```
# Importing libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.preprocessing import StandardScaler

from sklearn.feature_selection import SelectKBest, f_classif

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import StackingClassifier, VotingClassifier

from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, accuracy_score,
classification_report

import matplotlib.pyplot as plt

import seaborn as sns


# Load dataset

file_path = "/content/heart_disease.csv"  # Upload your dataset

data = pd.read_csv(file_path)


# Inspecting the dataset

print("Dataset Shape:", data.shape)

print("Dataset Sample:\n", data.head())


# Data preprocessing

# Handle missing values if any

data = data.dropna()
```

```
# Splitting features and target
X = data.drop("target", axis=1)  # Replace 'target' with your dataset's target column name
y = data["target"]

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Feature selection
selector = SelectKBest(score_func=f_classif, k=10)
X_selected = selector.fit_transform(X_scaled, y)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", selected_features)

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.3, random_state=42)

# Model building
knn = KNeighborsClassifier()
nb = GaussianNB()
dt = DecisionTreeClassifier(random_state=42)

# Stacking Ensemble
stacking = StackingClassifier(
```

```
    estimators=[('knn', knn), ('nb', nb), ('dt', dt)],
    final_estimator=DecisionTreeClassifier(random_state=42)
)

# Voting Ensemble
voting = VotingClassifier(
    estimators=[('knn', knn), ('nb', nb), ('dt', dt)],
    voting='soft'
)

# Training models
models = {"KNN": knn, "Naive Bayes": nb, "Decision Tree": dt, "Stacking": stacking, "Voting": voting}

for name, model in models.items():
    model.fit(X_train, y_train)
    print(f"{name} trained successfully.")

# Evaluating models
for name, model in models.items():
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print(f"{name} Accuracy: {acc:.2f}")
    print(f"{name} Classification Report:\n", classification_report(y_test, y_pred))
```

```
    # Plotting confusion matrix
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"{name} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.show()

    # ROC Curve
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_prob)
        auc_score = roc_auc_score(y_test, y_prob)
        plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")
    else:
        print(f"{name} does not support predict_proba.")

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```