Dataset Selection

The file i just uploaded is named heart.csv

Data Prepocessing

```
Start coding or generate with AI.
```

Step 1: Load the Dataset

```
Start coding or generate with AI.
```

Double-click (or enter) to edit

```
import pandas as pd

# Load the dataset
file_path = "/content/heart.csv"
df = pd.read_csv(file_path)

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

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	tha
0	63	1	3	145	233	1	0	150	0	2.3	0	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	

Next steps: Generate code with df | View recommended plots | New interactive sheet

Step 2: Summary Statistics

```
# Display information about the dataset
df.info()

# Summary statistics for numerical columns
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

	age	sex	cp	trestbps	chol	fbs	restec
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.52805
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.52586
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.00000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.00000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.00000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.00000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.00000

Step 3: Check for Missing Values python Copy code

```
# Check for missing values
missing_values = df.isnull().sum()
missing_values
```

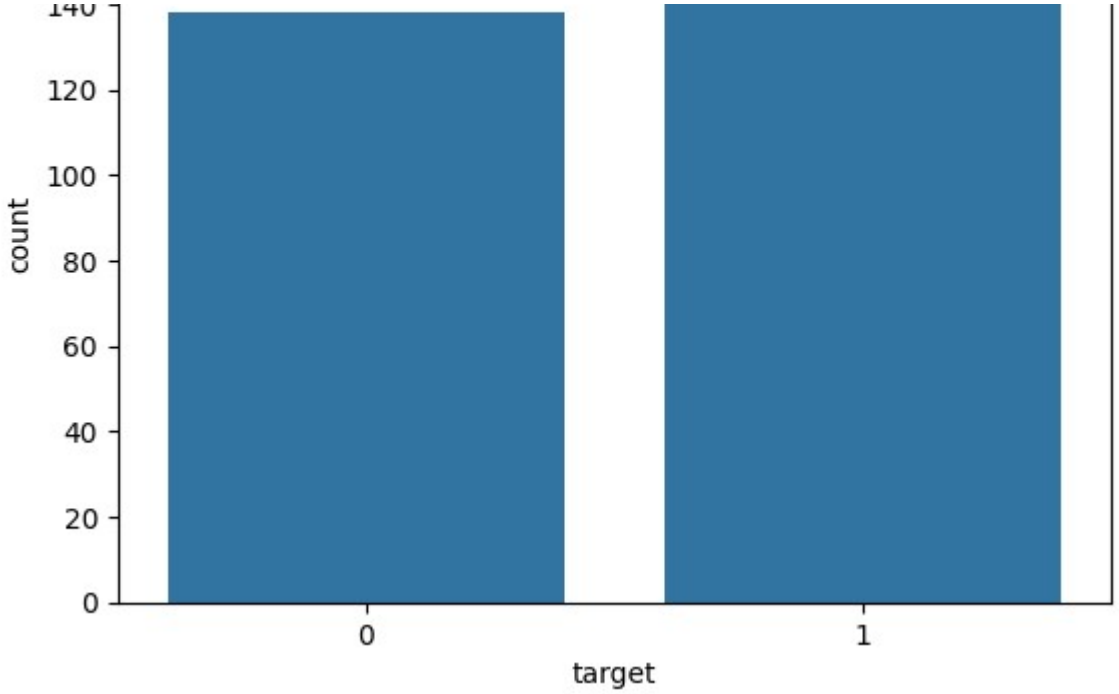
	0
age	0
sex	0
cp	0
trestbps	0
chol	0
fbs	0
restecg	0
thalach	0
exang	0
oldpeak	0
slope	0
ca	0
thal	0
target	0

dtype: int64

Double-click (or enter) to edit

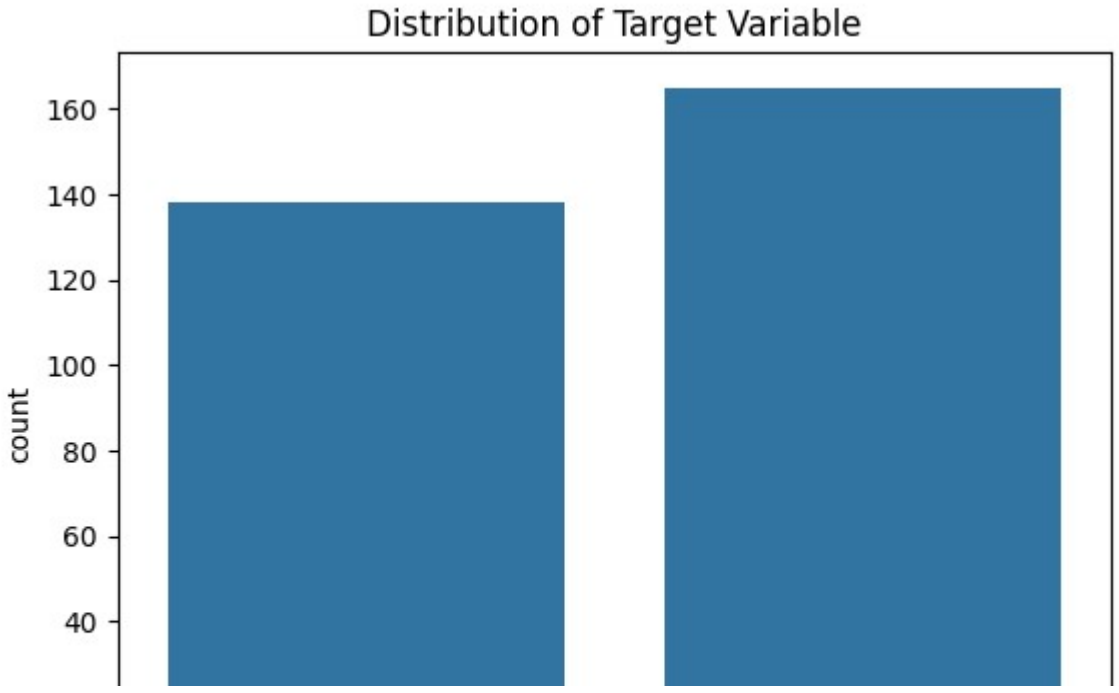
Step 4: Data Visualizations

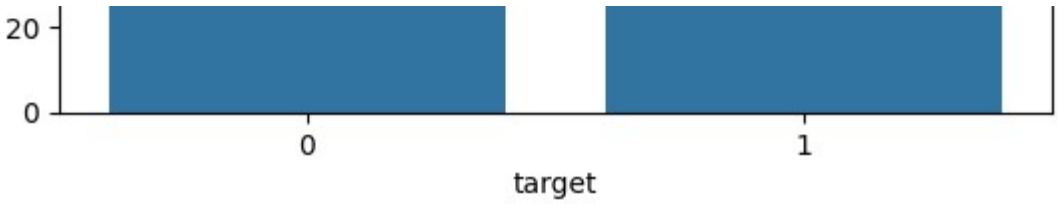
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x="target", data=df)
plt.title("Distribution of Target Variable")
plt.show()
```



a. Distribution of the Target Variable

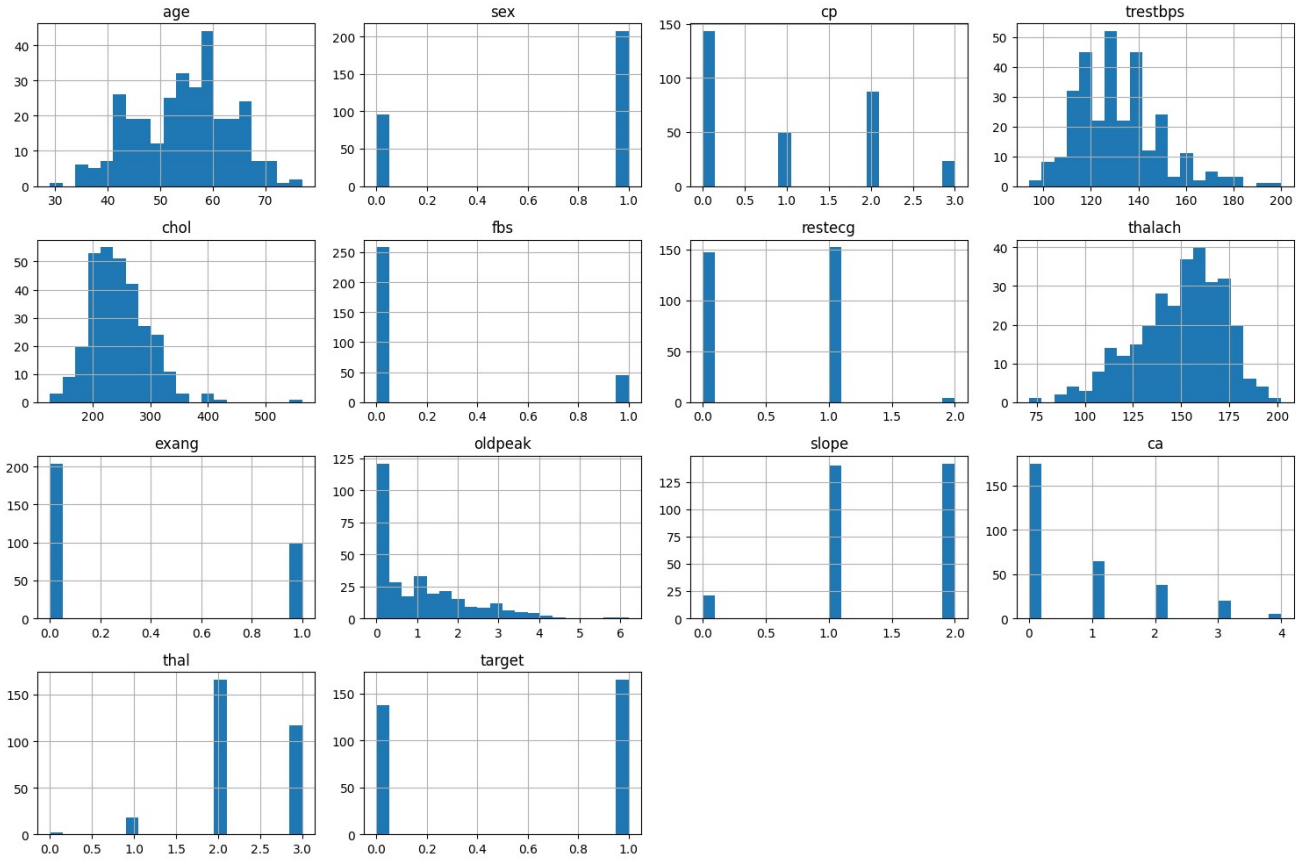
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x="target", data=df)
plt.title("Distribution of Target Variable")
plt.show()
```





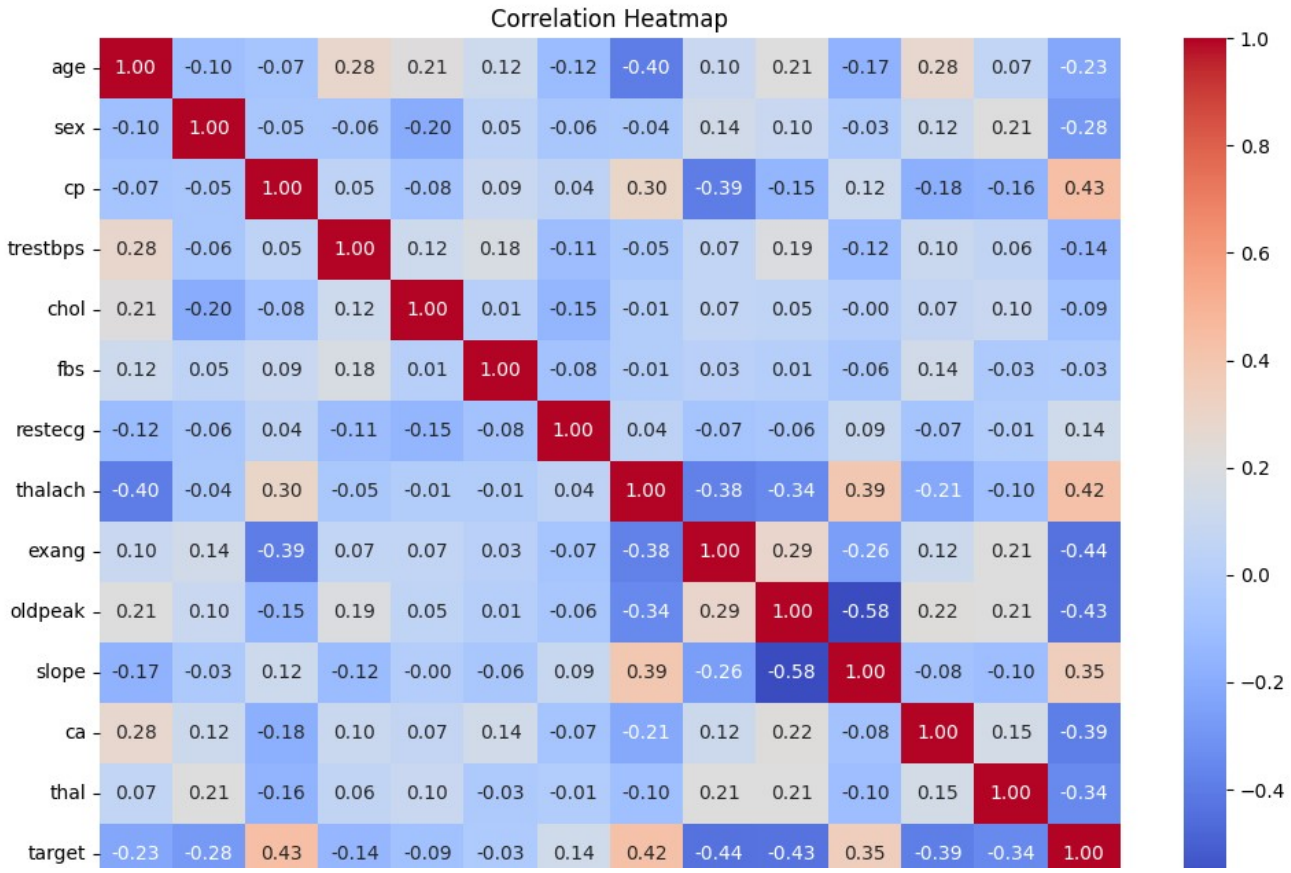
b. Histograms for Numerical Features

```
df.hist(figsize=(15, 10), bins=20)
plt.tight_layout()
plt.show()
```



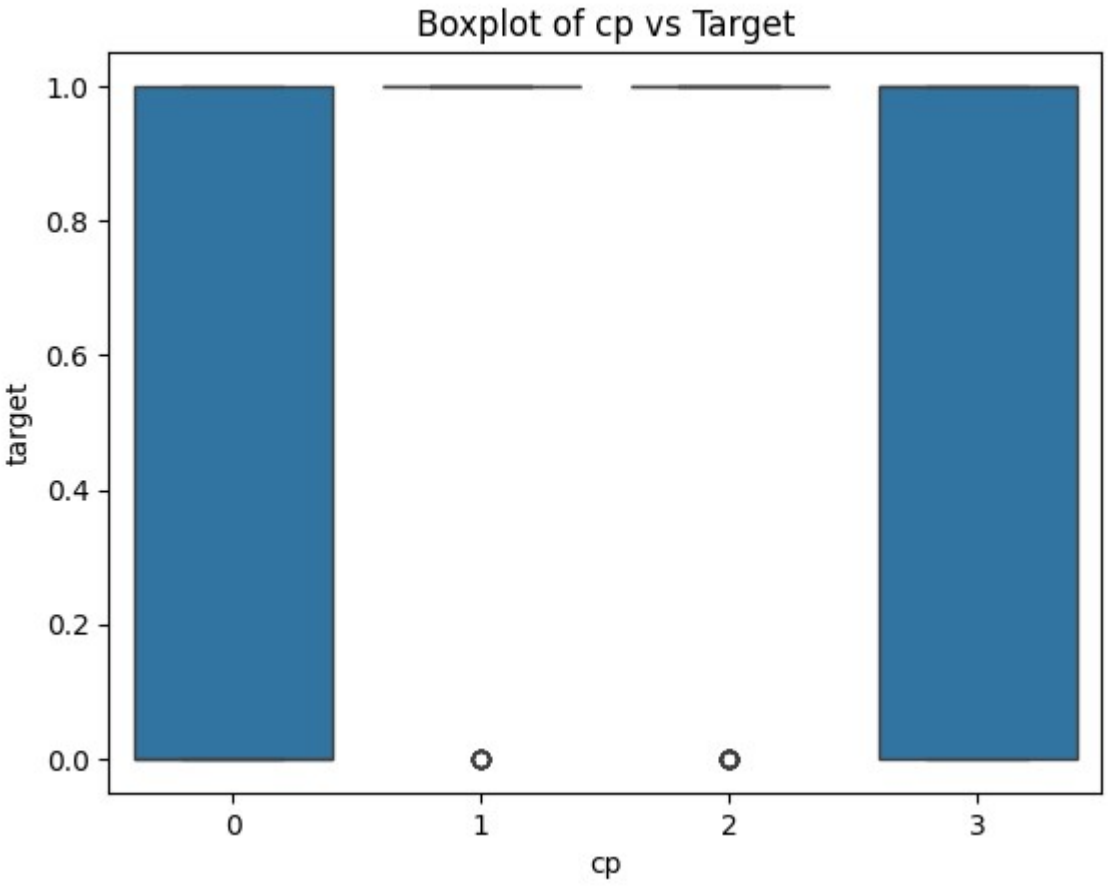
c. Correlation Heatmap

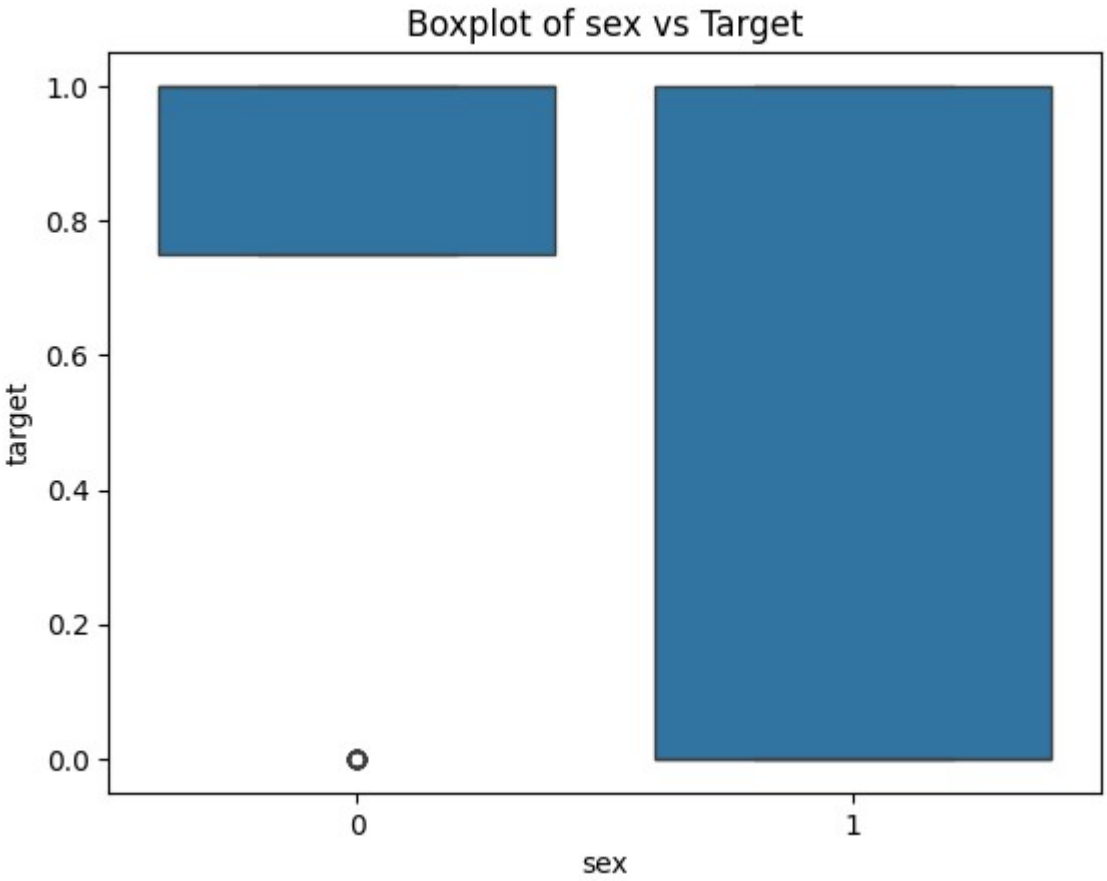
```
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```

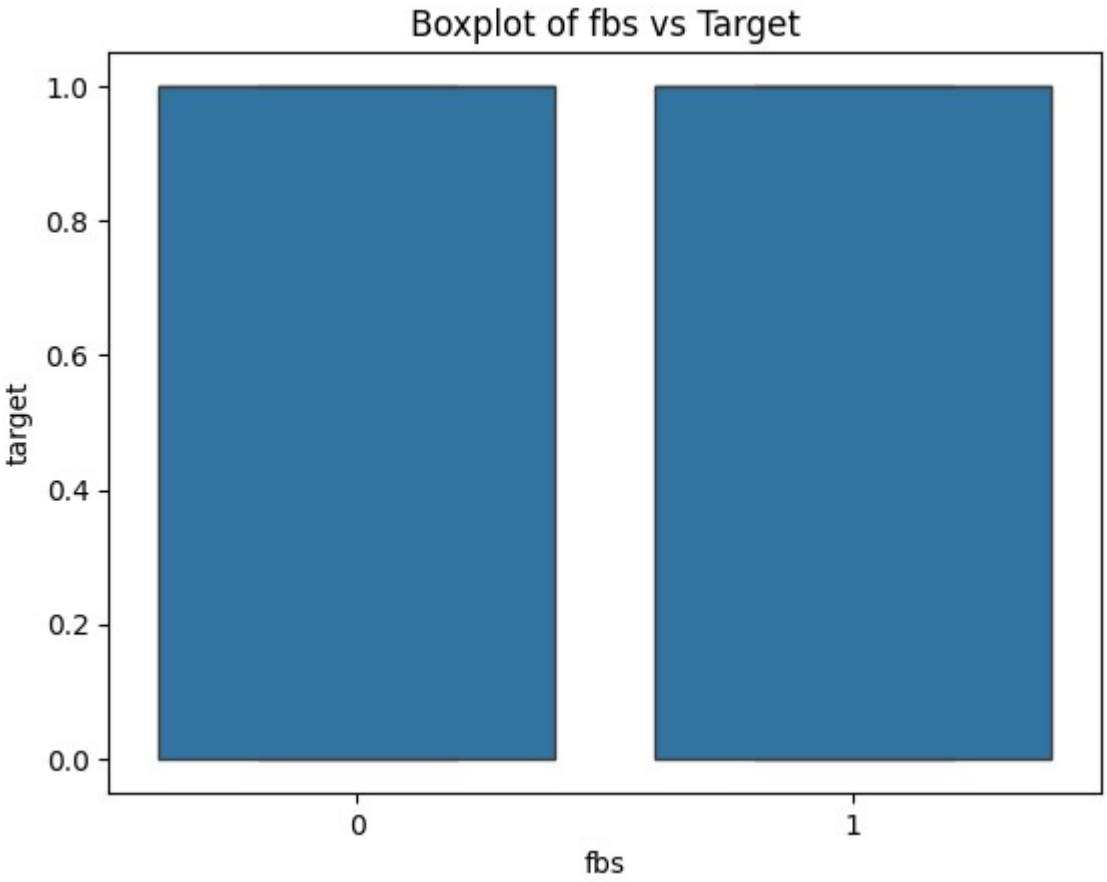


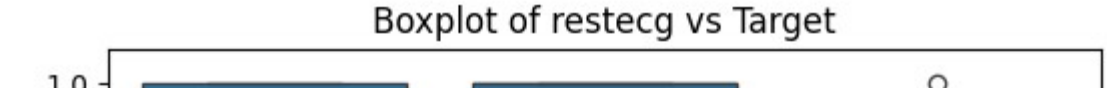
d. Boxplots for Categorical Variables

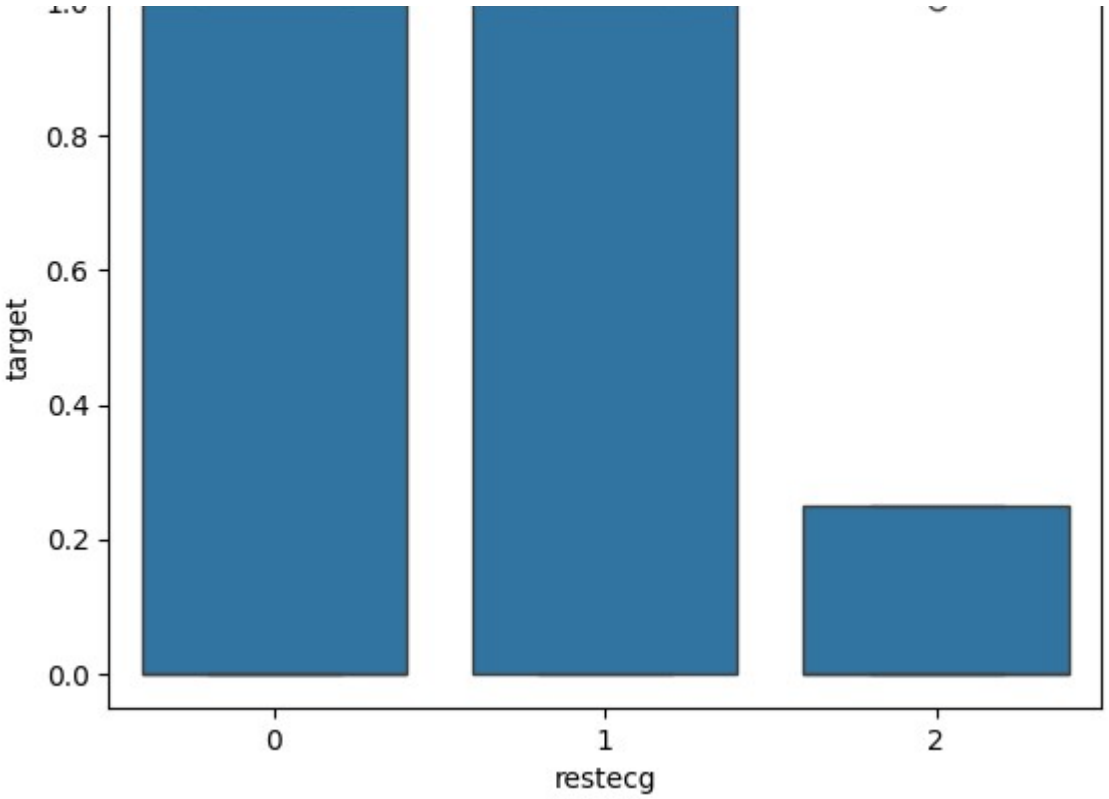
```
categorical_features = ["cp", "sex", "fbs", "restecg", "exang", "slope", "ca", "thal"]

for feature in categorical_features:
    sns.boxplot(x=feature, y="target", data=df)
    plt.title(f"Boxplot of {feature} vs Target")
    plt.show()
```

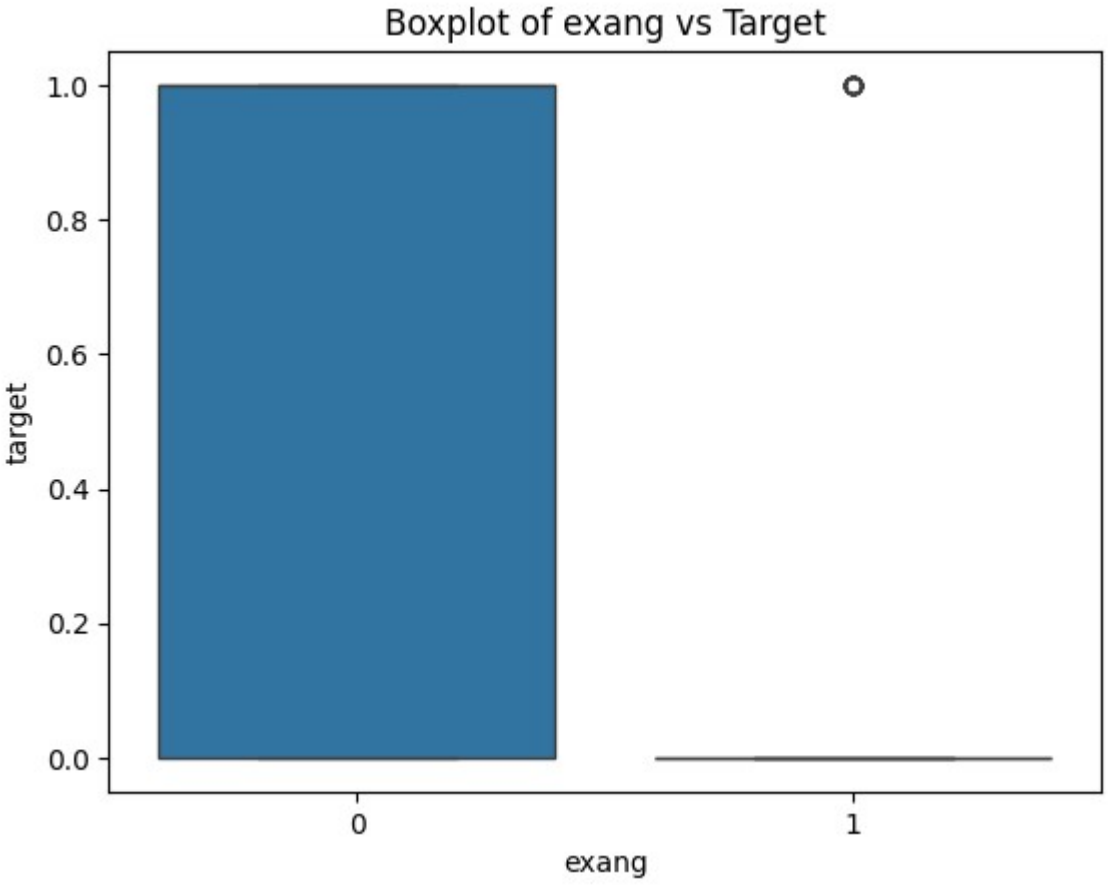



Boxplot of sex vs Target
target
1.0
0.8
0.6
0.4
0.2
0.0
0
1
sex

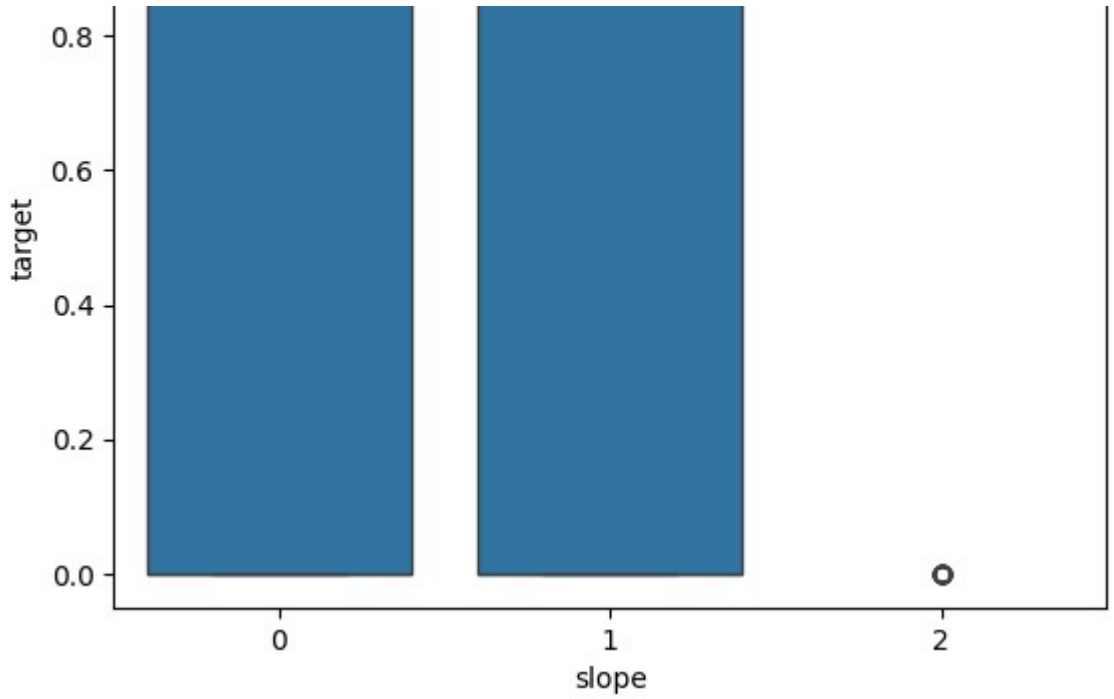

Boxplot of fbs vs Target
target
1.0
0.8
0.6
0.4
0.2
0.0
0
1
fbs

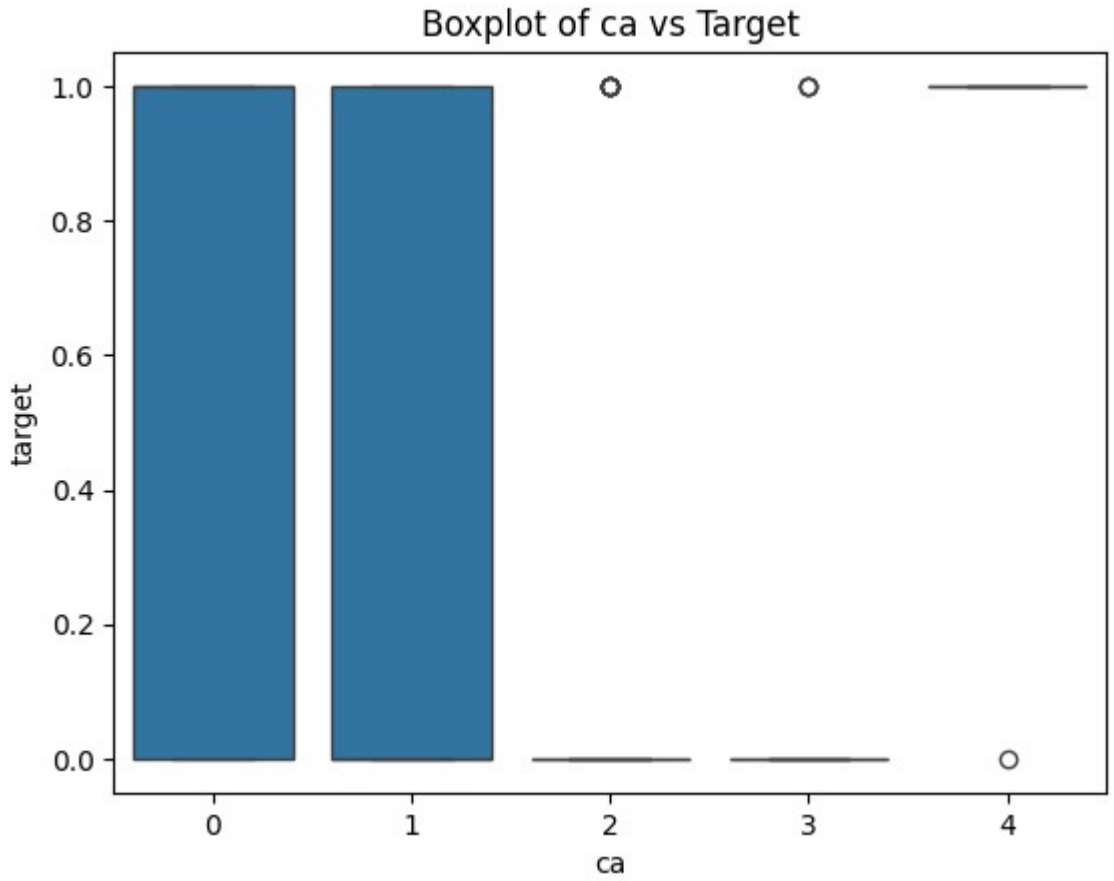

Boxplot of restecg vs Target

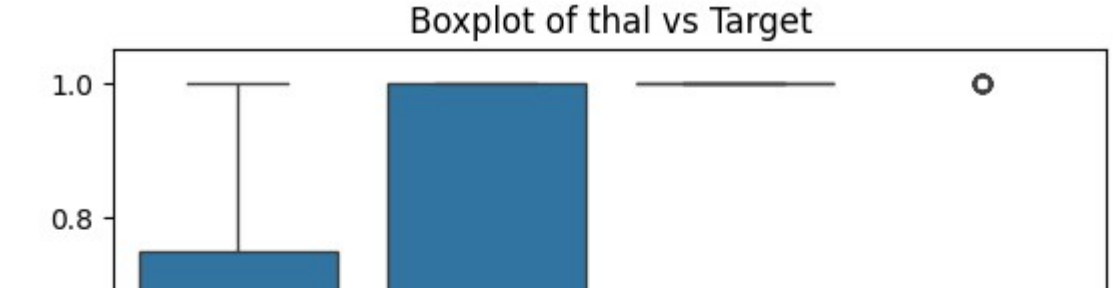

target
0.8
0.6
0.4
0.2
0.0
0
1
2
restecg

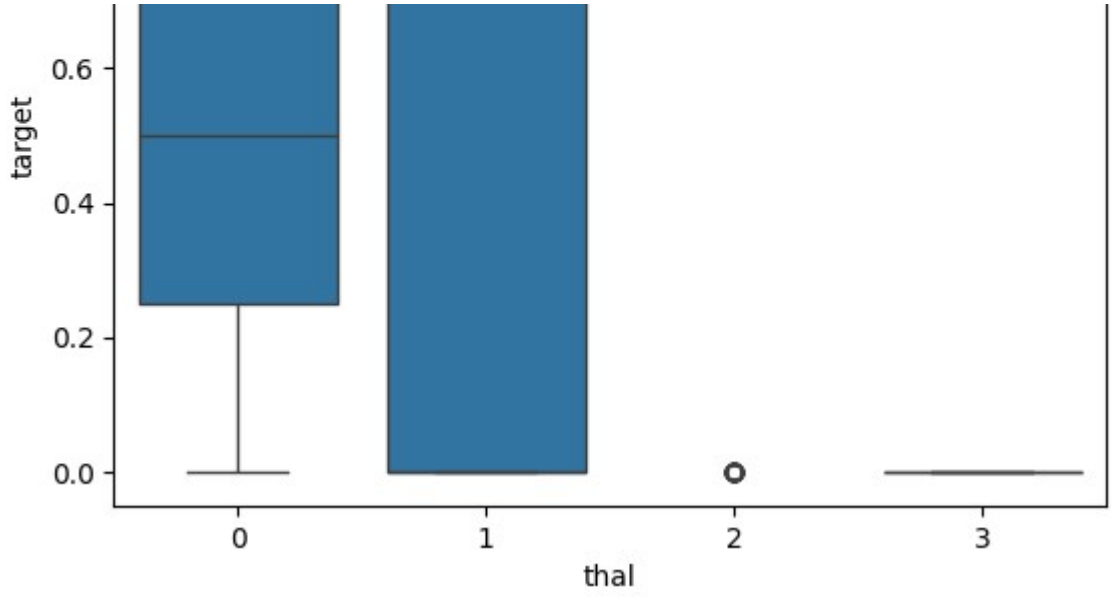

Boxplot of exang vs Target
target
1.0
0.8
0.6
0.4
0.2
0.0
0
1
exang


Boxplot of slope vs Target
1.0


0.8
0.6
0.4
0.2
0.0
target
0
1
2
slope


Boxplot of ca vs Target
1.0
0.8
0.6
0.4
0.2
0.0
target
0
1
2
3
4
ca


Boxplot of thal vs Target
1.0
0.8



Step 5: Handle Missing Values

```
# Handle missing values by filling with the median
df.fillna(df.median(), inplace=True)
```

Step 6: Encode Categorical Variables

```
# One-hot encode categorical variables if needed
df = pd.get_dummies(df, columns=categorical_features, drop_first=True)
df.head()
```

	age	trestbps	chol	thalach	oldpeak	target	cp_1	cp_2	cp_3	sex_1	...	exang
0	63	145	233	150	2.3	1	False	False	True	True	...	Fa
1	37	130	250	187	3.5	1	False	True	False	True	...	Fa
2	41	130	204	172	1.4	1	True	False	False	False	...	Fa
3	56	120	236	178	0.8	1	True	False	False	True	...	Fa
4	57	120	354	163	0.6	1	False	False	False	False	...	Tr

5 rows × 23 columns

Step 7: Normalize/Standardize Features

```
from sklearn.preprocessing import StandardScaler

# Standardize numerical features
numerical_features = ["age", "trestbps", "chol", "thalach", "oldpeak"]
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])

df.head()
```

	age	trestbps	chol	thalach	oldpeak	target	cp_1	cp_2	cp_3	sex_1
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	False	False	True	True
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	False	True	False	True
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	True	False	False	False
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	True	False	False	True
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	False	False	False	False

5 rows × 23 columns

Model Implementation

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix



# Load the dataset
file_path = '/content/heart.csv'  # Replace with the correct file path in Colab
data = pd.read_csv(file_path)


# Inspect the dataset
print("First five rows of the dataset:")
print(data.head())
```

```
First five rows of the dataset:
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

```
# Split features and target variable (assumes target column is named 'target')
X = data.drop(columns=['target'])  # Replace 'target' with the actual target column name
y = data['target']


# Split the dataset into training and testing sets (70/30 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)



# Standardize the features (for Logistic Regression and Gradient Boosting)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Logistic Regression

```
# Logistic Regression
print("\nLogistic Regression:")
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train_scaled, y_train)
lr_predictions = lr_model.predict(X_test_scaled)
print("Accuracy:", accuracy_score(y_test, lr_predictions))
print(classification_report(y_test, lr_predictions))
```

```
Logistic Regression:
Accuracy: 0.8131868131868132
              precision    recall  f1-score   support

           0       0.80      0.78      0.79        41
           1       0.82      0.84      0.83        50
```

```
    accuracy                           0.81        91
   macro avg       0.81      0.81      0.81        91
weighted avg       0.81      0.81      0.81        91
```

Decision Tree

```
print("\nDecision Tree:")
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
dt_predictions = dt_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, dt_predictions))
print(classification_report(y_test, dt_predictions))
```

```
Decision Tree:
Accuracy: 0.7362637362637363
              precision    recall  f1-score   support

           0       0.68      0.78      0.73        41
           1       0.80      0.70      0.74        50

    accuracy                           0.74        91
   macro avg       0.74      0.74      0.74        91
weighted avg       0.74      0.74      0.74        91
```

Start coding or generate with AI.

Random Forest

```
print("\nRandom Forest:")
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, rf_predictions))
print(classification_report(y_test, rf_predictions))
```

```
Random Forest:
Accuracy: 0.8241758241758241
              precision    recall  f1-score   support

           0       0.80      0.80      0.80        41
           1       0.84      0.84      0.84        50
```

```
          accuracy                           0.82        91
         macro avg       0.82      0.82      0.82        91
      weighted avg       0.82      0.82      0.82        91
```

Q-04

Model Training and Evaluation:

```
# Import necessary libraries
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confus
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load the dataset
from google.colab import files
uploaded = files.upload()

# Load the dataset into a DataFrame
file_name = list(uploaded.keys())[0]  # Get the name of the uploaded file
df = pd.read_csv(file_name)

# Ensure the dataset has a "target" column for classification purposes.
# Replace "target" with the actual column name if it is different.
if "target" not in df.columns:
    raise ValueError("The dataset must contain a 'target' column for classification.")

# Splitting data into features (X) and target (y)
X = df.drop(columns="target")
y = df["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)

# Train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]  # For ROC curve

# Calculate metrics
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Display results
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Disease", "Disease"], y
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

••• Browse... No files selected. Cancel upload

Hyperparameter Tunung

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from xgboost import XGBClassifier  # Ensure this is imported
from sklearn.ensemble import RandomForestClassifier

# Define hyperparameter grids for each model
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.5, 0.7, 1.0],
}

# Perform Grid Search for Random Forest
print("Performing Grid Search for Random Forest...")
grid_rf = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid_rf,
    scoring='accuracy',
    cv=3,
    verbose=2,
    n_jobs=-1
)
grid_rf.fit(X_train, y_train)

# Perform Random Search for XGBoost
print("Performing Random Search for XGBoost...")
random_xgb = RandomizedSearchCV(
    estimator=XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=
    param_distributions=param_grid_xgb,
    n_iter=20,
    scoring='accuracy',
```

```
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
random_xgb.fit(X_train, y_train)

# Display the best parameters and their corresponding scores
print("\nBest parameters for Random Forest:", grid_rf.best_params_)
print("Best score for Random Forest:", grid_rf.best_score_)

print("\nBest parameters for XGBoost:", random_xgb.best_params_)
print("Best score for XGBoost:", random_xgb.best_score_)

# Evaluate the best models on the test set
best_rf = grid_rf.best_estimator_
best_xgb = random_xgb.best_estimator_

# Predict the values using the best models
y_pred_rf = best_rf.predict(X_test)
y_pred_xgb = best_xgb.predict(X_test)

# Evaluate the models
print("\nEvaluating the best Random Forest model...")
evaluate_classification_model(y_test, y_pred_rf, data.target_names)

print("\nEvaluating the best XGBoost model...")
evaluate_classification_model(y_test, y_pred_xgb, data.target_names)
```

Conclusion

```
# ... (Existing code)

# Evaluate the model and print metrics
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))

# Evaluate and print results for Logistic Regression
print("\nLogistic Regression:")
evaluate_model(lr_model, X_test_scaled, y_test)

# Evaluate and print results for Decision Tree
```

```
# Evaluate and print results for Decision Tree
print("\nDecision Tree:")
evaluate_model(dt_model, X_test, y_test)

# Evaluate and print results for Random Forest
print("\nRandom Forest:")
evaluate_model(rf_model, X_test, y_test)

# ... (Rest of your code)


# ... (Existing code)

# Evaluate the best models and print metrics
def evaluate_model(model, X_test, y_test):
  # ... (Same as above)

print("\nEvaluating the best Random Forest model...")
evaluate_model(best_rf, X_test, y_test)

print("\nEvaluating the best XGBoost model...")
evaluate_model(best_xgb, X_test, y_test)

# ... (Rest of your code)
```