# ***Heart Disease Prediction Pipeline:***

*Here's the code broken down into clear parts with simple explanations:*

## 1. Importing Libraries:

```
import warnings
warnings.filterwarnings("ignore")
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.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve, classifi
cation_report
from sklearn.feature_selection import SelectKBest, f_classif
import joblib
```

**What it does:**

- Imports all necessary tools for data processing, visualization and machine learning
- Silences warnings to keep output clean
- Key libraries: pandas (data), sklearn (ML), matplotlib/seaborn (plots)

## 2. Loading and Checking Data:

```
def load_and_inspect(path="heart.csv"):
    df = pd.read_csv(path)
    print("Dataset shape:", df.shape)
    print("\nFirst 5 rows:\n", df.head())
    print("\nMissing values:\n", df.isnull().sum())
    print("\nDuplicates:", df.duplicated().sum())
    return df
```

**What it does:**

- Loads heart disease data from CSV file
- Shows basic info: number of rows/columns
- Checks for missing data and duplicates
- Returns clean dataframe

## 3. Data Visualization:

```
def basic_eda(df):
    # Plot target distribution
    sns.countplot(x='target', data=df)
    # Show correlations
    sns.heatmap(df.corr(), annot=True)
    # Plot age distribution
    sns.histplot(data=df, x='age', hue='target')
```

**What it does:**

Creates 3 types of plots:

1. Heart disease cases count (target variable)
2. Correlation between features
3. Age distribution split by disease status

## 4. Preparing Data for Modeling:

```
def preprocess(df):
    # Remove duplicates
    df = df.drop_duplicates()
    # Split data
    X = df.drop('target', axis=1)
    y = df['target']
    # Create train/test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    return X_train, X_test, y_train, y_test, scaler
```

**What it does:**

- Cleans data by removing duplicates
- Separates features (X) from target (y)
- Splits data into training (80%) and testing (20%) sets
- Normalizes all features to same scale

## 5. Selecting Important Features:

```
def feature_selection(X_train, y_train, X_test):
    selector = SelectKBest(score_func=f_classif, k=8)
    X_train = selector.fit_transform(X_train, y_train)
    X_test = selector.transform(X_test)
    return X_train, X_test, selector
```

**What it does:**

- Chooses top 8 most important features
- Uses statistical test (ANOVA) to find best predictors
- Reduces dataset to only these key features

## 6. Building and Testing Models:

```
def train_and_evaluate(X_train, y_train, X_test, y_test):
    # Logistic Regression
    lr = LogisticRegression()
    lr.fit(X_train, y_train)

    # Decision Tree
    dt = DecisionTreeClassifier()
    dt.fit(X_train, y_train)

    # Evaluate both models
    print("Logistic Regression Accuracy:", accuracy_score(y_test, lr.predict(X_test)))
    print("Decision Tree Accuracy:", accuracy_score(y_test, dt.predict(X_test))))

    # Save models
    joblib.dump(lr, "lr_model.joblib")
    joblib.dump(dt, "dt_model.joblib")
```

**What it does:**

- Trains two different models:
    1. Logistic Regression (simpler)
    2. Decision Tree (more complex)
- Checks accuracy on test data
- Saves trained models for future use

## 7. Running the Full Pipeline:

```python
def main():
    df = load_and_inspect("heart.csv")
    basic_eda(df)
    X_train, X_test, y_train, y_test, scaler = preprocess(df)
    X_train, X_test, selector = feature_selection(X_train, y_train, X_test)
    train_and_evaluate(X_train, y_train, X_test, y_test)

if __name__ == "__main__":
    main()
```

**What it does:**

- Runs all steps in order:
    1. Load data
    2. Explore data
    3. Prepare data
    4. Select features
    5. Train models
    6. Evaluate results
- Can be run with one command

*Each part has a clear purpose and connects to the next step, creating a complete machine learning pipeline from raw data to trained models. I trust that the explanation has given you a clear view of how each part of the code works.*
*From data loading to visualization, every step was structured to meet the task requirements.*
*This breakdown should help in understanding both the logic and the purpose behind the implementation.*

# ***Full Code:***

```
# heart_disease_full_pipeline.py

# Make sure 'heart.csv' is in the same folder and run: python
heart_disease_full_pipeline.py

# Requirements: pandas, numpy, matplotlib, seaborn, scikit-learn, joblib

# Install with: pip install pandas numpy matplotlib seaborn scikit-learn joblib

import warnings

warnings.filterwarnings("ignore") # Suppress warnings for cleaner output

# Importing libraries for data handling, visualization, and machine learning

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.tree import DecisionTreeClassifier

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

from sklearn.feature_selection import SelectKBest, f_classif

import joblib

RANDOM_STATE = 42  # For reproducibility

#  1. Load and Inspect Data

def load_and_inspect(path="heart.csv"):

    df = pd.read_csv(path)  # Read CSV file into DataFrame
```

```
    print("Dataset shape:", df.shape)  # Shape of dataset
    print("\nFirst 5 rows:\n", df.head())  # Preview data
    print("\nData types and info:")
    print(df.info())  # Data types and null info
    print("\nSummary statistics:\n", df.describe().T)  # Summary stats
    print("\nMissing values per column:\n", df.isnull().sum())  # Missing values check
    print("\nNumber of duplicate rows:", df.duplicated().sum())  # Duplicate rows check
    return df
# 2. Exploratory Data Analysis
def basic_eda(df):
    # Count plot for target variable distribution
    plt.figure(figsize=(6,4))
    sns.countplot(x='target', data=df)
    plt.title("Heart Disease Distribution (0 = No, 1 = Yes)")
    plt.xlabel("target")
    plt.tight_layout()
    plt.show()
    # Correlation heatmap
    plt.figure(figsize=(12,10))
    sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap="coolwarm")
    plt.title("Feature Correlation Matrix")
    plt.tight_layout()
    plt.show()
    # Pairplot for selected continuous features
    cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'target']
```

```
    sns.pairplot(df[cols], hue='target', diag_kind='kde', corner=True)
    plt.suptitle("Pairplot (selected features)", y=1.02)
    plt.show()
    # Age distribution by target
    plt.figure(figsize=(10,6))
    sns.histplot(data=df, x='age', hue='target', bins=30, kde=True,
multiple='stack')
    plt.title('Age distribution by target')
    plt.tight_layout()
    plt.show()
# 3. Preprocessing
def preprocess(df, test_size=0.2):
    df = df.drop_duplicates().reset_index(drop=True)  # Remove duplicates
    X = df.drop('target', axis=1)  # Features
    y = df['target']  # Target variable
    # Train-test split with stratification to preserve class balance
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=RANDOM_STATE, stratify=y
    )
    # Scale features for better ML performance
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    return X, X_train, X_test, X_train_scaled, X_test_scaled, y_train, y_test,
scaler
# 4. Feature Selection
def feature_selection(X_columns, X_train_scaled, y_train, X_test_scaled, k=8):
```

```
    k = min(k, X_train_scaled.shape[1])  # Prevent selecting more features than available
    selector = SelectKBest(score_func=f_classif, k=k)
    selector.fit(X_train_scaled, y_train)
    # Get boolean mask of selected features
    support = selector.get_support()
    selected_features = list(X_columns[support])
    print("\nSelected top-{} features: {}".format(k, selected_features))
    # Transform datasets to keep only selected features
    X_train_sel = selector.transform(X_train_scaled)
    X_test_sel = selector.transform(X_test_scaled)
    return selector, selected_features, X_train_sel, X_test_sel
# 5. Model Training and Evaluation
def train_and_evaluate(X_train_sel, y_train, X_test_sel, y_test, selected_features, scaler, selector):
    results = {}
    # ----- Logistic Regression -----
    lr = LogisticRegression(max_iter=2000, random_state=RANDOM_STATE)
    lr.fit(X_train_sel, y_train)
    y_pred_lr = lr.predict(X_test_sel)
    y_prob_lr = lr.predict_proba(X_test_sel)[:,1]
    acc_lr = accuracy_score(y_test, y_pred_lr)
    auc_lr = roc_auc_score(y_test, y_prob_lr)
    print("\n=== Logistic Regression ===")
    print("Accuracy: {:.4f}".format(acc_lr))
    print("ROC AUC : {:.4f}".format(auc_lr))
    print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

```
# Confusion matrix heatmap
cm_lr = confusion_matrix(y_test, y_pred_lr)
plt.figure(figsize=(5,4))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues')
plt.title("Logistic Regression Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
# ROC curve
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob_lr)
plt.figure(figsize=(6,4))
plt.plot(fpr_lr, tpr_lr, label=f'LR (AUC = {auc_lr:.2f})')
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
# ----- Decision Tree -----
dt = DecisionTreeClassifier(random_state=RANDOM_STATE,
max_depth=5)
dt.fit(X_train_sel, y_train)
y_pred_dt = dt.predict(X_test_sel)
y_prob_dt = dt.predict_proba(X_test_sel)[:,1]
```

```
acc_dt = accuracy_score(y_test, y_pred_dt)
auc_dt = roc_auc_score(y_test, y_prob_dt)
print("\n=== Decision Tree ===")
print("Accuracy: {:.4f}".format(acc_dt))
print("ROC AUC : {:.4f}".format(auc_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
# Confusion matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(5,4))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Greens')
plt.title("Decision Tree Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
# ROC curve
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_prob_dt)
plt.figure(figsize=(6,4))
plt.plot(fpr_dt, tpr_dt, label=f'DT (AUC = {auc_dt:.2f})')
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```

```python
# Feature importance plots
lr_coef_df = pd.DataFrame({
    'feature': selected_features,
    'coefficient': lr.coef_[0]
}).sort_values(by='coefficient', ascending=False)
dt_imp_df = pd.DataFrame({
    'feature': selected_features,
    'importance': dt.feature_importances_
}).sort_values(by='importance', ascending=False)
plt.figure(figsize=(8,5))
sns.barplot(x='coefficient', y='feature', data=lr_coef_df)
plt.title('Logistic Regression - Feature Coefficients')
plt.tight_layout()
plt.show()
plt.figure(figsize=(8,5))
sns.barplot(x='importance', y='feature', data=dt_imp_df)
plt.title('Decision Tree - Feature Importances')
plt.tight_layout()
plt.show()
# Model comparison bar chart
results_df = pd.DataFrame({
    'Model': ['Logistic Regression', 'Decision Tree'],
    'Accuracy': [acc_lr, acc_dt],
    'ROC AUC': [auc_lr, auc_dt]
}).set_index('Model')
```

```
    print("\nModel comparison:\n", results_df)
    results_df.plot(kind='bar', figsize=(8,5))
    plt.title('Model Performance Comparison')
    plt.ylabel('Score')
    plt.ylim(0,1)
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.show()
    # Save models and preprocessing objects for later use
    joblib.dump(lr, "logistic_regression_model.joblib")
    joblib.dump(dt, "decision_tree_model.joblib")
    joblib.dump(scaler, "scaler.joblib")
    joblib.dump(selector, "feature_selector.joblib")
    print("\nSaved: logistic_regression_model.joblib, decision_tree_model.joblib, scaler.joblib, feature_selector.joblib")
 # Return results
    results = {
        'lr': lr, 'dt': dt,
        'scaler': scaler, 'selector': selector,
        'lr_metrics': {'accuracy': acc_lr, 'roc_auc': auc_lr},
        'dt_metrics': {'accuracy': acc_dt, 'roc_auc': auc_dt},
        'selected_features': selected_features
    }
    return results
```

```python
#  6. Main Execution
def main(filepath="heart.csv"):
 # Load the dataset and show its basic structure
    df = load_and_inspect(filepath)
    # Perform initial data exploration (EDA) like shape, summary, and missing values
    basic_eda(df)
    # Preprocess data: split into train/test, scale features
    X, X_train, X_test, X_train_scaled, X_test_scaled, y_train, y_test, scaler = preprocess(df, test_size=0.2)
    # Perform feature selection to choose top 8 features for the model
    selector, selected_features, X_train_sel, X_test_sel = feature_selection(X.columns, X_train_scaled, y_train, X_test_scaled, k=8)
    # Train multiple models, evaluate performance, and display results
    results = train_and_evaluate(X_train_sel, y_train, X_test_sel, y_test, selected_features, scaler, selector)
if __name__ == "__main__": # Ensures the main function runs only when this script is executed directly, not when imported as a module
    # Run the main function using the 'heart.csv' dataset
    main("heart.csv")
    # Confirmation messages after successful code execution
    print("\n--- Task Completed ---")
```

*The End....!!!*