📌 Report on the Development of the Obesity Diagnosis Application

This report documents the steps taken in developing our obesity diagnosis aid application, from data processing to the final implementation of the solution.

I. Introduction

Obesity is a major health issue requiring precise evaluation to ensure effective medical follow-up.
As part of **Coding Week**, our team developed a **medical decision support application** using **machine learning** to predict a patient's obesity level based on their physical and behavioural characteristics.

The main objectives of our project were to:

- **Optimize the performance of the classification model** by testing different methods.
- **Compare several Machine Learning algorithms** to identify the one offering the best accuracy.
- **Deploy an interactive interface** allowing doctors to effectively use our solution.

II. Data presentation

We worked on a dataset titled **ObesityDataSet_raw_and_data_sinthetic.csv**, containing **over 2000 observations** and the following variables:

- **Physical data**: age, height, weight.
- **Eating habits**: consumption of vegetables, water, high-calorie foods.
- **Physical activity and lifestyle**: frequency of sports, time spent in front of a screen.
- **Family factors**: history of obesity in the family.

III. Data processing

A. Cleaning and Preprocessing

- Checking and managing **missing values** .
  - ➔ Result: None detected
- Transforming **categorical variables** into numerical variables via **One-Hot Encoding**.
- Normalizing continuous variables to homogenize the data scale.
- Checking outliers.

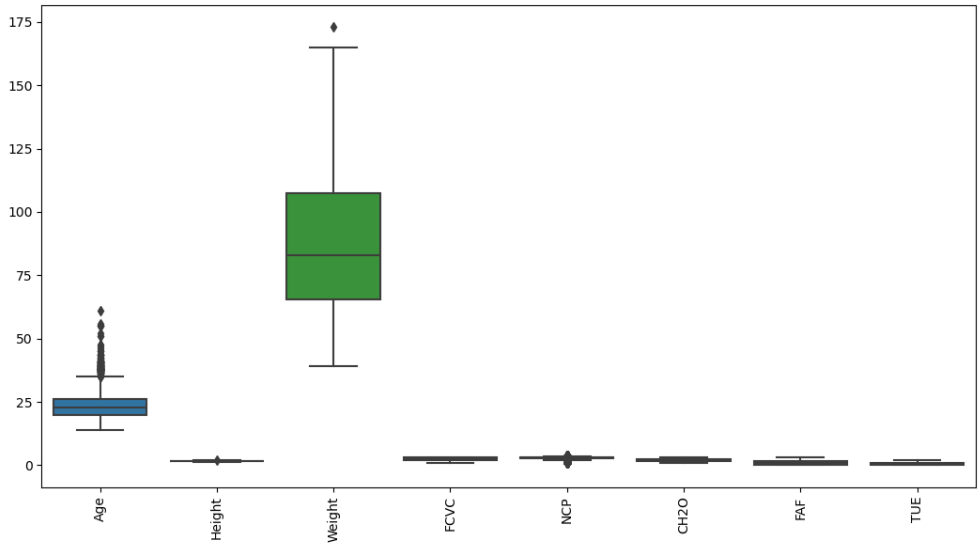


Figure 1: outliers visualization before cleaning

- Managing outliers.

  ➔ Result: 2 cleaned datasets were generated : the first one was obtained by approaching the outliers to the samples' mean value (winsoring) , the second one by filtering and deleting them.

  ➔ Objective: Comparing the performance of both datasets combined with the different models.

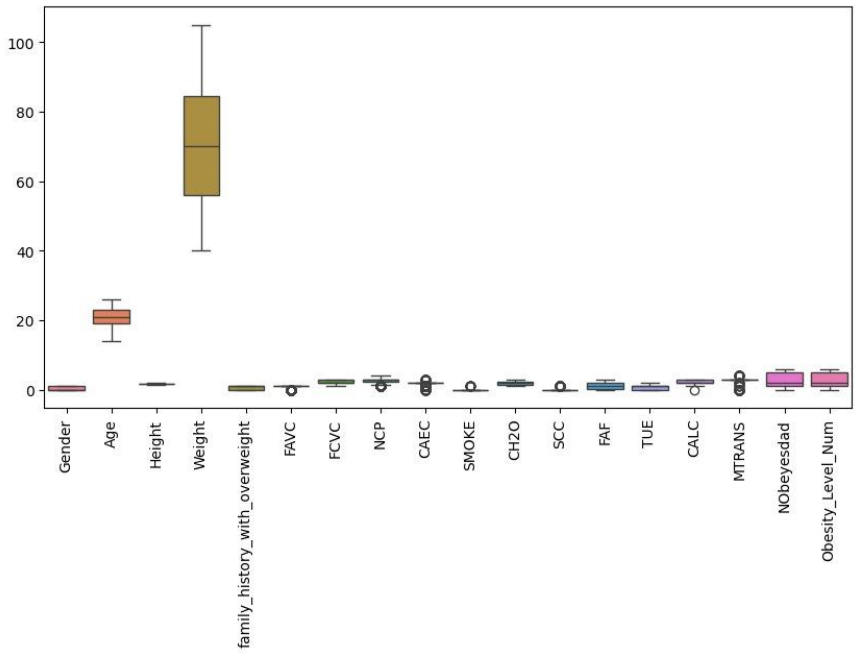
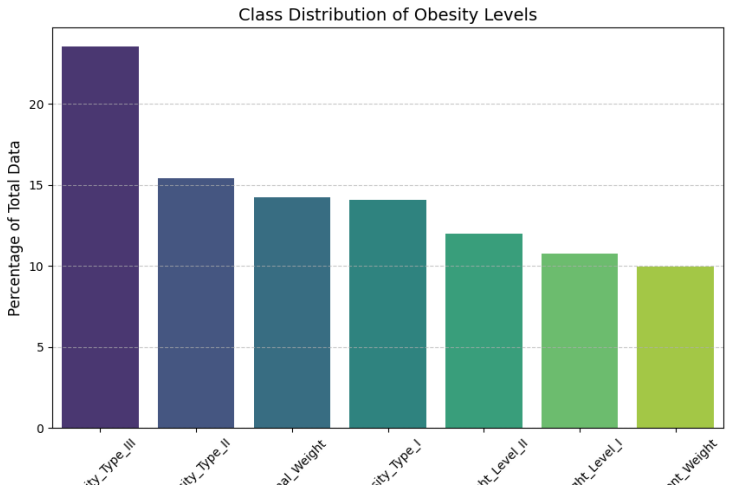


Figure 2: outliers visualization after cleaning

- Data distribution.
  ➔ Result: there are 7 obesity classes slightly imbalanced.



- Handling data distribution:
  ➔ Since the class imbalance could not affect the model's precision we decided to evaluate each model's performance before and after class distribution methods:

  1. Oversampling (SMOTE)
  2. Undersampling
  3. Combination of Oversampling + Undersampling
  4. Class weights

IV. Model training and comparison

A. Tested models:

To identify the most effective model for our task, we trained and evaluated the following machine learning classifiers:

- **Random Forest Classifier**
- **XGBoost Classifier**
- **LightGBM Classifier**
- **CatBoost Classifier**

Each of these models was trained on two variations of our dataset:

1. **Approached Outlier Dataset:** In this version, extreme values were adjusted to be more consistent with the rest of the data.
2. **Filtered Dataset:** Here, the extreme outliers were completely removed to enhance data reliability.

Each one of these Datasets was tested Before and after Applying class imbalance handling methods.

B. Final Choice of Dataset and Balancing Method

Each member chose the best combination for their model by testing different balancing methods.
Finally, **we compared the performance of the 4 models on the following metrics**:

- Accuracy
- ROC-AUC Score
- F1-Score
- Confusion Matrix
- Shap Graph: most influential feature should be relevant to medical data.

Model	Dataset used	Balancing method	Accuracy	F1-Score	ROC-AUC
XGBoost	Approached Outlier Dataset	Class Weights	0.9550	0.84	0.9968
Random Forest	Filtered Data	Oversampling + Undersampling	0.95	0.88	0.9985
LightGBM	Filtered Data	Oversampling	0.9574	0.80	0.9983
CatBoost	Filtered Data	Undersampling	0.9598	0.75	0.9981

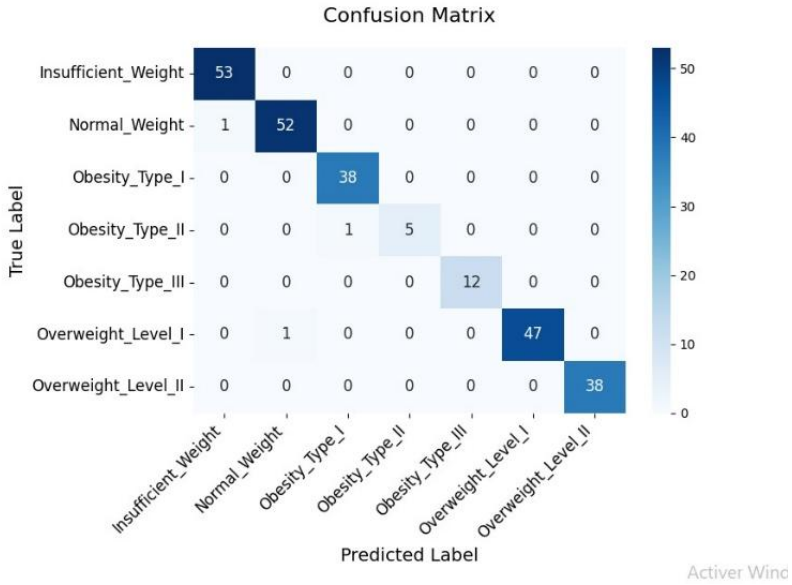


Figure 3: Confusion Matrix of the chosen combinaison

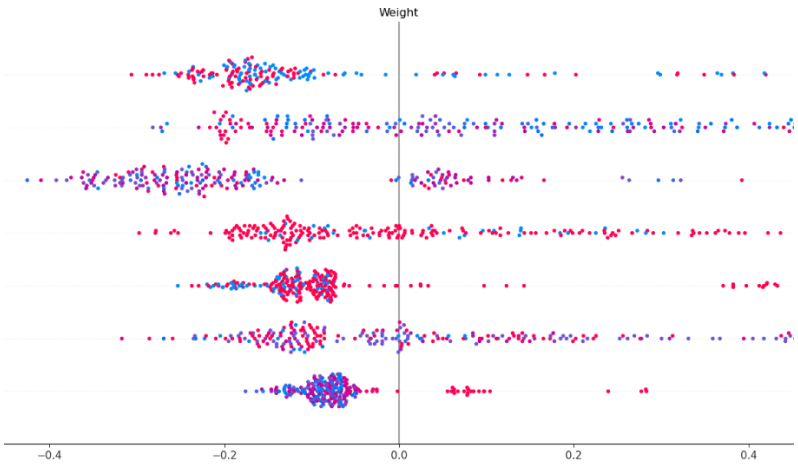


Figure 4: Shap graph of the chosen model

➔ **Results**

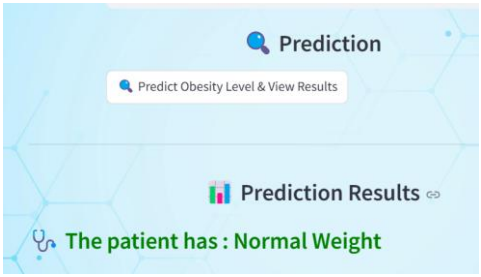
- **Random Forest** with the **Filtered dataset** and the **Oversampling + Undersampling:** (adding to rare classes and subtracting from dominant ones) method gave the best results.
- We therefore retained this configuration for further implementation.

V. Application features

➔ **Doctor Interface**

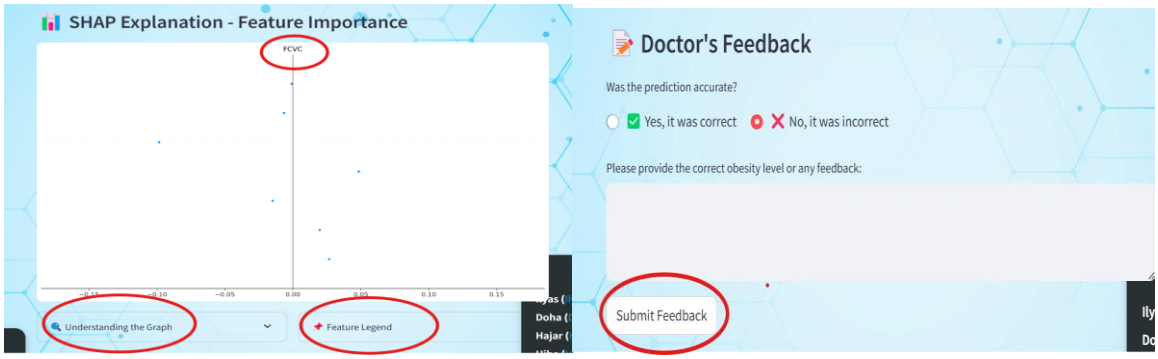
We Integrated the model into a **Streamlit** application for real-time predictions. We Developed an interactive interface tailored for doctors, featuring:

- Real-time obesity level prediction.
- Shap explanations on the classification.
- Doctor feedback on the model's prediction to support the prediction if accurate, to correct it if false.



➔ **Optimized User Experience**

- Clear interface with differentiated icons and colors.
- “Feature Legend” bar shows on demand to guarantee a better understanding of the parameters.
- “understanding the shap graph” mode activatable on demand.
- Doctor’s feedback feature sends the feedback text to a document to collect data.



VI. Application implementation

A. Deployment and Containerization

- Created a **Dockerfile** and **docker-compose.yml** for deployment.
- Integrated the project on **GitHub** for team sharing.
- Used **GitHub Actions** for CI/CD automation.
- The following MVC architecture was chosen:

```
Coding-Week/
|—— model/              # Machine Learning model
|   |—— train_model.py     # Model training and saving
|   |—— evaluate_model.py  # Model evaluation and testing
|   |—— explain_model.py   # SHAP-based model explanation
|   |—— obesity_model.pkl  # Trained Random Forest model
|—— controller/         # Business logic (API and processing)
|   |—— predict.py         # Prediction handling
|   |—— feedback_handler.py # Recording doctor feedback
|—— view/               # Streamlit user interface
|   |—— app.py             # Main application file
|—— data/               # Dataset and processed data
```

```
|   ├── raw/              # Raw dataset
|   ├── processed/        # Cleaned dataset (`dataset.csv`)
|── 📂 docker/            # Docker configuration
|   ├── Dockerfile
|   ├── docker-compose.yml
|── 📂 tests/             # Unit tests
|   ├── test_model.py     # Model testing
|   ├── test_api.py       # API testing
|   ├── test_data.py      # Data validation
|── requirements.txt      # Project dependencies
|── README.md + Report          # Project documentation
|── .gitignore            # Files to ignore in Git
|── .github/workflows/    # CI/CD automation
|   ├── ci-cd.yml
```

VII. Conclusion

➔ **Project summary**

- A reliable and **optimized model**, chosen after thorough comparison.
- An **intuitive interface** adapted to doctors' needs.
- Facilitated deployment via **Docker and GitHub**.

➔ **Next Improvements**

- **Adding a REST API** for integration with other medical tools.
- **Training a neural network-based model** to improve accuracy.
- **Adding an intelligent feedback system** to dynamically adapt the model to new data.

Note: We decided to test each model on every machine to give everyone a chance to practice. Ultimately, we chose one model to optimize together over the rest of the week. The reason why we started uploading once on Github.

© Made by: Douha MAJRI, Ilyas BAJJA, Hajar NAJIB, Meriem LAAROUSSI, Hiba CHUIMI