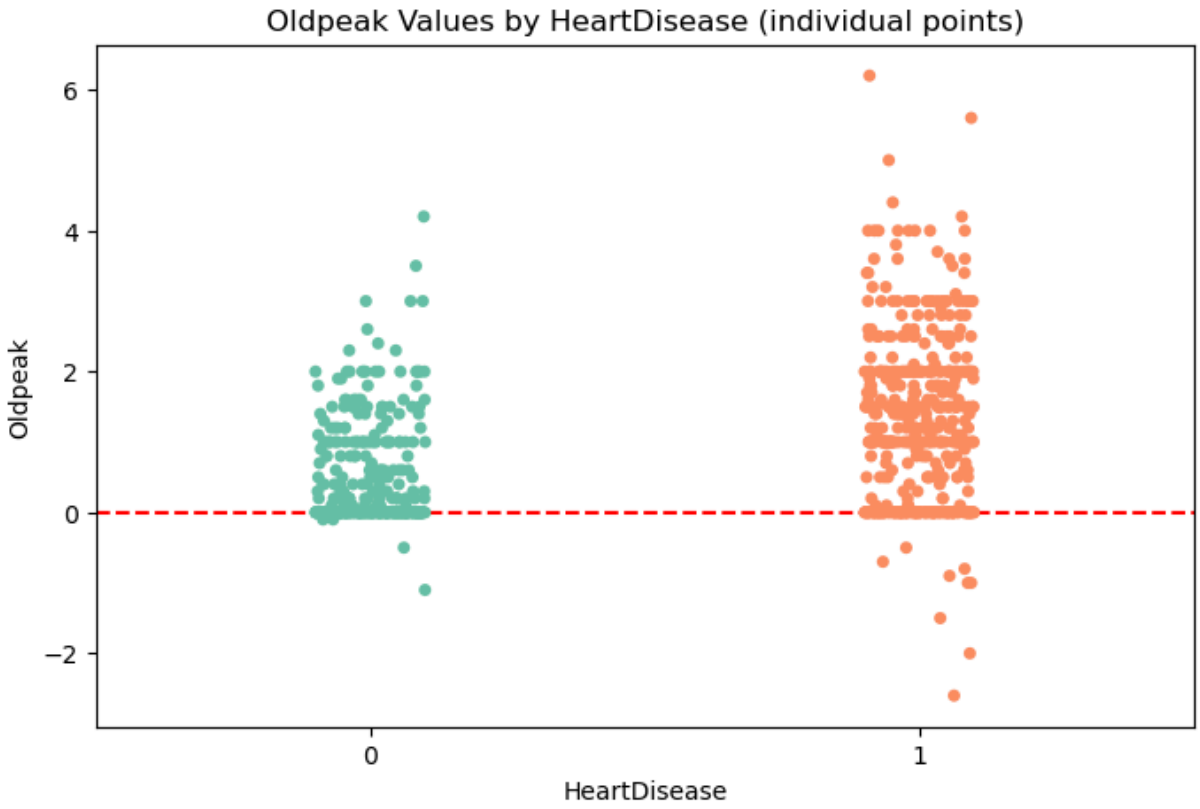
Out[64]:

	count	mean	std	min	25%	50%	75%	max
Age	917.0	53.51	9.44	28.0	47.0	54.0	60.0	77.0
RestingBP	917.0	132.54	18.00	80.0	120.0	130.0	140.0	200.0
Cholesterol	917.0	244.57	53.39	85.0	214.0	246.0	267.0	603.0
FastingBS	917.0	0.23	0.42	0.0	0.0	0.0	0.0	1.0
MaxHR	917.0	136.79	25.47	60.0	120.0	138.0	156.0	202.0
Oldpeak	917.0	0.89	1.07	-2.6	0.0	0.6	1.5	6.2
HeartDisease	917.0	0.55	0.50	0.0	0.0	1.0	1.0	1.0

In [66]:
```
# Count negative Oldpeak values

neg_oldpeak_count = (df["Oldpeak"] < 0).sum()
print(f"Number of negative Oldpeak values: {neg_oldpeak_count}")
```

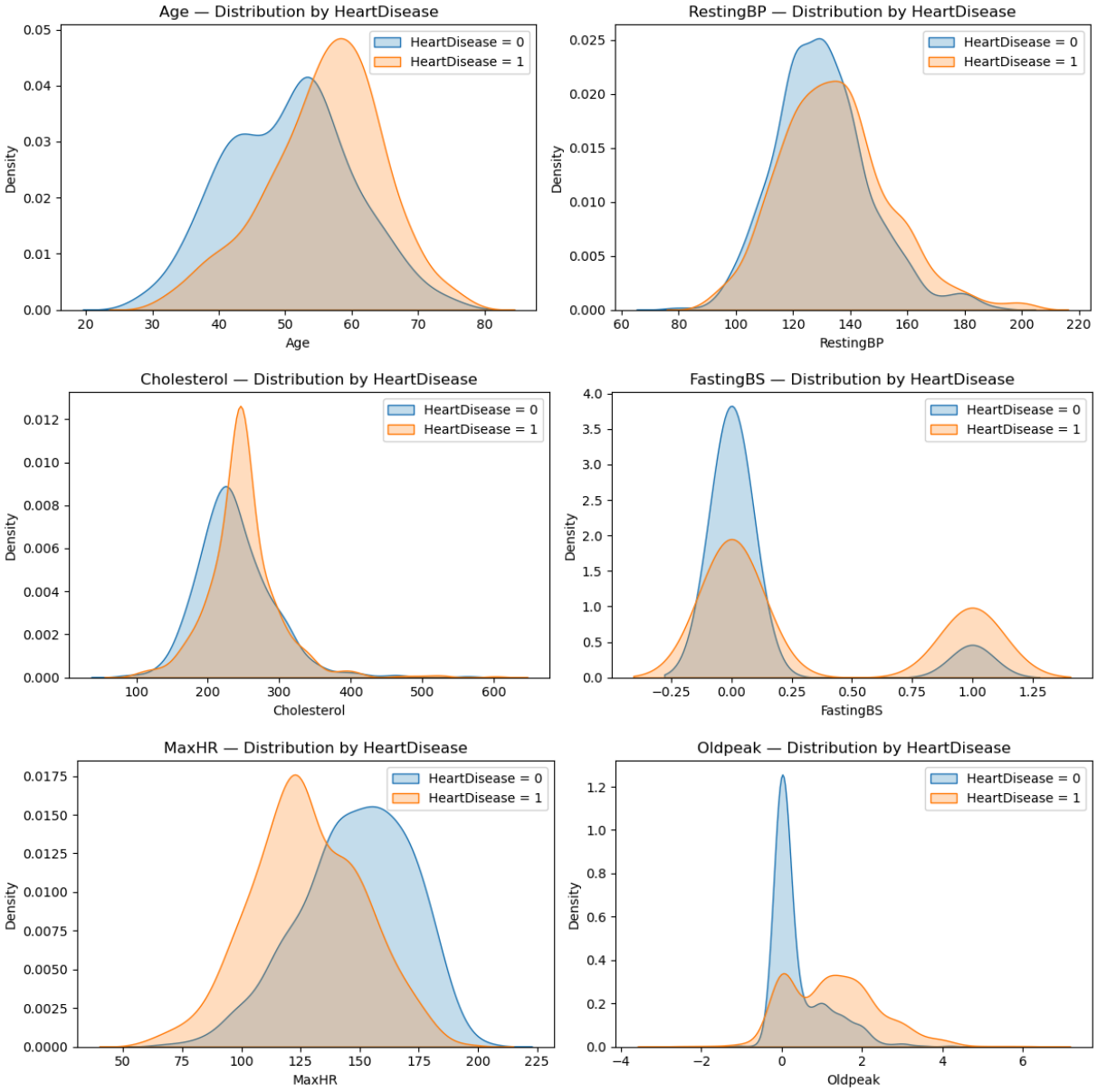
```
Number of negative Oldpeak values: 13
```

In [68]:
```
# Stripplot showing distribution of Oldpeak values by HeartDisease, with ref

plt.figure(figsize=(8,5))
sns.stripplot(x="HeartDisease", y="Oldpeak", data=df, jitter=True, palette="
plt.axhline(0, color="red", linestyle="--")
plt.title("Oldpeak Values by HeartDisease (individual points)")
plt.show()
```

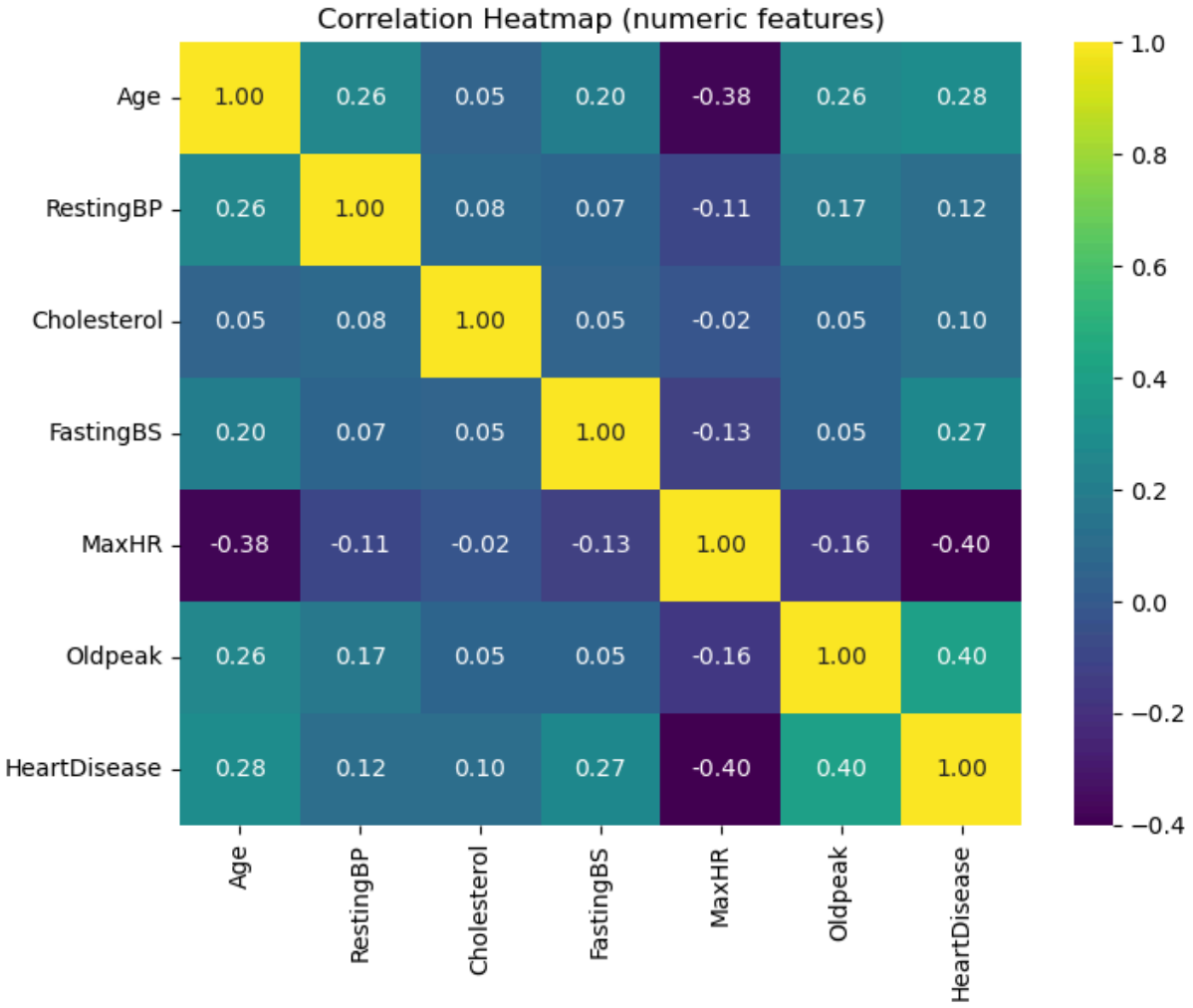


In [453…

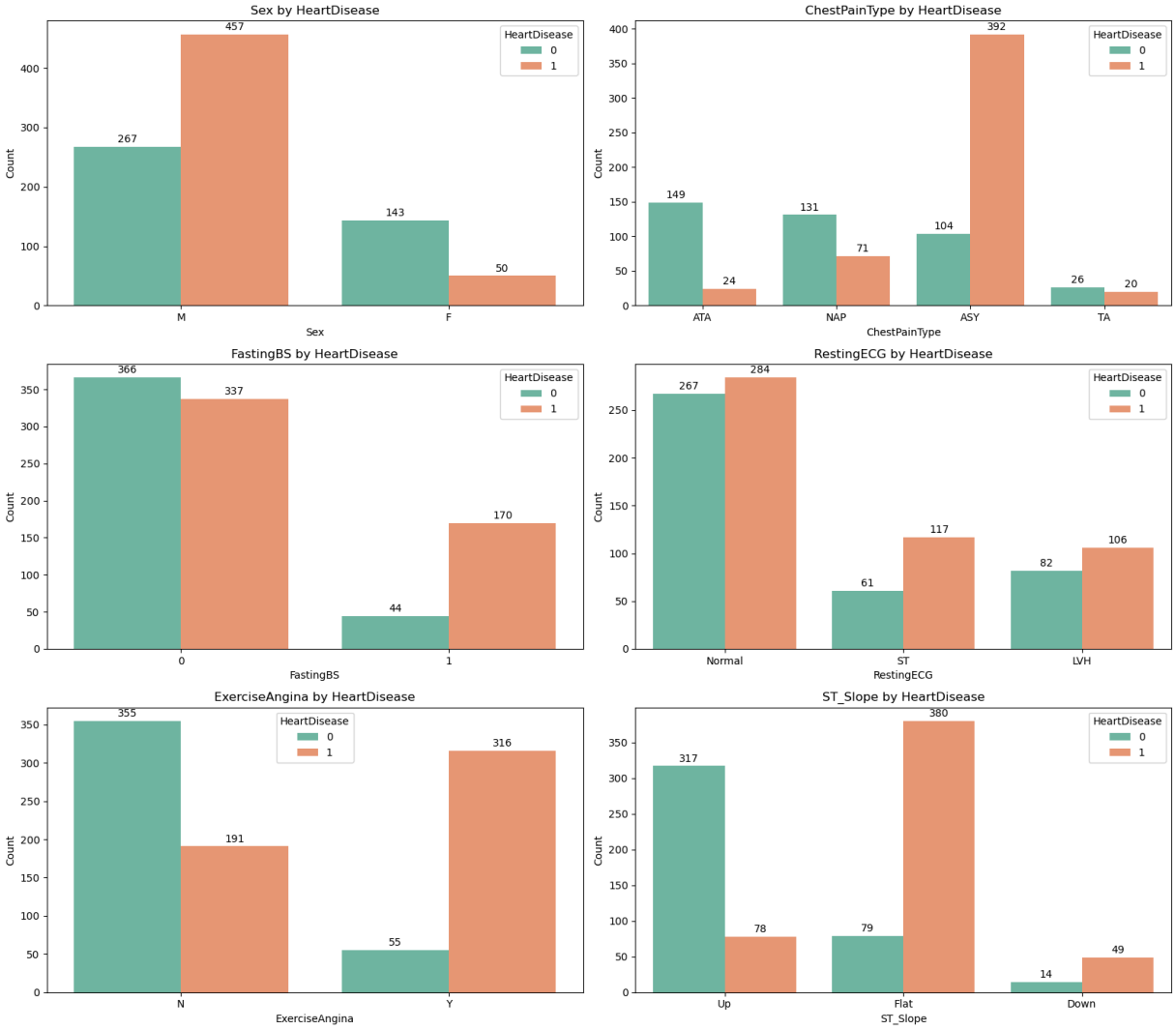
```
# Plot numeric feature distributions by target, two-at-a-time
num_cols = df_clean.select_dtypes(include="number").columns.drop("HeartDisea
cols = list(num_cols)

for i in range(0, len(cols), 2):
    pair = cols[i:i+2]  # up to 2 columns per figure

    fig, axes = plt.subplots(1, len(pair), figsize=(12, 4))
    if len(pair) == 1:
        axes = [axes]  # make iterable if only one axis

    for ax, col in zip(axes, pair):
        sns.kdeplot(
            df_clean.loc[df_clean["HeartDisease"] == 0, col].dropna(),
            label="HeartDisease = 0", fill=True, ax=ax
        )
        sns.kdeplot(
            df_clean.loc[df_clean["HeartDisease"] == 1, col].dropna(),
            label="HeartDisease = 1", fill=True, ax=ax
        )
        ax.set_title(f"{col} — Distribution by HeartDisease")
        ax.set_xlabel(col); ax.set_ylabel("Density")
        ax.legend()



    plt.tight_layout()
    plt.show()
```



In [72]:
```
# Select only numerical columns and to check correlation
num_cols = df_clean.select_dtypes(include=[np.number]).columns

plt.figure(figsize=(8,6))
sns.heatmap(df_clean[num_cols].corr(), annot=True, cmap="viridis", fmt=".2f"
plt.title("Correlation Heatmap (numeric features)")
plt.show()
```



In [74]:
```
# Distribution of categorical variables by the target variable
cat_cols = ["Sex", "ChestPainType", "FastingBS", "RestingECG", "ExerciseAngi
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(16, 14))
axes = axes.flatten()

for ax, col in zip(axes, cat_cols):
    g = sns.countplot(data=df_clean, x=col, hue="HeartDisease", palette="Set
    ax.set_title(f"{col} by HeartDisease")
    ax.set_xlabel(col); ax.set_ylabel("Count")
    # label name
    for c in g.containers:
        g.bar_label(c, padding=2, fmt="%.0f")

# Remove extra axes
for ax in axes[len(cat_cols):]:
    fig.delaxes(ax)

plt.tight_layout()
plt.show()
```

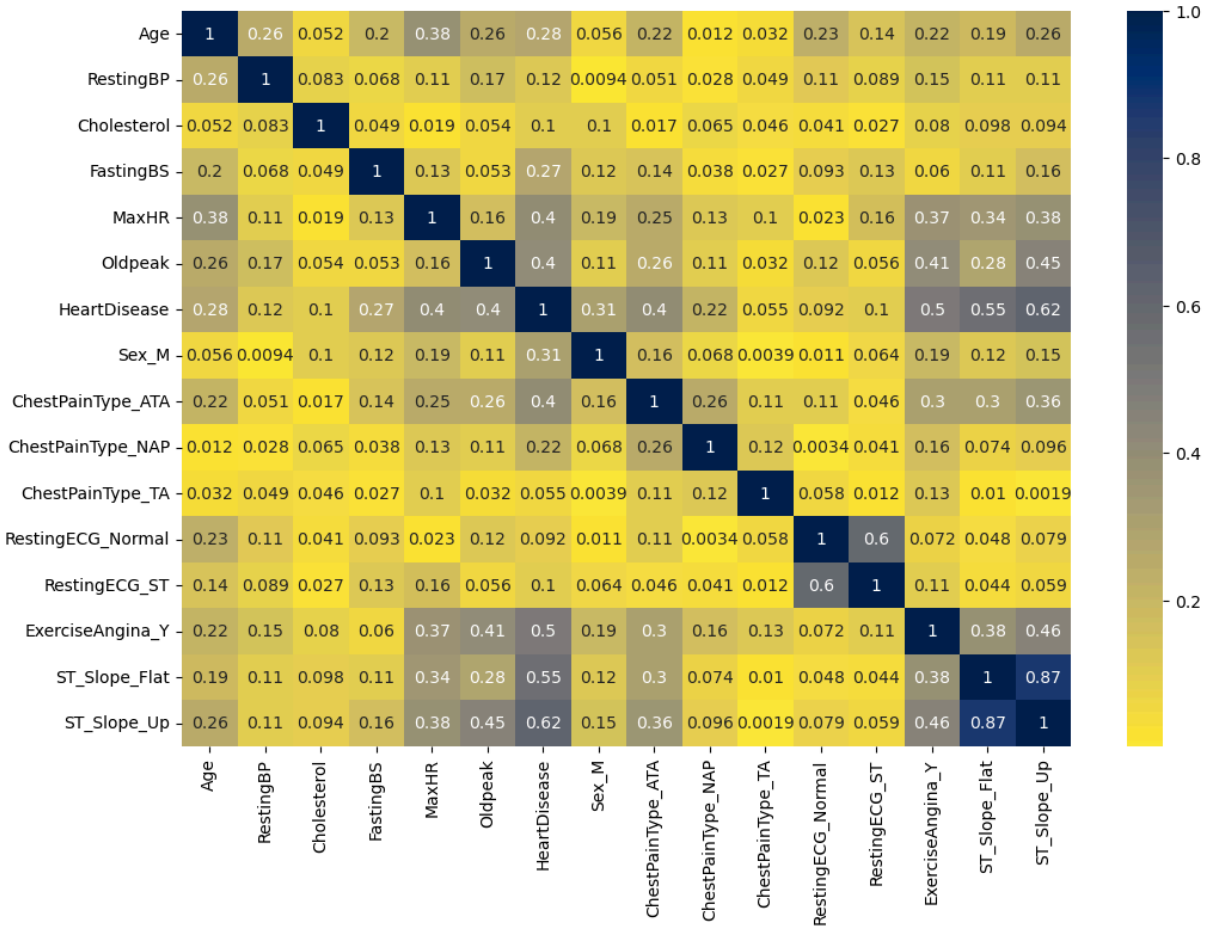


In [76]:
```
# One-hot encoding was applied to transform categorical variables into dummy
DUMMY = pd.get_dummies(df_clean, drop_first=True)
DUMMY.head()
```

Out[76]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease	Sex_M	C
0	40	140	289.0	0	172	0.0	0	True	
1	49	160	180.0	0	156	1.0	1	False	
2	37	130	283.0	0	98	0.0	0	True	
3	48	138	214.0	0	108	1.5	1	False	
4	54	150	195.0	0	122	0.0	0	True	

In [78]:
```
## Compute absolute pairwise correlations (after one-hot encoding) and visua
correlations = abs(DUMMY.corr())
plt.figure(figsize=(12,8))
sns.heatmap(correlations, annot=True, cmap="cividis_r")
plt.show()
```



In [395…

```
# Create a copy of the cleaned dataset
codedf = df_clean.copy()

# 1) Convert binary categorical columns into 0/1 format
if codedf['Sex'].dtype == 'object':
    codedf['Sex'] = codedf['Sex'].str.strip().map({'F': 0, 'M': 1}).astype('

if codedf['ExerciseAngina'].dtype == 'object':
    codedf['ExerciseAngina'] = codedf['ExerciseAngina'].str.strip().map({'N'

# (If they are already boolean True/False, convert them to integers)
for col in ['Sex', 'ExerciseAngina']:
    if codedf[col].dtype == 'bool':
        codedf[col] = codedf[col].astype(int)

# 2) Apply one-hot encoding for multi-class categorical columns
to_onehot = ['ChestPainType', 'RestingECG', 'ST_Slope']
codedf = pd.get_dummies(codedf, columns=to_onehot, drop_first=True)

# Convert any remaining boolean columns into 0/1 integers
for col in codedf.select_dtypes(include='bool').columns:
    codedf[col] = codedf[col].astype(int)

codedf.dtypes
```

Out[395…
```
Age                   int64
Sex                   Int64
RestingBP             int64
Cholesterol         float64
FastingBS             int64
MaxHR                 int64
ExerciseAngina        Int64
Oldpeak             float64
HeartDisease          int64
ChestPainType_ATA     int64
ChestPainType_NAP     int64
ChestPainType_TA      int64
RestingECG_Normal     int64
RestingECG_ST         int64
ST_Slope_Flat         int64
ST_Slope_Up           int64
dtype: object
```

In [399…
```
# Standardize continuous variables (mean = 0, std = 1)
# This ensures that all numeric predictors are on the same scale,
# which is especially important for distance-based algorithms (e.g., KNN, SV
numcolsc = ['Age','RestingBP','Cholesterol','MaxHR','Oldpeak']
scaler = StandardScaler()
codedf[numcolsc] = scaler.fit_transform(codedf[numcolsc])

codedf.head()
```

Out[399…

	Age	Sex	RestingBP	Cholesterol	FastingBS	MaxHR	ExerciseAngina	Ol
0	-1.432206	1	0.414627	0.832639	0	1.383339	0	-0.8
1	-0.478057	0	1.526360	-1.210238	0	0.754736	0	0.1
2	-1.750256	1	-0.141240	0.720187	0	-1.523953	0	-0.8
3	-0.584074	0	0.303453	-0.573010	0	-1.131075	1	0.5
4	0.052026	1	0.970493	-0.929108	0	-0.581047	0	-0.8

PREDICTION

In [401…
```
# Split the dataset into features (X) and target (y)
X = codedf.drop(columns=["HeartDisease"])
y = codedf["HeartDisease"]

# Train-test split: 80% training, 20% testing
# Stratify ensures the target class distribution (0/1) is preserved in both
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

In [403…
```
X_test.shape , y_test.shape # Check the shape of the test sets
```

Out[403…
```
((184, 15), (184,))
```

Logistic Regression

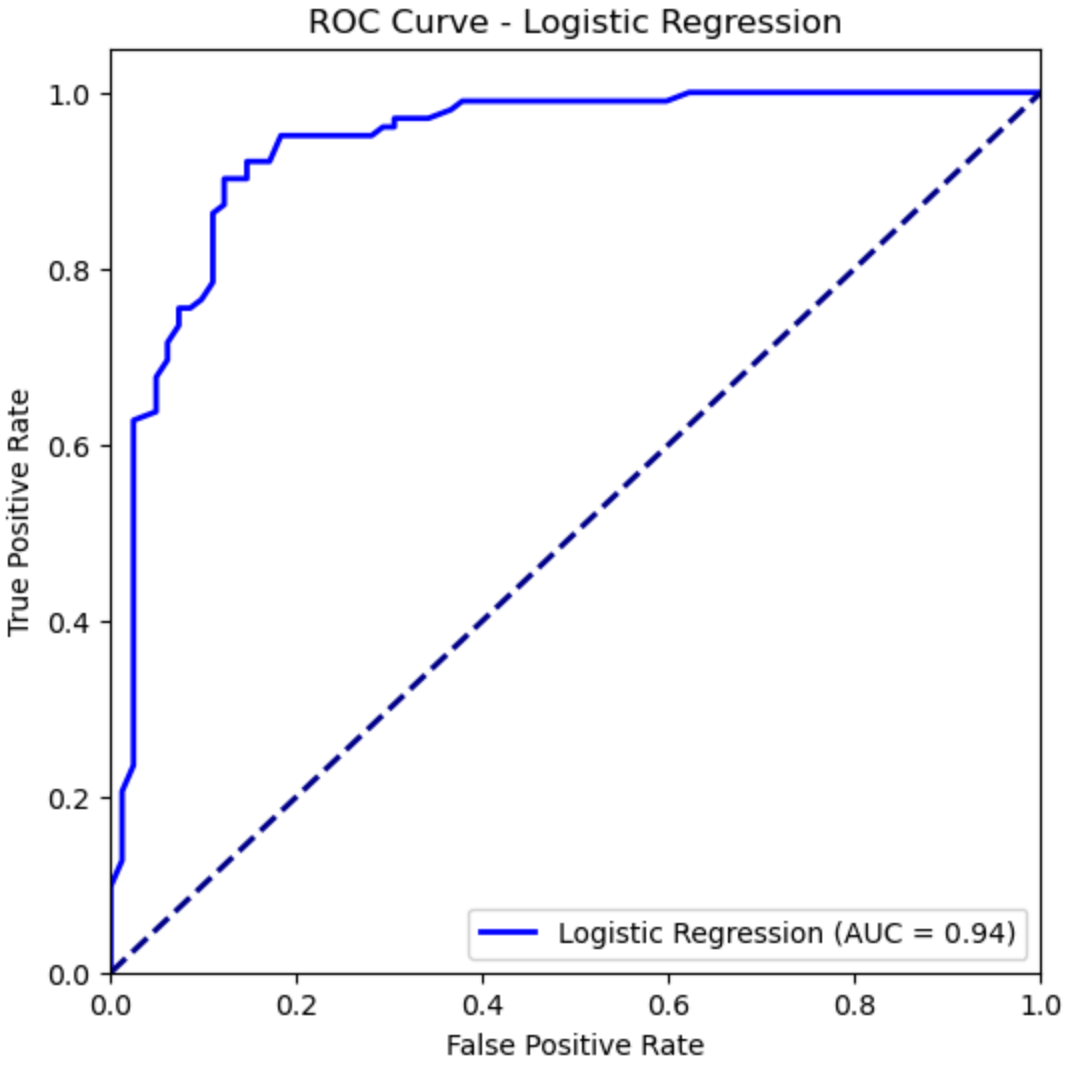
In [405…
```
# Logistic Regression Model
# max_iter=1000 ensures convergence during optimization
logreg = LogisticRegression(max_iter=1000)

# Train the model on the training set
logreg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = logreg.predict(X_test)

# Calculate accuracy of the model
logregAcc = accuracy_score(y_test, y_pred)
logregAcc
```

Out[405… 0.8858695652173914

In [407…
```
# Generate a detailed classification report
# Includes precision, recall, f1-score, and support for each class
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Predict probabilities for the positive class (1 = Heart Disease)
y_proba = logreg.predict_proba(X_test)[:, 1]

# Calculate the ROC AUC score to evaluate the model's discriminative ability
print("ROC AUC:", roc_auc_score(y_test, y_proba))
```

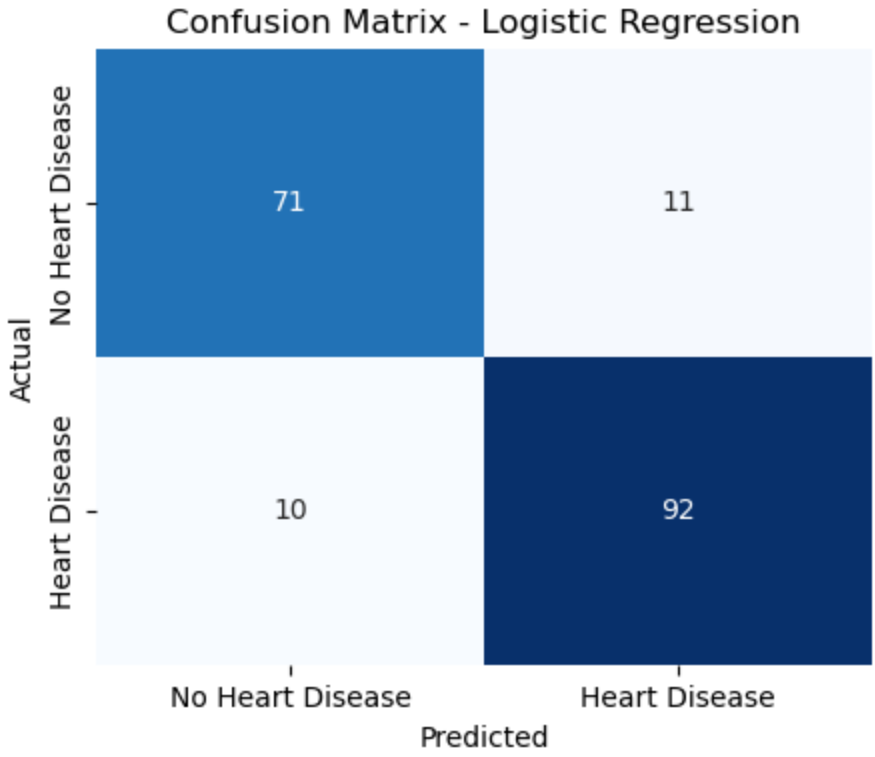
```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.87      0.87        82
           1       0.89      0.90      0.90       102

    accuracy                           0.89       184
   macro avg       0.88      0.88      0.88       184
weighted avg       0.89      0.89      0.89       184

ROC AUC: 0.9423720707795313
```

In [409…
```
# ROC CURVE
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='blue', lw=2,
         label='Logistic Regression (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='darkblue', lw=2, linestyle='--')
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 Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```



In [318…

```
## Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['No Heart Disease','Heart Disease'],
            yticklabels=['No Heart Disease','Heart Disease'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
```



In [411…

```
# Create a dataframe of Logistic Regression coefficients
# This shows the direction (+/-) and relative magnitude of each feature's ef
# Positive coefficients → increase likelihood of heart disease
# Negative coefficients → decrease likelihood of heart disease

coefficients = pd.DataFrame({
    'Feature': X_train.columns,
    'Coefficient': logreg.coef_[0]
}).sort_values(by='Coefficient', ascending=False)

coefficients
```

Out [411…

	Feature	Coefficient
1	Sex	1.358389
4	FastingBS	1.156245
13	ST_Slope_Flat	0.948000
6	ExerciseAngina	0.828883
7	Oldpeak	0.322847
3	Cholesterol	0.074785
0	Age	0.039656
2	RestingBP	-0.008738
12	RestingECG_ST	-0.173987
5	MaxHR	-0.268010
11	RestingECG_Normal	-0.309394
14	ST_Slope_Up	-1.280401
10	ChestPainType_TA	-1.329603
9	ChestPainType_NAP	-1.485096
8	ChestPainType_ATA	-1.554887

SUPPORT VECTOR MACHINE

In [413…

```
# Support Vector Machine (SVM) Models
# Train and evaluate both Linear and RBF kernel SVMs

# Linear SVM
svm_linear = SVC(kernel="linear", probability=True, random_state=1)
svm_linear.fit(X_train, y_train)
print("Linear SVM Accuracy:", accuracy_score(y_test, svm_linear.predict(X_te

# RBF SVM (non-linear)
svm_rbf = SVC(kernel="rbf", probability=True, random_state=1)
svm_rbf.fit(X_train, y_train)
print("RBF SVM Accuracy:", accuracy_score(y_test, svm_rbf.predict(X_test)))
```

```
Linear SVM Accuracy: 0.8586956521739131
RBF SVM Accuracy: 0.8858695652173914
```

In [415…

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Note:
# The ROC curve and confusion matrix visuals are not repeated here for SVM,
# as their performance and outputs were nearly identical to Logistic Regress
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.87      0.87        82
           1       0.89      0.90      0.90       102

    accuracy                           0.89       184
   macro avg       0.88      0.88      0.88       184
weighted avg       0.89      0.89      0.89       184
```

DECISION TREE

In [417…
```
# Build and train a Decision Tree model
clf = tree.DecisionTreeClassifier(random_state=0)  # reproducibility ensured
clf.fit(X_train, y_train)  # fit the model to training data

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy on the test set
clfAcc = accuracy_score(y_test, y_pred)
clfAcc
```

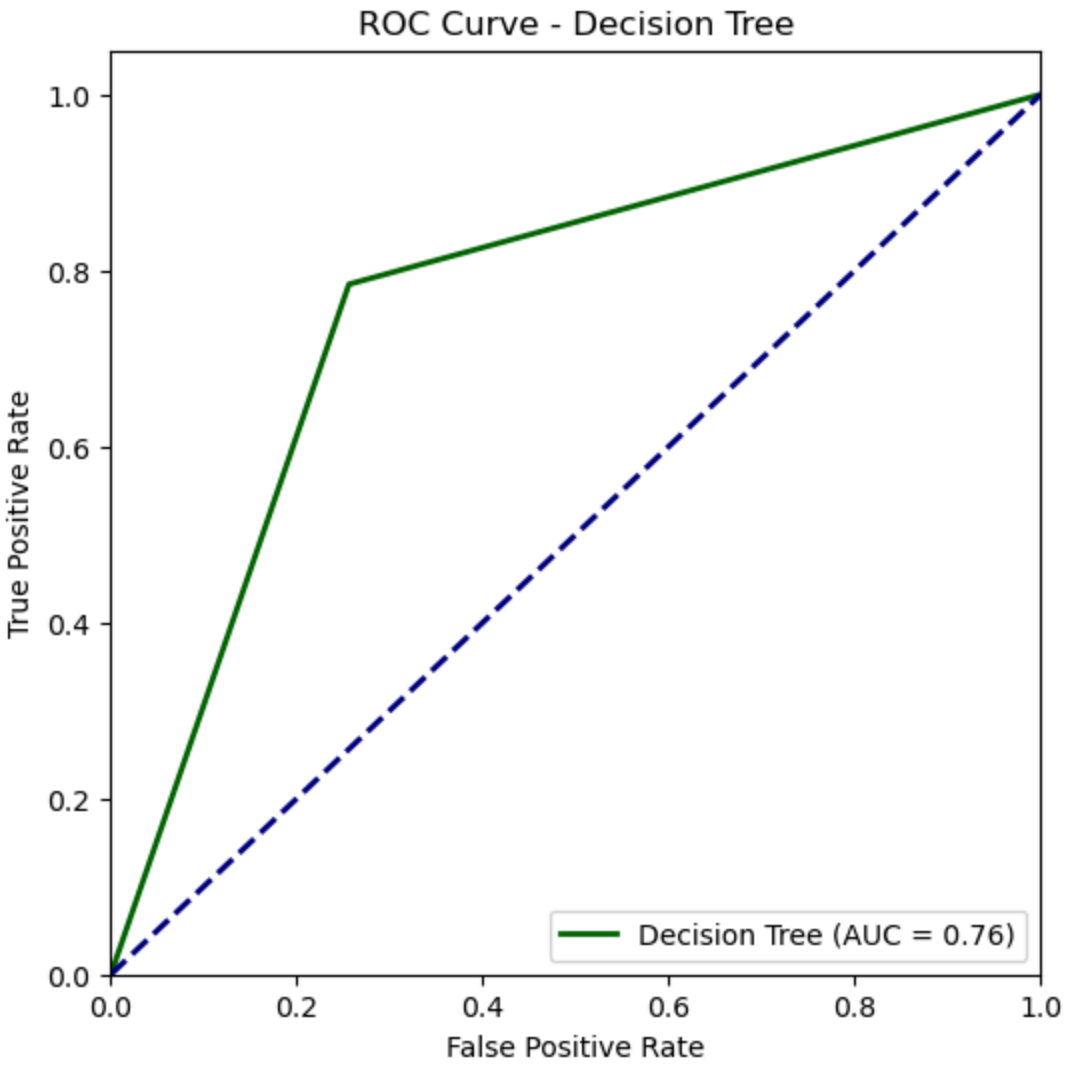
Out[417…
```
0.7663043478260869
```

In [419…
```
# Print classification metrics and ROC AUC for the Decision Tree
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Predict probabilities for the positive class (heart disease = 1)
y_proba = clf.predict_proba(X_test)[:, 1]

# Calculate and print ROC AUC score
print("ROC AUC:", roc_auc_score(y_test, y_proba))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.74      0.74        82
           1       0.79      0.78      0.79       102

    accuracy                           0.77       184
   macro avg       0.76      0.76      0.76       184
weighted avg       0.77      0.77      0.77       184

ROC AUC: 0.7641080822572931
```

In [421…
```
# Predict probability estimates for the positive class (heart disease = 1)
y_proba_tree = clf.predict_proba(X_test)[:, 1]

# Compute ROC curve values
fpr, tpr, thresholds = roc_curve(y_test, y_proba_tree)

# Calculate AUC (Area Under the Curve)
roc_auc_tree = roc_auc_score(y_test, y_proba_tree)
```

```
# Plot ROC curve for Decision Tree
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='darkgreen', lw=2,
         label='Decision Tree (AUC = %0.2f)' % roc_auc_tree)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 Curve - Decision Tree')
plt.legend(loc="lower right")
plt.show()

# Print final AUC value
print("ROC AUC (Decision Tree):", roc_auc_tree)
```

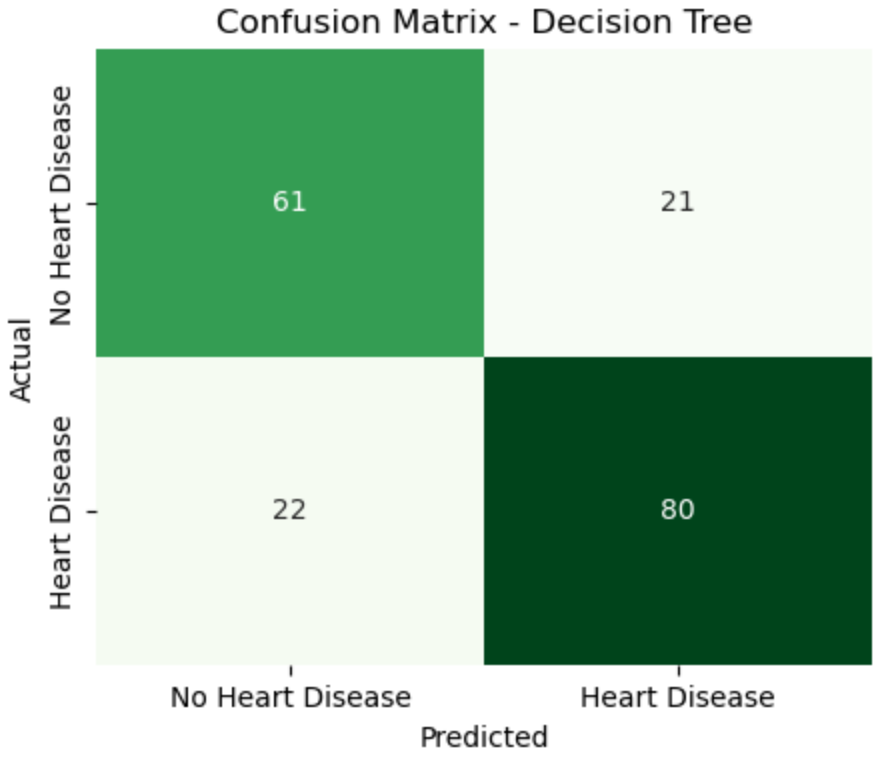



```
ROC AUC (Decision Tree): 0.7641080822572931
```

In [423…

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", cbar=False,
```

```
            xticklabels=['No Heart Disease','Heart Disease'],
            yticklabels=['No Heart Disease','Heart Disease'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Decision Tree")
plt.show()
```



RANDOM FOREST

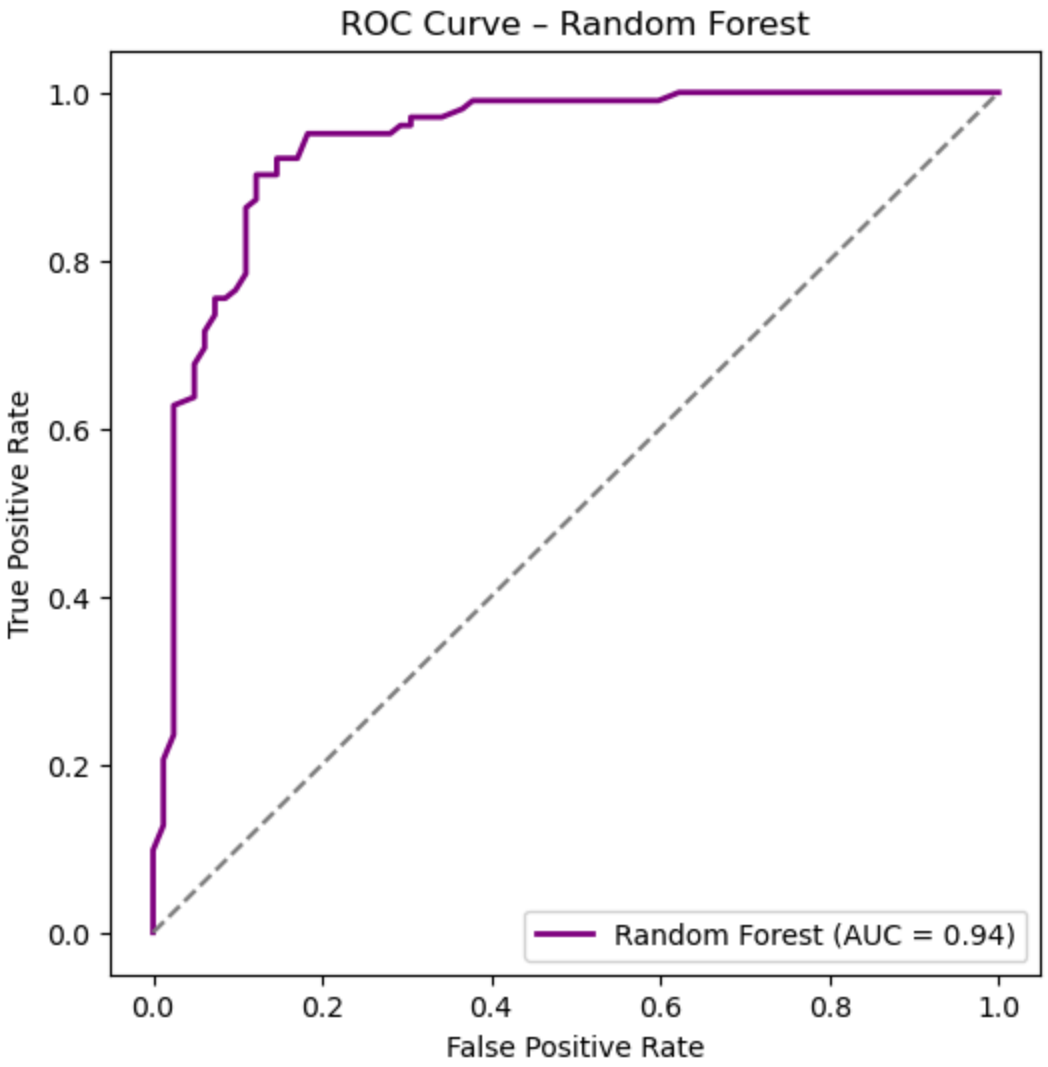
In [427…

```
# 1) Build and train the model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# 2) Predictions on the test set
y_pred = rf.predict(X_test)
y_proba = rf.predict_proba(X_test)[:, 1]  # probability for the positive cla

# 3) Evaluation
rfAcc = accuracy_score(y_test, y_pred)          # store accuracy
roc_auc_rf = roc_auc_score(y_test, y_proba)     # store AUC
report = classification_report(y_test, y_pred) # precision/recall/F1 per cla

print("Random Forest Accuracy:", rfAcc)
print("\nClassification Report:\n", report)
print("ROC AUC:", roc_auc_rf)
```

```
Random Forest Accuracy: 0.8804347826086957

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.85      0.86        82
           1       0.88      0.90      0.89       102

    accuracy                           0.88       184
   macro avg       0.88      0.88      0.88       184
weighted avg       0.88      0.88      0.88       184

ROC AUC: 0.93824725011956
```

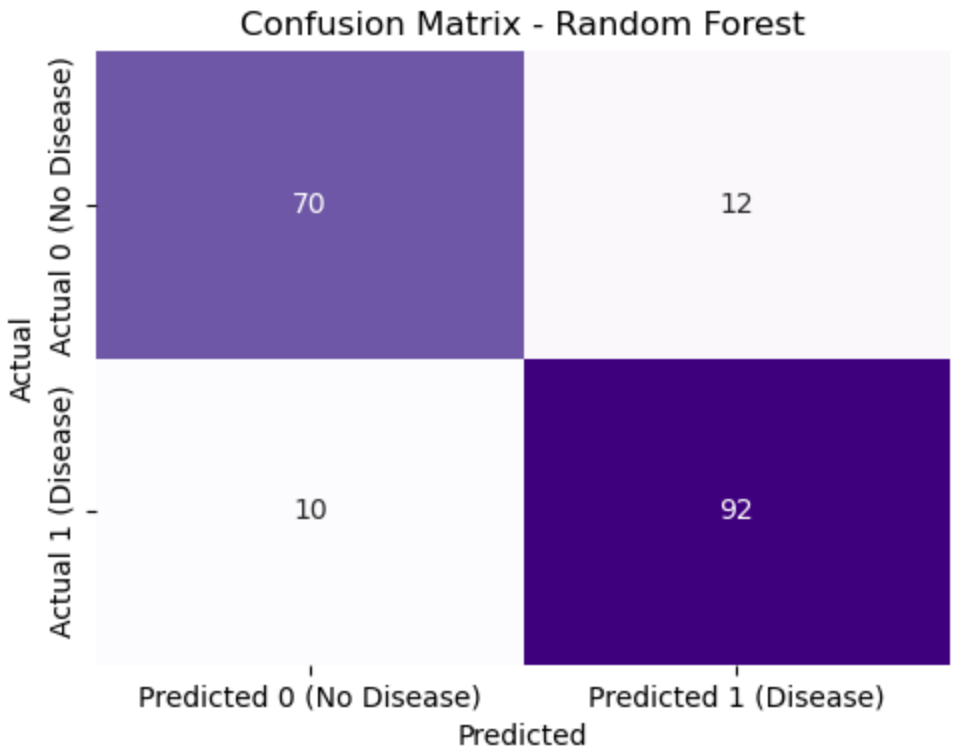
In [429…

```python
# ROC Curve for Random Forest

y_proba_rf = rf.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_proba_rf)
auc_rf = roc_auc_score(y_test, y_proba_rf)

plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, color="purple", lw=2, label=f"Random Forest (AUC = {auc_r
plt.plot([0,1],[0,1],'--', color='gray')
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.title("ROC Curve – Random Forest"); plt.legend(loc="lower right")
plt.show()
```



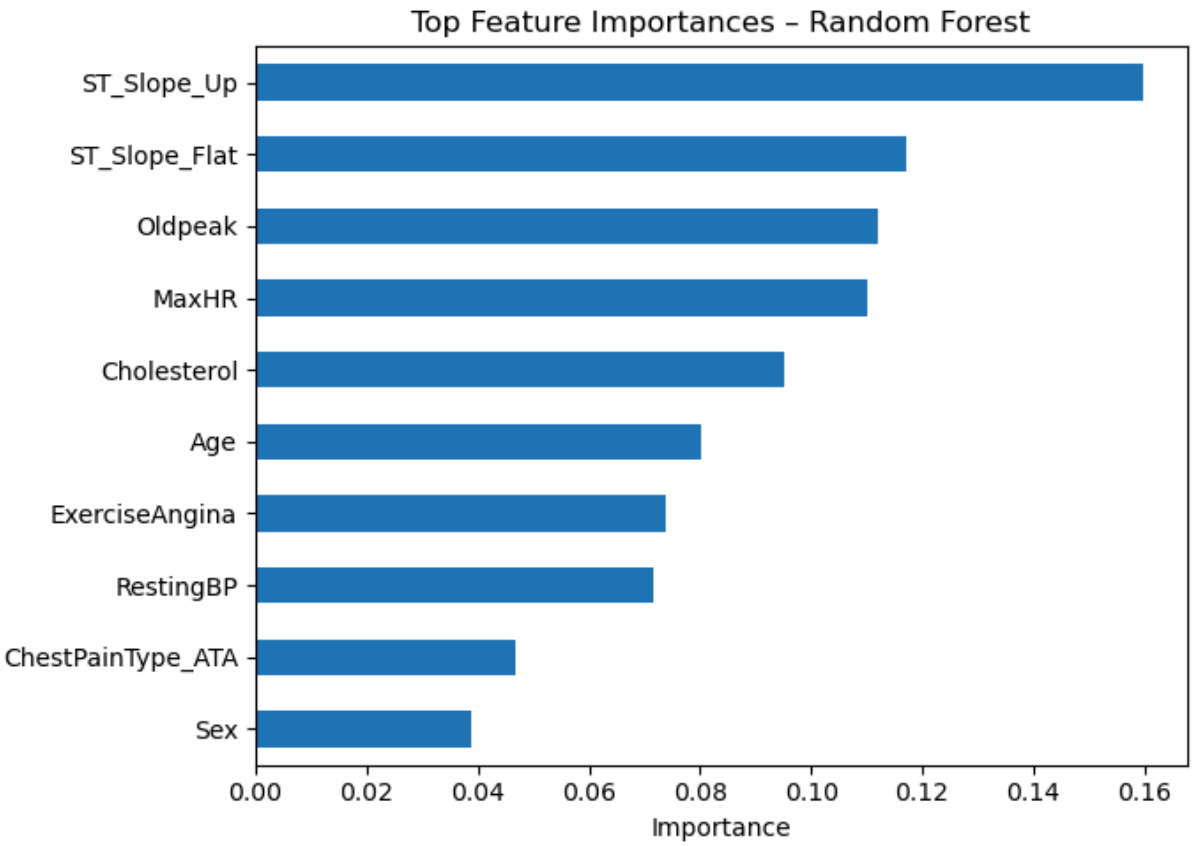
In [431…

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Purples", cbar=False,
            xticklabels=["Predicted 0 (No Disease)", "Predicted 1 (Disease)"
            yticklabels=["Actual 0 (No Disease)", "Actual 1 (Disease)"])

plt.title("Confusion Matrix – Random Forest")
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.tight_layout()
plt.show()
```



In [433…

```
# Extract and sort feature importances from the trained Random Forest model
fi = pd.Series(rf.feature_importances_, index=X_train.columns).sort_values()

# Select the top 10 most important features
top = fi.tail(10)

# Plot horizontal bar chart of feature importances
plt.figure(figsize=(7,5))
top.plot(kind="barh")
plt.title("Top Feature Importances – Random Forest")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



KNN

In [435…
```
# Define the range of k values to test
k_values = range(1, 21)
cv_scores = []

# Loop through each k and perform 5-fold cross-validation
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=5, scoring='accuracy'
    cv_scores.append(scores.mean())

# Identify the k with the highest mean accuracy
best_k = k_values[cv_scores.index(max(cv_scores))]

print("Best k:", best_k)
```

```
Best k: 10
```

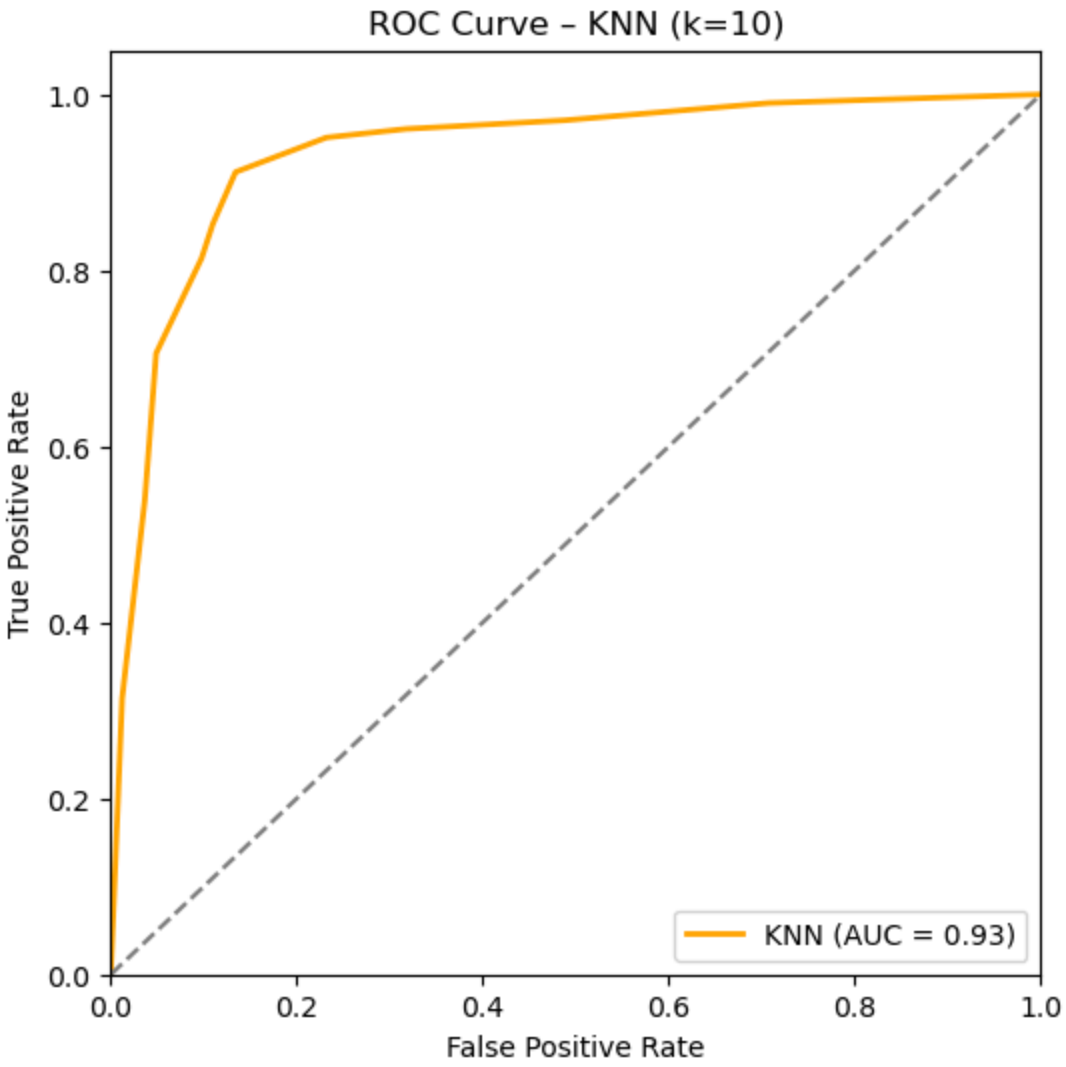
In [437…
```
# Building a model using KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 10)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
knnAcc = accuracy_score(y_test,y_pred)
knnAcc
```

Out[437… `0.8695652173913043`

In [439…
```
# Generate probability predictions for the positive class
y_proba = knn.predict_proba(X_test)[:, 1]

# Evaluate the model with ROC AUC and classification report
print("ROC AUC:", roc_auc_score(y_test, y_proba))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

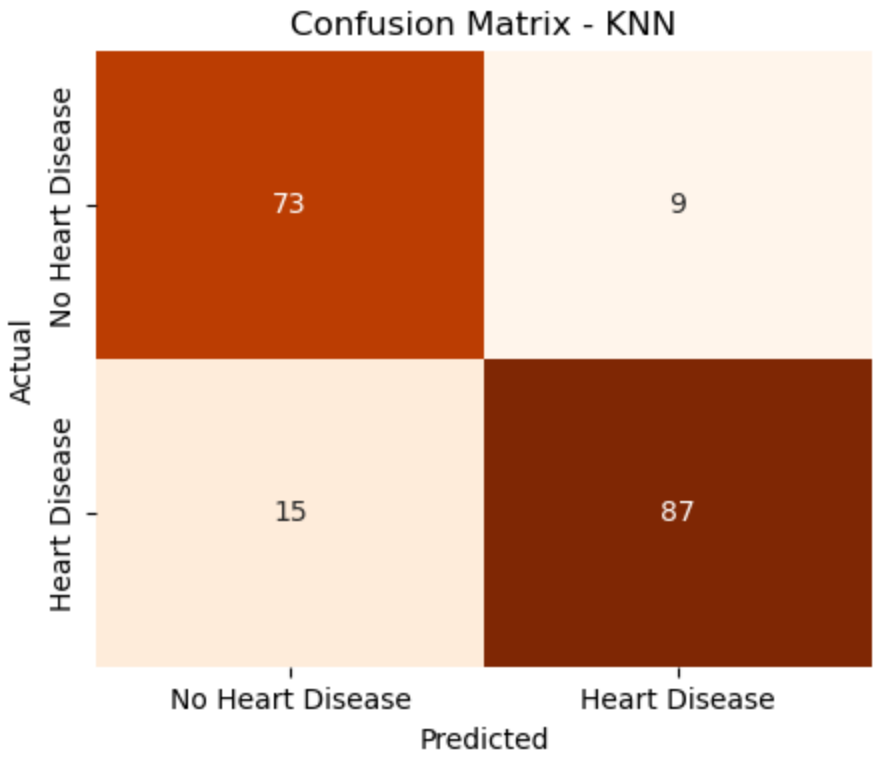
```
ROC AUC: 0.9324485891917742

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        82
           1       0.91      0.85      0.88       102

    accuracy                           0.87       184
   macro avg       0.87      0.87      0.87       184
weighted avg       0.87      0.87      0.87       184
```

In [445…
```
# Probability predictions for the positive class
y_proba_knn = knn.predict_proba(X_test)[:, 1]

# ROC curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_proba_knn)
auc_knn = roc_auc_score(y_test, y_proba_knn)

# Plot ROC curve
plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, color='orange', lw=2, label="KNN (AUC = %0.2f)" % auc_knn
plt.plot([0,1],[0,1],'--', color='gray')
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 Curve – KNN (k=10)")
plt.legend(loc="lower right")
plt.show()
```



In [441…

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Oranges", cbar=False,
            xticklabels=['No Heart Disease','Heart Disease'],
            yticklabels=['No Heart Disease','Heart Disease'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix – KNN")
plt.show()
```



In [447…

```
# Define model names and their corresponding accuracy scores
data = {
    'Estimators': [
        'Logistic Regression',
        'K-Nearest Neighbor',
        'Decision Tree',
        'Support Vector Machine',
        'Random Forest'
    ],
    'Accuracy': [logregAcc, knnAcc, clfAcc, svmAcc, rfAcc]  # rfAcc = Random
}

# Create a DataFrame for comparison
results_df = pd.DataFrame(data)

# Sort models by accuracy in descending order
results_df = results_df.sort_values('Accuracy', ascending=False).reset_index

# Display the results table
print(results_df)
```

```
              Estimators  Accuracy
0    Logistic Regression  0.885870
1  Support Vector Machine  0.885870
2          Random Forest  0.880435
3     K-Nearest Neighbor  0.869565
4          Decision Tree  0.766304
```

In [389…

```
# Step 1: Create a dictionary with model performance results
# Each key is a metric, and the values are the results for each model
data = {
    "Model": [
```

```
        "Logistic Regression",
        "Random Forest",
        "KNN (k=10)",
        "SVM (RBF)",
        "Decision Tree"
    ],
    "Accuracy": [0.89, 0.88, 0.87, 0.89, 0.77],
    "Precision": [0.89, 0.88, 0.91, 0.89, 0.79],
    "Recall": [0.90, 0.90, 0.85, 0.90, 0.78],
    "F1-score": [0.90, 0.89, 0.88, 0.90, 0.78],
    "ROC AUC": [0.94, 0.94, 0.93, 0.94, 0.76]
}

# Step 2: Convert the dictionary into a pandas DataFrame
df_perf = pd.DataFrame(data)

# Step 3: Print the DataFrame as a table in the console
print(df_perf)

# Step 4: Create a heatmap to visualize the performance
plt.figure(figsize=(10, 5))
# .set_index("Model") → makes 'Model' the row index so it's displayed on the
sns.heatmap(df_perf.set_index("Model"), annot=True, cmap="Reds", fmt=".2f",
plt.title("Model Performance Comparison", fontsize=14)
plt.yticks(rotation=0)  # keeps model names horizontal for better readabilit
plt.show()
```

```
                Model  Accuracy  Precision  Recall  F1-score  ROC AUC
0  Logistic Regression      0.89       0.89    0.90      0.90     0.94
1        Random Forest      0.88       0.88    0.90      0.89     0.94
2           KNN (k=10)      0.87       0.91    0.85      0.88     0.93
3            SVM (RBF)      0.89       0.89    0.90      0.90     0.94
4        Decision Tree      0.77       0.79    0.78      0.78     0.76
```