Preprocessing Report on Heart Disease Dataset

Name: Kalubowila K S U

IT No: IT23256378

Course: Fundamentals of Data Mining

Date: August 30, 2025

GitHub Repository: https://github.com/SusheniUmayangana/HeartDisease-Preprocessing

1. Table of Contents

2. Abstract

This report outlines the preprocessing steps applied to the Heart Disease dataset in preparation for classification modeling. The process includes data cleaning, feature selection, encoding of categorical variables, feature scaling, and train-test splitting. These steps ensure the dataset is structured and optimized for machine learning applications.

3. Introduction

Preprocessing is a foundational step in any data mining workflow. Raw datasets often contain inconsistencies, missing values, and unstructured features that can hinder model performance. This report documents the systematic approach taken to clean and transform the Heart Disease dataset, ensuring its suitability for predictive modeling and analysis.

All project files - including the original dataset (heart.csv), preprocessed dataset (heart_preprocessed.csv), Colab notebook, and final report - are available in the GitHub repository: HeartDisease-Preprocessing

4. Dataset Description

- **Source:** Kaggle – Heart Disease Dataset by John Smith
- **Format:** CSV
- **Rows and columns:** ~300 rows, 14 columns
- **Target variable:** target (1 = presence of heart disease, 0 = absence)
- **Features:** age, sex, chest pain type, cholesterol, resting blood pressure, etc.
- **Goal:** Predict likelihood of heart disease based on patient attributes

5. Preprocessing Steps

a. Data Inspection

- Used .head(), .info(), .describe(), and .isnull().sum() to understand the structure and check for missing values.

```
[14] # Import libraries
     import pandas as pd
     import numpy as np
     from sklearn.preprocessing import StandardScaler
     from sklearn.model_selection import train_test_split
```

```
[15] # Load the dataset
     df = pd.read_csv('heart.csv')
```

```
[3]  print(df.head())
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

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

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None
```

```
print(df.describe())
```

```
                age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000   
mean     54.434146     0.695610     0.942439   131.611707   246.00000   
std       9.072290     0.460373     1.029641    17.516718    51.59251   
min      29.000000     0.000000     0.000000    94.000000   126.00000   
25%      48.000000     0.000000     0.000000   120.000000   211.00000   
50%      56.000000     1.000000     1.000000   130.000000   240.00000   
75%      61.000000     1.000000     2.000000   140.000000   275.00000   
max      77.000000     1.000000     3.000000   200.000000   564.00000   

               fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000   
mean      0.149268     0.529756   149.114146     0.336585     1.071512   
std       0.356527     0.527878    23.005724     0.472772     1.175053   
min       0.000000     0.000000    71.000000     0.000000     0.000000   
25%       0.000000     0.000000   132.000000     0.000000     0.000000   
50%       0.000000     1.000000   152.000000     0.000000     0.800000   
75%       0.000000     1.000000   166.000000     1.000000     1.800000   
max       1.000000     2.000000   202.000000     1.000000     6.200000   

            slope           ca         thal       target  
count  1025.000000  1025.000000  1025.000000  1025.000000  
mean      1.385366     0.754146     2.323902     0.513171  
std       0.617755     1.030798     0.620660     0.500070  
min       0.000000     0.000000     0.000000     0.000000  
25%       1.000000     0.000000     2.000000     0.000000  
50%       1.000000     0.000000     2.000000     1.000000  
75%       2.000000     1.000000     3.000000     1.000000  
max       2.000000     4.000000     3.000000     1.000000  
```

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

```
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
```

b. Handling Missing Values

- No missing values were found, but code included dropna() as a precaution.

```
[18] # Handle missing values & duplicates
     df = df.dropna()
     df.drop_duplicates(inplace=True)
```

c. Encoding Categorical Variables

- Checked for object-type columns.
- Applied pd.get_dummies() to encode categorical features (if any).

```
[8]  df['sex'] = df['sex'].astype('category')
     df['cp'] = df['cp'].astype('category')
```

```
[17] # Encode categorical variables
     categorical_cols = df.select_dtypes(include='object').columns
     print("\nCategorical columns:", categorical_cols)
```

```
Categorical columns: Index([], dtype='object')
```

```
     df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

d. Feature Scaling

- Used StandardScaler to normalize numerical features for better model performance.

```
[11] scaler = StandardScaler()
     scaled_features = scaler.fit_transform(df.drop('target', axis=1))
```

e. Train-Test Split

- Split the dataset into training and testing sets (80/20 ratio).

```
[12] # Train-test split
     X = scaled_features
     y = df['target']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

f. Saving the Preprocessed Dataset

- Saved the final version for future modeling

```
[13] # save the scaled features
     processed_df = pd.DataFrame(X, columns=df.drop('target', axis=1).columns)
     processed_df['target'] = y.values

     # Save to CSV
     processed_df.to_csv('heart_preprocessed.csv', index=False)
```

6. Conclusion

The Heart Disease dataset has been successfully cleaned, transformed, and split for modeling. These preprocessing steps ensure that the data is ready for classification algorithms and further analysis. The structured approach enhances reproducibility and model accuracy.

7. References

- Kaggle Dataset: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset