Big Data & AI Course: Classification Lab

Presented By: Meriem MEKKI

04/12/2025

Introduction

This lab is part of the Big Data and AI course. For this exercise, I chose the Heart Disease dataset from Kaggle.

The workflow included:

1. Exploratory Data Analysis (EDA) to understand the dataset and identify patterns.
2. Data preprocessing, including handling missing values, scaling, and encoding categorical features.
3. Experimenting with different classifiers such as Logistic Regression, Random Forest, K-Nearest Neighbors, Support Vector Machine, XGBoost, and Multi-Layer Perceptron.

EDA

The dataset contains 21 columns: 20 features (a mix of numerical and categorical variables) and 1 target variable indicating the presence of heart disease.

#	Column	Non-Null Count	Dtype
0	Age	9971 non-null	float64
1	Gender	9981 non-null	object
2	Blood Pressure	9981 non-null	float64
3	Cholesterol Level	9970 non-null	float64
4	Exercise Habits	9975 non-null	object
5	Smoking	9975 non-null	object
6	Family Heart Disease	9979 non-null	object
7	Diabetes	9970 non-null	object
8	BMI	9978 non-null	float64
9	High Blood Pressure	9974 non-null	object
10	Low HDL Cholesterol	9975 non-null	object
11	High LDL Cholesterol	9974 non-null	object
12	Alcohol Consumption	7414 non-null	object
13	Stress Level	9978 non-null	object
14	Sleep Hours	9975 non-null	float64
15	Sugar Consumption	9970 non-null	object
16	Triglyceride Level	9974 non-null	float64
17	Fasting Blood Sugar	9978 non-null	float64
18	CRP Level	9974 non-null	float64
19	Homocysteine Level	9980 non-null	float64
20	Heart Disease Status	10000 non-null	object

EDA

This dataset contains null values

Alcohol Consumption	2586
Cholesterol Level	30
Sugar Consumption	30
Diabetes	30
Age	29
High LDL Cholesterol	26
CRP Level	26
Triglyceride Level	26
High Blood Pressure	26
Sleep Hours	25
Low HDL Cholesterol	25
Smoking	25
Exercise Habits	25
BMI	22
Stress Level	22
Fasting Blood Sugar	22
Family Heart Disease	21
Homocysteine Level	20
Gender	19
Blood Pressure	19

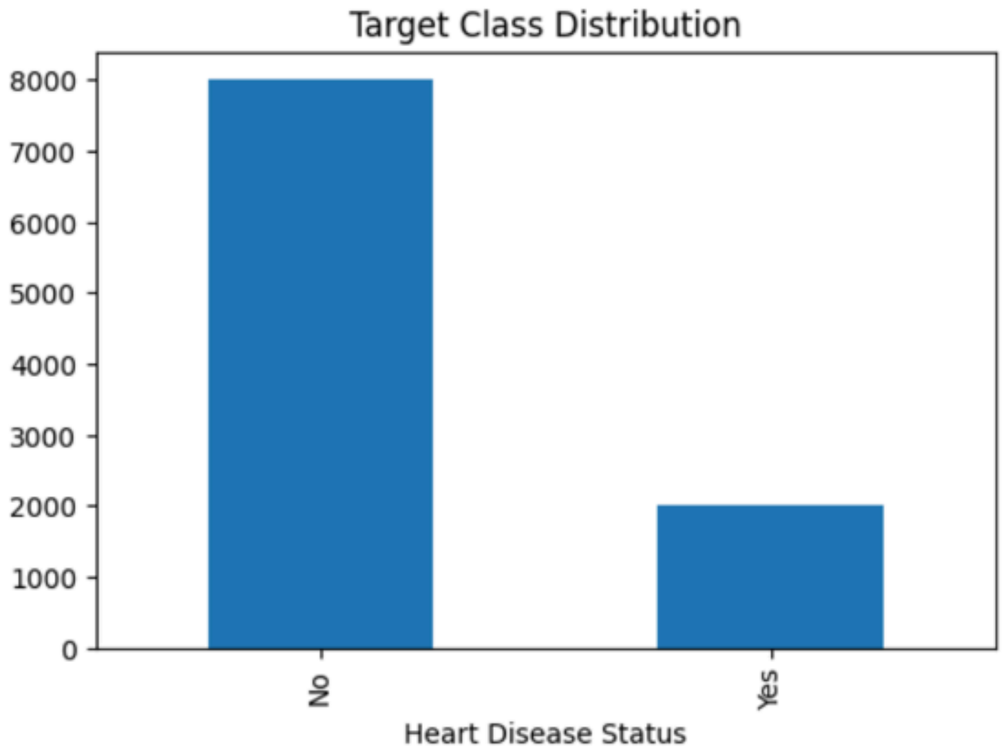
EDA

This dataset contains null values

Alcohol Consumption	2586
Cholesterol Level	30
Sugar Consumption	30
Diabetes	30
Age	29
High LDL Cholesterol	26
CRP Level	26
Triglyceride Level	26
High Blood Pressure	26
Sleep Hours	25
Low HDL Cholesterol	25
Smoking	25
Exercise Habits	25
BMI	22
Stress Level	22
Fasting Blood Sugar	22
Family Heart Disease	21
Homocysteine Level	20
Gender	19
Blood Pressure	19

EDA

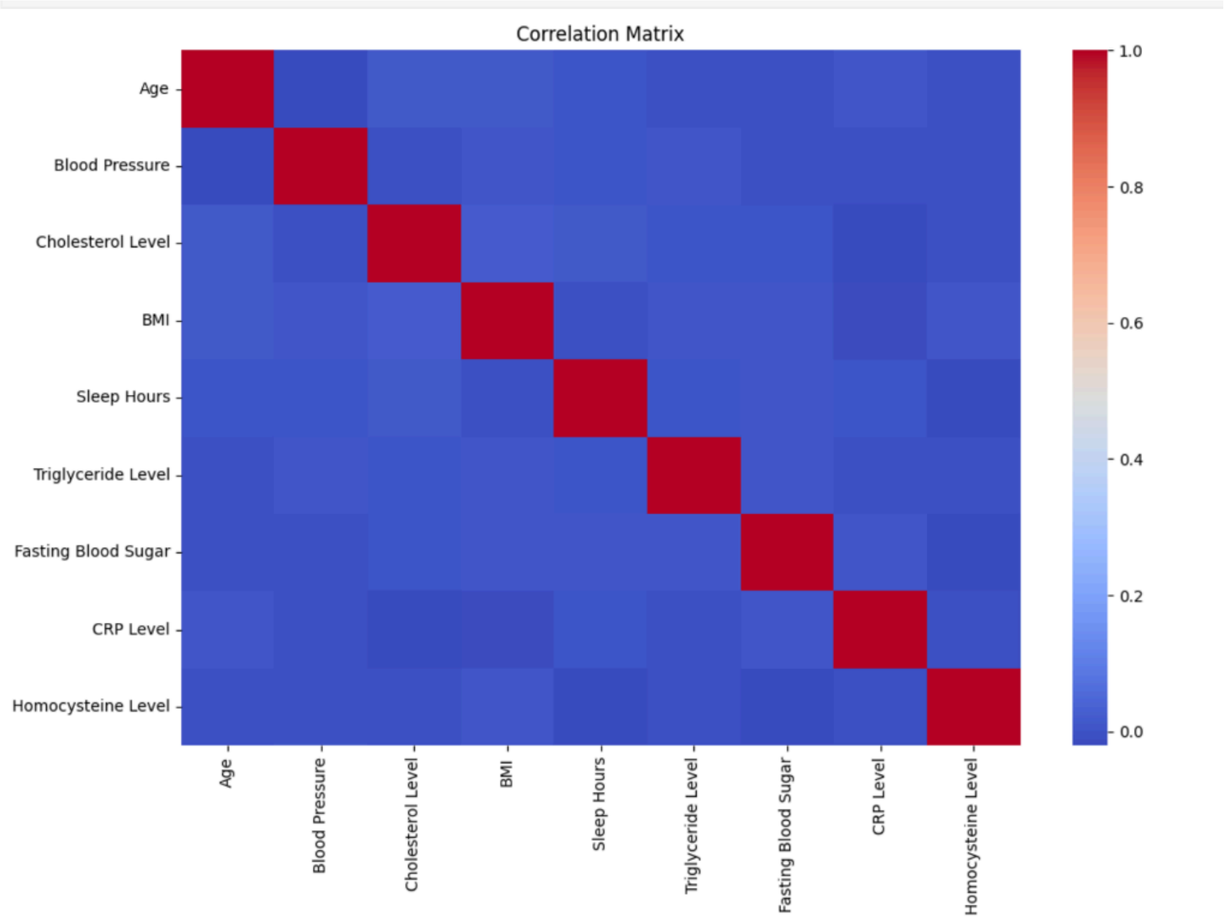
The dataset is highly imbalanced, with a smaller proportion of positive cases, which poses a challenge for classification.




```
Heart Disease Status
No     80.0
Yes    20.0
Name: proportion, dtype: float64
```

EDA

Numerical columns are not strongly correlated according to the correlation matrix, this means each numerical feature captures different aspects of the data, which is generally good for machine learning because the model can benefit from multiple independent signals.



EDA

This dataset doesn’t contain any outliers (no need to handle outliers in the preprocessing phase)

Handle missing values

- **Categorical Columns**

Approach: Added an indicator column and filled missing values with "missing".
Benefits:
Preserves the information that a value was missing, which can be informative for the model.
Avoids losing samples due to missing categorical data.
Allows the model to treat missing as a distinct category, useful for tree-based models like Random Forest or XGBoost.

- **Numerical Columns**

Approach: Imputed missing values with the mean of the column.
Benefits:
Simple and fast to implement.
Maintains the overall distribution of the data.
Works well when the fraction of missing values is small.

Encode Categorical Columns

Utilized approach: Ordinal Encoding

Why Encoding Is Needed

- Machine learning models require numerical inputs.

Ordinal Encoding

- Each category is replaced by an integer value (e.g., "male" → 0, "female" → 1).
- Although Ordinal Encoding is originally designed for features with a natural order, it can also be applied to unordered categorical variables when using models that are not sensitive to the numeric ordering (e.g., Random Forest, XGBoost, MLP).
- Works well when:
  - The dataset contains many categories, making one-hot encoding too sparse or high-dimensional.

Normalize numerical features

Why Normalize?

- Many machine learning models are sensitive to the scale of numerical features.
- Without scaling, features with large ranges can dominate the learning process.
- Scaling helps models converge faster and improves stability.

Utilized approach: MinMax Normalization

This transformation rescales each numerical feature to a fixed range:
Range after scaling: 0 to 1
Preserves the shape of the original distribution
Keeps relative distances between values

Split the dataset

Why Split the Data?

- To evaluate model performance on unseen data.
- Prevents overfitting by ensuring models generalize well.

Utilized approach: Stratified Splitting

Because the dataset is highly imbalanced (80% “No”, 20% “Yes”), a standard random split would distort the class distribution in the train and test sets.

To avoid this, I used Stratified Train/Test Split, which ensures:

- The same class proportions are preserved in both training and testing sets.
- Each subset represents the overall dataset fairly.
- More reliable and stable evaluation metrics, especially for minority classes.

Classification models

Objective:
Evaluate multiple classification algorithms to identify the most effective model for predicting heart disease.

Experimented models:
I trained and evaluated six different classification models:

- Logistic Regression
- Random Forest Classifier
- K-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- XGBoost Classifier
- Multi-Layer Perceptron (MLP)

Classification models

Results

Model Name	Accuracy	F1 score - No	F1 score - Yes
Logistic regression	0.8	0.89	0.0
Random Forest	0.8	0.89	0.0
KNN	0.756	0.86	0.10
XGBoost	0.793	0.88	0.01
SVM	0.8	0.89	0.0
MLP	0.7315	0.84	0.16

Classification models

Interpretation of Results

1. **Accuracy appears acceptable, but it is misleading**
All models achieve around 75% to 80% accuracy, which at first glance looks good.
However, because the dataset is highly imbalanced (80% No, 20% Yes), a model can reach 80% accuracy simply by predicting “No” for every sample.
This is exactly what is happening for most models.

2. Models fail to generalize to the minority class
Looking at the F1-score of the “Yes” class, which represents patients with heart disease:
Logistic Regression → 0.00
Random Forest → 0.00
KNN → 0.10
XGBoost → 0.01
SVM → 0.00
MLP → 0.16 (best but still very poor)
This means:

- The models are not detecting positive cases.
- They learned only to predict the majority class (“No”).
- There is severe class imbalance impact.

Why accuracy is not a reliable metric here

For imbalanced datasets:

- Accuracy hides poor performance on the minority class
- F1-score for the positive class, recall, and ROC-AUC are more informative

You correctly focused on F1-score (Yes), and this reveals the true problem.

CV + Grid Search did NOT improve results

Even with hyperparameter tuning, the models still predicted mostly the majority class.
This shows that:

- The issue is not the model complexity,
- The root cause is the data imbalance, which dominates the learning process.

Downsampling did not help

Attempted downsampling (ClusterCentroids):

- It drastically reduced the dataset size
- Important information was lost
- Models were trained on too few samples → poor performance
- Accuracy dropped significantly

Conclusion

- Although the models achieved seemingly acceptable accuracy, they failed to detect the minority class due to strong dataset imbalance.
- Accuracy is therefore not a valid performance metric in this context.
- Hyperparameter tuning and downsampling did not solve the issue, as the imbalance is too severe and undersampling reduces the dataset too much.