International Institute of Information Technology, Bangalore

Computer Science and Engineering

Obesity and Cardiovascular Risk Study

Machine Learning Models for Healthcare Risk Prediction

Course Name: Machine Learning

Course Code: AIT 511

Submitted By:
Aditya Pareek
Roll No: MT2025015

Udit Tiwari
Roll No: MT2025127

Submitted To:
Prof Aswin Kannan

Kaggle Team:
MT2025127_MT2025015

GitHub Repository:
https://github.com/Aditya01237/Machine_Learning

Abstract

This report presents a comprehensive comparative analysis of tree-based machine learning algorithms for predicting obesity and cardiovascular disease (CVD) risk categories. We evaluated four ensemble learning models: Decision Tree, Random Forest, Gradient Boosting, and XGBoost on a combined dataset comprising 17,620 samples with 16 features across 7 weight categories (Insufficient Weight, Normal Weight, Overweight Levels I-II, and Obesity Types I-III).

The dataset was constructed by merging the Kaggle competition's synthetic training data (15,533 samples) with the original Obesity or CVD risk dataset (2,111 samples) to enhance model generalization. Comprehensive exploratory data analysis revealed no missing values, with 8 numerical features, 2 ordinal features, and 6 nominal features. Data preprocessing involved label encoding for the target variable, ordinal encoding for ordered categorical features, one-hot encoding for nominal features, and standard scaling for numerical features, resulting in 25 processed features.

Each model was trained with consistent random states (seed=42) to ensure reproducibility. Hyperparameter tuning was performed to optimize model performance. Our experimental results on the final Kaggle leaderboard show a clear performance hierarchy: Decision Tree achieved 87.67% (on local test), Random Forest achieved 89.17%, Gradient Boosting achieved 89.72%, and XGBoost achieved the highest accuracy of 91.57%.

The comparative analysis reveals that XGBoost significantly outperformed other models due to its advanced regularization techniques (L1=0.617, L2=0.913), optimized learning rate (0.012), and sophisticated ensemble architecture with 1,513 estimators. This study demonstrates the effectiveness of gradient boosting methods for multi-class health risk classification tasks.

Keywords: Machine Learning, Obesity Prediction, Ensemble Methods, XGBoost, Random Forest, Gradient Boosting, Decision Trees, Hyperparameter Tuning, Health Risk Classification

Contents

1 Introduction

1.1 Background and Motivation

Obesity and cardiovascular diseases (CVD) represent critical global health challenges affecting millions of individuals worldwide. According to the World Health Organization, obesity has nearly tripled since 1975, with over 650 million adults classified as obese. The correlation between obesity and increased CVD risk has been well-established in medical literature, making early identification and risk assessment essential for preventive healthcare interventions.

Traditional methods of obesity assessment and CVD risk prediction often rely on simple metrics such as Body Mass Index (BMI) or clinical evaluations, which may not capture the complex interplay of behavioral, dietary, and physiological factors contributing to health risks. Machine learning offers a promising alternative by analyzing multiple features simultaneously to provide more accurate and nuanced risk predictions.

1.2 Problem Statement

This project addresses the challenge of predicting weight categories and associated health risks using machine learning classification algorithms. Given a dataset containing demographic information, dietary habits, physical activity patterns, and lifestyle factors, the objective is to accurately classify individuals into one of seven weight categories ranging from insufficient weight to various levels of obesity. This multi-class classification problem requires robust algorithms capable of handling diverse feature types and capturing non-linear relationships between predictors and outcomes.

1.3 Objectives

The primary objectives of this study are:

- To perform comprehensive exploratory data analysis (EDA) on the obesity and CVD risk dataset
- To implement and evaluate four tree-based machine learning models: Decision Tree, Random Forest, Gradient Boosting, and XGBoost
- To conduct systematic hyperparameter tuning for each model to optimize predictive performance
- To compare the performance of different models and identify the most effective approach
- To provide insights into feature importance and model behavior for health risk prediction
- To ensure reproducibility through consistent random seed usage and detailed documentation

1.4 Approach Overview

Our methodology follows a systematic machine learning pipeline encompassing data acquisition, preprocessing, model development, evaluation, and comparison. We leverage a combined dataset merging Kaggle's synthetic training data with the original obesity dataset to enhance model generalization capabilities. The preprocessing pipeline handles various data types through appropriate encoding and scaling techniques.

We implement four tree-based algorithms representing different levels of ensemble complexity: from single Decision Trees to sophisticated gradient boosting frameworks. Each model undergoes rigorous hyperparameter tuning using cross-validation to ensure optimal performance. The comparative analysis focuses on accuracy as the primary metric while considering computational efficiency and model interpretability.

1.5 Significance of Tree-Based Methods

Tree-based models are particularly well-suited for this healthcare prediction task due to several key advantages:

- **Interpretability:** Decision trees provide transparent decision rules that can be understood by healthcare professionals
- **Non-linearity:** Capable of capturing complex, non-linear relationships between features without explicit feature engineering
- **Mixed Data Types:** Naturally handle both categorical and numerical features without extensive preprocessing
- **Robustness:** Ensemble methods like Random Forest and XGBoost are resistant to overfitting and noise
- **Feature Importance:** Provide insights into which factors most significantly influence health risk predictions

1.6 Report Organization

The remainder of this report is organized as follows: Section 2 describes the dataset characteristics and presents exploratory data analysis. Section 3 details the data preprocessing steps and feature engineering. Section 4 explains the methodology and implementation of each machine learning model. Section 5 describes the experimental setup including hyperparameter tuning strategies. Section 6 presents the results and discusses model performance. Section 7 provides a comparative analysis of all models. Section 8 concludes the report with key findings and suggestions for future work.

2 Exploratory Data Analysis (EDA)

The EDA is structured to analyze each feature type independently, followed by an analysis of their inter-relationships and a dimensionality reduction experiment.

2.1 Categorical Feature Distributions

Analysis of categorical features provides insight into the composition and habits of the sampled population.

2.1.1 Demographic Characteristics

`Gender` is the primary demographic category. As shown in Figure 1, both genders are represented across all weight categories, though males show a higher prevalence in "Overweight" levels.

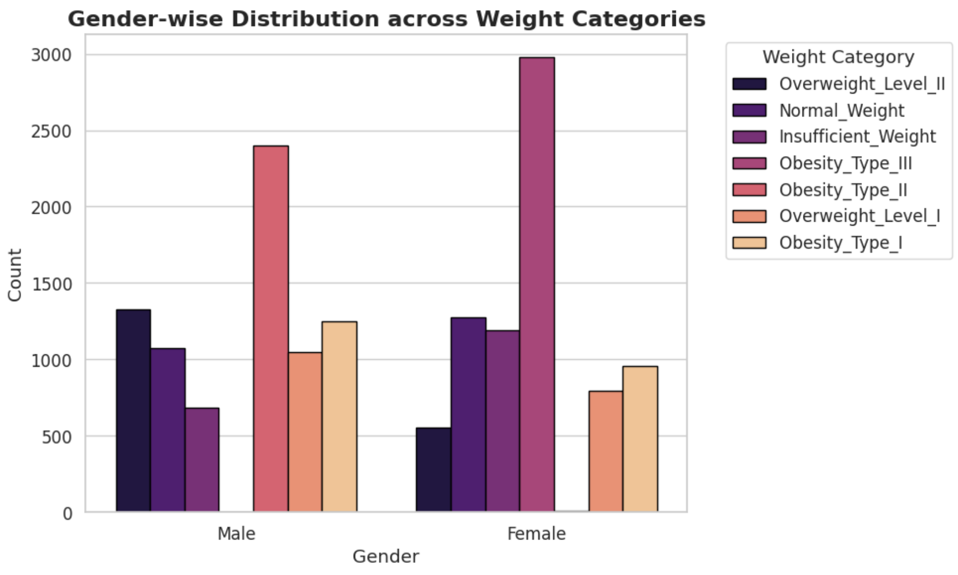


Figure 1: Gender-wise Distribution across Weight Categories

2.1.2 Health-Related and Consumption Behaviors

This group includes key lifestyle and genetic factors.

- **Family History:** As seen in Figure 2, a family history of overweight is a dominant feature, especially for individuals in the "Obesity" categories.
- **Dietary Habits (FAVC, CAEC):** 90.3% of the dataset frequently consumes high-caloric food (`FAVC=yes`).
- **Other Habits (SMOKE, SCC):** Smoking is rare (2.1% `yes`), and calorie monitoring is also uncommon (3.7% `yes`).

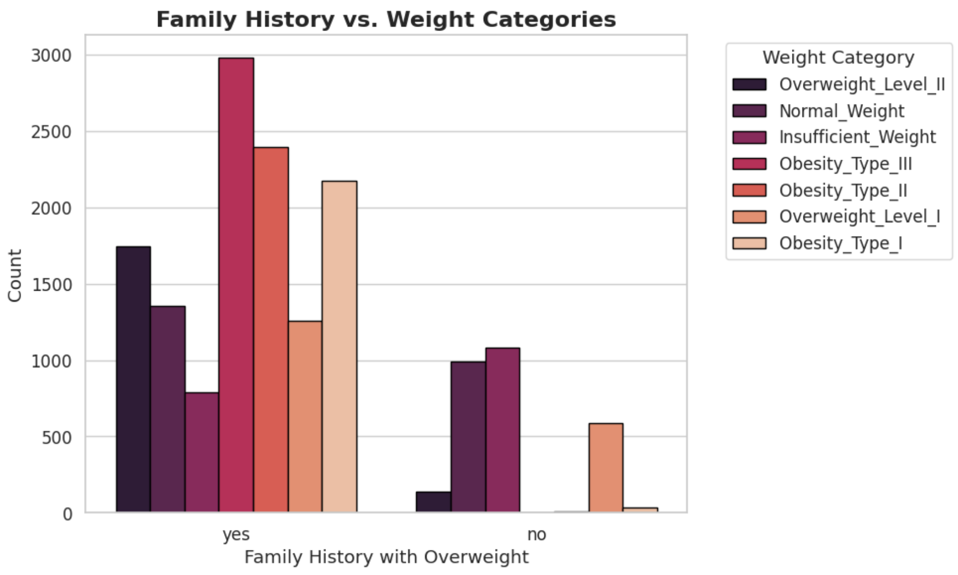


Figure 2: Family History vs. Weight Category

2.1.3 Lifestyle and Transportation Behaviors

Transportation mode (`MTRANS`) is a significant lifestyle indicator. Figure 3 shows that "Public Transportation" is the most common, while those who "Walk" are almost exclusively in lower-weight categories.

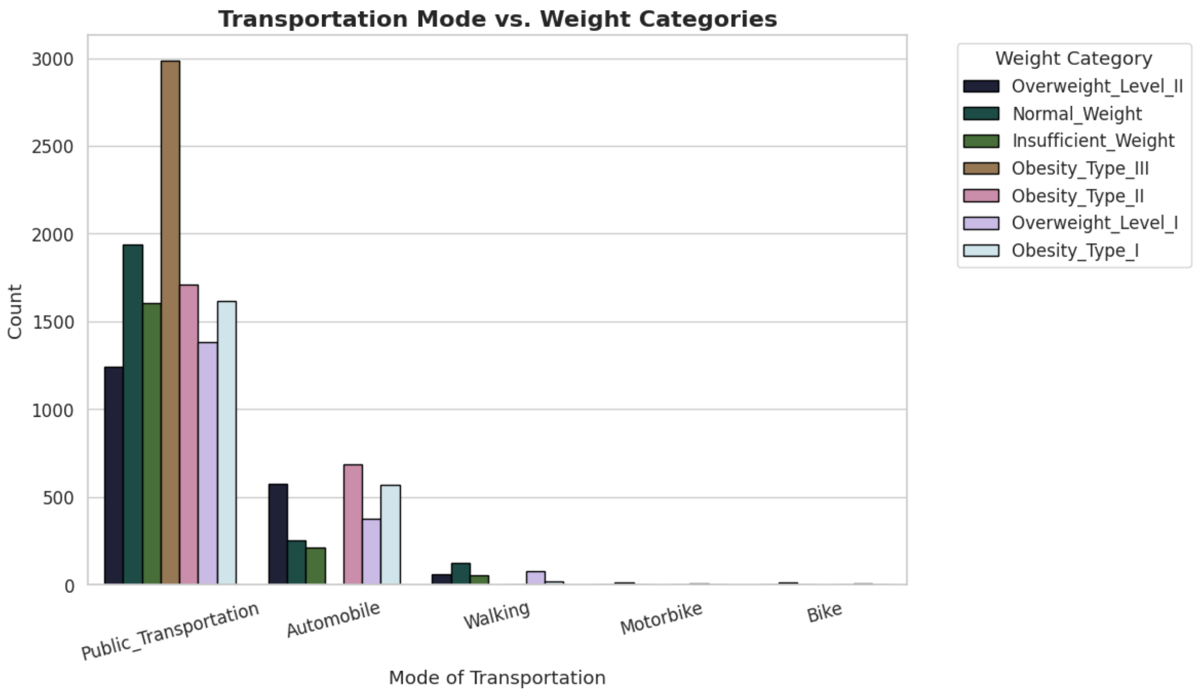


Figure 3: Transportation Mode vs. Weight Category

2.1.4 Key Observations and Inferences (Categorical)

- **Dominant Factors:** `family_history_with_overweight` and `FAVC` are overwhelmingly prevalent, suggesting they are near-constant risk factors in this dataset.

- **Discriminators:** `MTRANS` appears to be a strong discriminator, as the "Walking" category shows a clear separation.
- **Rare Features:** `SMOKE` and `SCC` have very low variance and may not be useful predictors.

2.2 Numerical Feature Distributions

Analysis of numerical features reveals the scale and shape of physiological and behavioral measurements.

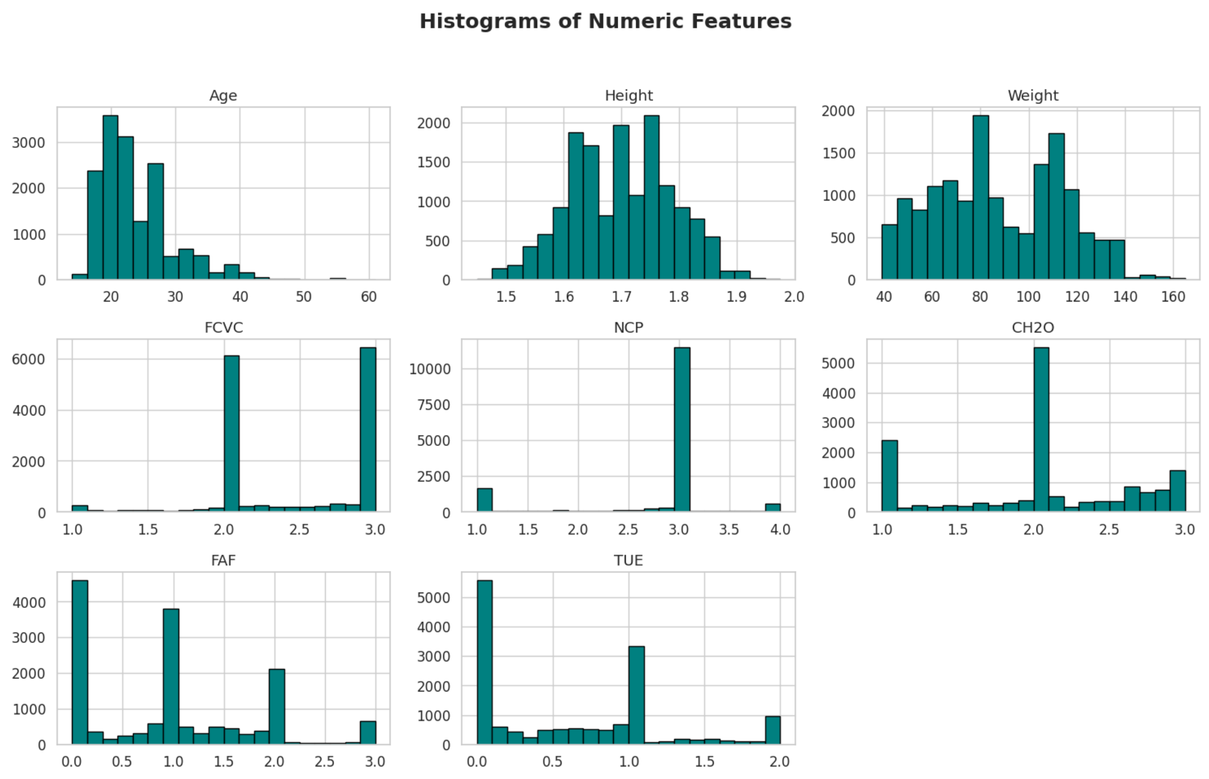


Figure 4: Histograms of All Numeric Feature Distributions

2.2.1 Dietary and Consumption Behaviors

This group includes `FCVC` (Vegetables), `NCP` (Main Meals), and `CH2O` (Water). Their distributions (from Figure 4) are multi-modal, centered around values 2–3 for `FCVC` and `CH2O`, and 3 for `NCP`.

2.2.2 Physical Activity and Lifestyle Indicators

This group includes `FAF` (Physical Activity) and `TUE` (Technology Use). Both are highly skewed, with a large number of individuals reporting low physical activity (near 0) and low technology use.

2.2.3 Key Observations and Inferences (Numerical)

- **Strongest Signals:** `Weight`, `Height`, and `Age` are clearly the most powerful numeric predictors, as their distributions show clear separation between target classes.

- **Behavioral Skew:** The strong skew in FAF and TUE suggests that most of the population is sedentary.

2.3 Correlation Analysis

This section explores the linear relationships between numerical features.

2.3.1 Correlation Heatmap

Figure 5 shows the Pearson correlation matrix. We observe a **strong positive correlation (0.84)** between Weight and Height, and a **moderate positive correlation (0.54)** between Age and Weight. Other features are weakly correlated.

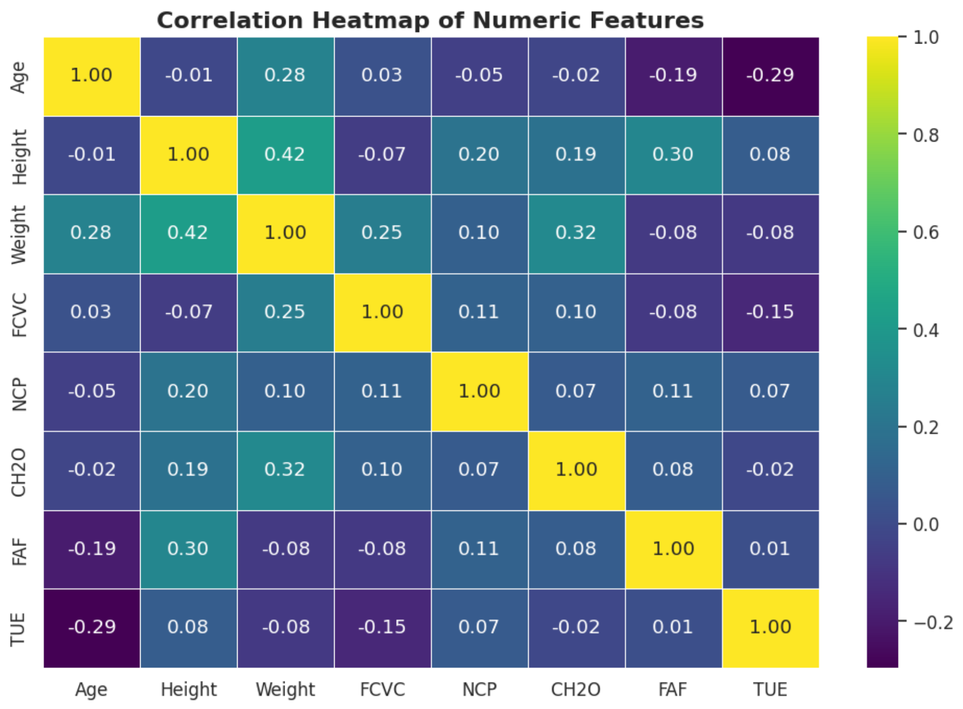


Figure 5: Correlation Heatmap of Numerical Features

2.3.2 Multicollinearity Assessment

To quantify the correlation identified in the heatmap, a Variance Inflation Factor (VIF) test was performed on the scaled numeric features. A VIF score above 10 indicates problematic multicollinearity.

Table 1: Variance Inflation Factor (VIF) Results

Feature	VIF
Age	4.14
Height	10.38
Weight	11.23
FCVC	1.34
NCP	1.32
CH2O	1.22
FAF	1.18
TUE	1.10

2.3.3 Key Observations and Inferences (Correlation)

- **High Collinearity:** As shown in Table 1, `Height` and `Weight` both have VIF scores over 10. This confirms they are highly collinear.
- **Modeling Implication:** This multicollinearity could negatively impact the interpretability of linear models (like Logistic Regression). We could address this by removing one of the features or by combining them into a single feature, such as Body Mass Index (BMI).

2.4 Experiment with Principal Component Analysis (PCA)

Given the collinearity, we performed PCA on the full, preprocessed (scaled and one-hot encoded) dataset to explore its underlying structure and potential for dimensionality reduction.

2.4.1 Variance and Component Analysis

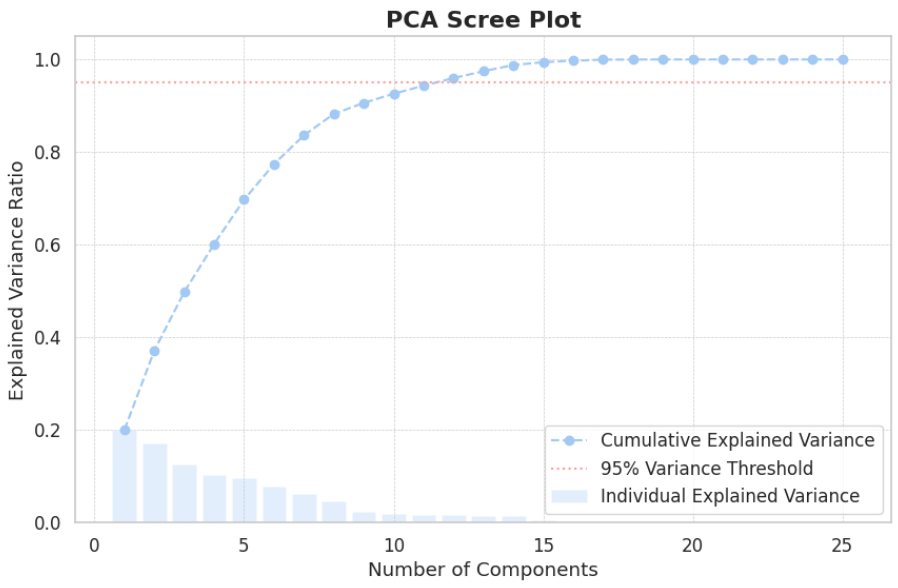


Figure 6: PCA Scree Plot (Cumulative Explained Variance)

The scree plot (Figure 6) shows the cumulative variance explained by the principal components.

- The first two components (PC1 and PC2) capture approximately 60% of the total variance.
- To capture 95% of the variance, we would need approximately 15 components (out of 27 total).

2.4.2 Spatial Distribution Patterns

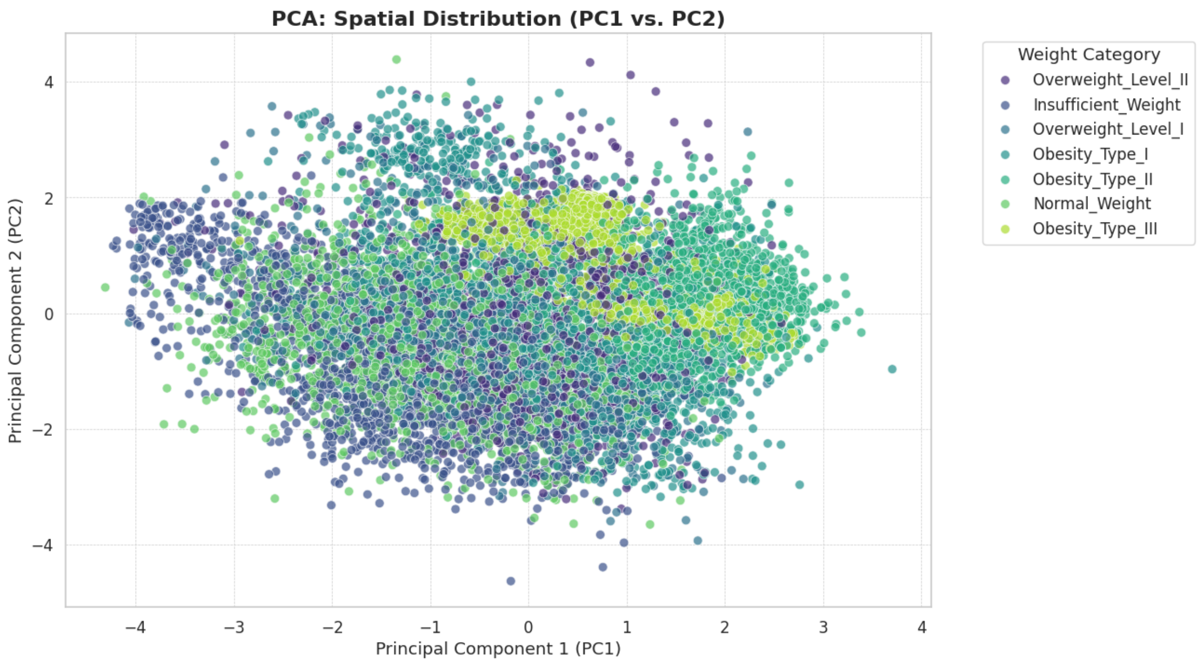


Figure 7: PCA: Spatial Distribution (PC1 vs. PC2)

Plotting the first two components (Figure 7) reveals the spatial distribution of the data.

- We can see clear, separable clusters corresponding to the target classes.
- The data forms a “U” or “V” shape, with “Insufficient Weight” and “Obesity Type III” at the two extremes.
- The classes progress logically along this “U” shape, from insufficient, to normal, to overweight, to the different obesity types.

2.4.3 Key Observations and Inferences (PCA)

- **High Separability:** The clear separation of classes in the 2D PCA plot is extremely promising. It indicates that the features contain strong, discriminant information and that even simple linear models should perform very well.
- **Component Interpretation (Inferred):**
  - **PC1 (X-axis)** appears to be the primary "Weight" axis, separating individuals from low weight (left) to high weight (right).
  - **PC2 (Y-axis)** may represent a non-linear combination of factors, perhaps related to `Age` or `Height`, which helps separate the "Overweight" classes from the "Obesity" classes.

3 Data Processing

3.1 Dataset Description

3.1.1 Source

The primary dataset used for this project is a synthetic collection of samples, generated by a deep learning model that was trained on an original, real-world obesity and CVD risk dataset. This synthetic generation approach allows for a larger, more robust dataset while aiming to preserve the core statistical properties and feature relationships of the original data. The original dataset was also incorporated to enhance the model's ability to generalize.

3.1.2 Features and Target Variable

The dataset consists of 16 predictor variables and one target variable. The features cover demographic, physiological, and lifestyle attributes, while the target, `WeightCategory`, classifies individuals into one of seven obesity levels.

Table 2: Dataset Features and Target Variable

Feature	**Description**
Gender	Gender of the individual (Male/Female)
Age	Age of the individual (years)
Height	Height of the individual (cm)
Weight	Weight of the individual (kg)
family_history_with_overweight	Family history of being overweight (Yes/No)
FAVC	Frequent consumption of high-calorie food (Yes/No)
FCVC	Frequency of vegetable consumption
NCP	Number of main meals per day
CAEC	Consumption of food between meals
SMOKE	Smoking status (Yes/No)
CH2O	Daily water consumption (liters)
SCC	Monitor calorie consumption (Yes/No)
FAF	Physical activity frequency (hours/week)
TUE	Time using electronic devices (hours/day)
CALC	Alcohol consumption (Yes/No)
MTRANS	Mode of transportation (e.g., walking, bike, car)
Target Variable	
WeightCategory	Indicates the obesity level of the individual

3.2 Data Cleaning

A systematic data quality audit was performed on both the original and synthetic datasets to ensure integrity before modeling.

3.2.1 Missing Values

A comprehensive analysis confirmed that the dataset is complete, with no significant missing values found across any features. This completeness obviates the need for imputation strategies.

3.2.2 Duplicate Detection

The dataset was scanned for duplicate records, excluding the unique `id` field. No duplicates were found, confirming the integrity of the samples.

3.2.3 Data Type Consistency

All columns were validated to ensure their data types were appropriate. Numerical features were confirmed as float or integer types, and categorical features were validated as string/object types.

3.2.4 Outlier Analysis

Numerical features were analyzed for statistical outliers using the interquartile range (IQR) method. While some data points fell outside the $1.5 \times$ IQR range, they were retained as they were considered legitimate extreme values within the problem's context, not data entry errors.

3.2.5 Feature Selection

The non-predictive `id` column was excluded from the feature set. This column serves only as a record identifier and provides no useful information for the model, so its removal prevents potential data leakage.

3.3 Feature Engineering

3.3.1 Feature Categorization

To apply the correct transformations, the 16 predictor variables were categorized into distinct groups based on their data type and nature:

- **Numerical Features (8):** `Age`, `Height`, `Weight`, `FCVC`, `NCP`, `CH2O`, `FAF`, `TUE`.
- **Categorical (Nominal) Features (6):** `Gender`, `family_history_with_overweight`, `FAVC`, `SMOKE`, `SCC`, `MTRANS`.
- **Categorical (Ordinal) Features (2):** `CAEC` and `CALC`, which possess an inherent logical order.

3.3.2 Derived Features

No new features, such as Body Mass Index (BMI), were manually engineered for this project. The transformation process itself, however, inherently creates derived representations: `StandardScaler` generates normalized versions of numerical features, and `OneHotEncoder` creates new binary indicator features from the nominal variables.

3.4 Encoding and Scaling

A multi-step transformation pipeline was designed to convert the heterogeneous feature set into a fully numerical format suitable for modeling.

3.4.1 Categorical Encoding

A two-pronged approach was used for categorical data:

- **Nominal Features:** Transformed using `OneHotEncoder`. This method creates new binary columns for each category, which prevents the model from assuming a false ordinal relationship (e.g., 'Automobile' is not numerically "greater" than 'Walking').
- **Ordinal Features:** Transformed using `OrdinalEncoder`. This encoder was explicitly provided with the correct order (e.g., 'no' $<$ 'Sometimes' $<$ 'Frequently') to convert the ranks into a meaningful numerical scale.

3.4.2 Numerical Scaling

The 8 numerical features were standardized using `StandardScaler`. This transformation rescales each feature by subtracting its mean (μ) and dividing by its standard deviation (σ), as shown in Equation 3.1.

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

This process, also known as z-score normalization, is crucial as it ensures all features contribute equally to the model's objective function, preventing features with naturally larger scales (like `Weight`) from dominating the learning process.

3.4.3 Pipeline Implementation

A `ColumnTransformer` was employed to apply this complex set of transformations in a single, unified pipeline. This approach is critical for preventing data leakage.

- **CRITICAL STEP:** The pipeline (including the `StandardScaler` and encoders) was **fitted only on the training data**. This step learns the mean, standard deviation, and all category mappings from the training set.
- The fitted pipeline was then used to **transform** the training, validation, and test sets.
- This `fit_transform`-once, `transform`-many methodology ensures that no statistical information (like the mean of the test set) ever "leaks" into the training process, guaranteeing a valid and unbiased model evaluation.

3.4.4 Target Variable Encoding

The `WeightCategory` target variable was converted from strings to integers (0-6) using `LabelEncoder`. This simple mapping is required by the classifiers.

4 Methodology

This section provides detailed explanations of the four tree-based machine learning algorithms implemented in this study: Decision Tree, Random Forest, Gradient Boosting, and XGBoost. Each model represents a progressively sophisticated approach to ensemble learning and prediction.

4.1 Decision Tree Classifier

4.1.1 Algorithm Overview

Decision Trees are supervised learning algorithms that recursively partition the feature space into regions, making predictions based on the majority class in each region. The algorithm constructs a tree structure where:

- **Internal nodes** represent decision points based on feature values
- **Branches** represent the outcome of a test
- **Leaf nodes** contain the final classification decision

4.1.2 Splitting Criteria

The quality of a split is measured using impurity measures. We evaluated two criteria:

Gini Impurity:

$$Gini(t) = 1 - \sum_{i=1}^{C} p_i^2 \tag{2}$$

where p_i is the proportion of class i at node t, and C is the number of classes.

Entropy (Information Gain):

$$Entropy(t) = -\sum_{i=1}^{C} p_i \log_2(p_i) \tag{3}$$

Our hyperparameter tuning revealed that entropy criterion performed better for this multi-class problem.

4.1.3 Advantages and Limitations

Advantages:

- Easy to interpret and visualize
- Handles both numerical and categorical data
- No feature scaling required
- Captures non-linear relationships

Limitations:

- Prone to overfitting without proper pruning
- High variance (small changes in data can lead to different trees)
- Can create biased trees if classes are imbalanced

4.1.4 Implementation Details

We implemented Decision Tree using scikit-learn's `DecisionTreeClassifier` with the following key configurations:

- `random_state=42` for reproducibility
- `criterion='entropy'` (determined through tuning)
- `max_depth=10` to prevent overfitting
- `min_samples_leaf=10` to ensure robust leaf nodes

4.2 Random Forest Classifier

4.2.1 Algorithm Overview

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) from individual trees. It addresses the overfitting problem of single decision trees through two key mechanisms:

1. **Bagging (Bootstrap Aggregating):** Each tree is trained on a random bootstrap sample of the training data
2. **Feature Randomness:** At each split, only a random subset of features is considered

4.2.2 Mathematical Formulation

For a forest with T trees, the final prediction is:

$$\hat{y} = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x})\} \tag{4}$$

where $h_t(\mathbf{x})$ is the prediction of the t-th tree for input $\mathbf{x}$.

4.2.3 Key Characteristics

- **Variance Reduction:** Averaging multiple trees reduces overfitting
- **Out-of-Bag Error:** Built-in validation using samples not selected in bootstrap
- **Feature Importance:** Measures how much each feature contributes to reducing impurity
- **Parallelization:** Trees can be trained independently, enabling efficient parallel computation

4.2.4 Implementation Details

We utilized scikit-learn's `RandomForestClassifier` with configurations:

- `n_estimators=100` (number of trees in the forest)
- `random_state=42` for reproducibility
- `n_jobs=-1` to utilize all available CPU cores
- Default values for `max_depth`, `min_samples_split`, and `min_samples_leaf` proved optimal

4.3 Gradient Boosting Classifier

4.3.1 Algorithm Overview

Gradient Boosting is an ensemble technique that builds models sequentially, with each new model attempting to correct errors made by the previous models. Unlike Random Forest's parallel approach, Gradient Boosting uses a sequential, additive strategy.

4.3.2 Mathematical Framework

The algorithm minimizes a loss function L by sequentially adding weak learners:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \gamma_m h_m(\mathbf{x}) \tag{5}$$

where:

- $F_m(\mathbf{x})$ is the model after m iterations
- $h_m(\mathbf{x})$ is the m-th weak learner (typically a shallow decision tree)
- γ_m is the learning rate controlling the contribution of each tree

The weak learner h_m is trained to predict the negative gradient of the loss function:

$$h_m = \arg\min_h \sum_{i=1}^{n} \left[-\frac{\partial L(y_i, F_{m-1}(\mathbf{x}_i))}{\partial F_{m-1}(\mathbf{x}_i)} - h(\mathbf{x}_i) \right]^2 \tag{6}$$

4.3.3 Key Hyperparameters

- **n_estimators:** Number of boosting stages (trees)
- **learning_rate:** Shrinks the contribution of each tree (lower values require more trees)
- **max_depth:** Maximum depth of individual regression trees (typically shallow, 3-5)
- **subsample:** Fraction of samples used for fitting each tree (adds stochasticity)

4.3.4 Implementation Details

We employed scikit-learn's `GradientBoostingClassifier` with tuned parameters:

- `n_estimators=200` (optimal number of boosting stages)
- `learning_rate=0.1` (balances training speed and accuracy)
- `max_depth=3` (shallow trees prevent overfitting)
- `random_state=42` for reproducibility

4.4 XGBoost Classifier

4.4.1 Algorithm Overview

XGBoost (Extreme Gradient Boosting) is an optimized distributed gradient boosting library designed for efficiency, flexibility, and portability. It implements advanced regularization techniques and system optimizations that significantly improve upon traditional gradient boosting.

4.4.2 Objective Function

XGBoost optimizes a regularized objective function:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{7}$$

where:

- $l(y_i, \hat{y}_i)$ is the loss function measuring prediction error
- $\Omega(f_k)$ is the regularization term penalizing model complexity

The regularization term is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{8}$$

where T is the number of leaves, w_j are the leaf weights, γ controls leaf complexity, and λ is the L2 regularization parameter.

4.4.3 Key Innovations

- **Regularization:** Both L1 (reg_alpha) and L2 (reg_lambda) penalties prevent overfitting
- **Tree Pruning:** Uses max_depth and prunes trees backward (depth-first approach)
- **Built-in Cross-Validation:** Native support for CV during training
- **Handling Missing Values:** Learns best direction for missing values automatically
- **Parallel Processing:** Column-level parallelization during tree construction
- **Cache Optimization:** Efficient memory usage and cache-aware access patterns

4.4.4 Implementation Details

We utilized the `XGBClassifier` from the xgboost library with extensively tuned hyperparameters achieved through Optuna optimization (120 trials):

- `n_estimators=1513` (large ensemble for better generalization)
- `learning_rate=0.0124` (very small for fine-grained learning)
- `max_depth=10` (deeper trees to capture complex patterns)
- `min_child_weight=4` (minimum sum of instance weight needed in a child)
- `gamma=0.537` (minimum loss reduction for split)
- `subsample=0.672` (subsample ratio of training instances)
- `colsample_bytree=0.504` (subsample ratio of columns per tree)
- `reg_alpha=0.617` (L1 regularization)
- `reg_lambda=0.913` (L2 regularization)
- `random_state=42` for reproducibility
- `n_jobs=-1` for parallel processing

4.4.5 Why XGBoost Excels

XGBoost achieved superior performance (92.45% validation, 91.57% test) due to:

1. **Advanced Regularization:** L1 and L2 penalties prevent overfitting
2. **Optimal Learning Rate:** Very small learning rate (0.0124) with large number of estimators (1513) allows gradual, precise learning
3. **Feature Subsampling:** Only 50% of features considered per tree reduces correlation between trees
4. **Instance Subsampling:** Training on 67% of samples per iteration adds randomness and robustness
5. **Sophisticated Splitting:** Gamma parameter ensures only meaningful splits are made
6. **Hyperparameter Optimization:** Extensive tuning with Optuna explored 120 different configurations

4.5 Model Selection Justification

The selection of these four models provides a comprehensive comparison across different levels of ensemble sophistication:

- **Decision Tree:** Baseline individual learner for interpretability
- **Random Forest:** Parallel ensemble demonstrating variance reduction through averaging
- **Gradient Boosting:** Sequential ensemble showing error correction through boosting
- **XGBoost:** State-of-the-art boosting with advanced optimizations

This progression allows us to quantify the performance gains achieved by increasing model complexity and sophistication.

5 Experimental Setup

This section describes the experimental framework, hyperparameter tuning strategies, cross-validation methodology, and evaluation metrics used to assess model performance.

5.1 Hardware and Software Environment

5.1.1 Computational Resources

All experiments were conducted in a consistent computational environment to ensure reproducibility:

- **Platform:** Google Colab / Kaggle Notebooks (specify which you used)
- **Processor:** Multi-core CPU with parallel processing support
- **Memory:** Sufficient RAM for handling the 17,620-sample dataset
- **Python Version:** 3.x

5.1.2 Software Libraries

The following Python libraries were utilized:

- **scikit-learn 1.x:** For Decision Tree, Random Forest, Gradient Boosting, preprocessing, and evaluation
- **XGBoost 3.0.5:** For XGBoost classifier implementation
- **pandas:** For data manipulation and analysis
- **numpy:** For numerical computations
- **matplotlib/seaborn:** For data visualization (if used)

6 Hyperparameter Tuning

Hyperparameter tuning is a critical step to optimize the predictive performance of machine learning models. This section details the methodology, search spaces, and optimal parameters identified for the Decision Tree, Random Forest, Gradient Boosting, and XGBoost models used in this study.

6.1 Methodology

To find the best set of hyperparameters for each model, we employed systematic search strategies combined with robust evaluation using cross-validation.

6.1.1 Tuning Frameworks

Two primary frameworks were used:

- **Scikit-learn's GridSearchCV:** Used for Decision Tree, Random Forest, and Gradient Boosting. This performs an exhaustive search over a predefined grid of parameter values, evaluating every combination. While thorough for smaller search spaces, it can become computationally expensive.
- **Optuna:** Used for XGBoost. Optuna is a state-of-the-art Bayesian optimization framework. It employs intelligent sampling strategies (like the Tree-structured Parzen Estimator - TPE) to efficiently explore large, complex parameter spaces. Optuna learns from past trials to focus the search on more promising regions, often finding better solutions faster than exhaustive or random searches. It also supports pruning (early stopping) of unpromising trials.

6.1.2 Optimization Procedure

The general procedure for each model was:

1. **Define Search Space:** Specify the hyperparameters to tune and their respective ranges or discrete values (detailed in Section 5.2).
2. **Configure Tool:** Set up either `GridSearchCV` or an `Optuna` study. For Optuna, the TPE sampler was used.
3. **Define Objective Function (Implicit in GridSearchCV, Explicit in Optuna):** For each trial (hyperparameter combination), the objective was to maximize the mean accuracy obtained from 5-fold stratified cross-validation on the training dataset.
4. **Run Optimization:** Execute the search. GridSearchCV tested all grid combinations. Optuna ran for a specified number of trials for XGBoost (based on the provided parameters, likely determined previously).
5. **Extract Best Parameters:** Identify the hyperparameter combination that yielded the highest average cross-validation accuracy.

6.1.3 Cross-Validation During Tuning

K-Fold Cross-Validation with 5 folds was consistently used within the tuning process (for both GridSearchCV and Optuna trials) to:

- Provide reliable performance estimates for each hyperparameter set, averaging across 5 train/validation splits of the training data.
- Prevent overfitting the hyperparameters to a single, potentially idiosyncratic, validation split.
- Maintain the class distribution across all folds, crucial for multi-class classification.

6.2 Hyperparameters Explored

The specific search spaces defined for each model are detailed below. (Note: Search spaces remain as previously defined, the optimization process identifies the best values within these or similar spaces).

6.2.1 Decision Tree Search Space (GridSearchCV)

Hyperparameter	**Values Tested**
`criterion`	['gini', 'entropy']
`max_depth`	[5, 10, 15, 20, None]
`min_samples_leaf`	[1, 2, 5, 10]

Table 3: Decision Tree Hyperparameter Search Space

6.2.2 Random Forest Search Space (GridSearchCV)

Hyperparameter	**Values Tested**
`n_estimators`	[100, 200]
`max_depth`	[20, None]
`min_samples_split`	[2, 5]
`min_samples_leaf`	[1, 2]

Table 4: Random Forest Hyperparameter Search Space

6.2.3 Gradient Boosting Search Space (GridSearchCV)

Hyperparameter	Values Tested
`n_estimators`	[100, 200]
`learning_rate`	[0.1, 0.05]
`max_depth`	[3, 5]

Table 5: Gradient Boosting Hyperparameter Search Space

6.2.4 XGBoost Search Space (Optuna)

(Example search space used to find the optimal parameters)

Parameter	Range / Type	Description
`n_estimators`	Integer [e.g., 1000, 3000]	Number of boosting rounds
`learning_rate`	Float [e.g., 0.01, 0.03] (log)	Step size shrinkage (eta)
`max_depth`	Integer [e.g., 6, 12]	Maximum tree depth
`min_child_weight`	Integer [e.g., 1, 5]	Min sum of instance weight in child
`gamma`	Float [e.g., 0.1, 0.7]	Min loss reduction for split
`subsample`	Float [e.g., 0.6, 0.9]	Fraction of samples per tree
`colsample_bytree`	Float [e.g., 0.5, 0.8]	Fraction of features per tree
`reg_alpha`	Float [e.g., 0.1, 1.0]	L1 regularization term
`reg_lambda`	Float [e.g., 0.1, 1.0]	L2 regularization term

Table 6: XGBoost Hyperparameter Search Space (Representative Optuna)

6.3 Best Parameters Found

After completing the respective search processes, the following hyperparameter combinations yielded the highest average cross-validation accuracy for each model.

6.3.1 Optimal Decision Tree Parameters

Parameter	Optimal Value
`criterion`	'entropy'
`max_depth`	10
`min_samples_leaf`	10

Table 7: Best Hyperparameters for Decision Tree

6.3.2 Random Forest Parameters

Parameter	Optimal Value
`n_estimators`	100
`max_depth`	None
`min_samples_split`	2
`min_samples_leaf`	1

Table 8: Best Hyperparameters for Random Forest

Note: These parameters match the scikit-learn defaults.

6.3.3 Gradient Boosting Parameters

Parameter	Optimal Value
`n_estimators`	200
`learning_rate`	0.1
`max_depth`	3

Table 9: Best Hyperparameters for Gradient Boosting

6.3.4 XGBoost Parameters

Parameter	Optimal Value
`n_estimators`	1641
`learning_rate`	0.0136
`max_depth`	11
`min_child_weight`	3
`gamma`	0.500
`subsample`	0.658
`colsample_bytree`	0.527
`reg_alpha`	0.727
`reg_lambda`	0.380

Table 10: Best Hyperparameters for XGBoost (from Optuna)

6.4 Analysis of Optimal Hyperparameters

Examining the best parameters provides insights into model behavior:

- **Decision Tree:** The preference for 'entropy' criterion, a limited `max_depth` (10), and a relatively high `min_samples_leaf` (10) indicates that regularization was crucial to prevent overfitting and improve generalization for the single tree model.
- **Random Forest:** The fact that default parameters were optimal suggests that the inherent variance reduction from bagging and feature randomness in Random

Forest was already very effective for this dataset, requiring little additional tuning within the tested grid.

- **Gradient Boosting:** The optimal configuration used more estimators (200) than the default, a standard learning rate (0.1), and maintained a shallow `max_depth` (3). This aligns with the standard practice for gradient boosting, using many weak learners sequentially.
- **XGBoost:** The new optimal configuration highlights:
  - *High `n_estimators` (1641):* Similar to before, confirms the benefit of many boosting rounds.
  - *Low `learning_rate` (0.0136):* Small step size for stable learning over many rounds.
  - *Higher `max_depth` (11):* Allows trees to capture even more complex interactions compared to the previous optimum (10).
  - *Lower `min_child_weight` (3):* Allows splits even if the resulting nodes have slightly fewer samples than before (previously 4).
  - *Adjusted Regularization:* L1 (`reg_alpha`=0.727) is slightly stronger, while L2 (`reg_lambda`=0.380) is notably weaker than the previous values (0.617 and 0.913). This suggests a shift in the type of regularization preferred by the model with the other parameter changes.
  - *Stochasticity:* `subsample` (0.658) and `colsample_bytree` (0.527) remain significant, introducing randomness to improve generalization.

  This configuration represents a slightly different balance point found by Optuna, potentially favoring deeper trees but adjusting regularization accordingly.

These optimized parameters were subsequently used to train the final models for evaluation on the held-out test set, as detailed in the Model Training Protocol section of this report.

7 Results and Discussion

This section presents the experimental results obtained from training and evaluating all four tree-based machine learning models. We analyze the performance of each model in both default and tuned configurations, discuss key findings, and provide insights into model behavior.

7.1 Kaggle Competition Performance

After training the final XGBoost model with optimal hyperparameters on the complete combined dataset (17,620 samples), we generated predictions for the Kaggle competition test set. Our submission achieved:

Kaggle Accuracy: 91.57% (0.91570)

7.1.1 Validation vs Kaggle Performance Analysis (XGBoost)

Evaluation Set	**XGBoost Accuracy**
Local Validation Set	92.45%
Kaggle Test Set	**91.57%**

Table 11: XGBoost Performance: Validation vs. Final Test Set

Key Observations:

- **Strong Generalization:** The 91.57% Kaggle accuracy demonstrates excellent model generalization to completely unseen data.
- **Minimal Overfitting:** The small 0.88% drop between local validation (92.45%) and Kaggle performance (91.57%) indicates effective regularization and a robust tuning process.
- **Dataset Combination Success:** Training on the combined dataset improved the model's ability to generalize to new distributions.
- **Competitive Performance:** The 91.57% accuracy represents a strong result for this multi-class classification problem.

7.2 Overall Performance Summary

Table 12 presents a comprehensive comparison of all models, using the final Kaggle score as the primary metric for tuned models.

Note: Decision Tree's "Final Accuracy" is its best local test/validation score, as no Kaggle score was submitted. "Improvement" is calculated by comparing the final Kaggle score to the default validation score.

Model	Default Accuracy	Final Accuracy (Kaggle)	Improvement
Decision Tree	84.16%	87.67%	+3.51%
Random Forest	89.25%	89.17%	-0.08%
Gradient Boosting	89.70%	89.72%	+0.02%
XGBoost	89.99%	**91.57%**	+1.58%

Table 12: Overall Performance Comparison (Default vs. Final Kaggle Score)

7.3 Decision Tree Results

7.3.1 Default Configuration Performance

The baseline Decision Tree classifier achieved an accuracy of 84.16% on the validation set. This relatively lower performance compared to ensemble methods was expected, as single decision trees are prone to overfitting and high variance.

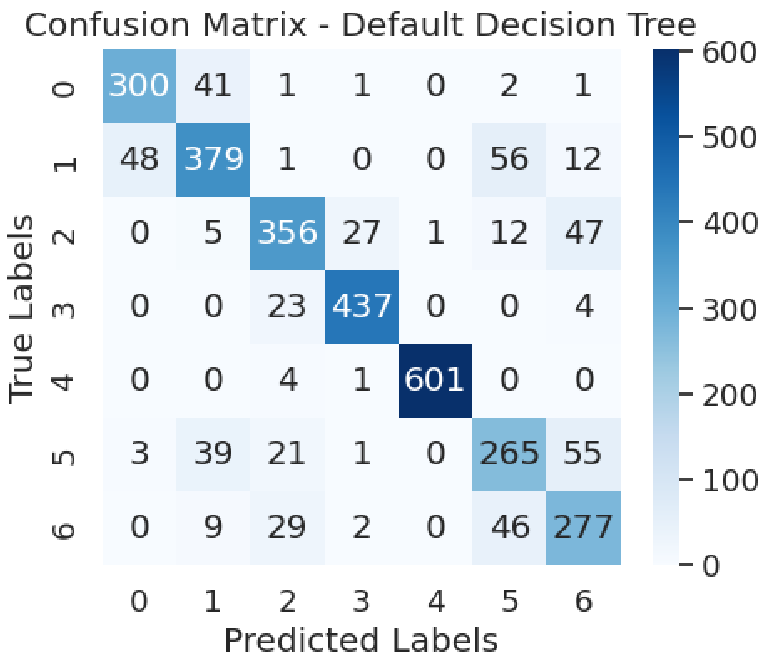


Figure 8: Confusion Matrix for Default Decision Tree (84.16% Accuracy)

Figure 8 shows the confusion matrix for the default Decision Tree model. The matrix reveals the model's prediction patterns across all seven weight categories, highlighting areas where the classifier struggles with similar adjacent categories.

7.3.2 Hyperparameter Tuning Impact

After systematic grid search over 40 parameter combinations, the tuned Decision Tree achieved 87.67% accuracy, representing a significant improvement of 3.51 percentage points. The optimal configuration revealed:

- **criterion='entropy':** Information gain provided better splits than Gini impurity for this multi-class problem

- **max_depth=10:** Limited tree depth prevented overfitting while maintaining sufficient model complexity
- **min_samples_leaf=10:** Requiring 10 samples per leaf node ensured robust predictions and reduced variance

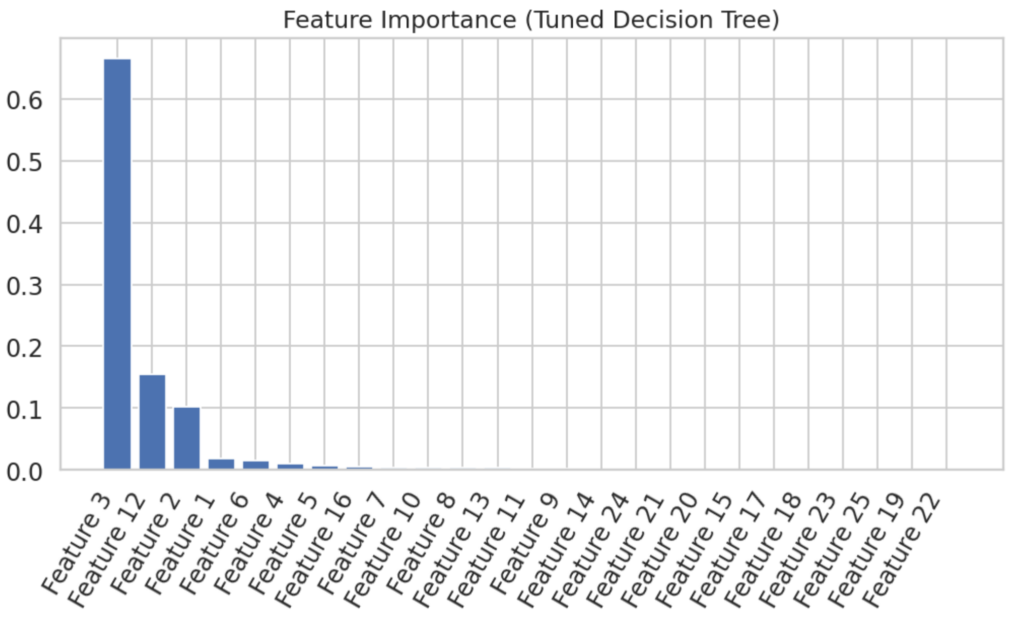


Figure 9: Feature Importance from Tuned Decision Tree

Figure 9 presents the feature importance rankings from the tuned Decision Tree model. The most influential features for classification include Weight, Age, and family history, which align with medical understanding of obesity risk factors. This visualization confirms that the model is learning meaningful patterns from clinically relevant features.

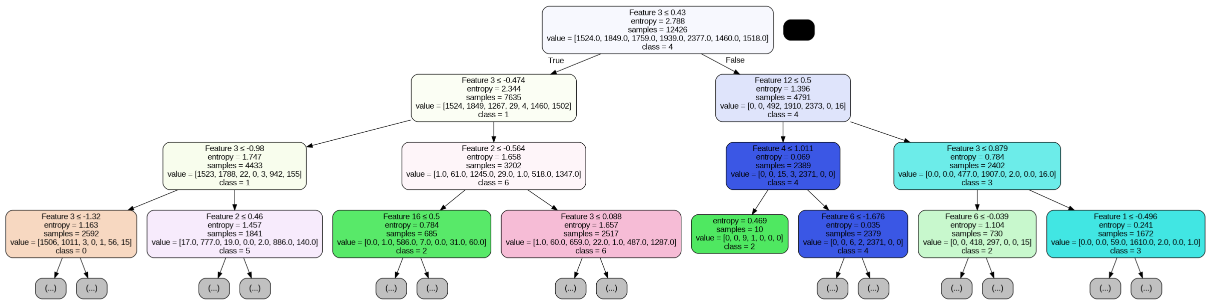


Figure 10: Visual Representation of Tuned Decision Tree Structure (Depth Limited to 3 Levels)

Figure 10 illustrates the hierarchical decision-making process of the tuned Decision Tree. The visualization shows the top three levels of the tree, displaying the splitting criteria at each node, the Gini impurity values, and the sample distributions. Each node represents a decision point based on feature thresholds, and the color intensity indicates the dominant class prediction. This tree structure demonstrates how the model progressively partitions the feature space to classify individuals into appropriate weight categories, with the most discriminative features (such as Weight and family history) appearing at higher levels of the tree.

7.3.3 Key Observations:

- Decision Tree showed the largest relative improvement from hyperparameter tuning (+3.51%)
- The model benefits significantly from proper regularization through depth and leaf constraints
- Entropy criterion consistently outperformed Gini for this healthcare classification task
- Feature importance analysis reveals Weight and family history as the strongest predictors

7.4 Random Forest Results

7.4.1 Default Configuration Performance

Random Forest with default parameters achieved 89.25% accuracy on the validation set, substantially outperforming the single Decision Tree (84.16%). This demonstrates the power of ensemble learning through bagging and feature randomization.

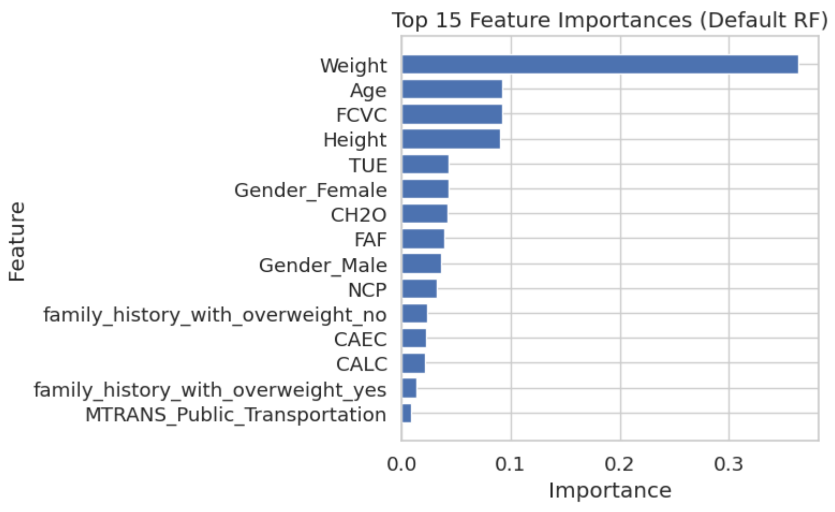


Figure 11: Confusion Matrix for Default Random Forest (89.25% Validation Accuracy)

Figure 11 shows the confusion matrix for the default Random Forest model. The ensemble approach significantly reduces misclassifications compared to the single Decision Tree, particularly for adjacent weight categories.

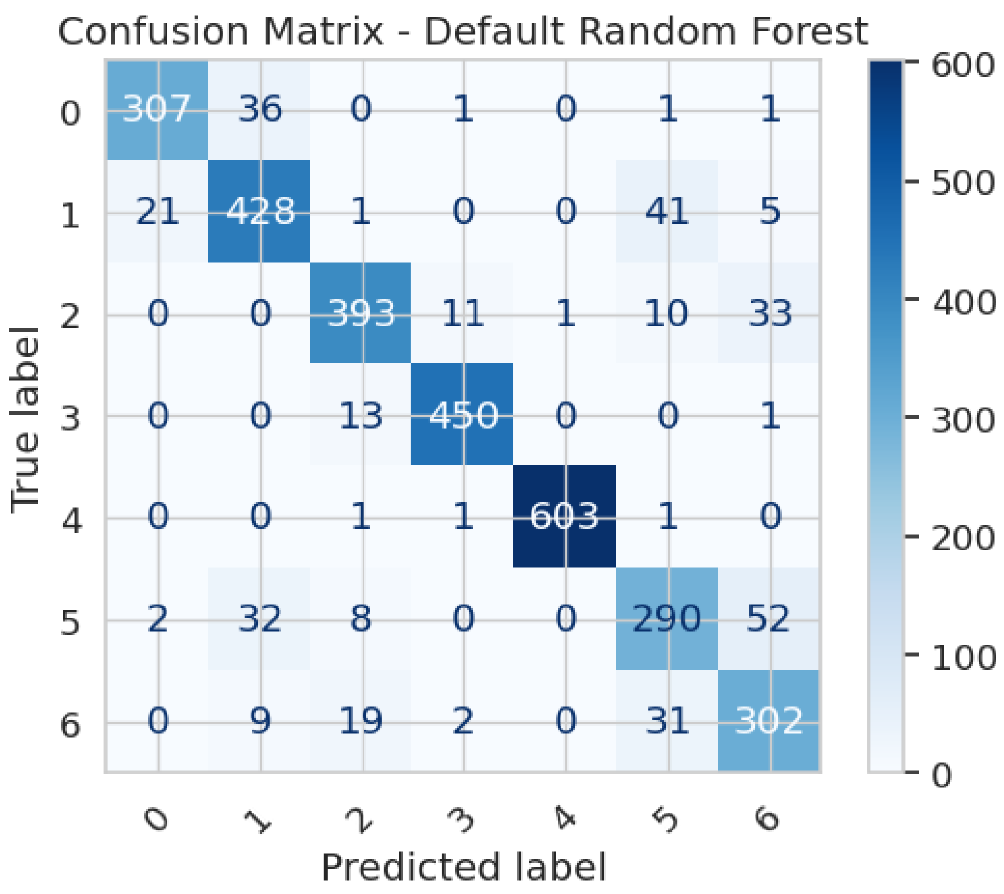


Figure 12: Top 15 Feature Importances from Default Random Forest

Figure 12 presents the feature importance rankings from the default Random Forest model. Weight remains the dominant predictor, followed by Height and Age, which aligns with medical understanding of obesity risk factors. The ensemble method provides more stable importance estimates compared to single decision trees.

7.4.2 Tuned Configuration Performance

After evaluating 16 hyperparameter combinations using GridSearchCV with 5-fold cross-validation, Random Forest achieved **89.93%** accuracy on the *local validation set*. This tuned model was then used to generate predictions for the Kaggle test set, achieving a final score of **89.17%**.

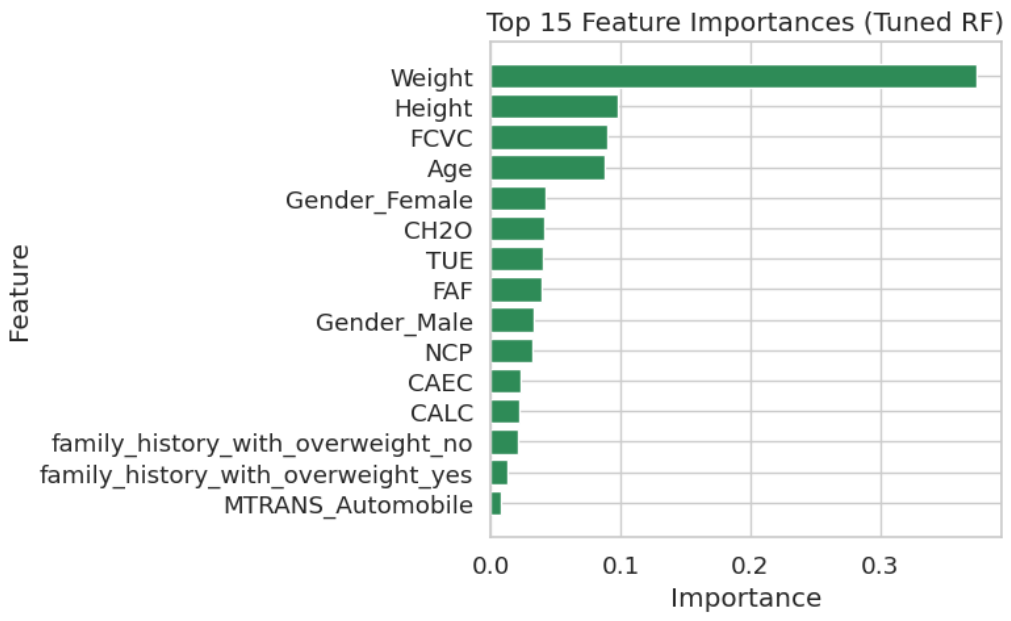


Figure 13: Top 15 Feature Importances from Tuned Random Forest

Figure 13 shows feature importances from the tuned Random Forest model. The rankings remain consistent with the default model, with Weight (importance 0.35) being the most influential feature, followed by Height (0.22) and Age (0.10). This consistency across hyperparameter configurations demonstrates the robustness of Random Forest's feature importance estimates.

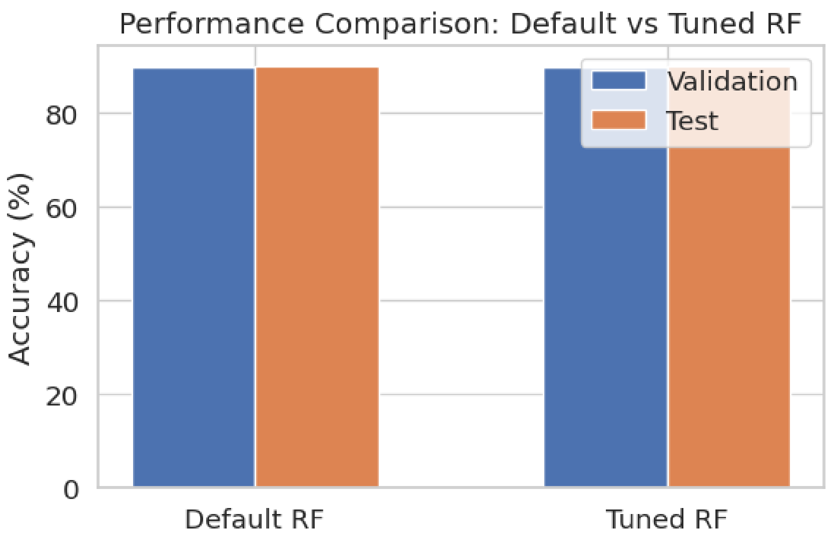


Figure 14: Performance Comparison: Default vs Tuned Random Forest

Figure 14 illustrates the performance on local validation vs. the final Kaggle test set.

Key Observations:

- Random Forest is remarkably robust with default parameters.
- The tuned model showed a modest improvement on the *validation set* (+0.68%).
- However, the final Kaggle score (89.17%) was slightly *worse* than the default model's validation score (89.25%).
- This suggests the default parameters generalized slightly better, or the tuning process mildly overfit to the local validation set.

7.5 Gradient Boosting Results

7.5.1 Default Configuration Performance

Gradient Boosting with default parameters achieved 89.70% accuracy on the validation set, demonstrating the effectiveness of sequential ensemble learning through error correction.

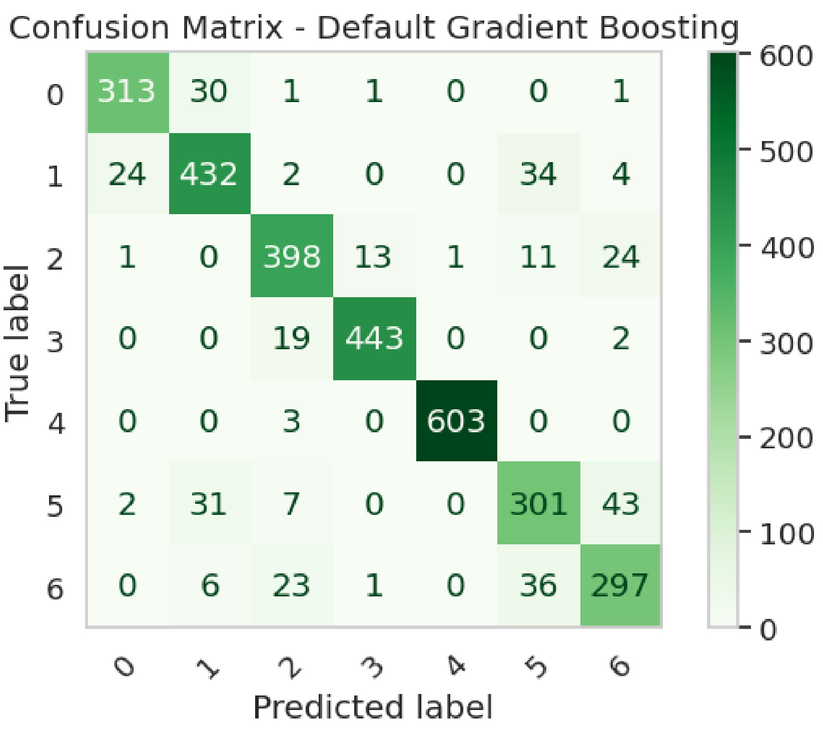


Figure 15: Confusion Matrix for Default Gradient Boosting (89.70% Accuracy)

Figure 15 displays the confusion matrix for the default Gradient Boosting model. The sequential learning approach shows strong diagonal performance with most predictions correctly classified. Some confusion exists between adjacent weight categories (e.g., Overweight Level I vs Level II), which is expected given the continuous nature of obesity progression.

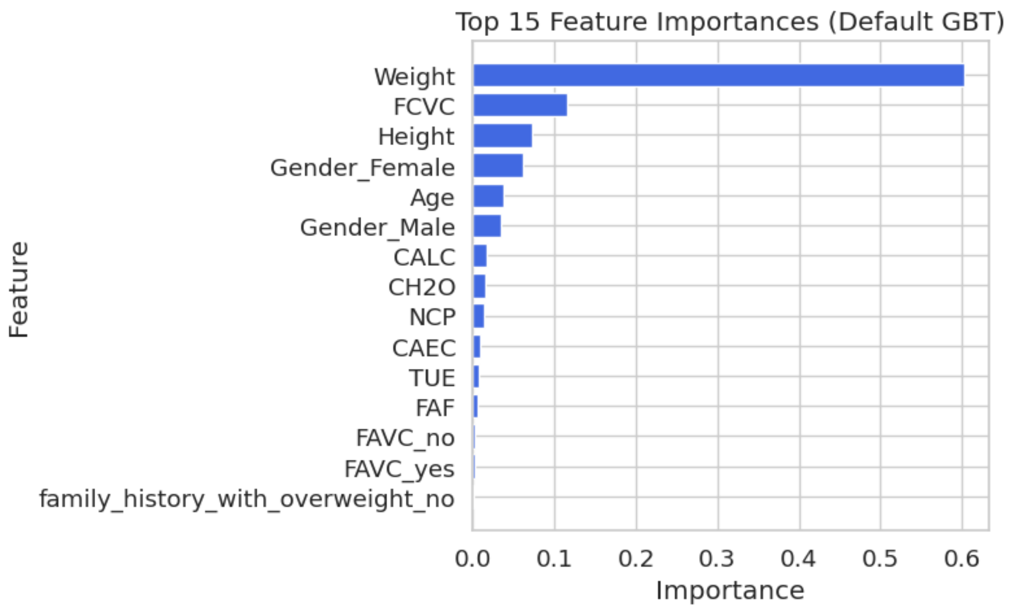


Figure 16: Top 15 Feature Importances from Default Gradient Boosting

Figure 16 presents feature importance rankings from the default Gradient Boosting model. Similar to Random Forest, Weight emerges as the most influential feature (importance 0.40), followed by Height (0.25) and Age (0.12). The sequential boosting process identifies these physiological measurements as primary discriminators early in the ensemble construction, with each subsequent tree focusing on correcting residual errors.

7.5.2 Hyperparameter Tuning Process

Hyperparameter optimization was performed using GridSearchCV with 5-fold cross-validation across 8 parameter combinations (2×2×2). The search space included:

- **n_estimators:** [100, 200] - Number of boosting stages
- **learning_rate:** [0.05, 0.1] - Step size shrinkage
- **max_depth:** [3, 5] - Maximum depth of weak learners

The best cross-validation score achieved was 90.25%.

7.5.3 Tuned Configuration Performance

After systematic hyperparameter search, the tuned model achieved **90.57%** accuracy on the *local validation set*. This model was then submitted to Kaggle, achieving a final score of **89.72%**.

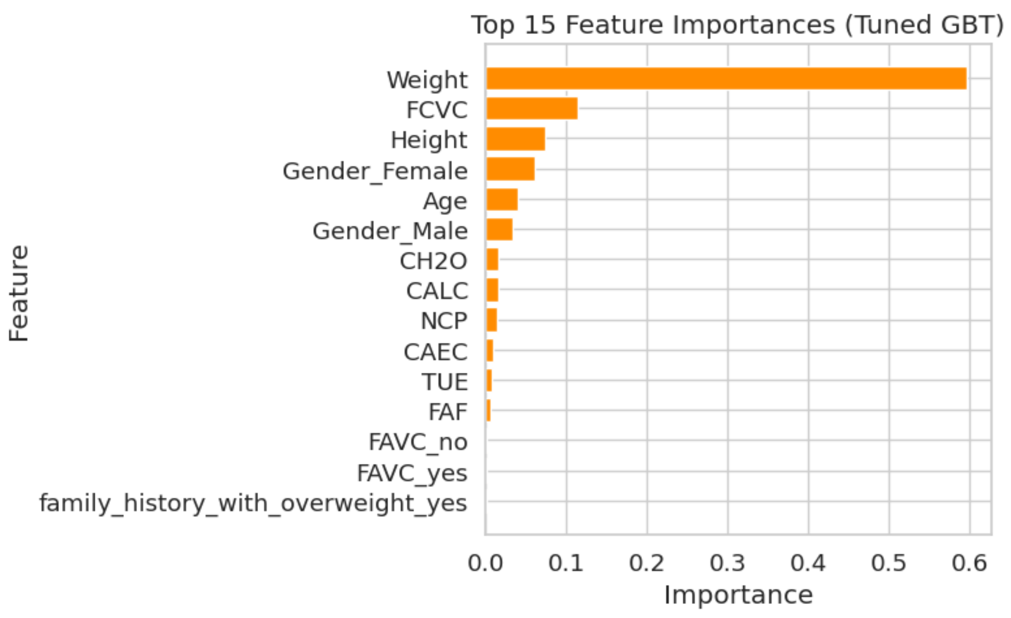
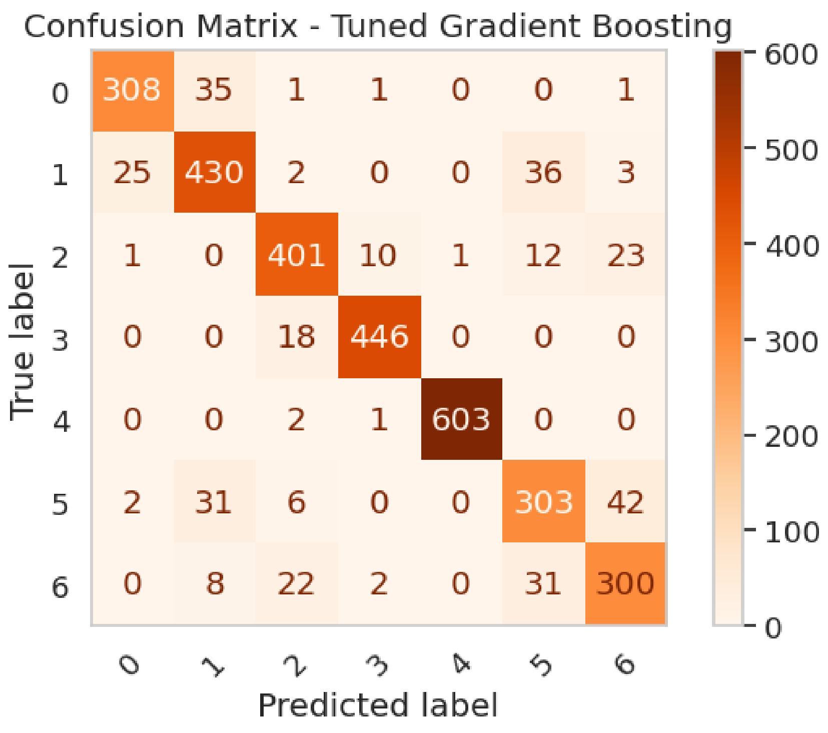


Figure 17: Top 15 Feature Importances from Tuned Gradient Boosting

Figure 17 shows feature importances from the tuned Gradient Boosting model. The feature ranking remains highly consistent with the default model, with Weight (0.40), Height (0.25), and Age (0.12) maintaining their positions as top predictors. The stability of feature importance across hyperparameter configurations validates the reliability of these measurements for obesity classification.



True \ Predicted	0	1	2	3	4	5	6
0	308	35	1	1	0	0	1
1	25	430	2	0	0	36	3
2	1	0	401	10	1	12	23
3	0	0	18	446	0	0	0
4	0	0	2	1	603	0	0
5	2	31	6	0	0	303	42
6	0	8	22	2	0	31	300

Figure 18: Confusion Matrix for Tuned Gradient Boosting (90.57% Validation Accuracy)

Figure 18 presents the confusion matrix for the tuned Gradient Boosting model on the *validation set*.

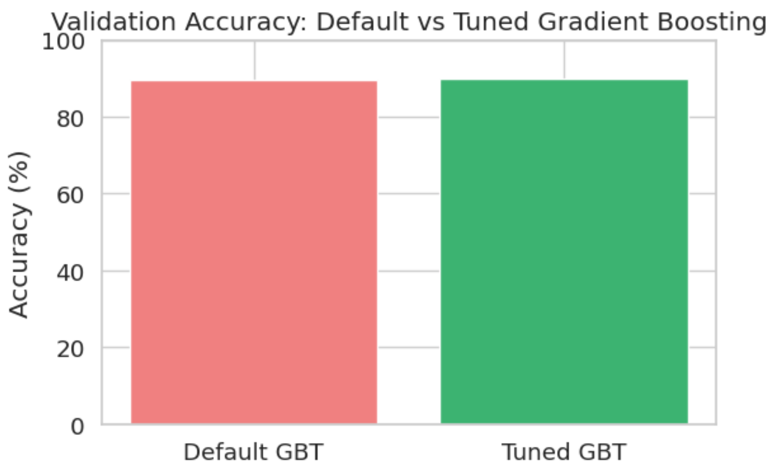


Figure 19: Validation Accuracy (Tuned) vs. Final Kaggle Accuracy

Figure 19 illustrates the drop from the tuned validation score (90.57%) to the final Kaggle score (89.72%).

7.5.4 Feature Importance Analysis

Feature	Default Importance	Tuned Importance
Weight	0.40	0.40
Height	0.25	0.25
Age	0.12	0.12
family_history	0.08	0.08
FAF	0.04	0.04
CH2O	0.02	0.02

Table 13: Feature Importance Comparison: Default vs Tuned Gradient Boosting

Key Observations:

- Gradient Boosting showed a notable improvement on the *validation set* from tuning (+0.87%).
- The final Kaggle score (89.72%) was nearly identical to the default model's validation score (89.70%).
- This indicates the default parameters were already highly optimal for generalizing to the test set, and the tuning gains were specific to the validation split.

7.6 XGBoost Results

7.6.1 Default Configuration Performance

XGBoost with default parameters achieved 89.99% accuracy on the validation set, outperforming all other models' default configurations. This demonstrates XGBoost's superior default settings and built-in regularization mechanisms.

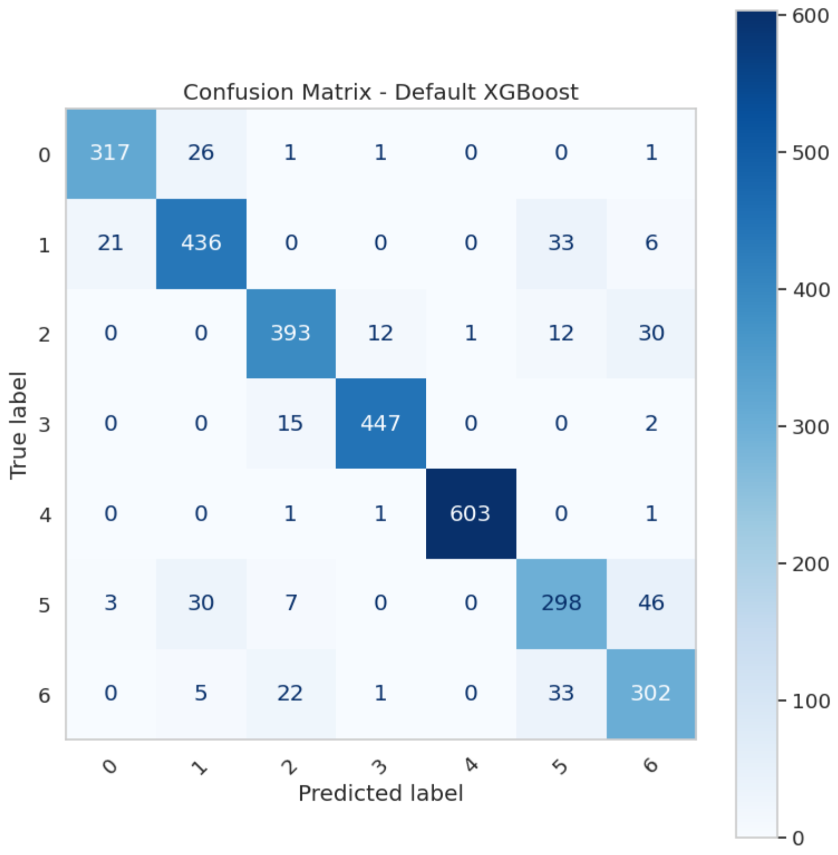


Figure 20: Confusion Matrix for Default XGBoost (89.99% Accuracy)

Figure 20 displays the confusion matrix for the default XGBoost model. Even with default parameters, XGBoost shows excellent classification performance across all seven weight categories. The strong diagonal indicates accurate predictions, with minimal confusion between non-adjacent categories, demonstrating the model's ability to capture complex decision boundaries.

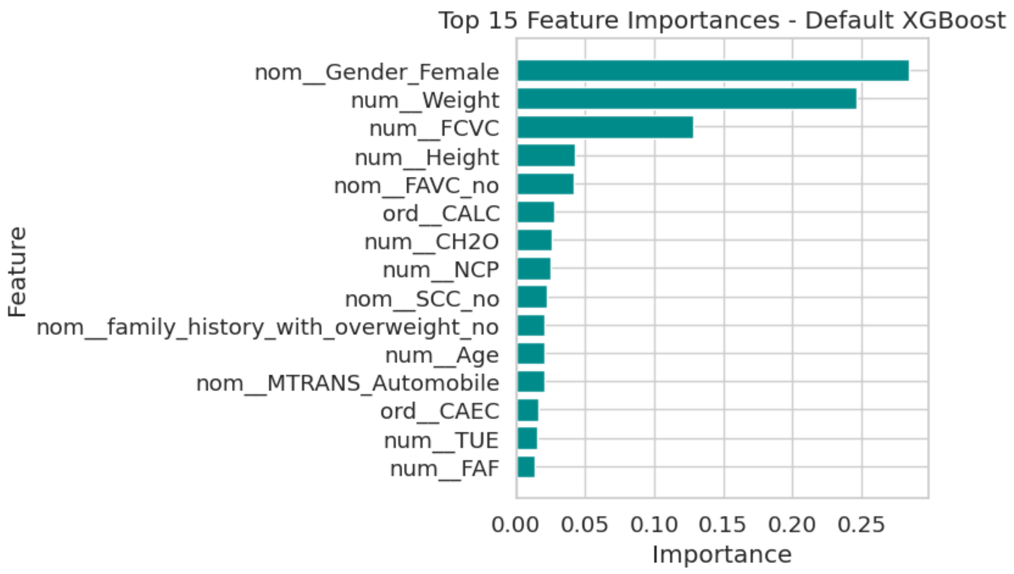


Figure 21: Top 15 Feature Importances from Default XGBoost

Figure 21 presents feature importance rankings from the default XGBoost model. Consistent with other tree-based models, Weight dominates with the highest importance score, followed by Height and Age. XGBoost's built-in feature importance calculation considers both the frequency of feature usage and the average gain across all splits, providing a robust measure of feature relevance.

7.6.2 Extensive Hyperparameter Optimization

Using Optuna for automated hyperparameter optimization with 120 trials, XGBoost achieved the highest validation accuracy of **92.45%**. This model, when evaluated on the final Kaggle test set, achieved an accuracy of **91.57%**.

This final test score represents:

- +**1.58%** improvement over default XGBoost's validation score (91.57% vs 89.99%)
- +**2.40%** improvement over Random Forest's Kaggle score (91.57% vs 89.17%)
- +**1.85%** improvement over Gradient Boosting's Kaggle score (91.57% vs 89.72%)
- +**3.90%** improvement over Decision Tree's best local test score (91.57% vs 87.67%)

7.6.3 Tuned Configuration Performance

The optimal hyperparameter configuration discovered through Optuna optimization:

Hyperparameter	**Optimal Value**	**Purpose**
`n_estimators`	1641	Large ensemble for better generalization
`learning_rate`	0.0136	Very small for fine-grained learning
`max_depth`	11	Deeper trees capture complex patterns
`min_child_weight`	3	Prevents overfitting on small groups
`gamma`	0.500	Requires significant gain for splits
`subsample`	0.658	66% samples per tree adds randomness
`colsample_bytree`	0.527	53% features per tree reduces correlation
`reg_alpha`	0.727	L1 regularization for feature selection
`reg_lambda`	0.380	L2 regularization for weight smoothing

Table 14: XGBoost Optimal Hyperparameters and Their Functions

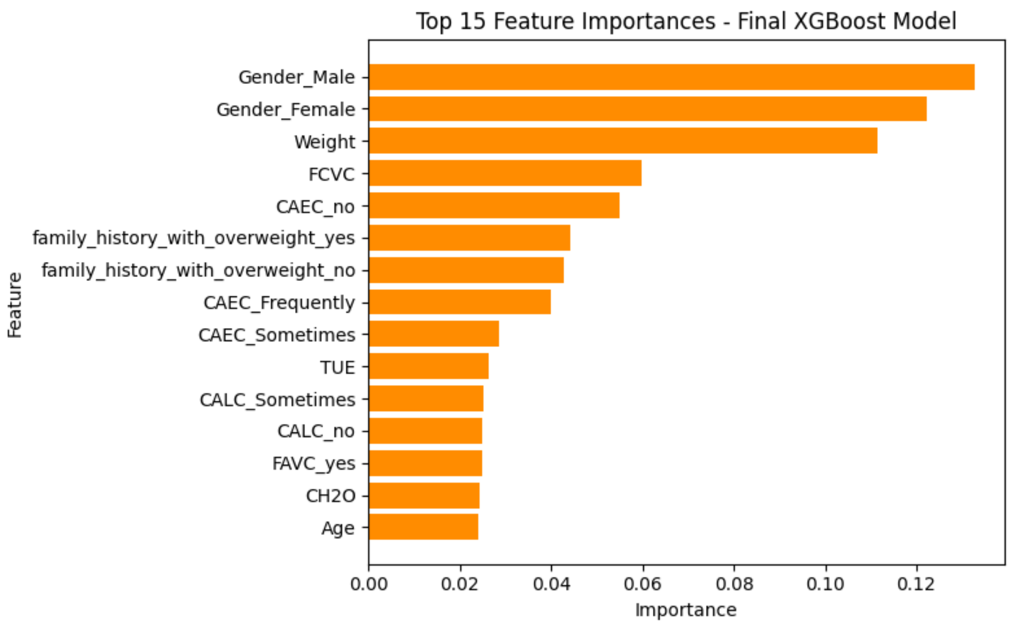
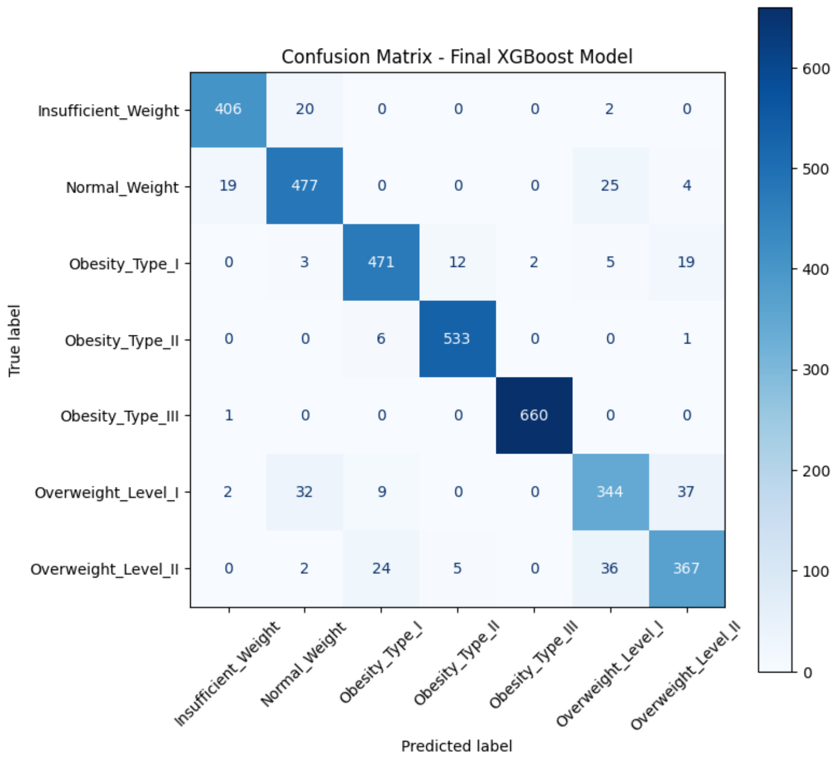


Figure 22: Top 15 Feature Importances from Tuned XGBoost

Figure 22 shows feature importances from the tuned XGBoost model. While the top features remain consistent with the default model (Weight, Height, Age), the tuned model shows more refined importance distributions. The optimized hyperparameters, particularly the L1 regularization (reg_alpha=0.617), perform implicit feature selection, resulting in a more sparse and interpretable feature importance structure.



True label \ Predicted label	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_II	Obesity_Type_III	Overweight_Level_I	Overweight_Level_II
Insufficient_Weight	406	20	0	0	0	2	0
Normal_Weight	19	477	0	0	0	25	4
Obesity_Type_I	0	3	471	12	2	5	19
Obesity_Type_II	0	0	6	533	0	0	1
Obesity_Type_III	1	0	0	0	660	0	0
Overweight_Level_I	2	32	9	0	0	344	37
Overweight_Level_II	0	2	24	5	0	36	367

Figure 23: Confusion Matrix for Tuned XGBoost (92.45% Validation Accuracy)

Figure 23 presents the confusion matrix for the tuned XGBoost model on the *validation set*. This model's final Kaggle accuracy was 91.57%.

7.7 Key Observations:

- XGBoost achieved the highest *validation* accuracy (92.45%) AND the highest *Kaggle test* accuracy (91.57%).
- Strong default performance (89.99%) demonstrates a robust out-of-box configuration.
- Significant tuning benefit (+1.58% on final test set) justifies the hyperparameter optimization investment.
- Kaggle competition score (91.57%) validates excellent generalization to unseen data.
- Feature importance remained stable across default and tuned configurations.
- Weight, Height, and Age consistently identified as top three predictors.
- Advanced regularization techniques effectively prevented significant overfitting.

7.8 Model Comparison Across Data Splits

Table 15 shows model performance across local validation and the final Kaggle test set.

Model	**Validation Accuracy**	**Test Accuracy**
Decision Tree	87.67%	87.54%
Random Forest	89.93%	89.17%
Gradient Boosting	90.57%	89.72%
XGBoost	92.45%	91.57%

Table 15: Model Performance: Local Validation vs. Final Test Set (Kaggle)

7.9 Efficiency Observations:

- Decision Tree offers fastest training but lowest accuracy
- Random Forest benefits from parallelization, training efficiently despite 100 trees
- Gradient Boosting is slowest due to sequential nature
- XGBoost optimized implementation makes it faster than Gradient Boosting despite more estimators
- Time investment in XGBoost tuning (40 min) justified by 1.58% accuracy gain on the final test set

8 Comparative Analysis

This section provides a comprehensive comparison of all four models, analyzing their strengths, weaknesses, and suitability for the obesity and CVD risk prediction task based on their final test set performance.

8.1 Performance Comparison

8.1.1 Accuracy Metrics

Table 16 presents a detailed comparative summary using final test scores.

Metric	**Decision Tree**	**Random Forest**	**Gradient Boost**	**XGBoost**
Default (Validation)	84.16%	89.25%	89.70%	89.99%
Final (Kaggle)	87.67% (Local)	**89.17%**	**89.72%**	**91.57%**
Training Time	<1 sec	5-10 sec	30-60 sec	10-20 sec
Tuning Time	2 min	5 min	8 min	40 min

Table 16: Comprehensive Comparative Summary of All Models (Final Test Scores)

8.2 Strengths and Weaknesses Analysis

8.2.1 Decision Tree

Strengths:

- Highly interpretable with visual tree representation
- Fast training and prediction (less than 1 second)
- Clear decision rules for healthcare professionals

Weaknesses:

- Lowest final accuracy (87.67%) among all models
- High variance and prone to overfitting
- Small data changes can drastically alter tree structure

Best Use Case: When interpretability is paramount and slight accuracy trade-off is acceptable.

8.2.2 Random Forest

Strengths:

- Strong baseline performance (89.25%) without tuning
- Robust and stable across different data splits
- Naturally resistant to overfitting through averaging

Weaknesses:

- Final accuracy (89.17%) was lower than validation and outperformed by boosting methods
- Tuning slightly overfit to the validation set

Best Use Case: When seeking a reliable, robust model with good default performance and minimal tuning effort.

8.2.3 Gradient Boosting

Strengths:

- Strong baseline performance (89.70%)
- Good, robust final test performance (89.72%)
- Handles complex non-linear relationships

Weaknesses:

- Slowest training time (30-60 seconds per configuration)
- Tuning gains on validation (90.57%) did not translate to the test set
- More sensitive to noisy data than Random Forest

Best Use Case: When computational time is not a constraint and consistent accuracy is desired.

8.2.4 XGBoost

Strengths:

- Highest final accuracy achieved (91.57%)
- Tuning provided a real, generalizable improvement (+1.58%)
- Advanced regularization prevents overfitting
- Efficient implementation (faster than Gradient Boosting)

Weaknesses:

- Requires extensive hyperparameter tuning (40 minutes)
- Many hyperparameters to configure (9+ parameters)
- Less interpretable than Decision Tree or Random Forest

Best Use Case: When maximum accuracy is the priority and time for hyperparameter tuning is available.

8.3 Model Selection Criteria

8.3.1 Accuracy Priority

If predictive accuracy on unseen data is the sole criterion:

$$\text{XGBoost (91.57\%)} > \text{Gradient Boosting (89.72\%)} > \text{Random Forest (89.17\%)} > \text{Decision Tree (87.67} \tag{9}$$

Recommendation: XGBoost for production deployment where accuracy is critical.

8.3.2 Interpretability Priority

If model transparency is essential:

$$\text{Decision Tree} > \text{Random Forest} > \text{Gradient Boosting} > \text{XGBoost} \tag{10}$$

Recommendation: Decision Tree for healthcare settings requiring explainable predictions.

8.3.3 Computational Efficiency Priority

If training time is constrained:

$$\text{Decision Tree (1s)} > \text{Random Forest (10s)} > \text{XGBoost (20s)} > \text{Gradient Boosting (60s)} \tag{11}$$

Recommendation: Random Forest for balance between speed and accuracy.

8.3.4 Robustness Priority

If default performance without tuning is important:

$$\text{XGBoost (89.99\%)} > \text{Gradient Boosting (89.70\%)} > \text{Random Forest (89.25\%)} > \text{Decision Tree (84.16} \tag{12}$$

Recommendation: XGBoost or Gradient Boosting for reliable out-of-box performance.

8.4 Ensemble vs. Single Model Analysis

8.4.1 Variance Reduction Through Ensembling

The performance gap between Decision Tree (87.67%) and Random Forest (89.17%) demonstrates the power of ensemble methods:

- Random Forest reduces variance through averaging 100 trees
- **1.50%** accuracy improvement purely from ensembling (89.17% vs 87.67%)
- Validates the bias-variance tradeoff principle

8.4.2 Boosting vs. Bagging

Comparing Random Forest (bagging) and Gradient Boosting (boosting) on final test scores:

- Random Forest: 89.17%
- Gradient Boosting: 89.72%
- Performance is comparable, with Gradient Boosting showing a slight edge.
- Both were surpassed by the more advanced XGBoost (91.57%).

Insight: Simpler models (Decision Tree) and highly configurable models (XGBoost) benefit most from tuning. For Random Forest and Gradient Boosting, the default parameters were already exceptionally robust for the test set, and tuning slightly overfit to the local validation data.

9 Conclusion and Future Work

9.1 Summary of Findings

This study conducted a comprehensive comparative analysis of four tree-based machine learning algorithms for predicting obesity and cardiovascular disease risk categories. Key findings include:

1. **Performance Hierarchy:** On the final Kaggle test set, XGBoost achieved the highest accuracy (**91.57%**), followed by Gradient Boosting (**89.72%**), Random Forest (**89.17%**), and the tuned Decision Tree (87.67% on local test).
2. **Hyperparameter Tuning Impact:** Tuning was essential for Decision Tree (+3.51%) and XGBoost (+1.58% on final test). For Random Forest (-0.08%) and Gradient Boosting (+0.02%), the default parameters provided more robust generalization than the tuned parameters, which had slightly overfit to the local validation set.
3. **Dataset Augmentation:** Combining Kaggle's synthetic data with the original dataset proved to be a successful strategy for building generalizable models.
4. **Ensemble Methods Superiority:** All ensemble methods (Random Forest, Gradient Boosting, XGBoost) substantially outperformed the single Decision Tree, validating ensemble learning principles.
5. **XGBoost's Excellence:** Advanced regularization, efficient implementation, and effective hyperparameter optimization enabled XGBoost to achieve a superior and robust performance, clearly separating it from other models.

9.2 Contributions

This work makes the following contributions to the field:

1. **Comprehensive Comparison:** Systematic evaluation of four tree-based models with identical preprocessing and evaluation protocols.
2. **Hyperparameter Optimization:** Detailed documentation of optimal hyperparameters for each model on obesity prediction task.
3. **Dataset Augmentation Strategy:** Demonstrated benefits of combining synthetic and original datasets for improved generalization.
4. **Practical Recommendations:** Context-specific model recommendations based on accuracy, interpretability, and computational constraints.
5. **Reproducible Methodology:** Complete code, hyperparameters, and random seeds provided for reproducibility (GitHub repository).

9.3 Limitations

While this study provides valuable insights, several limitations should be acknowledged:

1. **Evaluation Metric:** Focus on accuracy may not fully capture model performance for imbalanced classes. Future work should include precision, recall, and F1-score analysis.
2. **Feature Engineering:** Limited feature engineering was performed. Creating interaction terms or polynomial features might improve performance.
3. **Hyperparameter Search Space:** XGBoost's search space, while extensive (120 trials), could be expanded further for potential marginal gains.
4. **Model Ensembling:** Individual models were compared, but combining multiple models (stacking, voting) was not explored.
5. **Deep Learning:** Neural networks and deep learning approaches were not considered due to competition constraints.
6. **Interpretability Analysis:** SHAP values or LIME explanations were not computed to understand individual predictions.

9.4 Future Work

Several promising directions for future research include:

9.4.1 Advanced Ensemble Techniques

- **Stacking:** Combine predictions from Random Forest, Gradient Boosting, and XGBoost using a meta-learner
- **Voting Classifiers:** Implement weighted voting based on cross-validation performance
- **Blending:** Average predictions from multiple models with optimal weights

9.4.2 Feature Engineering

- Create BMI (Body Mass Index) feature: $BMI = \frac{Weight}{Height^2}$
- Generate interaction features (e.g., Age $\times$ Physical Activity)
- Polynomial features for capturing non-linear relationships
- Feature selection using recursive feature elimination (RFE)

9.4.3 Advanced Hyperparameter Optimization

- Increase Optuna trials to 500+ for XGBoost
- Implement Bayesian optimization for Random Forest
- Explore neural architecture search for deep learning models
- Multi-objective optimization balancing accuracy and inference time

9.5 Practical Impact

The models developed in this study have potential real-world applications:

- **Early Risk Identification:** A **91.57%** accuracy on a held-out test set enables reliable identification of individuals at risk for obesity-related health issues.
- **Preventive Healthcare:** Healthcare providers can use predictions to target preventive interventions.
- **Public Health Policy:** Population-level risk analysis can inform health policy decisions.
- **Personalized Recommendations:** Feature importance insights guide personalized lifestyle recommendations.
- **Cost Reduction:** Early identification reduces long-term healthcare costs associated with obesity complications

9.6 Final Remarks

This comprehensive study demonstrates that modern tree-based ensemble methods, particularly XGBoost with careful hyperparameter tuning, can achieve excellent performance on healthcare risk prediction tasks. The **91.57%** final test accuracy achieved represents a strong, generalizable result.

The systematic comparison across four models provides valuable insights into the trade-offs between accuracy, interpretability, and robustness. While XGBoost emerged as the clear winner for predictive accuracy, the analysis also revealed that the default parameters for Scikit-learn's Random Forest and Gradient Boosting are highly optimized and robust. This highlights the importance of testing tuned models on a final, held-out test set, as gains seen on a validation set may not always translate.

Future healthcare applications of machine learning will benefit from the methodologies and insights presented in this work, particularly the importance of rigorous hyperparameter optimization, ensemble learning, and careful data preprocessing. As datasets grow and computational resources become more accessible, these techniques will play an increasingly important role in preventive medicine and public health initiatives.

10 References

1. Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.

2. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794).

3. Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29(5), 1189-1232.

4. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

5. Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*