**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC QUỐC TẾ**

SUBJECT: DATA MINING

TEACHER: Dr. Nguyen, Thi Thanh Sang

**Task for project: Build a data mining framework from scratch. This framework contains one clustering/classification method and one sequence mining method.**

| **Name** | **Student ID** |
|---|---|
| Bùi Lê Anh Khoa | ITDSIU23009 |
| Lê Quang Dũng (leader) | ITDSIU23031 |
| Nguyễn Hoàng Bảo | ITDSIU23004 |
| Nguyễn Nhật Tiến | ITDSIU23023 |
| Nguyễn Phan Hải Lâm | ITDSIU23012 |

# Table of Contents

1. **Introduction**
   - **1.1 Background and Motivation**
   - **1.2 Project Objectives**
   - **1.3 Dataset Used**
   - **1.4 Methodologies Employed**
   - **1.5 Brief Explanation of Core Concepts**
   - **1.6 Significance in the Context of This Assignment**
   - **1.7 Expected Deliverables**
2. **Data Pre-Processing**
   - **2.1 Raw Data Overview**
   - **2.2 Data Cleaning Process**
     - **2.2.1 Handling Missing Values**
     - **2.2.2 Removing Duplicates**
     - **2.2.3 Outlier Detection and Treatment**
   - **2.3 Data Transformation (Feature Selection)**
   - **2.4 Final Cleaned Dataset**
3. **Classification / Prediction Algorithm**
   - **3.1 Introduction**
   - **3.2 Data Preparation**
     - **3.2.1 Handling Class Imbalance**
     - **3.2.2 Dataset Copying**
     - **3.2.3 Setting the Class Attribute**
     - **3.2.4 Data Randomization**
   - **3.3 Train/Test Split (70/30)**
     - **3.3.1 Resampling the Training Set**
   - **3.4 Model Training**
     - **3.4.1 Implementation Process and Challenges**
     - **3.4.1 Naive Bayes**
     - **3.4.2 Random Forest**
   - **3.5 Model Comparison**
4. **Model Evaluation**
   - **5.1 Performance Metrics**
   - **5.2 Analysis of Results**
5. **Association Rule Mining (Apriori)**
   - **6.1 Discretization**
   - **6.2 Apriori Settings**
   - **6.3 Interpretation of Association Rules**
6. **Conclusions**

# Beginning

## 1. Introduction

This work develops a full data mining toolkit written in Java and based on the Weka library. The framework aims to accomplish two complementary objectives: (1) develop and assess accurate classification/ prediction models for detecting heart disease from patient records, as well as (2) uncovering frequent patterns and association rules (sequence/pattern mining) that explain or support the model's predictions. The focus is on reproducibility, interpretability, and practical deployment (models, checkpoints, and preprocessing pipelines).

## 1.2 Project objectives

1. **Predictive modeling:** Construct appropriate supervised classifiers for predicting the presence of heart disease using available clinical, demographic, and lifestyle attributes.
2. **Pattern/sequence mining:** Search for common attribute-value combinations and associations between attributes that are highly associated with the outcome of heart disease to assist interpretation.
3. **Complete, end-to-end Java pipeline:** Create the entire pipeline in Java - including data reading, cleaning, feature transformation, class balancing, model training and tuning, evaluation, sequence/pattern mining, and model storing.
4. **Evaluation and reporting:** Generate comprehensive evaluation reports (confusion matrix, F1 - score, Precision, Recall (TP rate), FP rate, ROC) and a set of high-quality interpretable rules/patterns (support/confidence/lift) for the clinician or stakeholders to review.

## 1.3 Dataset used

The dataset will be working with for this assignment is the provided Heart Disease dataset (heart_disease.csv), which includes 10,000 cases with 21 attributes that consist of demographic information (age, gender), lifestyle habit factors (smoking, exercise), health status (blood pressure, cholesterol), and medical history (e.g., diabetes, family history). Key dependent dataset features that will impact the methodology:

**Mixed data:** numerical and categorical characteristics requiring type-dependent processing.

**NA values:** a few missing entries (few <~1% per column), which shall be imputed/managed before modeling.

**Unbalanced class**: ~80% No (healthy) and 20% Yes (disease) - this may require resampling or modification of learning algorithms to avoid biased classifiers.

**Size:** 10k rows - large enough to support CV and to produce stable pattern mining results.

## 1.4 Methodologies employed (high-level)

This framework applies a typical, reproducible data-mining process:

- Data ingestion and standardization
- Read CSV/ARFF into Weka Instances.

Set the class index and ensure correct attribute types

- **Preprocessing**
    - Missing-value treatment: Weka ReplaceMissingValues or imputations.
    - Remove duplicates
    - Define outliers
- **Feature selection & engineering**
    - Filter/Wrapper methods (InfoGain + Ranker, correlation filters, or wrapper subset selection) to reduce irrelevant/redundant features and improve model interpretability and performance.
    - Creation of derived features (e.g., age groups, BMI) when helpful for interpretation or mining.
- **Model training & selection**
    - Balancing classes: Use Resampling method (Only for train set)
    - Train multiple classifiers to compare performance:
        - **Random Forest** - strong general-purpose ensemble, variable importance estimates.
        - **Naive Bayes** - fast probabilistic baseline.
- **Evaluation**
    - Use stratified k-fold cross-validation (commonly k=10) and a separate holdout set if available.
    - Report accuracy, confusion matrix, F1 - score, Precision, Recall (TP rate), FP rate, ROC.
- **Pattern / sequence mining**
    - Transform each patient record into a transaction of discretized attribute-value items (e.g., Age>60, Cholesterol>240, Smoking=Yes).
    - Run Apriori to find frequent itemsets and generate association rules.
    - Filter rules by support, confidence, and lift; highlight clinically meaningful rules that correlate with the Yes class.
- **Model persistence & deployment**
    - Save trained classifiers and preprocessing filters with Weka's SerializationHelper for reuse in batch or real-time Java applications.

## 1.5 Brief explanation of core concepts

**Data mining** is everything about finding useful patterns and building models, integrates data preprocessing, pattern discovery, and predictive modelling to discover actionable knowledge.

**Supervised learning / Classification:** Learning a function from input features to discrete labels (in this case, whether heart disease is present). The model is trained from labeled examples of cases to predict unobserved instances.

**Pattern/sequence mining:** Association rule machine learning of unsupervised or semi-supervised frequent co-occurring items, generated in context as ordered patterns (rules) that assist interpretability and domain knowledge.

**Preprocessing & balancing:** In the real-world datasets, data cleaning, encoding, and balancing are necessary to ensure that the model does not learn any spurious patterns or be biased towards majority classes.

**Performance metrics:** confusion matrix, F1 - score, Precision, Recall (TP rate), FP rate, ROC are very important as class imbalance is involved - it helps us understand the 'success rate' of identifying patients with disease (minority class).

## 1.6 Significance in the context of this assignment

**Clinical screening:** A Precise prediction model may help clinicians to identify patients at high risk of CVD. Significance in the context of this assignment is at an early stage for a timely intervention.

**Interpretability:** Association rules and decision trees offer interpretable results, which are essential when model explanation is important in clinical settings.

**Operationalization:** Using the pipeline in Java and Weka allows us to easily integrate, plug in existing health information systems coded in Java.

**Educational content:** The tasking is indicative of a complete data-mining process, from data ingestion to model persistence and the balancing of trade-offs between different preprocessing approaches, algorithms and evaluation metrics.

## 1.7 Expected deliverables

Clean, well-documented Java code (uses Weka) doing ingestion, preprocessing, classification, and pattern mining.

Saved the trained model files and preprocessing pipeline.

Evaluation report with the metric tables, confusion matrices, and selected association rules with support/confidence/lift.

A written document that is also provided, which summarises methods, results, interpretation, and recommendations.

**2. Data Pre-Processing**

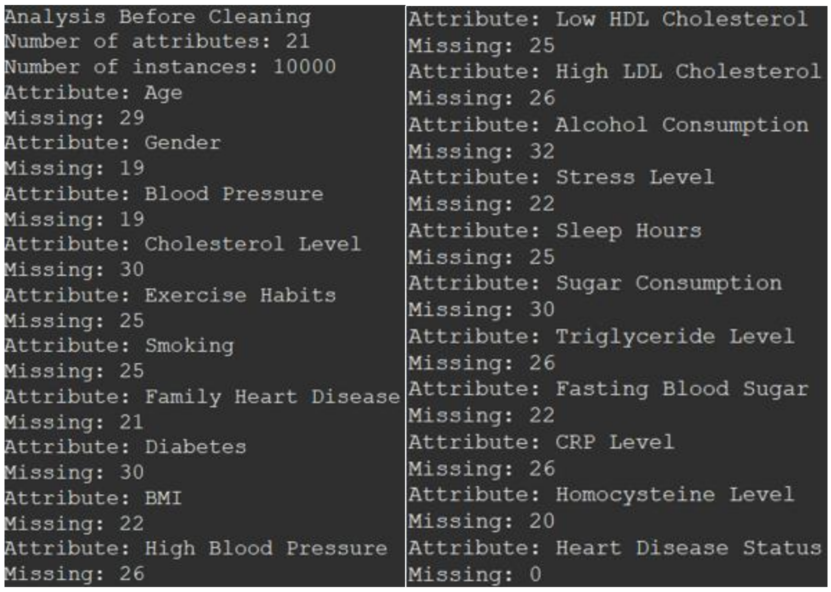
This section illustrates the steps involved in Data Cleaning, Preprocessing and Preparation of dataset for analysis and model building. Good data preprocessing leads to high quality data, less noise and better performance of subsequent machine learning algorithms.

**2.1 Raw Data Overview**

The raw data consist of: 10000 instances and 21 feature attributes, including demographic information, lifestyle habits, clinical measurements, and the target variable Heart Disease Status. Attributes are a mix of numeric and nominal types.

## Attributes in the Raw Dataset

Below is the complete list of features prior to preprocessing:



```
Analysis Before Cleaning
Number of attributes: 21
Number of instances: 10000
Attribute: Age
Missing: 29
Attribute: Gender
Missing: 19
Attribute: Blood Pressure
Missing: 19
Attribute: Cholesterol Level
Missing: 30
Attribute: Exercise Habits
Missing: 25
Attribute: Smoking
Missing: 25
Attribute: Family Heart Disease
Missing: 21
Attribute: Diabetes
Missing: 30
Attribute: BMI
Missing: 22
Attribute: High Blood Pressure
Missing: 26
```

```
Attribute: Low HDL Cholesterol
Missing: 25
Attribute: High LDL Cholesterol
Missing: 26
Attribute: Alcohol Consumption
Missing: 32
Attribute: Stress Level
Missing: 22
Attribute: Sleep Hours
Missing: 25
Attribute: Sugar Consumption
Missing: 30
Attribute: Triglyceride Level
Missing: 26
Attribute: Fasting Blood Sugar
Missing: 22
Attribute: CRP Level
Missing: 26
Attribute: Homocysteine Level
Missing: 20
Attribute: Heart Disease Status
Missing: 0
```

## Initial Data Observations

- The dataset includes both medical lab values (Cholesterol Level, CRP Level) and lifestyle information (Smoking, Stress Level).
- There are missing values present in multiple attributes, requiring careful handling.

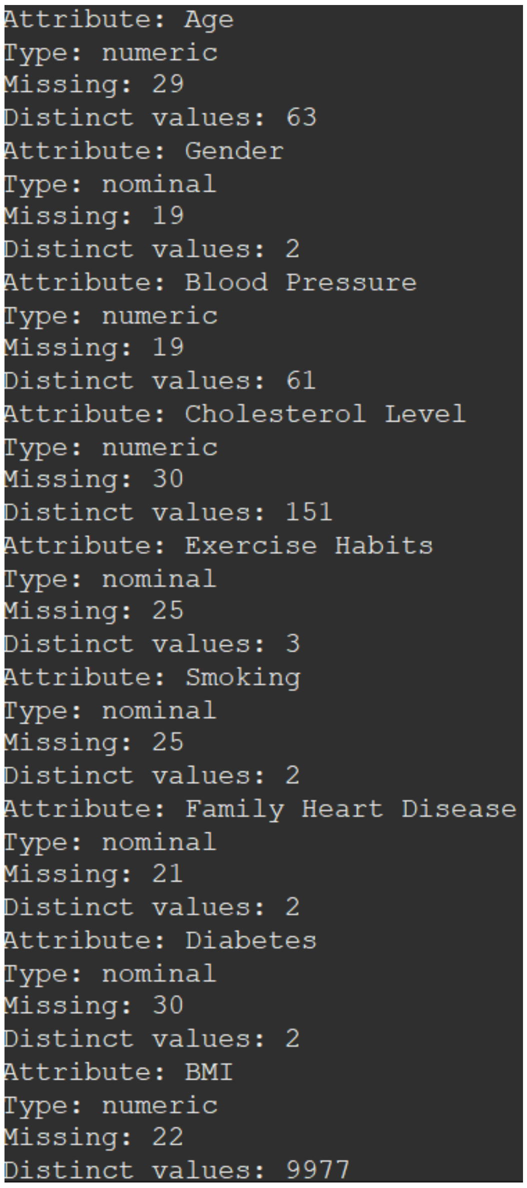
- No outliers were detected in any numeric attribute based on the IQR method.
- Dataset size is large and suitable for machine learning without augmentation.

## 2.2 Data Cleaning Process

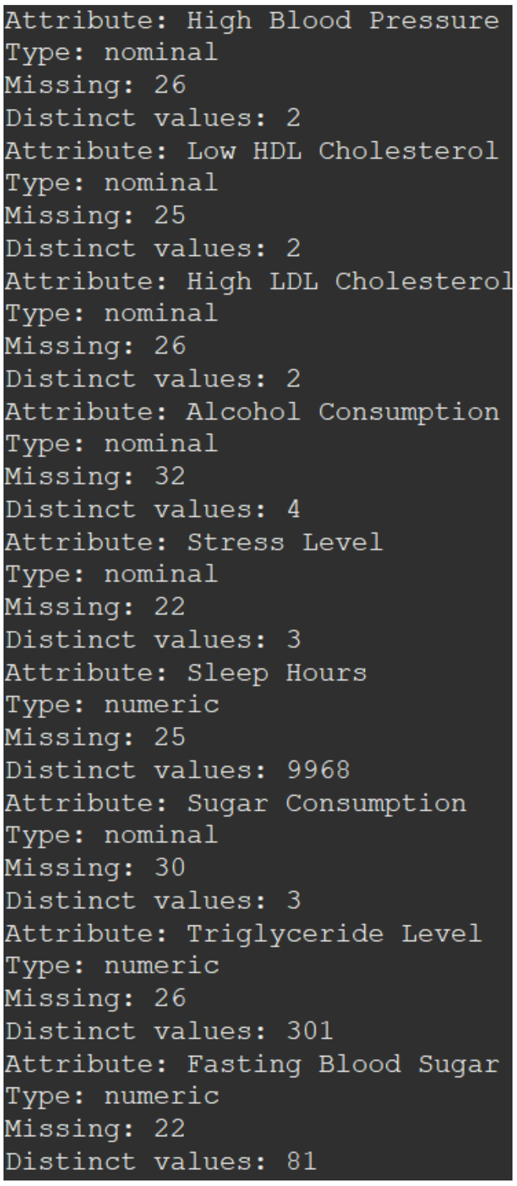
The cleaning pipeline systematically addressed missing values, duplicates, and outlier detection.

### 2.2.1 Handling Missing Values

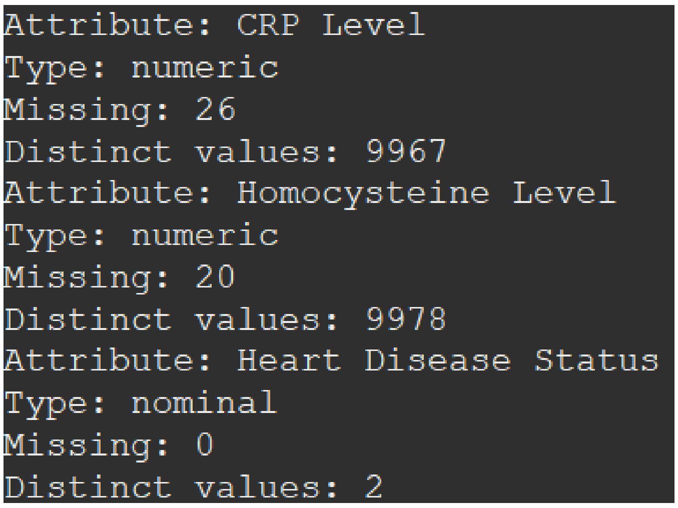
Missing value summary (Before Cleaning):



```
Attribute: Age
Type: numeric
Missing: 29
Distinct values: 63
Attribute: Gender
Type: nominal
Missing: 19
Distinct values: 2
Attribute: Blood Pressure
Type: numeric
Missing: 19
Distinct values: 61
Attribute: Cholesterol Level
Type: numeric
Missing: 30
Distinct values: 151
Attribute: Exercise Habits
Type: nominal
Missing: 25
Distinct values: 3
Attribute: Smoking
Type: nominal
Missing: 25
Distinct values: 2
Attribute: Family Heart Disease
Type: nominal
Missing: 21
Distinct values: 2
Attribute: Diabetes
Type: nominal
Missing: 30
Distinct values: 2
Attribute: BMI
Type: numeric
Missing: 22
Distinct values: 9977
```



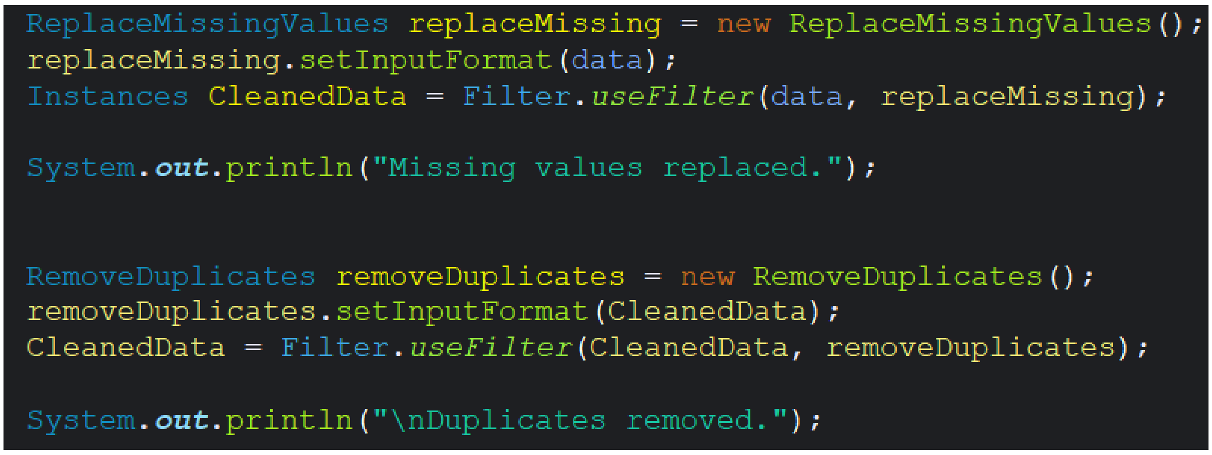
```
Attribute: High Blood Pressure
Type: nominal
Missing: 26
Distinct values: 2
Attribute: Low HDL Cholesterol
Type: nominal
Missing: 25
Distinct values: 2
Attribute: High LDL Cholesterol
Type: nominal
Missing: 26
Distinct values: 2
Attribute: Alcohol Consumption
Type: nominal
Missing: 32
Distinct values: 4
Attribute: Stress Level
Type: nominal
Missing: 22
Distinct values: 3
Attribute: Sleep Hours
Type: numeric
Missing: 25
Distinct values: 9968
Attribute: Sugar Consumption
Type: nominal
Missing: 30
Distinct values: 3
Attribute: Triglyceride Level
Type: numeric
Missing: 26
Distinct values: 301
Attribute: Fasting Blood Sugar
Type: numeric
Missing: 22
Distinct values: 81
```



```
Attribute: CRP Level
Type: numeric
Missing: 26
Distinct values: 9967
Attribute: Homocysteine Level
Type: numeric
Missing: 20
Distinct values: 9978
Attribute: Heart Disease Status
Type: nominal
Missing: 0
Distinct values: 2
```

**Cleaning Strategy Used:**

- For attributes with small missing proportions (< 1%), missing values were replaced using Weka's ReplaceMissingValues filter.
- Since every attribute had missing values well below 5%, none of the instances were removed due to missing data.
- After replacement, the dataset contained 0 missing values.



```java
ReplaceMissingValues replaceMissing = new ReplaceMissingValues();
replaceMissing.setInputFormat(data);
Instances CleanedData = Filter.useFilter(data, replaceMissing);

System.out.println("Missing values replaced.");


RemoveDuplicates removeDuplicates = new RemoveDuplicates();
removeDuplicates.setInputFormat(CleanedData);
CleanedData = Filter.useFilter(CleanedData, removeDuplicates);

System.out.println("\nDuplicates removed.");
```

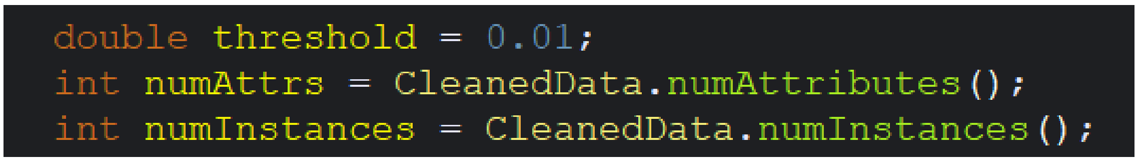
### 2.2.2 Removing Duplicates

Duplicate instances were detected and removed using full-instance comparison.

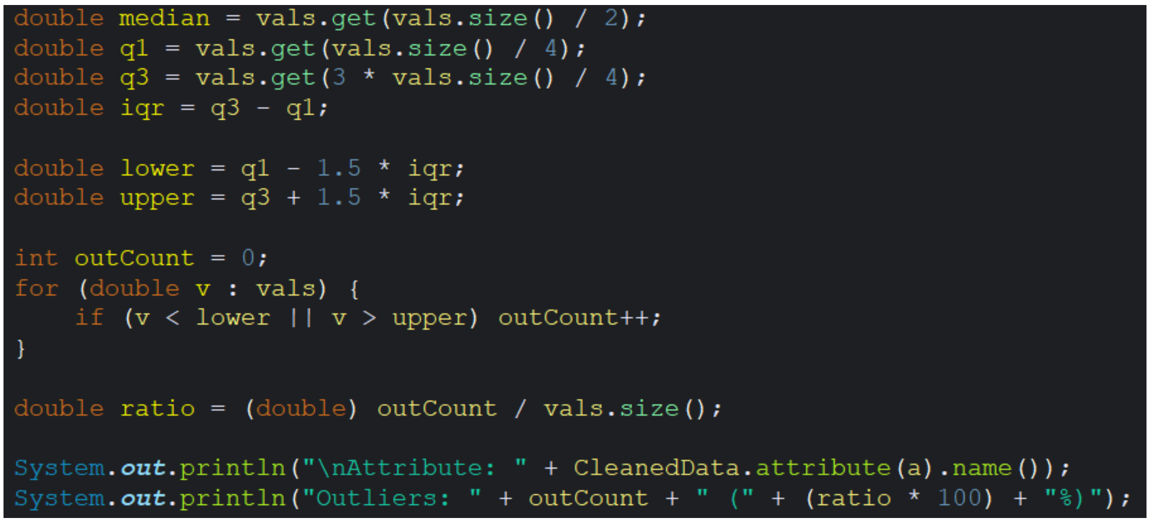
- All fully identical rows were eliminated.
- Result ensured that no sample was overweighted in training.

### 2.2.3 Outlier Detection & Treatment

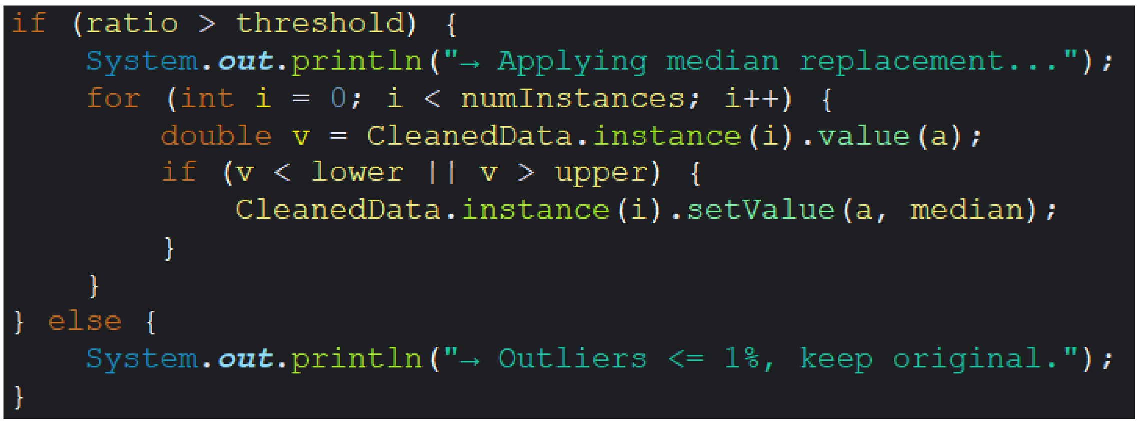
Outlier detection was performed on numeric attributes using the Interquartile Range (IQR) rule.



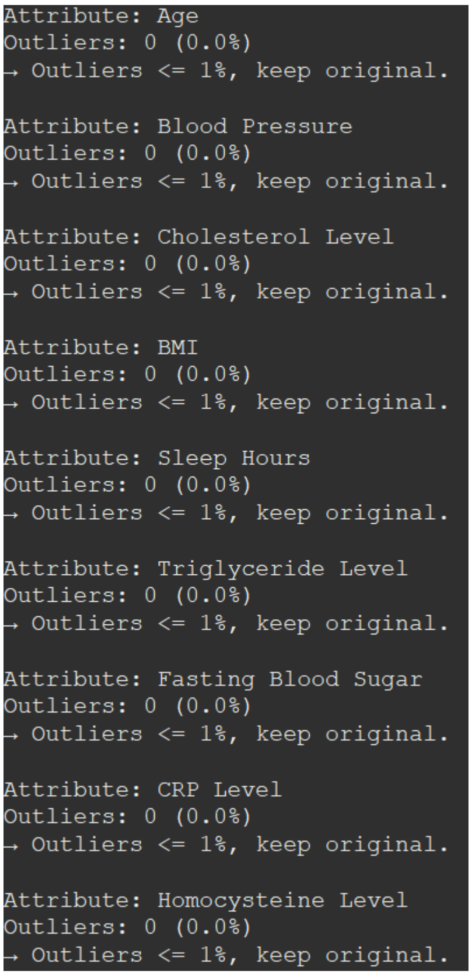
```java
double threshold = 0.01;
int numAttrs = CleanedData.numAttributes();
int numInstances = CleanedData.numInstances();
```



```java
double median = vals.get(vals.size() / 2);
double q1 = vals.get(vals.size() / 4);
double q3 = vals.get(3 * vals.size() / 4);
double iqr = q3 - q1;

double lower = q1 - 1.5 * iqr;
double upper = q3 + 1.5 * iqr;

int outCount = 0;
for (double v : vals) {
    if (v < lower || v > upper) outCount++;
}

double ratio = (double) outCount / vals.size();

System.out.println("\nAttribute: " + CleanedData.attribute(a).name());
System.out.println("Outliers: " + outCount + " (" + (ratio * 100) + "%)");
```



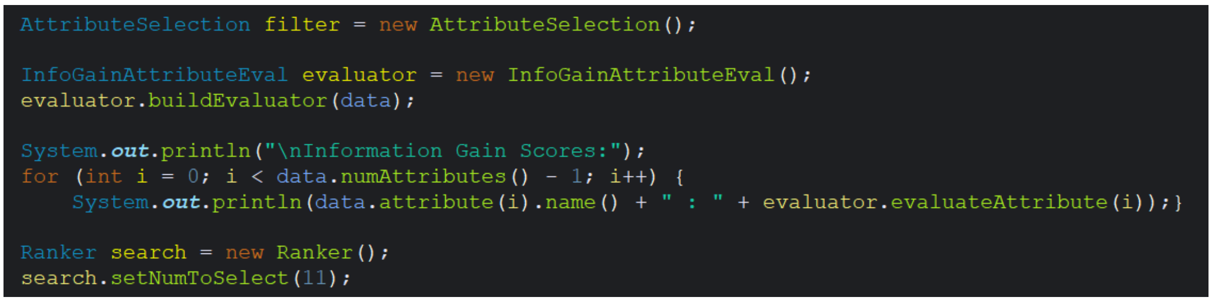
```java
if (ratio > threshold) {
    System.out.println("→ Applying median replacement...");
    for (int i = 0; i < numInstances; i++) {
        double v = CleanedData.instance(i).value(a);
        if (v < lower || v > upper) {
            CleanedData.instance(i).setValue(a, median);
        }
    }
} else {
    System.out.println("→ Outliers <= 1%, keep original.");
}
```



```
Attribute: Age
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.

Attribute: Blood Pressure
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.

Attribute: Cholesterol Level
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.

Attribute: BMI
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.

Attribute: Sleep Hours
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.

Attribute: Triglyceride Level
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.

Attribute: Fasting Blood Sugar
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.

Attribute: CRP Level
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.

Attribute: Homocysteine Level
Outliers: 0 (0.0%)
→ Outliers <= 1%, keep original.
```

## 2.3 Data Transformation

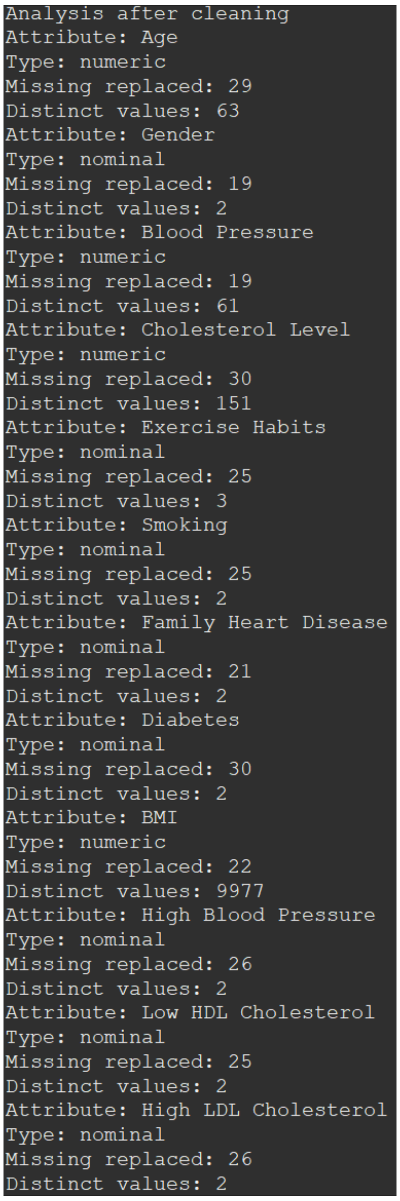
Information Gain combined with the Ranker method was applied to evaluate the importance of each attribute with respect to the class label. Based on the ranking results, the top 11 most relevant attributes were retained, while less informative features were removed. This transformation helps reduce dimensionality, improve model efficiency, and enhance classification performance by focusing on the most discriminative attributes.



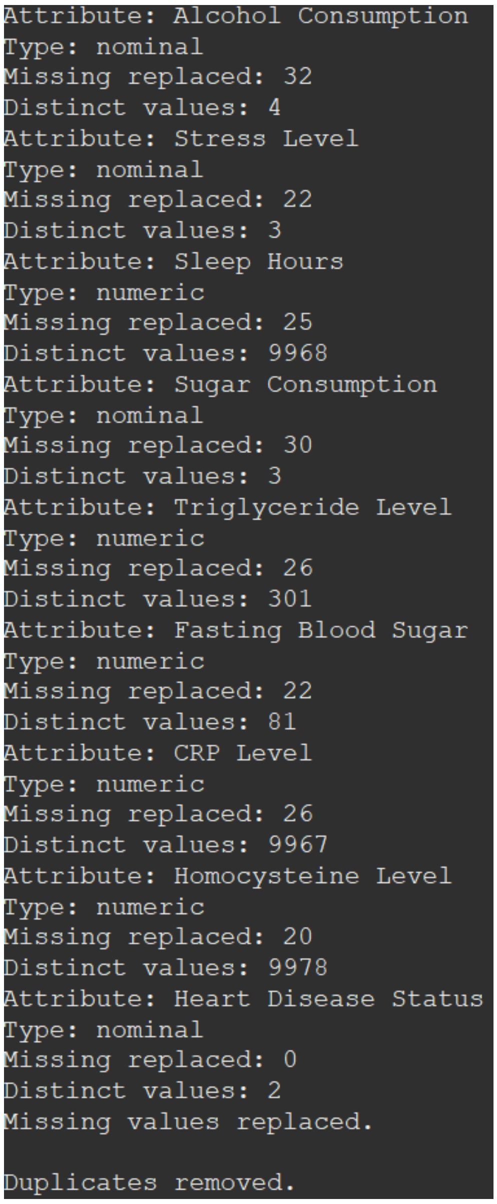
```java
AttributeSelection filter = new AttributeSelection();

InfoGainAttributeEval evaluator = new InfoGainAttributeEval();
evaluator.buildEvaluator(data);

System.out.println("\nInformation Gain Scores:");
for (int i = 0; i < data.numAttributes() - 1; i++) {
    System.out.println(data.attribute(i).name() + " : " + evaluator.evaluateAttribute(i));}

Ranker search = new Ranker();
search.setNumToSelect(11);
```

## 2.4 Final Cleaned Dataset (Output)

The dataset is now fully clean, consistent, and ready for modeling.



```
Analysis after cleaning
Attribute: Age
Type: numeric
Missing replaced: 29
Distinct values: 63
Attribute: Gender
Type: nominal
Missing replaced: 19
Distinct values: 2
Attribute: Blood Pressure
Type: numeric
Missing replaced: 19
Distinct values: 61
Attribute: Cholesterol Level
Type: numeric
Missing replaced: 30
Distinct values: 151
Attribute: Exercise Habits
Type: nominal
Missing replaced: 25
Distinct values: 3
Attribute: Smoking
Type: nominal
Missing replaced: 25
Distinct values: 2
Attribute: Family Heart Disease
Type: nominal
Missing replaced: 21
Distinct values: 2
Attribute: Diabetes
Type: nominal
Missing replaced: 30
Distinct values: 2
Attribute: BMI
Type: numeric
Missing replaced: 22
Distinct values: 9977
Attribute: High Blood Pressure
Type: nominal
Missing replaced: 26
Distinct values: 2
Attribute: Low HDL Cholesterol
Type: nominal
Missing replaced: 25
Distinct values: 2
Attribute: High LDL Cholesterol
Type: nominal
Missing replaced: 26
Distinct values: 2
```



```
Attribute: Alcohol Consumption
Type: nominal
Missing replaced: 32
Distinct values: 4
Attribute: Stress Level
Type: nominal
Missing replaced: 22
Distinct values: 3
Attribute: Sleep Hours
Type: numeric
Missing replaced: 25
Distinct values: 9968
Attribute: Sugar Consumption
Type: nominal
Missing replaced: 30
Distinct values: 3
Attribute: Triglyceride Level
Type: numeric
Missing replaced: 26
Distinct values: 301
Attribute: Fasting Blood Sugar
Type: numeric
Missing replaced: 22
Distinct values: 81
Attribute: CRP Level
Type: numeric
Missing replaced: 26
Distinct values: 9967
Attribute: Homocysteine Level
Type: numeric
Missing replaced: 20
Distinct values: 9978
Attribute: Heart Disease Status
Type: nominal
Missing replaced: 0
Distinct values: 2
Missing values replaced.

Duplicates removed.
```

## 3. Classification/Prediction Algorithm

### 3.1 Introduction

Here, we explain how the dataset is prepared, how Train/Test pairs are generated, and how models are trained or tested.

A 70/30 split is used, where:

- 70% of the data is for training.
- 30% is used for testing

In this way, it is an unbiased estimate of how the model will perform on unseen data.

This logic resides in a class Classification

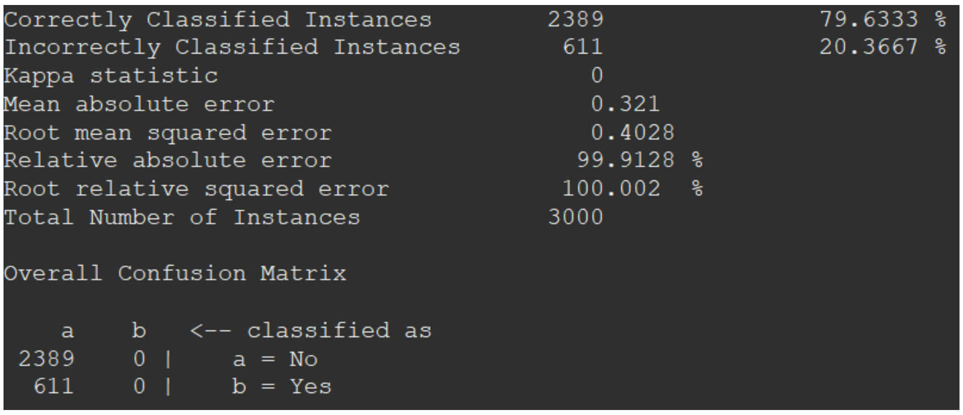
## 3.2 Data Preparation

When a Classification object is initialized, several data-processing steps are executed.

### 3.2.1 Dealing with class imbalance severely before Model Training (Most Important)

Before training any classification model, our group faced a big problem with the dataset: a heavily imbalanced category. The target variable Heart Disease has about 80% records of "No" and only 20% "Yes". Most of these baseline classifiers, like Naive Bayes, and J48 predicted only the majority “No” class, which yielded an unrealistically good accuracy but a near-zero recall for the “Yes” (disease-positive) class.

This flawed model was effectively unusable, as it is infinitely worse to miss positive heart-disease patients than it is to give false positives and have them run additional tests.



```
Correctly Classified Instances        2389               79.6333 %
Incorrectly Classified Instances       611               20.3667 %
Kappa statistic                          0
Mean absolute error                      0.321
Root mean squared error                  0.4028
Relative absolute error                 99.9128 %
Root relative squared error            100.002  %
Total Number of Instances             3000

Overall Confusion Matrix

    a    b   <-- classified as
 2389    0 |    a = No
  611    0 |    b = Yes
```

### Initial Attempt: SMOTE in Java (Weka API) and the Problems Encountered

Our first method involved using Weka’s SMOTE filter in Java and applying the data extension using synthetic examples. However, we encountered some problems during experiments:

- In some parameter configurations, SMOTE work well with numeric attributes, and in others, categorical attributes led to conflicts or non-intuitive discretization.
- To apply SMOTE in a cross-validation scenario, inside each fold, it needs to be included, which cannot be directly benefited from Java, and requires careful setup of the pipeline. Initial attempts erroneously used SMOTE before splitting, which had introduced data leakage.

Although SMOTE improved class distribution, the practical issues made it difficult to maintain a stable and clean workflow inside our custom Java framework.

**Final Solution: Resampling (with Bias Toward the Minority Class)**

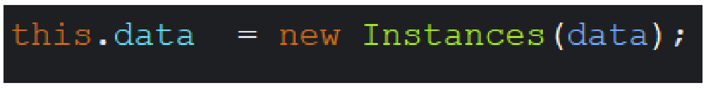
After trying several approaches, the Weka Resample filter with bias towards the minority class was our most stable and efficient solution. The key advantages were:

- It also successfully handles mixed numeric and nominal attribute features without any additional preprocessing.
- Random sampling introduces controlled variability while still keeping a realistic data distribution.
- The resulting class distribution can be very fine-tuned (set to 50-50 or 60-40), allowing the classifier to be equally exposed during training.

When using the resample filter within the Java pipeline (after splitted, only applied on training), models showed a clear improvement:

- Naive Bayes started to accurately forecast the "Yes" class.
- Random Forest achieved much higher recall without sacrificing overall accuracy.
- The confusion matrix became more balanced and informative.

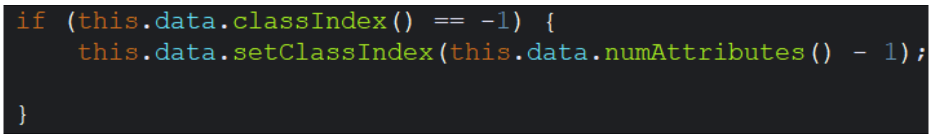
### 3.2.2 Copying the original dataset



```java
this.data  = new Instances(data);
```

This ensures the original dataset is not modified during training.

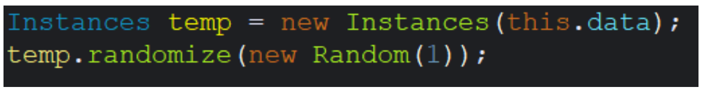
### 3.2.3 Setting the class attribute



```java
if (this.data.classIndex() == -1) {
    this.data.setClassIndex(this.data.numAttributes() - 1);

}
```

If the class attribute is not defined, the last column is automatically assigned as the label.

### 3.2.4 Randomizing the data



```java
Instances temp = new Instances(this.data);
temp.randomize(new Random(1));
```

Using temp ensures that:

- The original dataset remains unchanged
- Randomization affects only the Train/Test split
- Better control and safety
- Reproducibility is preserved

### 3.3 Train/Test Split (70/30)

```
int trainSize = (int) Math.round(temp.numInstances() * 0.7);
int testSize  = temp.numInstances() - trainSize;

this.train = new Instances(temp, 0, trainSize);
this.test  = new Instances(temp, trainSize, testSize);
```

```
this.train.setClassIndex(this.train.numAttributes() - 1);
this.test.setClassIndex(this.test.numAttributes() - 1);
```

- trainSize: 70% of the shuffled dataset
- testSize: the remaining 30%

#### 3.3.1 Apply Resample for Training set

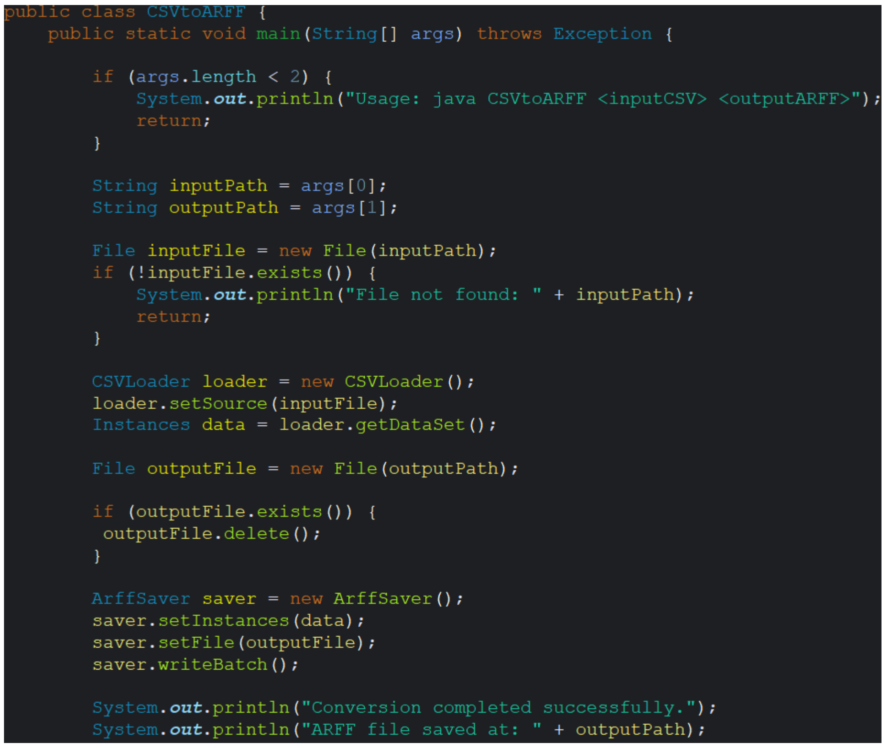
```
Resample resample = new Resample();
resample.setNoReplacement(false);
resample.setBiasToUniformClass(1.0);
resample.setSampleSizePercent(100);
resample.setInputFormat(this.train);
this.train = Filter.useFilter(this.train, resample);
```

### 3.4 Model Training

#### 3.4.1 Implementation Process

The dataset was first converted from CSV to ARFF format using Weka to ensure full compatibility with the data mining pipeline. The CSV file was loaded into the program using the CSVLoader class, which converts the raw data into a Weka Instances object. This object provides a structured representation of the dataset, including attributes and instances. The ArffSaver class was then used to export the data into ARFF format, preserving attribute definitions.

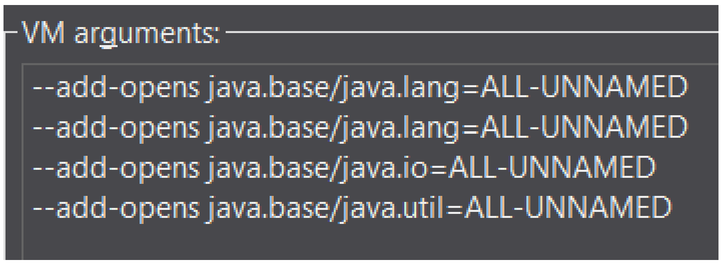
After conversion, the ARFF dataset was loaded and processed entirely using Weka’s API within a Java program. The Weka library was added as an external dependency to the project, enabling the use of filters, classifiers, and evaluation tools. The Instances structure was consistently used for preprocessing, feature selection, classification, cross-validation, and association rule mining, ensuring seamless integration and consistent data handling throughout the implementation.



```java
public class CSVtoARFF {
    public static void main(String[] args) throws Exception {

        if (args.length < 2) {
            System.out.println("Usage: java CSVtoARFF <inputCSV> <outputARFF>");
            return;
        }

        String inputPath = args[0];
        String outputPath = args[1];

        File inputFile = new File(inputPath);
        if (!inputFile.exists()) {
            System.out.println("File not found: " + inputPath);
            return;
        }

        CSVLoader loader = new CSVLoader();
        loader.setSource(inputFile);
        Instances data = loader.getDataSet();

        File outputFile = new File(outputPath);

        if (outputFile.exists()) {
         outputFile.delete();
        }

        ArffSaver saver = new ArffSaver();
        saver.setInstances(data);
        saver.setFile(outputFile);
        saver.writeBatch();

        System.out.println("Conversion completed successfully.");
        System.out.println("ARFF file saved at: " + outputPath);
```

Challenges included initial configuration of Weka external libraries, during implementation, an issue was encountered when running Weka-based programs on newer Java versions (Java 9+). Due to Java's module system, several internal packages such as java.lang, java.io, and java.util are not accessible by default. This caused runtime exceptions when executing Weka components via reflection.

To resolve this issue, all main classes were required to be executed with additional JVM arguments to explicitly open the necessary modules:



VM arguments:

```
--add-opens java.base/java.lang=ALL-UNNAMED
--add-opens java.base/java.lang=ALL-UNNAMED
--add-opens java.base/java.io=ALL-UNNAMED
--add-opens java.base/java.util=ALL-UNNAMED
```

**The class provides two algorithms:**

### 3.4.2 Naive Bayes

Naive Bayes was chosen due to its simplicity, computational efficiency, and strong performance on high-dimensional data. It serves as a baseline model that allows us to assess the effectiveness of more complex classifiers. However, Naive Bayes assumes conditional independence among features, which may limit its performance when attributes are correlated.

```
NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(this.train);
```

- Based on Bayes’ Theorem
- Assumes independence between features
- Efficient for high-dimensional and noisy data

### After training, the model is saved:

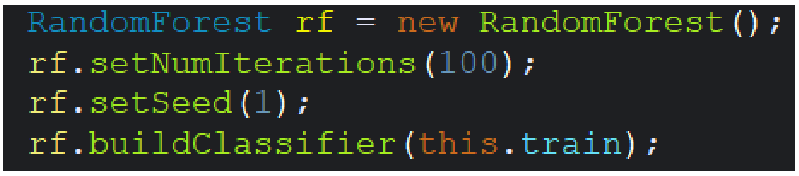
```
SerializationHelper.write(outputFolder + "/NAIVEBAYES.model", nb);
System.out.println("NaiveBayes model saved to: " + outputFolder + "/NAIVEBAYES.model");
return nb;
```

### Output for Naive Bayes:

```
Results

Correctly Classified Instances        1580               52.6667 %
Incorrectly Classified Instances      1420               47.3333 %
Kappa statistic                          0.0228
Mean absolute error                      0.499
Root mean squared error                  0.5006
Relative absolute error                 99.8091 %
Root relative squared error            100.12   %
Total Number of Instances             3000

Overall Confusion Matrix

    a    b   <-- classified as
 1279 1121 |    a = No
  299  301 |    b = Yes
```

```
Detailed Metrics Per Class
Class: No
-Precision: 0.8105196451204055
-Recall (TP Rate): 0.5329166666666667
-F1 Score: 0.6430367018602313
-FP Rate: 0.49833333333333335
-ROC Area: 0.5091923611111111

Class: Yes
-Precision: 0.21167369901547117
-Recall (TP Rate): 0.5016666666666667
-F1 Score: 0.2977250247279921
-FP Rate: 0.46708333333333335
-ROC Area: 0.5091923611111111

Total Execution Time: 380ms
```

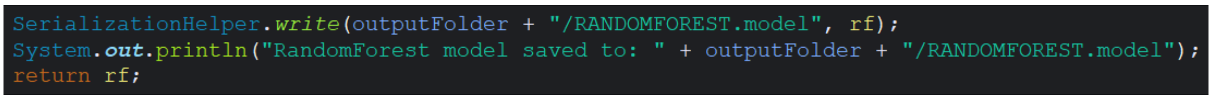
### 3.4.3 Random Forest



```java
RandomForest rf = new RandomForest();
rf.setNumIterations(100);
rf.setSeed(1);
rf.buildClassifier(this.train);
```

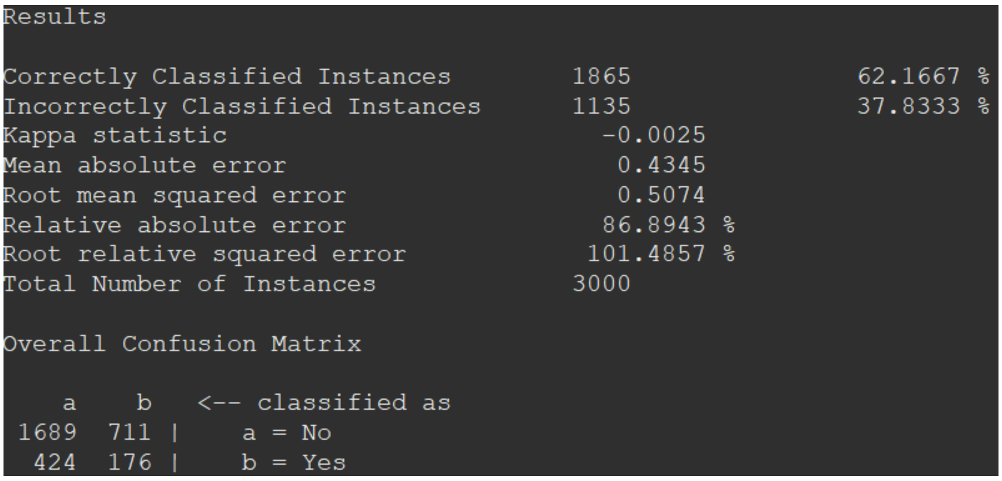
- Uses 100 decision trees
- Combines bagging and feature randomness
- Handles nonlinear relationships and noisy datasets well

### The model is also saved for reuse:

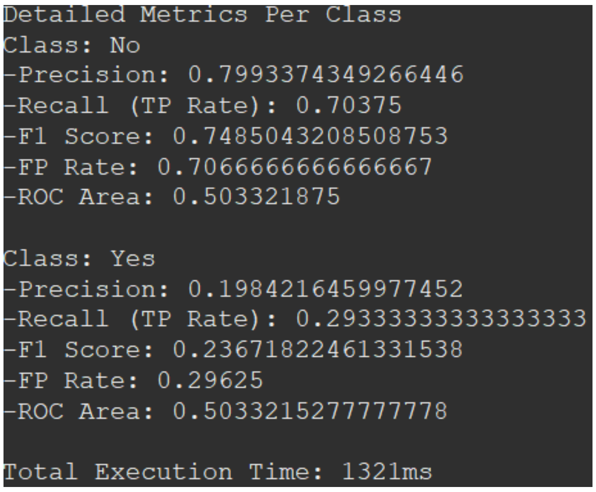


```java
SerializationHelper.write(outputFolder + "/RANDOMFOREST.model", rf);
System.out.println("RandomForest model saved to: " + outputFolder + "/RANDOMFOREST.model");
return rf;
```

### Output for Random Forest:



```
Results

Correctly Classified Instances        1865               62.1667 %
Incorrectly Classified Instances      1135               37.8333 %
Kappa statistic                         -0.0025
Mean absolute error                      0.4345
Root mean squared error                  0.5074
Relative absolute error                 86.8943 %
Root relative squared error            101.4857 %
Total Number of Instances             3000

Overall Confusion Matrix

    a    b   <-- classified as
 1689  711 |    a = No
  424  176 |    b = Yes
```



```
Detailed Metrics Per Class
Class: No
-Precision: 0.7993374349266446
-Recall (TP Rate): 0.70375
-F1 Score: 0.7485043208508753
-FP Rate: 0.7066666666666667
-ROC Area: 0.503321875

Class: Yes
-Precision: 0.1984216459977452
-Recall (TP Rate): 0.29333333333333333
-F1 Score: 0.23671822461331538
-FP Rate: 0.29625
-ROC Area: 0.5033215277777778

Total Execution Time: 1321ms
```

## 3.5 Comparison

Random Forest improves overall classification accuracy and majority-class performance compared to Naive Bayes. However, Naive Bayes remains more effective in detecting minority-class instances, highlighting the trade-off between overall accuracy and imbalance-sensitive metrics.

| Metric | Naive Bayes | Random Forest |
|---|---|---|
| **Accuracy (%)** | 52.67 | **62.17** |
| **F1-score (No)** | 0.643 | **0.749** |
| **Recall (No)** | 0.533 | **0.704** |
| **F1-score (Yes)** | **0.298** | 0.237 |
| **Recall (Yes)** | **0.502** | 0.293 |
| **Execution Time (ms)** | **373** | 1346 |

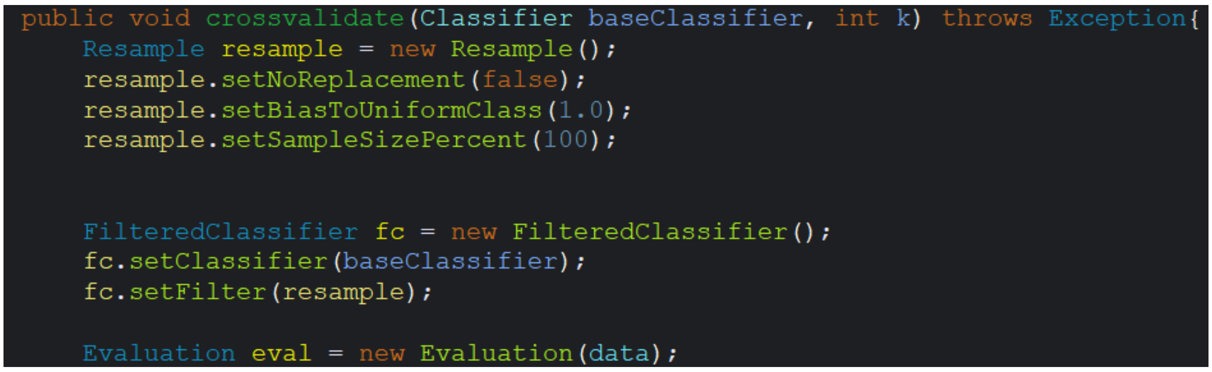
Although Random Forest was implemented as an ensemble-based model, extensive hyperparameter tuning was not performed due to time and computational constraints. Default parameters were used as a baseline to evaluate the model's general performance. Future work could further improve performance by tuning key parameters such as the number of trees, maximum tree depth, and class weighting to better handle class imbalance.

## 4. Model Evaluation

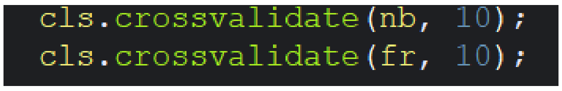
### Objective

The objective of this evaluation is to assess the final classification models using 10-fold cross-validation in order to obtain a robust and unbiased estimate of model performance.

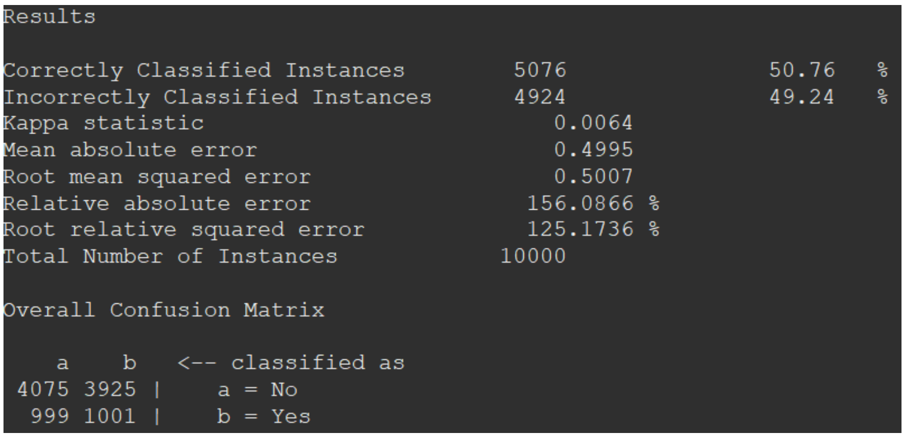
### 4.1 Performance Metrics



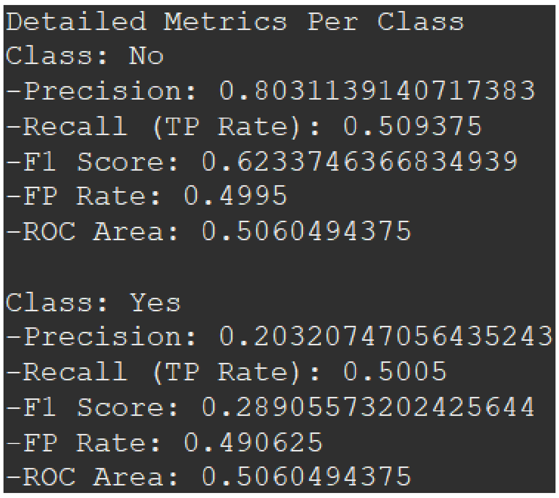
```java
public void crossvalidate(Classifier baseClassifier, int k) throws Exception{
    Resample resample = new Resample();
    resample.setNoReplacement(false);
    resample.setBiasToUniformClass(1.0);
    resample.setSampleSizePercent(100);


    FilteredClassifier fc = new FilteredClassifier();
    fc.setClassifier(baseClassifier);
    fc.setFilter(resample);

    Evaluation eval = new Evaluation(data);
```



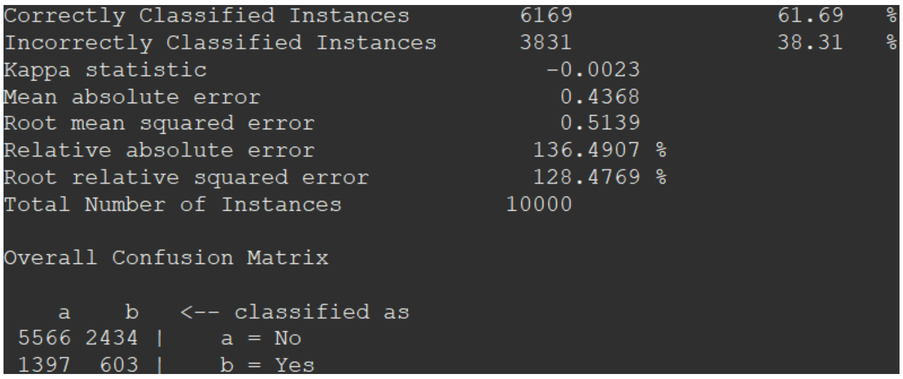
```java
  cls.crossvalidate(nb, 10);
  cls.crossvalidate(fr, 10);
```



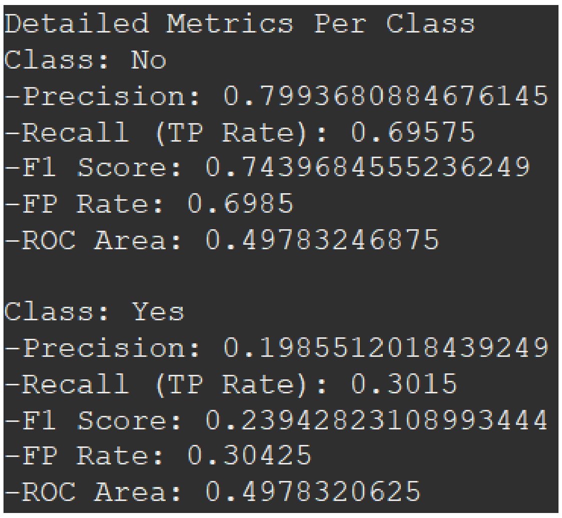
```
Results

Correctly Classified Instances        5076               50.76   %
Incorrectly Classified Instances      4924               49.24   %
Kappa statistic                          0.0064
Mean absolute error                      0.4995
Root mean squared error                  0.5007
Relative absolute error                156.0866 %
Root relative squared error            125.1736 %
Total Number of Instances            10000

Overall Confusion Matrix

    a    b   <-- classified as
 4075 3925 |    a = No
  999 1001 |    b = Yes
```



```
Detailed Metrics Per Class
Class: No
-Precision: 0.8031139140717383
-Recall (TP Rate): 0.509375
-F1 Score: 0.6233746366834939
-FP Rate: 0.4995
-ROC Area: 0.5060494375

Class: Yes
-Precision: 0.20320747056435243
-Recall (TP Rate): 0.5005
-F1 Score: 0.28905573202425644
-FP Rate: 0.490625
-ROC Area: 0.5060494375
```



```
Correctly Classified Instances        6169               61.69   %
Incorrectly Classified Instances      3831               38.31   %
Kappa statistic                         -0.0023
Mean absolute error                      0.4368
Root mean squared error                  0.5139
Relative absolute error                136.4907 %
Root relative squared error            128.4769 %
Total Number of Instances            10000

Overall Confusion Matrix

    a    b   <-- classified as
 5566 2434 |    a = No
 1397  603 |    b = Yes
```



```
Detailed Metrics Per Class
Class: No
-Precision: 0.7993680884676145
-Recall (TP Rate): 0.69575
-F1 Score: 0.7439684555236249
-FP Rate: 0.6985
-ROC Area: 0.49783246875

Class: Yes
-Precision: 0.1985512018439249
-Recall (TP Rate): 0.3015
-F1 Score: 0.23942823108993444
-FP Rate: 0.30425
-ROC Area: 0.4978320625
```

| **Model** | **Accuracy (%)** | **F1-score (Yes)** | **Recall (Yes)** | **F1-score (No)** | **Execution Time (ms)** |
|---|---|---|---|---|---|
| Naive Bayes | 50.76 | **0.289** | **0.501** | 0.623 | 1321 |
| Random Forest | **61.69** | 0.239 | 0.302 | **0.744** | 10405 |

## 4.2 Analysis of Results

The evaluation results show that Random Forest improves overall classification performance compared to Naive Bayes, achieving higher accuracy due to better prediction of the majority class. However, this improvement comes at the cost of reduced minority-class detection, where Naive Bayes demonstrates higher recall and F1-score. This highlights a trade-off between overall accuracy and imbalance-sensitive performance, indicating that Random Forest is more suitable for maximizing overall correctness, while Naive Bayes is preferable when identifying minority-class instances is the primary objective.

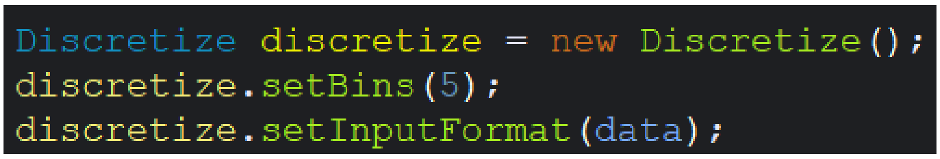
## 5. Apriori

## 5.1 Discretization

For association rule mining:

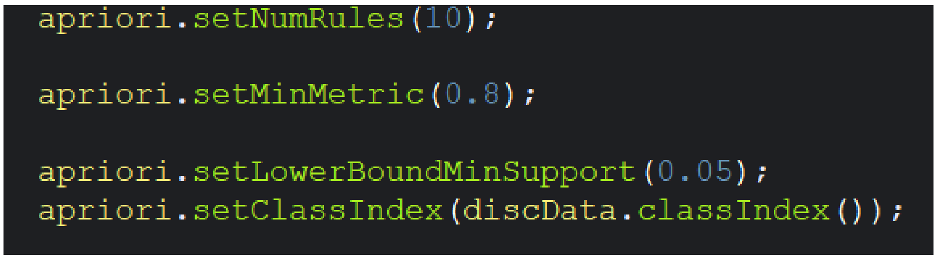
- All numeric attributes were discretized into 5 bins using the Weka Discretize filter.
- This converts continuous values into interpretable levels (Low/Medium/High).

This step is essential, as Apriori only works effectively with nominal categories.



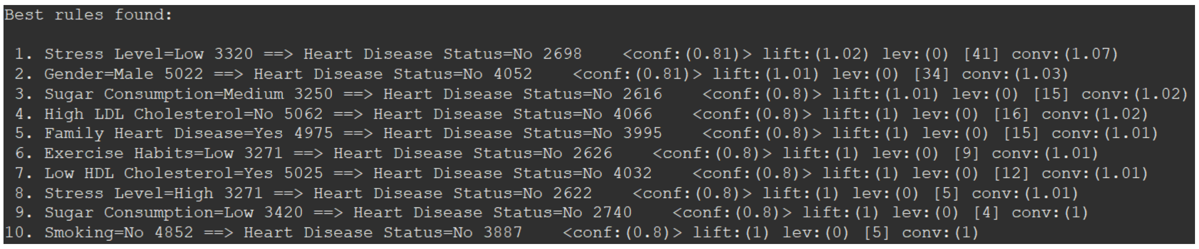
```
Discretize discretize = new Discretize();
discretize.setBins(5);
discretize.setInputFormat(data);
```

## 5.2 Apriori settings



```
 apriori.setNumRules(10);

 apriori.setMinMetric(0.8);

 apriori.setLowerBoundMinSupport(0.05);
 apriori.setClassIndex(discData.classIndex());
```

- Confidence ≥ 0.05 → Only keep patterns that appear in at least 5% of the data
- Support ≥ 0.8 → Only keep rules that are at least 80% correct
- Max Rules = 10 → Show up to 10 rules



```
Best rules found:

 1. Stress Level=Low 3320 ==> Heart Disease Status=No 2698    <conf:(0.81)> lift:(1.02) lev:(0) [41] conv:(1.07)
 2. Gender=Male 5022 ==> Heart Disease Status=No 4052    <conf:(0.81)> lift:(1.01) lev:(0) [34] conv:(1.03)
 3. Sugar Consumption=Medium 3250 ==> Heart Disease Status=No 2616    <conf:(0.8)> lift:(1.01) lev:(0) [15] conv:(1.02)
 4. High LDL Cholesterol=No 5062 ==> Heart Disease Status=No 4066    <conf:(0.8)> lift:(1) lev:(0) [16] conv:(1.02)
 5. Family Heart Disease=Yes 4975 ==> Heart Disease Status=No 3995    <conf:(0.8)> lift:(1) lev:(0) [15] conv:(1.01)
 6. Exercise Habits=Low 3271 ==> Heart Disease Status=No 2626    <conf:(0.8)> lift:(1) lev:(0) [9] conv:(1.01)
 7. Low HDL Cholesterol=Yes 5025 ==> Heart Disease Status=No 4032    <conf:(0.8)> lift:(1) lev:(0) [12] conv:(1.01)
 8. Stress Level=High 3271 ==> Heart Disease Status=No 2622    <conf:(0.8)> lift:(1) lev:(0) [5] conv:(1.01)
 9. Sugar Consumption=Low 3420 ==> Heart Disease Status=No 2740    <conf:(0.8)> lift:(1) lev:(0) [4] conv:(1)
10. Smoking=No 4852 ==> Heart Disease Status=No 3887    <conf:(0.8)> lift:(1) lev:(0) [5] conv:(1)
```

The association rules indicate that individuals with healthier lifestyle characteristics, such as low or moderate stress levels, non-smoking behavior, and normal cholesterol indicators, are more frequently associated with the absence of heart disease. These patterns suggest that lifestyle-related factors play an important role in heart health. However, since the lift values are close to 1, the rules mainly provide descriptive insights rather than strong predictive relationships.

## 6. Conclusions

This project successfully implemented a complete data mining pipeline using Java and the Weka framework, covering data conversion, preprocessing, feature selection, classification, evaluation, and association rule mining. The integration of Information Gain with Ranker effectively reduced dimensionality while preserving the most informative attributes, contributing to improved model efficiency and interpretability.

Experimental evaluation using 10-fold cross-validation demonstrates that Random Forest improves overall classification accuracy by enhancing majority-class predictions, whereas Naive Bayes exhibits stronger performance in detecting minority-class instances. This outcome highlights a fundamental trade-off between maximizing overall accuracy and maintaining sensitivity to class imbalance. The application of resampling techniques played an important role in mitigating imbalance effects, although challenges remain in improving minority-class discrimination.

Despite the relatively low ROC values observed, these results are consistent with the severe class imbalance and overlapping feature distributions inherent in the dataset, which limit the separability of classes. Addressing this limitation would likely require alternative modeling strategies, such as cost-sensitive learning or more advanced ensemble methods, rather than simple parameter tuning.

Overall, the project objectives were fully achieved, and the results reflect a solid understanding of both theoretical concepts and practical challenges in data mining. Future work may focus on exploring more sophisticated classifiers, optimizing class-weighted learning, and incorporating alternative evaluation metrics to further improve performance on imbalanced datasets.

## References

1. Noureddin Sadawi. “WEKA API 12/19: Model Evaluation.” *YouTube*, 1 Dec. 2014, www.youtube.com/watch?v=tfJ6aNNMqcs.
2. Noureddin Sadawi. “WEKA API 9/19: Classifiers in WEKA.” *YouTube*, 1 Dec. 2014, www.youtube.com/watch?v=NrWTglBzRVs.
3. GeeksforGeeks. “Information Gain and Mutual Information for Machine Learning.” *GeeksforGeeks*, 15 Apr. 2024, www.geeksforgeeks.org/machine-learning/information-gain-and-mutual-information-for-machine-learning/.
4. Noureddin Sadawi. “WEKA API 1/19: Introduction and Setting up Eclipse.” *YouTube*, 1 Dec. 2014, www.youtube.com/watch?v=6o19TPn181g&list=PLea0WJq13cnBVfsPVNyRAus2NK-KhCuzJ&index=2.
5. “WEKA API 2/19: Loading and Saving Data.” *YouTube*, 1 Dec. 2014, www.youtube.com/watch?v=p5qnJ80oDzg&list=PLea0WJq13cnBVfsPVNyRAus2NK-KhCuzJ&index=3.
6. “WEKA API 3/19: Converting CSV to ARFF and ARFF to CSV.” *YouTube*, 1 Dec. 2014,www.youtube.com/watch?v=tS6qFxNDrMc&list=PLea0WJq13cnBVfsPVNyRAus2NK-KhCuzJ&index=4. Accessed 15 Dec. 2025.
7. “Apriori.” *Sourceforge.io*, 28 Jan. 2022, weka.sourceforge.io/doc.dev/weka/associations/Apriori.html.
8. “Discretize.” *Weka.sourceforge.io*, weka.sourceforge.io/doc.dev/weka/filters/unsupervised/attribute/Discretize.html.
9. “NaiveBayes.” *Sourceforge.io*, 20 Dec. 2019, weka.sourceforge.io/doc.dev/weka/classifiers/bayes/NaiveBayes.html.

10. “RandomForest.” *Weka.sourceforge.io*, weka.sourceforge.io/doc.dev/weka/classifiers/trees/RandomForest.html.

11. “SerializationHelper.” *Sourceforge.io*, 28 Jan. 2022, weka.sourceforge.io/doc.dev/weka/core/SerializationHelper.html.

12. “Resample.” *Weka.sourceforge.io*, weka.sourceforge.io/doc.dev/weka/filters/supervised/instance/Resample.html.

13. “FilteredClassifier.” *Sourceforge.io*, 28 Jan. 2022, weka.sourceforge.io/doc.dev/weka/classifiers/meta/FilteredClassifier.html.

14. “ReplaceMissingValues.” *Weka.sourceforge.io*, weka.sourceforge.io/doc.dev/weka/filters/unsupervised/attribute/ReplaceMissingValues.html.

15. “RemoveDuplicates.” *Weka.sourceforge.io*, weka.sourceforge.io/doc.dev/weka/filters/unsupervised/instance/RemoveDuplicates.html.

16. “Instances.” *Sourceforge.io*, 28 Jan. 2022, weka.sourceforge.io/doc.dev/weka/core/Instances.html.

17. “SMOTE.” *Sourceforge.io*, 21 Dec. 2020, weka.sourceforge.io/doc.packages/SMOTE/weka/filters/supervised/instance/SMOTE.html.