This process will include data loading, cleaning, visualization, and model creation.

---

1. Define the Problem and Objective

- **Objective**: Predict the likelihood of heart disease based on various features (age, sex, cholesterol levels, etc.).
- **Goal**: Perform EDA to understand the relationships between features and the target variable (`target`), which indicates if a person has heart disease (1) or not (0).

---

2. Load the Dataset

```
import pandas as pd

# Load Heart Disease dataset
df = pd.read_csv('heart_disease.csv')

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

---

3. Data Exploration (EDA)

Check the Shape and Data Types

```
# Check dataset shape
print(f"Dataset Shape: {df.shape}")

# Data types and missing values
print(df.info())

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

Visualize Distributions

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

# Distribution of Age
sns.histplot(df['age'], kde=True)
plt.title("Age Distribution")
plt.show()

# Target variable distribution (Heart Disease)
sns.countplot(x='target', data=df)
plt.title("Heart Disease Distribution")
plt.show()
```

4. Data Cleaning

Handle Missing Values

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

# Fill missing values (if any)
# Example: Fill missing 'chol' (cholesterol) with median value
df['chol'].fillna(df['chol'].median(), inplace=True)

# Check if missing values are handled
print(df.isnull().sum())
```

Handle Duplicates

```
# Remove duplicate rows if any
df.drop_duplicates(inplace=True)
```

---

5. Feature Engineering (Optional)

Create New Features (if needed)

```
# Example: Create a new feature 'age_group' based on age ranges
df['age_group'] = pd.cut(df['age'], bins=[0, 30, 50, 70, 100],
labels=['Young', 'Middle-Aged', 'Senior', 'Elderly'])

# Display unique age groups
print(df['age_group'].unique())
```

---

6. Data Preprocessing

Convert Categorical Variables to Numerical

```
# Convert categorical features like 'sex' and 'age_group' into numerical
values
df['sex'] = df['sex'].map({0: 'Female', 1: 'Male'})  # Optional: for better
understanding
df['sex'] = df['sex'].map({'Female': 0, 'Male': 1})

# One-hot encode 'age_group'
df = pd.get_dummies(df, columns=['age_group'], drop_first=True)

# Define features (X) and target (y)
X = df.drop(columns=['target'])
y = df['target']
```

Scale Numerical Features

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# Scale numerical columns: Age, Cholesterol, etc.
X[['age', 'chol', 'trestbps', 'thalach']] = scaler.fit_transform(X[['age',
'chol', 'trestbps', 'thalach']])
```

---

7. Split the Data

```
from sklearn.model_selection import train_test_split

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

---

8. Model Selection

Choose the Model

```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
```

---

9. Model Training

```
# Train the model on the training data
model.fit(X_train, y_train)
```

---

10. Model Evaluation

Predictions and Evaluation Metrics

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Confusion Matrix Visualization

```
# Visualize confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues', cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

---

11. Model Interpretation and Insights

Feature Importance

```
# Get feature importances from the trained model
importances = model.feature_importances_

# Create a DataFrame to display features and their importance
features = X.columns
feature_importance_df = pd.DataFrame({'Feature': features, 'Importance':
importances})

# Sort by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)

# Plot feature importances
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title("Feature Importance")
plt.show()

print(feature_importance_df)
```

---

12. Fine-Tuning and Optimization (Optional)

Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

# Set up grid search parameters for Random Forest
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,
n_jobs=-1, verbose=2)

# Fit grid search
```

```
grid_search.fit(X_train, y_train)

# Get best parameters
print("Best Parameters:", grid_search.best_params_)
```

---

13. Final Reporting

Summarize Insights

- **EDA Insights**: Discuss age, cholesterol, and sex distributions, as well as the correlation between features and the target variable.
- **Model Evaluation**: Summarize the accuracy, classification report, and confusion matrix.
- **Important Features**: List the most important features influencing heart disease predictions.
- **Next Steps**: Possible improvements like tuning hyperparameters or exploring other models (e.g., Logistic Regression, Gradient Boosting).

Additional Notes:

- Start with a **basic model** like Random Forest and evaluate its performance.
- Use **cross-validation** to assess model robustness.
- Perform **feature engineering** based on domain knowledge (e.g., creating age groups or new features).
