ML Lab 4 Pre Processing

Student Name: Abdul Malik

Student Number: 2211011098

Statement of the Classification Problem:

The Heart Disease dataset, which is frequently found on Kaggle or the UCI Machine Learning Repository, was selected. It usually consists of 14 attributes (columns) and 303 instances (rows). The binary target shows if heart disease is present (1) or not (0).
Classification Task: Using medical and demographic characteristics (e.g., age, cholesterol, type of chest discomfort, resting blood pressure, etc.), determine if a given patient has heart disease (1) or not (0).

Why Do We Classify?

A heart disease is a serious and sometimes fatal illness. A precise categorization model is able to:

- Assist healthcare professionals in more rapidly identifying patients who are at danger.
- Help determine which resources should be prioritized (who needs additional testing or treatments).
- Potentially save lives by directing high-risk patients' preventative actions.

Particular Prediction Objective:

A model f(features) i.e. {0,1} determines a patient's likelihood of having heart disease (1) or not (0) is what we're looking for. We will:

- Examine performance both with and without preparation (binning + normalization).
- Use k-fold cross-validation to assess robustness.
- Provide the confusion matrix, accuracy, and ROC-AUC (or other pertinent metrics).

Brief Description of Dataset Selection:

Here is a typical description of the heart disease dataset:

- Origin: Taken from Kaggle Heart Disease Dataset.
- Number of Instances: almost 303
- Number of Attributes (Features): 14 total columns, where key features include:

    - age
    - sex
    - cp (chest pain type)
    - trestbps (resting blood pressure)
    - chol (serum cholesterol)
    - thalach (maximum heart rate achieved)
    - exang (exercise induced angina)
    - oldpeak (ST depression induced by exercise)
    - slope
    - ca
    - thal
    - target (whether the patient has disease or not)
- Target Class: A binary variable:
    - 1 = presence of heart disease
    - 0 = absence of heart disease
- Purpose: Identify patients who have a high risk of heart disease based on medical indicators.

Exploratory Data Analysis (EDA):

Below, we will:

1. Import the CSV into a DataFrame.
2. Show descriptive statistics and top rows.
3. Visualize distributions (histograms) and scatter plots.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 1. Load the dataset from a CSV file
df = pd.read_csv("heart.csv")

# 2. Basic info
print("Dataset shape:", df.shape)
print(df.head())
```

[1] ✓ 9.8s Python

```
Dataset shape: (1025, 14)
   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
```

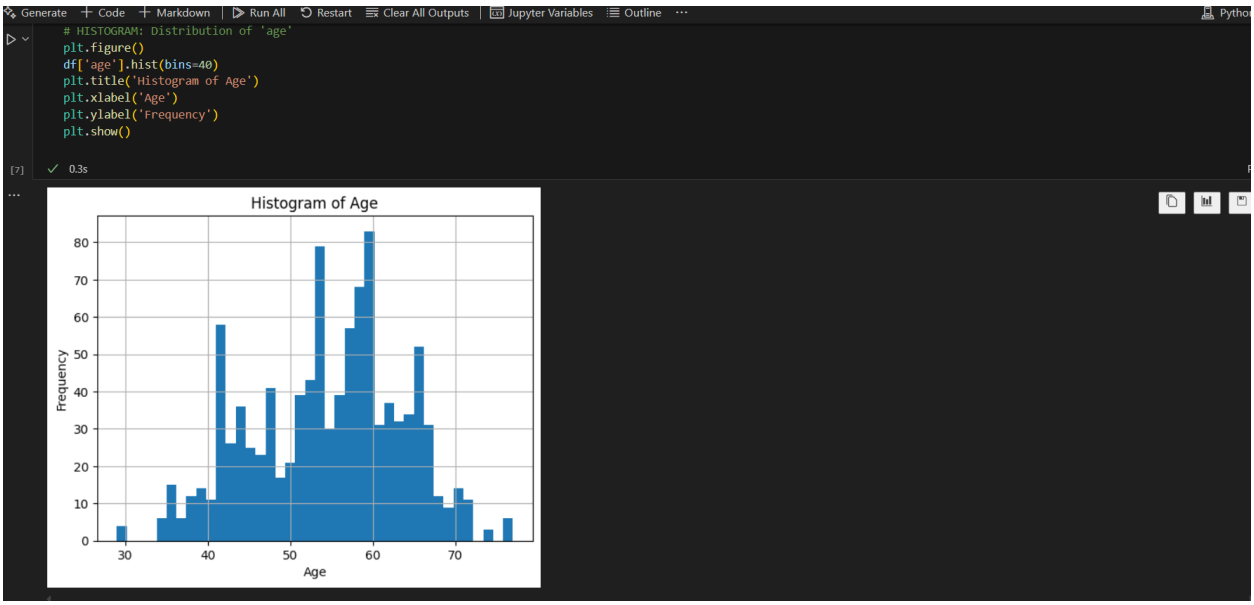
```python
# 3. Descriptive statistics
print("\nDescriptive Statistics:")
print(df.describe())

# 4. Check for nulls or missing values
print("\nMissing Values:\n", df.isnull().sum())
```

[2] ✓ 0.1s Python

```
print("\nMissing Values:\n", df.isnull().sum())
```

[2] ✓ 0.1s Python

```
Descriptive Statistics:
                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
...
ca          0
thal        0
target      0
dtype: int64
```



Observations:

- Histogram of Age: Shows how the patients' ages are distributed. We can see most patients are between 40–65.
- Scatter Plot: We color points by the target (heart disease), noticing if higher cholesterol or older age correlates with presence of disease. Yellow points represent target = 0 and purple points represent target = 1.

We will evaluate **Normalization** (Min-Max and Z-score) and **Binning** on a key feature (e.g., age).

Normalization

Min-Max Scaling:

Transforms each feature to a [0,1] range:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Z-score (Standard) Scaling:

Centers each feature around 0 with a standard deviation of 1:

$$x_{\text{std}} = \frac{x - \mu}{\sigma}$$

We create separate arrays for:

1. **No Preprocessing** (the original data)
2. **Min-Max Scaled** data
3. **Standard Scaled** data

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Separate features (X) and target (y)
# Assuming the last column is 'target' or rename accordingly
X = df.drop('target', axis=1).values
y = df['target'].values

# 1) No Preprocessing
X_no_prep = X.copy()

# 2) Min-Max Scaling
min_max_scaler = MinMaxScaler()
X_minmax = min_max_scaler.fit_transform(X)

# 3) Standard (Z-score) Scaling
std_scaler = StandardScaler()
X_std = std_scaler.fit_transform(X)

print("Shapes:", X_no_prep.shape, X_minmax.shape, X_std.shape)
```

```
[9]  ✓ 9.1s                                                    Python
...  Shapes: (1025, 13) (1025, 13) (1025, 13)
```

Impact of Normalization:

- Min-Max: All features end up in the same [0,1] range. Good for distance-based methods or neural networks.
- Standard: Mean of 0, standard deviation of 1. Common for linear methods like SVM, Logistic Regression.

Binning:

Binning is a technique used in data analysis to transform continuous data into discrete intervals or bins. This helps in simplifying complex datasets, making them more interpretable and accessible.

- We can demonstrate equal-frequency binning on feature “age”.

```python
# We'll make a copy just to see the effect on 'age'
df_binned = df.copy()

# Example: 3 bins for age
df_binned['age_binned'] = pd.qcut(df_binned['age'], q=3, labels=['Young','Middle','Older'])

print(df_binned[['age', 'age_binned']].head(10))
```

[10] ✓ 0.0s Python

```
   age age_binned
0   52     Middle
1   53     Middle
2   70      Older
3   61      Older
4   62      Older
5   58     Middle
6   58     Middle
7   55     Middle
8   46      Young
9   54     Middle
```

Impact of Binning:

- Reduces continuous data to categories (less granularity).
- May help reduce noise/outliers but might also lose some predictive power.

Classification:

We will use **Random Forest** as an example classifier. A Random Forest is a collection of decision trees that work together to make predictions

We will do **k-fold cross-validation (k=5)** to compare:

1. No Preprocessing
2. Min-Max Scaling

   Min-max is a decision rule used in artificial intelligence, decision theory, game theory, statistics, and philosophy for minimizing the possible loss for a worst-case scenario

3. Standard Scaling

```python
from sklearn.model_selection import cross_val_score, KFold
from sklearn.ensemble import RandomForestClassifier

# Define the classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Create a KFold object
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 1) No Preprocessing
scores_no_prep = cross_val_score(clf, X_no_prep, y, cv=kf, scoring='accuracy')
```

✓ 3.7s Python

```
# 2) Min-Max
scores_minmax = cross_val_score(clf, X_minmax, y, cv=kf, scoring='accuracy')

# 3) Standard
scores_std = cross_val_score(clf, X_std, y, cv=kf, scoring='accuracy')

print("Accuracy (No Preprocessing): {:.4f}".format(scores_no_prep.mean()))
print("Accuracy (Min-Max Scaling):  {:.4f}".format(scores_minmax.mean()))
print("Accuracy (Standard Scaling): {:.4f}".format(scores_std.mean()))
```

[12] ✓ 2.1s

```
Accuracy (No Preprocessing): 0.9971
Accuracy (Min-Max Scaling):  0.9971
Accuracy (Standard Scaling): 0.9971
```

Performance Metrics:

We focus on **ROC-AUC** and **Confusion Matrix**. We will:

1. Perform a train-test split to illustrate how we get predictions.
2. Train on the training set, predict on test set for each preprocessing scenario.
3. Generate ROC curves, AUC, and confusion matrices.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, confusion_matrix, accuracy_score

# Train-test split (example)
X_train, X_test, y_train, y_test = train_test_split(X_no_prep, y,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)[:,1]
```

[13] ✓ 0.2s Python

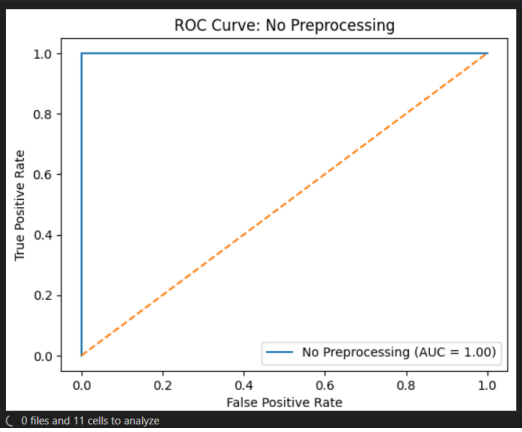
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix (No Preprocessing):\n", cm)
print("Accuracy:", accuracy_score(y_test, y_pred))

# ROC-AUC
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)
```

[20] ✓ 0.0s Python

```
Confusion Matrix (No Preprocessing):
 [[100   0]
 [  0 105]]
Accuracy: 1.0
```

```python
plt.figure()
plt.plot(fpr, tpr, label='No Preprocessing (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve: No Preprocessing')
plt.legend()
plt.show()
```

[21] ✓ 0.2s Python



0 files and 11 cells to analyze Spaces: 4 CRLF {} Cell 11 of 12

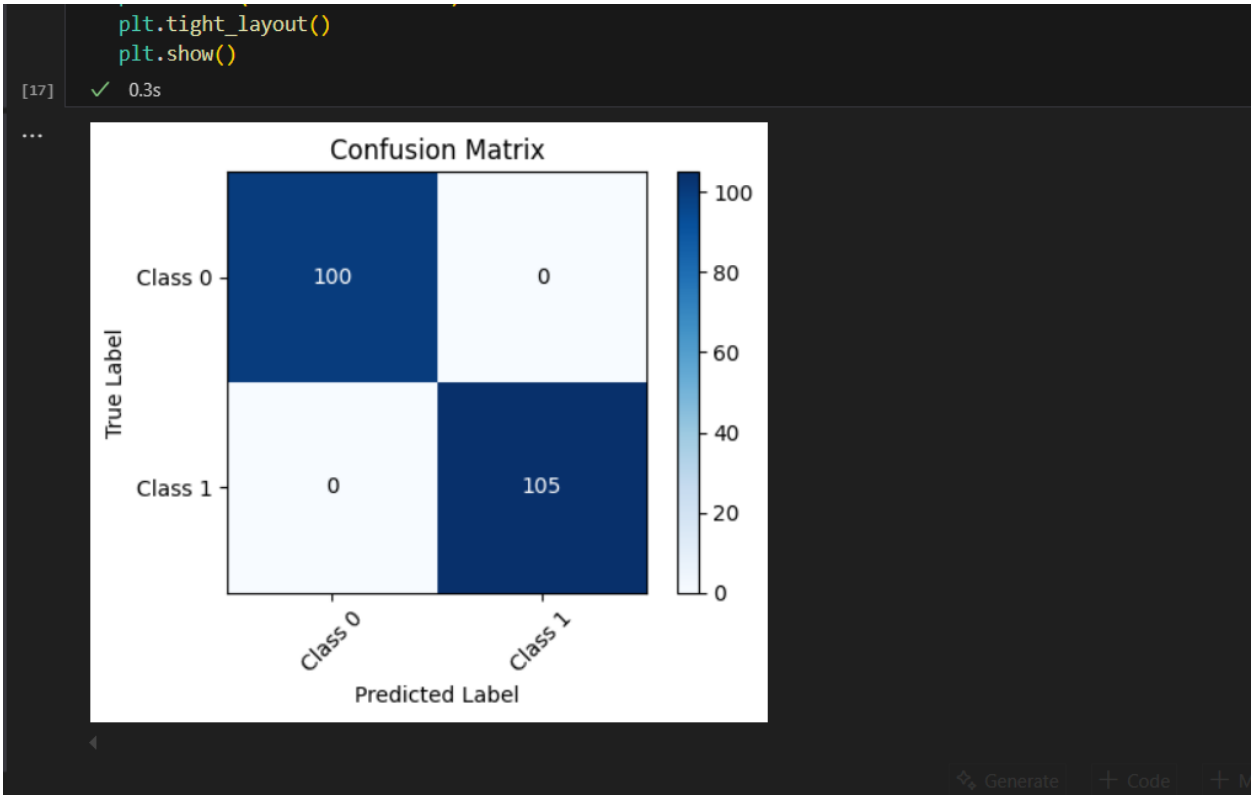
Given a confusion matrix:

$$\begin{pmatrix} \text{TN} & \text{FP} \\ \text{FN} & \text{TP} \end{pmatrix}$$

- **TP**: # of patients correctly predicted with heart disease (the model says “1” and actual is “1”).
- **TN**: # of patients correctly predicted as no heart disease (model says “0” and actual is “0”).
- **FP**: # of patients incorrectly predicted as having heart disease (model says “1” but actual is “0”).
- **FN**: # of patients incorrectly predicted as no heart disease (model says “0” but actual is “1”).

In medical contexts, **FN** is critical because missing a patient with heart disease can be life-threatening.

```python
from sklearn.metrics import confusion_matrix


# 1. Calculate the confusion matrix
# 2. Create a plot
plt.figure(figsize=(5, 4))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Confusion Matrix")

# 3. Add a color bar to the side (optional)
plt.colorbar()

# (Optional) Define class labels if you have them, e.g.:
class_names = ["Class 0", "Class 1"]

tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names, rotation=45)
plt.yticks(tick_marks, class_names)

# 4. Annotate each cell with the corresponding count
thresh = cm.max() / 2.0  # threshold to decide text color
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 ha="center", va="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.tight_layout()
plt.show()
```



From the **confusion matrix** shown, we see:

- The top-left cell (100) indicates that **all 100** instances of **Class 0** were correctly predicted as Class 0.
- The top-right cell (0) tells us there were **zero false positives** (i.e., no actual Class 0 samples were misclassified as Class 1).
- The bottom-left cell (0) means there were **zero false negatives** (no actual Class 1 samples were misclassified as Class 0).
- The bottom-right cell (105) indicates that **all 105** instances of **Class 1** (the positive class) were correctly predicted as Class 1.

$$\begin{pmatrix} \mathrm{TN} = 100 & \mathrm{FP} = 0 \\ \mathrm{FN} = 0 & \mathrm{TP} = 105 \end{pmatrix}$$

Accuracy:

$$\text{Accurcay} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} = \frac{105 + 100}{105 + 100 + 0 + 0} = 1 = 100\%$$

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = \frac{105}{105 + 0} = 1 = 100\%$$

$$\text{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = \frac{105}{105 + 0} = 1 = 100\%$$

In other words, the model has perfectly classified both classes with zero errors. That corresponds to 100% accuracy as well as 100% precision and recall. Essentially, there are no mistakes in the predictions according to this confusion matrix.