```
from google.colab import files
uploaded = files.upload()
import pandas as pd
import io
```

```
Choose files UCI Heart D…se Data.csv
UCI Heart Disease Data.csv(text/csv) - 79346 bytes, last modified: 10/12/2025 - 100% done
Saving UCI Heart Disease Data.csv to UCI Heart Disease Data (1).csv
```

```
df = pd.read_csv(io.BytesIO(uploaded["UCI Heart Disease Data (1).csv"]))
print("Loaded:", df.shape)
```

```
Loaded: (920, 16)
```

```
import pandas as pd
df = pd.read_csv(list(uploaded.keys())[0])
df.head()
```

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2.3	downsloping	0.0	
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1.5	flat	3.0	r
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2.6	flat	2.0	reve

Next steps: Generate code with df | New interactive sheet

Importing the Dependencies

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
from sklearn.model_selection import learning_curve
```

A. Data Cleaning

```
df = df.drop_duplicates()
df = df.replace(["", " ", "NA", "NaN"], np.nan)
```

```
num_cols = ["age","trestbps","chol","thalch","oldpeak","ca","num"]
for col in num_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors="coerce")
```

```
if "chol" in df.columns:
    df.loc[df["chol"] == 0, "chol"] = np.nan
```

```
for col in df.columns:
    if pd.api.types.is_numeric_dtype(df[col]):
        df[col] = df[col].fillna(df[col].median())
    else:
        mode = df[col].mode()
        if not mode.empty:
            df[col] = df[col].fillna(mode.iloc[0])
```

```
/tmp/ipython-input-2363592276.py:7: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is
  df[col] = df[col].fillna(mode.iloc[0])
```

B. Encoding

```
if "sex" in df.columns:
    df["sex"] = df["sex"].astype(str).str.upper().map({"MALE":1,"FEMALE":0}).fillna(0).astype(int)
for c in ["fbs","exang"]:
    if c in df.columns:
        df[c] = df[c].astype(str).str.upper().map({"TRUE":1,"FALSE":0}).fillna(0).astype(int)
```

```
if "thal" in df.columns:
    df["thal"] = (
        df["thal"].astype(str).str.strip()
        .replace({"reversable defect":"reversible defect"})
    )
```

```
cat_targets = ["cp","restecg","slope","thal","dataset"]
cat_cols = [c for c in cat_targets if c in df.columns]
for c in cat_cols:
    df[c] = df[c].astype(str).str.strip()
df = pd.get_dummies(df, columns=cat_cols, drop_first=True)
```

C.Feature Scaling

```
y = (df["num"] > 0).astype(int)
X = df.drop(columns=[c for c in ["id","num"] if c in df.columns])
```

```
X = X.replace([np.inf, -np.inf], np.nan).fillna(0)
```

D. Train- Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)

print("Preprocessing OK")
print("Train:", X_train_scaled.shape, "| Test:", X_test_scaled.shape, "| Features:", X.shape[1])
```

```
Preprocessing OK
Train: (736, 21) | Test: (184, 21) | Features: 21
```

Apply 5 Machine Learning Algo.

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_scaled, y_train)
y_pred_log = log_model.predict(X_test_scaled)
print("\n Logistic Regression Results")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_log))
print("\nClassification Report:\n", classification_report(y_test, y_pred_log, digits=4))
```

```
 Logistic Regression Results
Confusion Matrix:
 [[65 17]
 [10 92]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8667    0.7927    0.8280        82
           1     0.8440    0.9020    0.8720       102

    accuracy                         0.8533       184
   macro avg     0.8554    0.8473    0.8500       184
weighted avg     0.8541    0.8533    0.8524       184
```

```
# K-Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=7)
```

```
knn_model.fit(X_train_scaled, y_train)
y_pred_knn = knn_model.predict(X_test_scaled)

print("\n🔍 KNN Results")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("\nClassification Report:\n", classification_report(y_test, y_pred_knn, digits=4))
```

```
🔍 KNN Results
Confusion Matrix:
 [[67 15]
 [ 9 93]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8816    0.8171    0.8481        82
           1     0.8611    0.9118    0.8857       102

    accuracy                         0.8696       184
   macro avg     0.8713    0.8644    0.8669       184
weighted avg     0.8702    0.8696    0.8690       184
```

```
# Decision Tree

from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
dt_model.fit(X_train_scaled, y_train)
y_pred_dt = dt_model.predict(X_test_scaled)
print("\n🔍 Decision Tree Results")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("\nClassification Report:\n", classification_report(y_test, y_pred_dt, digits=4))
```

```
🔍 Decision Tree Results
Confusion Matrix:
 [[69 13]
 [20 82]]

Classification Report:
              precision    recall  f1-score   support

           0     0.7753    0.8415    0.8070        82
           1     0.8632    0.8039    0.8325       102

    accuracy                         0.8207       184
   macro avg     0.8192    0.8227    0.8198       184
weighted avg     0.8240    0.8207    0.8211       184
```

```
# Random Forest

from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
rf_model.fit(X_train_scaled, y_train)
y_pred_rf = rf_model.predict(X_test_scaled)
print("\n🔍 Random Forest Results")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf, digits=4))
```

```
🔍 Random Forest Results
Confusion Matrix:
 [[68 14]
 [15 87]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8193    0.8293    0.8242        82
           1     0.8614    0.8529    0.8571       102

    accuracy                         0.8424       184
   macro avg     0.8403    0.8411    0.8407       184
weighted avg     0.8426    0.8424    0.8425       184
```

```
# Support Vector Machine

from sklearn.svm import SVC
svm_model = SVC(kernel="rbf", probability=True, random_state=42)
```

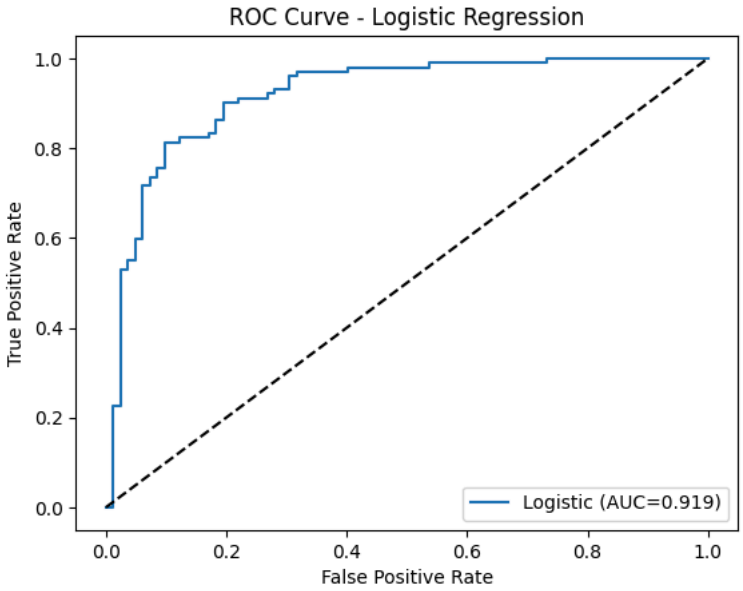
```
svm_model.fit(X_train_scaled, y_train)
y_pred_svm = svm_model.predict(X_test_scaled)
print("\n🔍 SVM Results")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("\nClassification Report:\n", classification_report(y_test, y_pred_svm, digits=4))
```

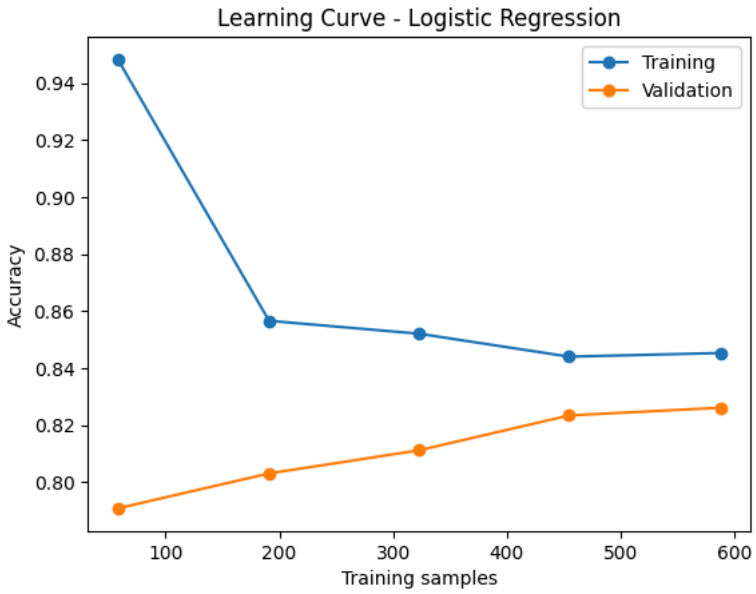
```
🔍 SVM Results
Confusion Matrix:
 [[63 19]
 [ 6 96]]

Classification Report:
              precision    recall  f1-score   support

           0     0.9130    0.7683    0.8344        82
           1     0.8348    0.9412    0.8848       102

    accuracy                         0.8641       184
   macro avg     0.8739    0.8547    0.8596       184
weighted avg     0.8697    0.8641    0.8624       184
```

Model result Analysis

```
# Logistic Regression Evaluation

from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import numpy as np
y_pred_log = log_model.predict(X_test_scaled)

#Confusion Matrix
print("Confusion Matrix (Logistic Regression):\n", confusion_matrix(y_test, y_pred_log))

#Precision, Recall, F1-score
print("\nClassification Report:\n", classification_report(y_test, y_pred_log, digits=4))

#ROC Curve
y_prob_log = log_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob_log)
plt.plot(fpr, tpr, label=f"Logistic (AUC={auc(fpr,tpr):.3f})")
plt.plot([0,1],[0,1],"k--")
plt.title("ROC Curve - Logistic Regression")
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.legend(); plt.show()

#Learning Curve
train_sizes, train_scores, val_scores = learning_curve(
    log_model, X_train_scaled, y_train, cv=5,
    train_sizes=np.linspace(0.1,1.0,5), scoring="accuracy"
)
plt.plot(train_sizes, train_scores.mean(axis=1), "o-", label="Training")
plt.plot(train_sizes, val_scores.mean(axis=1), "o-", label="Validation")
plt.title("Learning Curve - Logistic Regression")
plt.xlabel("Training samples"); plt.ylabel("Accuracy")
plt.legend(); plt.show()
```

```
Confusion Matrix (Logistic Regression):
 [[65 17]
 [10 92]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8667    0.7927    0.8280        82
           1     0.8440    0.9020    0.8720       102

    accuracy                         0.8533       184
   macro avg     0.8554    0.8473    0.8500       184
weighted avg     0.8541    0.8533    0.8524       184
```

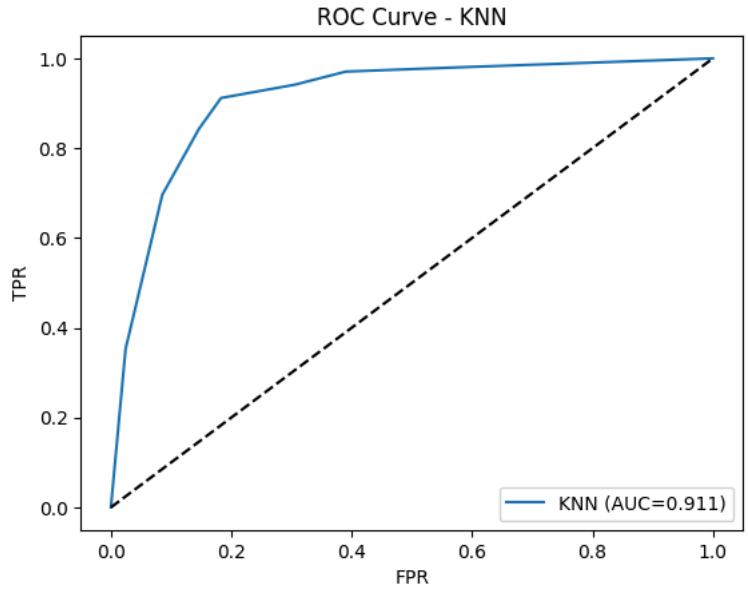


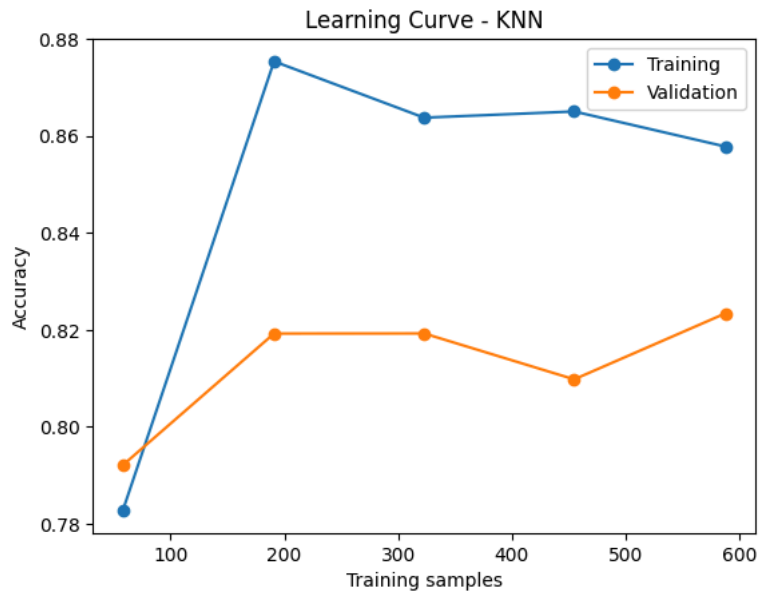



```
# KNN Evaluation
y_pred_knn = knn_model.predict(X_test_scaled)

print("Confusion Matrix (KNN):\n", confusion_matrix(y_test, y_pred_knn))
print("\nClassification Report:\n", classification_report(y_test, y_pred_knn, digits=4))

#ROC Curve
y_prob_knn = knn_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob_knn)
plt.plot(fpr, tpr, label=f"KNN (AUC={auc(fpr,tpr):.3f})")
plt.plot([0,1],[0,1],"k--")
plt.title("ROC Curve - KNN"); plt.xlabel("FPR"); plt.ylabel("TPR")
plt.legend(); plt.show()

#Learning Curve
train_sizes, train_scores, val_scores = learning_curve(
    knn_model, X_train_scaled, y_train, cv=5,
    train_sizes=np.linspace(0.1,1.0,5), scoring="accuracy"
)
```

```
plt.plot(train_sizes, train_scores.mean(axis=1), "o-", label="Training")
plt.plot(train_sizes, val_scores.mean(axis=1), "o-", label="Validation")
plt.title("Learning Curve - KNN")
plt.xlabel("Training samples"); plt.ylabel("Accuracy")
plt.legend(); plt.show()
```

```
Confusion Matrix (KNN):
 [[67 15]
 [ 9 93]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8816    0.8171    0.8481        82
           1     0.8611    0.9118    0.8857       102

    accuracy                         0.8696       184
   macro avg     0.8713    0.8644    0.8669       184
weighted avg     0.8702    0.8696    0.8690       184
```

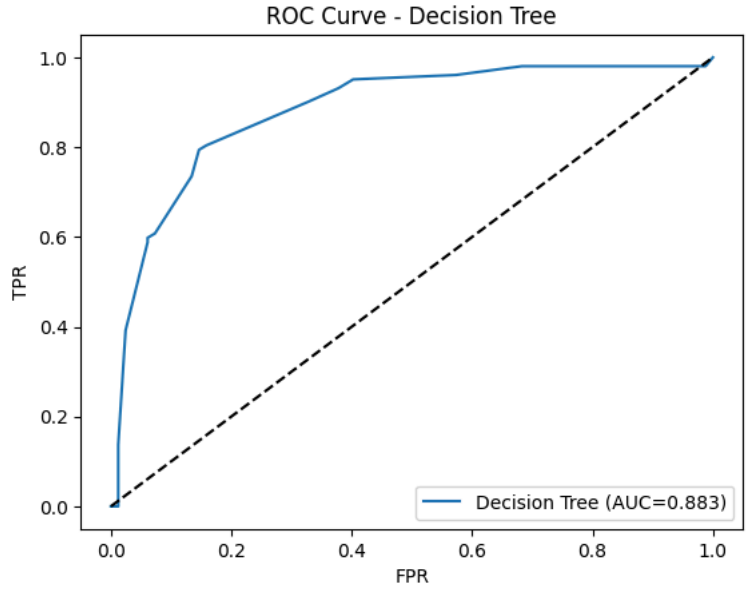


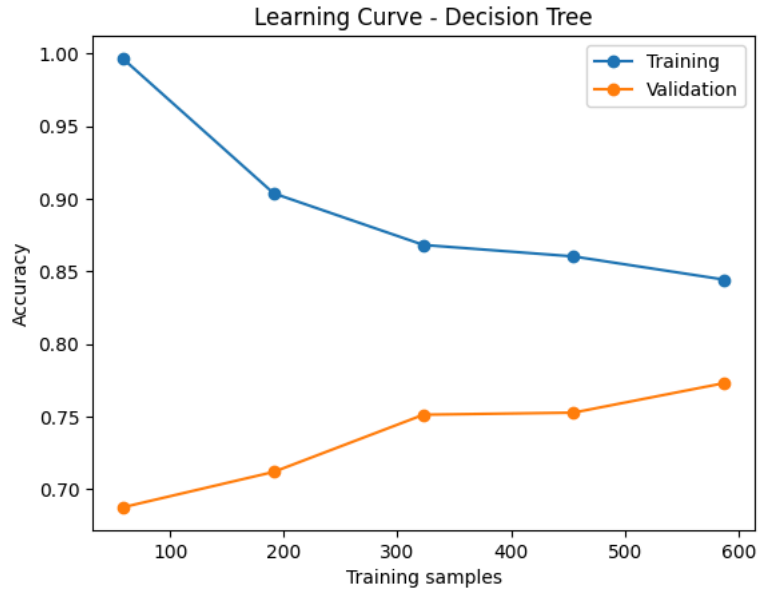



```
# Decision Tree Evaluation

y_pred_dt = dt_model.predict(X_test_scaled)

print("Confusion Matrix (Decision Tree):\n", confusion_matrix(y_test, y_pred_dt))
print("\nClassification Report:\n", classification_report(y_test, y_pred_dt, digits=4))

#ROC Curve
y_prob_dt = dt_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob_dt)
plt.plot(fpr, tpr, label=f"Decision Tree (AUC={auc(fpr,tpr):.3f})")
plt.plot([0,1],[0,1],"k--")
plt.title("ROC Curve - Decision Tree"); plt.xlabel("FPR"); plt.ylabel("TPR")
```

```
plt.legend(); plt.show()

#Learning Curve
train_sizes, train_scores, val_scores = learning_curve(
    dt_model, X_train_scaled, y_train, cv=5,
    train_sizes=np.linspace(0.1,1.0,5), scoring="accuracy"
)
plt.plot(train_sizes, train_scores.mean(axis=1), "o-", label="Training")
plt.plot(train_sizes, val_scores.mean(axis=1), "o-", label="Validation")
plt.title("Learning Curve - Decision Tree")
plt.xlabel("Training samples"); plt.ylabel("Accuracy")
plt.legend(); plt.show()
```

```
Confusion Matrix (Decision Tree):
 [[69 13]
 [20 82]]

Classification Report:
              precision    recall  f1-score   support

           0     0.7753    0.8415    0.8070        82
           1     0.8632    0.8039    0.8325       102

    accuracy                         0.8207       184
   macro avg     0.8192    0.8227    0.8198       184
weighted avg     0.8240    0.8207    0.8211       184
```

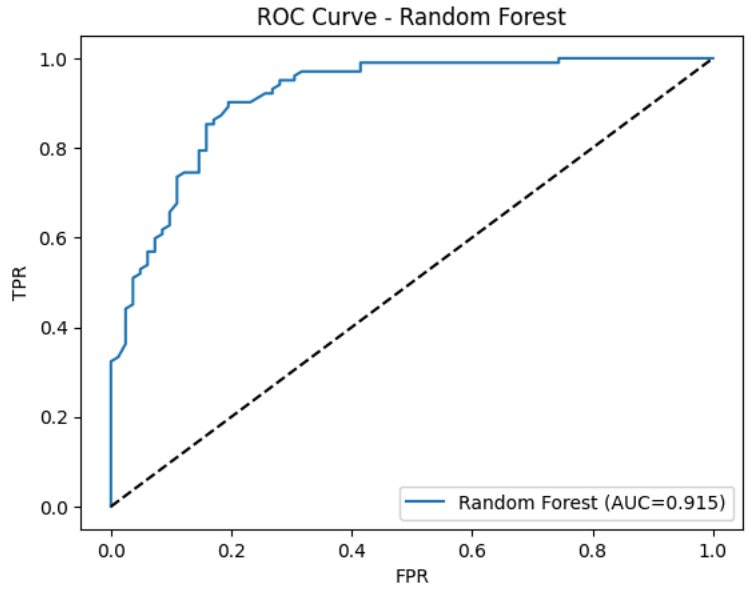


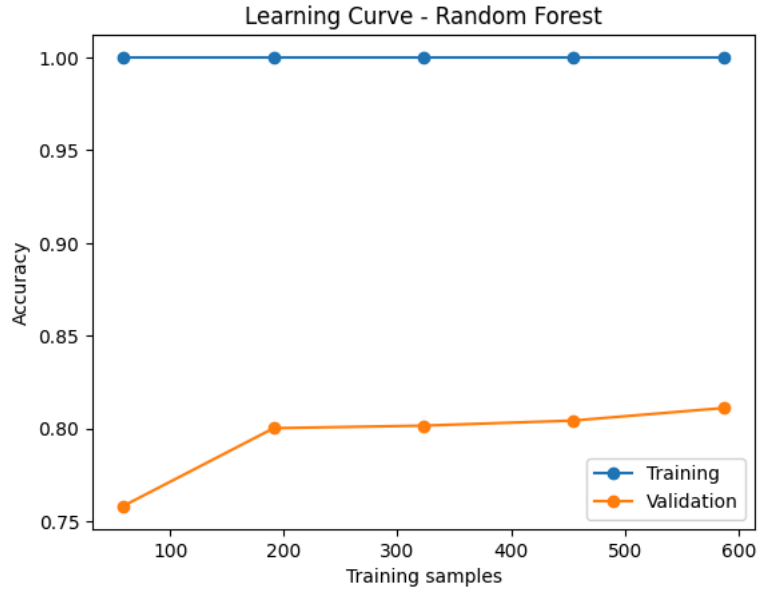



```
# Random Forest Evaluation

y_pred_rf = rf_model.predict(X_test_scaled)

print("Confusion Matrix (Random Forest):\n", confusion_matrix(y_test, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf, digits=4))
```

```
#ROC Curve
y_prob_rf = rf_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob_rf)
plt.plot(fpr, tpr, label=f"Random Forest (AUC={auc(fpr,tpr):.3f})")
plt.plot([0,1],[0,1],"k--")
plt.title("ROC Curve - Random Forest"); plt.xlabel("FPR"); plt.ylabel("TPR")
plt.legend(); plt.show()

#Learning Curve
train_sizes, train_scores, val_scores = learning_curve(
    rf_model, X_train_scaled, y_train, cv=5,
    train_sizes=np.linspace(0.1,1.0,5), scoring="accuracy"
)
plt.plot(train_sizes, train_scores.mean(axis=1), "o-", label="Training")
plt.plot(train_sizes, val_scores.mean(axis=1), "o-", label="Validation")
plt.title("Learning Curve - Random Forest")
plt.xlabel("Training samples"); plt.ylabel("Accuracy")
plt.legend(); plt.show()
```

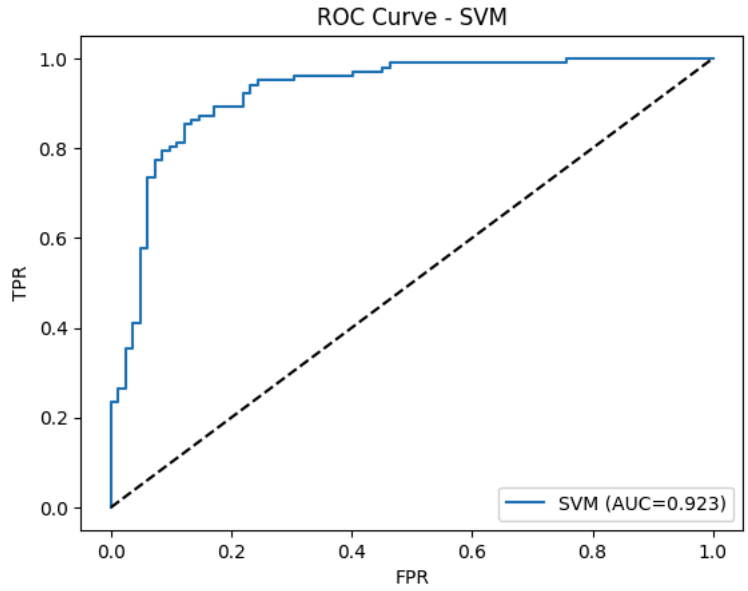
```
Confusion Matrix (Random Forest):
 [[68 14]
 [15 87]]

Classification Report:
              precision    recall  f1-score   support

           0     0.8193    0.8293    0.8242        82
           1     0.8614    0.8529    0.8571       102

    accuracy                         0.8424       184
   macro avg     0.8403    0.8411    0.8407       184
weighted avg     0.8426    0.8424    0.8425       184
```

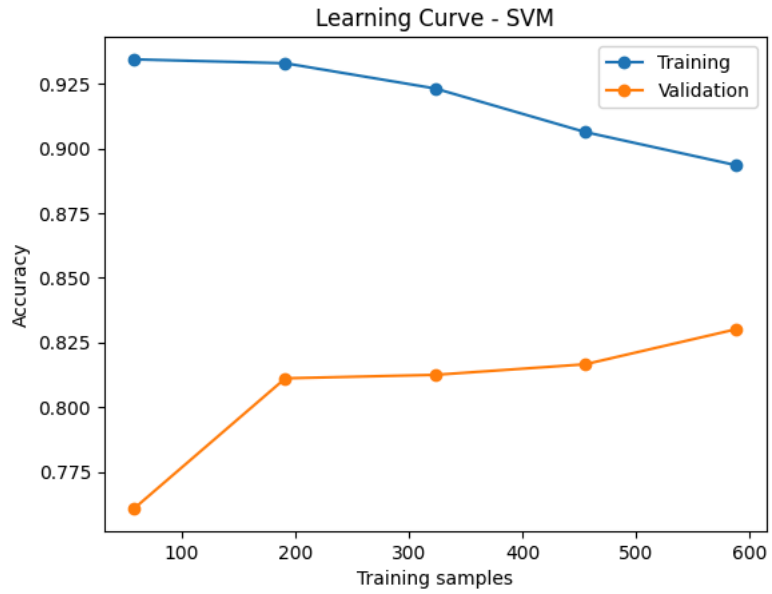




```
# SVM Evaluation

y_pred_svm = svm_model.predict(X_test_scaled)

print("Confusion Matrix (SVM):\n", confusion_matrix(y_test, y_pred_svm))
print("\nClassification Report:\n", classification_report(y_test, y_pred_svm, digits=4))

#ROC Curve
y_prob_svm = svm_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob_svm)
plt.plot(fpr, tpr, label=f"SVM (AUC={auc(fpr,tpr):.3f})")
plt.plot([0,1],[0,1],"k--")
plt.title("ROC Curve - SVM"); plt.xlabel("FPR"); plt.ylabel("TPR")
plt.legend(); plt.show()

#Learning Curve
train_sizes, train_scores, val_scores = learning_curve(
    svm_model, X_train_scaled, y_train, cv=5,
    train_sizes=np.linspace(0.1,1.0,5), scoring="accuracy"
)
plt.plot(train_sizes, train_scores.mean(axis=1), "o-", label="Training")
plt.plot(train_sizes, val_scores.mean(axis=1), "o-", label="Validation")
plt.title("Learning Curve - SVM")
plt.xlabel("Training samples"); plt.ylabel("Accuracy")
plt.legend(); plt.show()
```

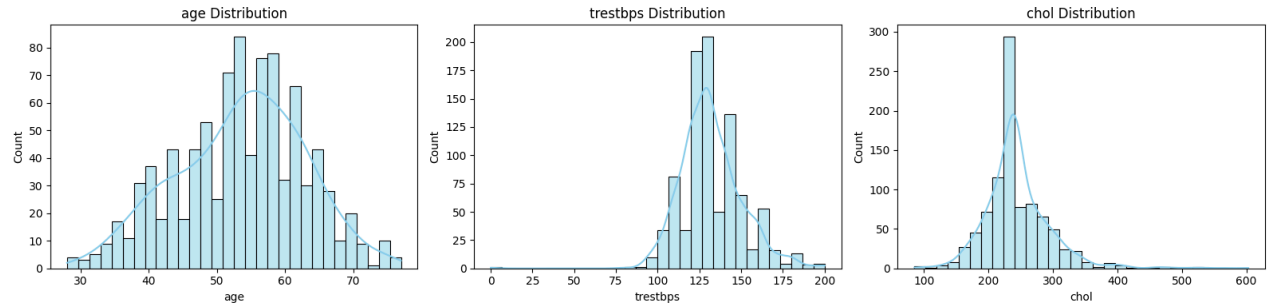
```
Confusion Matrix (SVM):
 [[63 19]
 [ 6 96]]

Classification Report:
              precision    recall  f1-score   support

           0     0.9130    0.7683    0.8344        82
           1     0.8348    0.9412    0.8848       102

    accuracy                         0.8641       184
   macro avg     0.8739    0.8547    0.8596       184
weighted avg     0.8697    0.8641    0.8624       184
```



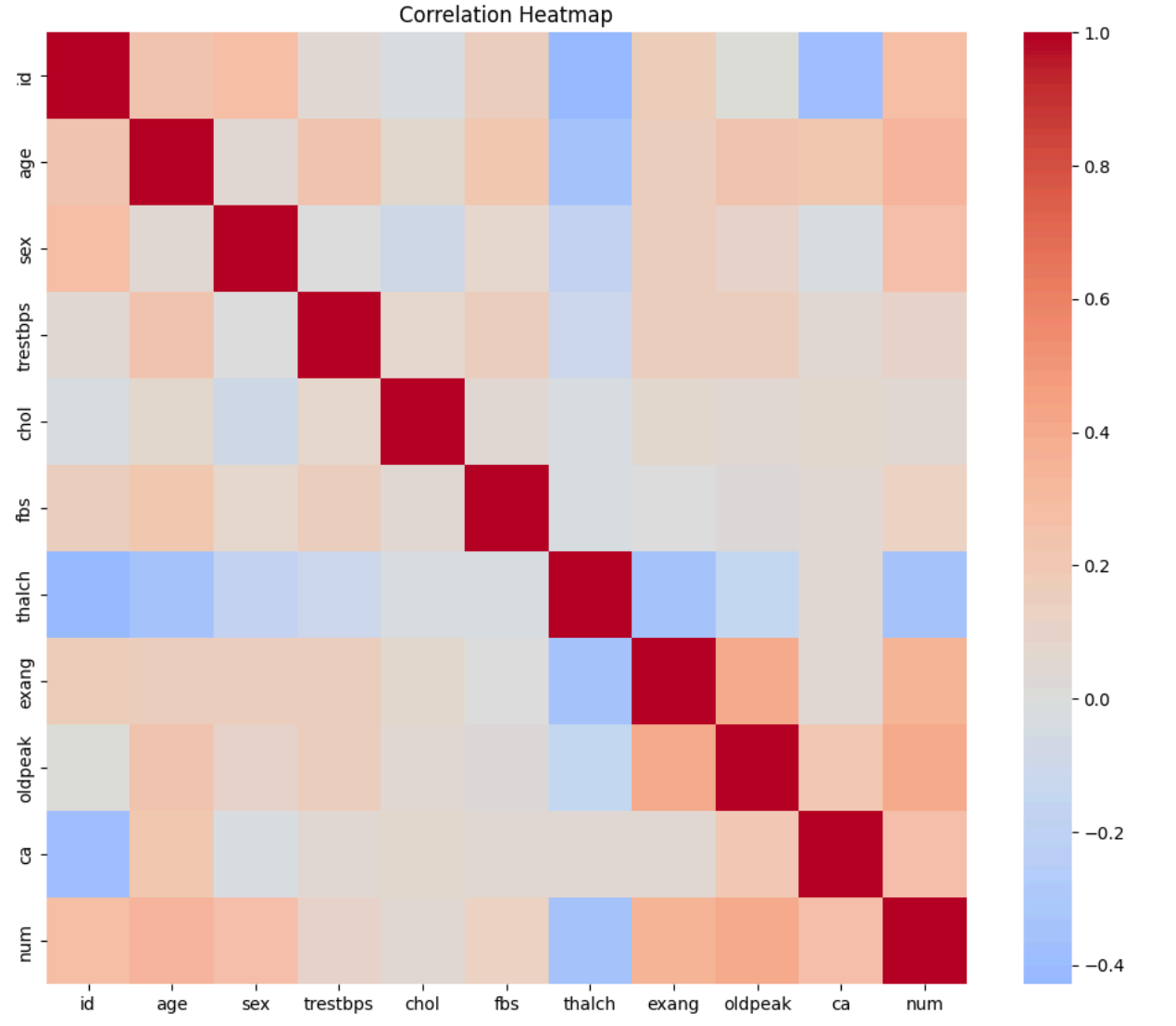


Visualization Requirements

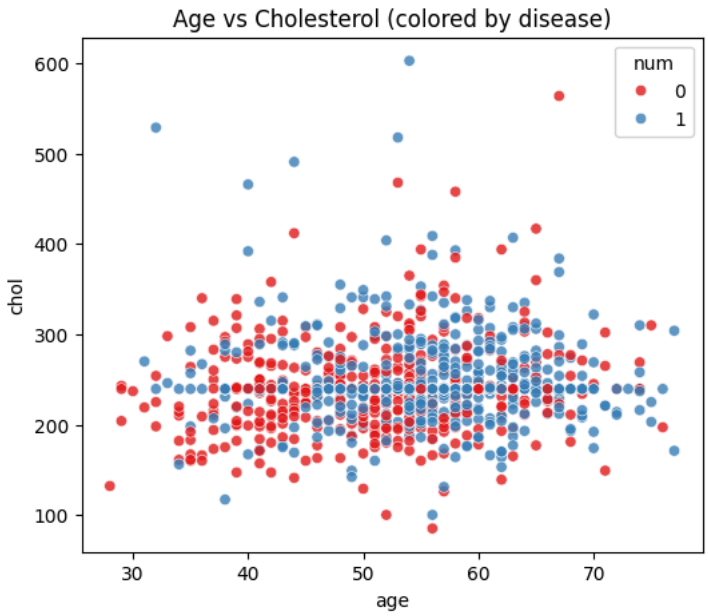
```
#Histograms
fig, axes = plt.subplots(1, 3, figsize=(16, 4))
for ax, col in zip(axes, ["age", "trestbps", "chol"]):
    if col in df.columns:
        sns.histplot(df[col], bins=30, kde=True, ax=ax, color="skyblue")
        ax.set_title(f"{col} Distribution")
plt.tight_layout()
plt.show()
```

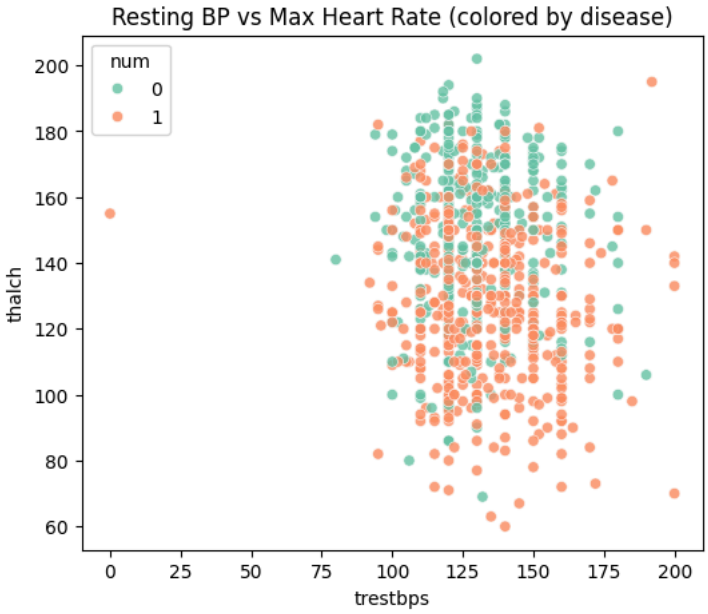



```
#Correlation Heatmap
corr = df.select_dtypes(include=[np.number]).corr()
plt.figure(figsize=(12, 10))
sns.heatmap(corr, cmap="coolwarm", center=0, annot=False)
plt.title("Correlation Heatmap")
plt.show()
```




```
#Scatter Plots
if {"age","chol"}.issubset(df.columns):
    plt.figure(figsize=(6, 5))
    sns.scatterplot(x=df["age"], y=df["chol"], hue=y, palette="Set1", alpha=0.8)
    plt.title("Age vs Cholesterol (colored by disease)")
    plt.show()

if {"trestbps","thalch"}.issubset(df.columns):
    plt.figure(figsize=(6, 5))
    sns.scatterplot(x=df["trestbps"], y=df["thalch"], hue=y, palette="Set2", alpha=0.8)
    plt.title("Resting BP vs Max Heart Rate (colored by disease)")
    plt.show()
```


Age vs Cholesterol (colored by disease)
num
0
1
chol
age
600
500
400
300
200
100
30
40
50
60
70


Resting BP vs Max Heart Rate (colored by disease)
num
0
1
thalch
trestbps
200
180
160
140
120
100
80
60
0
25
50
75
100
125
150
175
200