# AIT 511 Machine Learning Project on Weight category prediction

MT2025077 - Nir Shah, MT2025107 - Rushil Khatri

**Abstract**

This project aims at developing a predictive model to classify individuals into different weight categories by analyzing various behavioral factors rather than just the traditional metrics. It understands the complex relationship of non-linear daily habits like diet, physical activity etc. Our primary approach uses the technique of Extreme Gradient Boosting(XGBoost), augmented by internal bagging mechanisms, to provide highly robust and low-variance classification models. Before training, all numerical features were scaled and all categorical features were encoded to correctly represent them in the model space. This method successfully achieved a highly predictive model. It resulted in optimal classification accuracy of 91.21% on the testing dataset.

## 1 Introduction

### 1.1 Problem Definition

The core problem is to accurately predict an individual's specific *WeightCategory* based on multivariate set of non-linear behavioral features. For training the model, we also want to ensure that the model captures the subtle, high dimensional interactions between these lifestyle choices rather than simply modeling BMI.

### 1.2 Dataset Description

#### 1.2.1 Dataset Overview

Table 1: Dataset Overview

| **Property** | **Description** |
|---|---|
| Number of Samples (Rows) | 15,533 |
| Number of Features (Input Variables) | 17 |
| Target Variable | `WeightCategory` |

#### 1.2.2 Data Type Summary

Table 2: Data Type Summary

| **Type** | **Columns** |
|---|---|
| int64 | id |
| float64 | Age, Height, Weight, FCVC, NCP, CH2O, FAF, TUE |
| object | Gender, family_history_with_overweight, FAVC, CAEC, SMOKE, SCC, CALC, MTRANS, WeightCategory |

### 1.2.3 Feature Description

Table 3: Feature Description

| Column | Type | Description / Value Range |
|---|---|---|
| id | Numerical | Unique identifier (0–15,532) |
| Gender | Categorical | Male, Female |
| Age | Numerical | Age in years (14–61) |
| Height | Numerical | Height in meters (1.45–1.98) |
| Weight | Numerical | Weight in kilograms (39–165) |
| family_history_with_overweight | Categorical | Family history with overweight (Yes/No) |
| FAVC | Categorical | Frequent consumption of high-calorie food (Yes/No) |
| FCVC | Numerical | Frequency of vegetable consumption (1–3) |
| NCP | Numerical | Number of main meals per day (1–4) |
| CAEC | Categorical | Consumption of food between meals (No, Sometimes, Frequently, Always) |
| SMOKE | Categorical | Smoking habit (Yes/No) |
| CH2O | Numerical | Daily water intake (1–3 L/day) |
| SCC | Categorical | Self-calorie monitoring (Yes/No) |
| FAF | Numerical | Physical activity frequency (0–3 hours/week) |
| TUE | Numerical | Time using technology devices (0–2 hours/day) |
| CALC | Categorical | Alcohol consumption frequency (No, Sometimes, Frequently, Always) |
| MTRANS | Categorical | Mode of transportation (Walking, Bike, Motorbike, Public Transport, Automobile) |
| WeightCategory | Categorical | Target variable: Weight classification (Normal, Overweight, Obesity types) |

## 2 Data Preprocessing and Exploratory Data Ananlysis

Data preprocessing is a crucial step in preparing the dataset for modeling. It involves handling missing values, detecting and managing outliers, and applying necessary transformations to improve model performance.

## 2.1 Handling Missing Values

The dataset was first checked for missing values. A heatmap visualization (1) was used to inspect any gaps in the data.

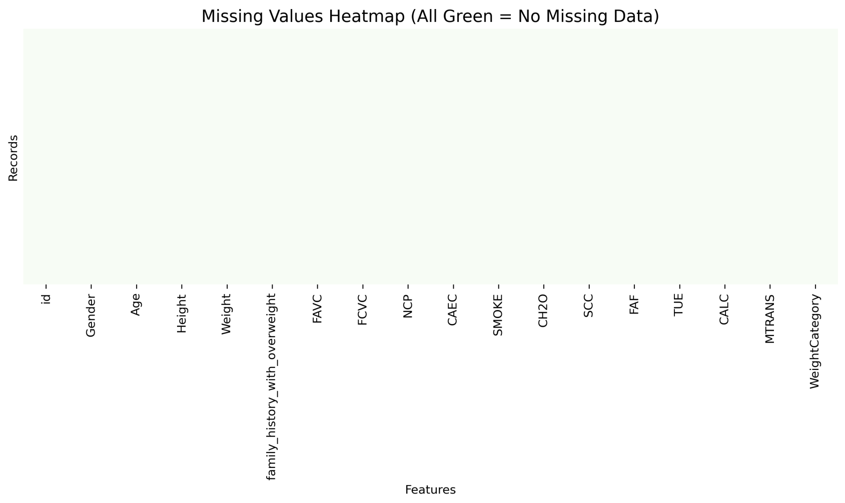


Figure 1: Heatmap of missing values in the dataset. All green indicates no missing data.

No missing values were found in the dataset, so no imputation was required.

## 2.2 Outlier Detection

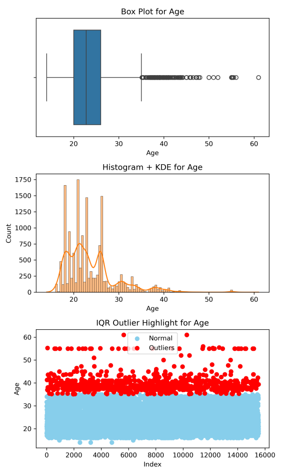
Outliers were detected separately for numerical and categorical features.

### 2.2.1 Numerical Features

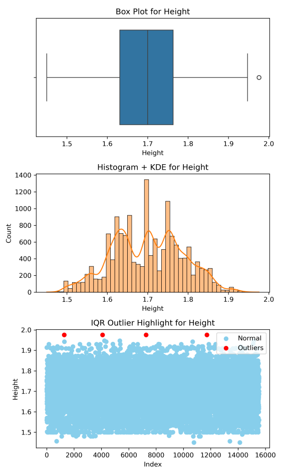
The numerical features considered were `Age`, `Height`, `Weight`, `FCVC`, `NCP`, `CH2O`, `FAF`, and `TUE`. Outlier detection was performed using:

- Boxplots
- Interquartile Range (IQR) method
- Z-scores (optional, not shown in plots)

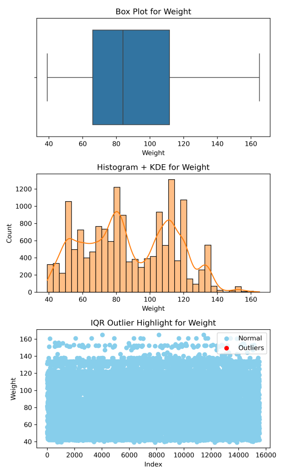
Example visualizations for each numerical feature are shown in Figure 2



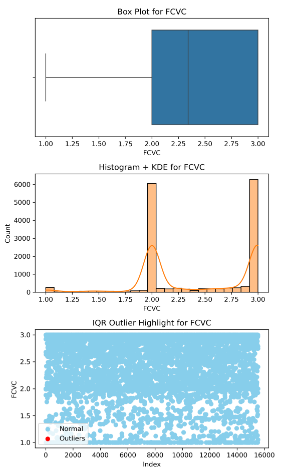
(a) Age



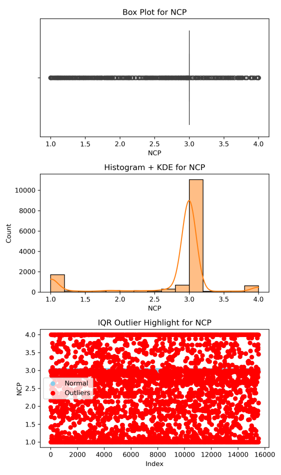
(b) Height



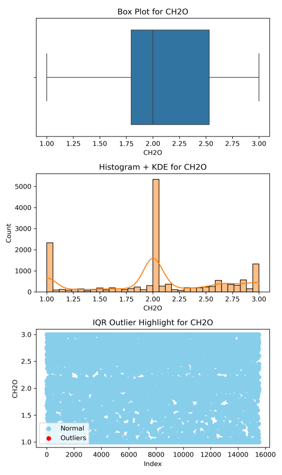
(c) Weight



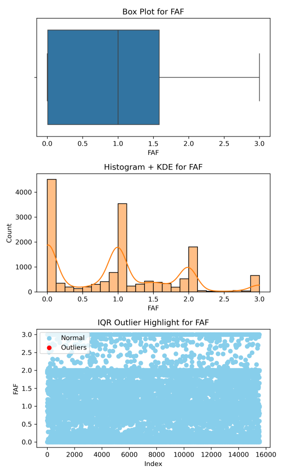
(d) FCVC



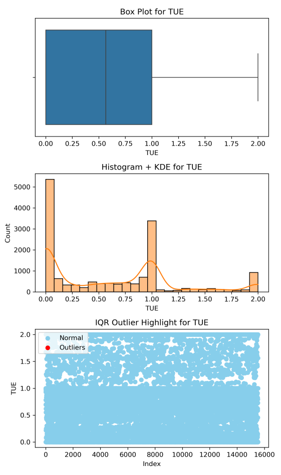
(e) NCP



(f) CH2O



(g) FAF



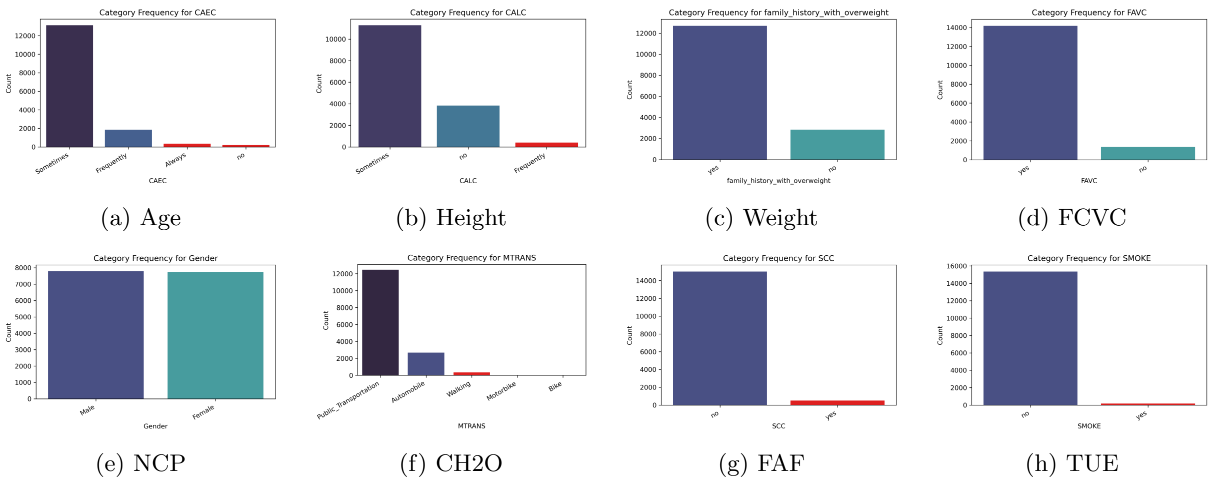
(h) TUE

Figure 2: Combined outlier analysis for numerical features: Boxplots, Histograms with KDE, and IQR highlights.

Each feature shows the distribution, potential outliers, and IQR highlights in a single combined figure. The features containing outliers (by IQR) are: Age, NCP, Height. To remediate it, we are applying BOXCOX transformation to reduce their skewness and reduce the influence of extreme values.

### 2.2.2 Categorical Features

Categorical features were analyzed for rare categories, defined as categories representing less than 5% of the dataset. Rare categories act as *categorical outliers*. This can be better visualized from the figure 3



(a) Age (b) Height (c) Weight (d) FCVC

(e) NCP (f) CH2O (g) FAF (h) TUE

Figure 3: Category count for all categorical features to find the rare categories

**Rare categories are:**

- CAEC: {Always, no}
- SMOKE: {yes}
- SCC: {yes}
- CALC: {Frequently}
- MTRANS: {Walking, Motorbike, Bike}

Although rare, these categories may carry valuable information about the target variable (`WeightCategory`) and were therefore needs testing before removing them, so we find the effect of outliers and compare the accuracy to decide whether to keep the outliers or remove them.

### 2.2.3 Effect of Outliers on Modeling

Two models were trained to analyze the impact of outliers:

- **Model A:** Trained on full dataset (with outliers).
- **Model B:** Trained after removing numerical outliers using Isolation Forest.

**Result:** Model A (full data) performed better or equal to Model B, suggesting that outliers in the dataset carried valuable information and should not be removed blindly. **To the very high extent we can say that it is a clean data, free from noise and outliers.**

### 2.2.4 Multivariate Outliers Detection

Multivariate outliers are observations in a dataset that deviate significantly from the overall pattern of the data when considering multiple features simultaneously. Unlike univariate outliers, which are extreme values in a single feature, multivariate outliers may appear normal in individual features but are anomalous when all features are considered together.

Why do we need to check for multi-variate outliers:

- Outliers can distort statistical estimates such as mean, covariance, and correlation.
- They may negatively affect machine learning models, leading to poor generalization.
- Identifying outliers helps ensure the robustness and reliability of predictive models.

**Algorithm 1** Effect of Outliers on Model Performance

```
Require: Dataset with features $X$ and target $y$
Ensure: Comparison of model accuracy with and without outliers
  Preprocess numerical features (Box-Cox for skewed features)
  Encode categorical features using ordinal and one-hot encoding
  Split the dataset into training and test sets
  Detect outliers in training data using Isolation Forest
  Separate inliers and outliers

  Model A: Train XGBoost on full training data
  Model B: Train XGBoost on cleaned training data (outliers removed)

  Evaluate both models on the same test set
  Compare accuracy and weighted F1-score
  if Model B performance > Model A performance then
      Conclude that removing outliers improved model performance
  else
      Conclude that outliers did not significantly affect performance
  end if
```

**Algorithm 2** Effect of Multivariate Outliers on Multiclass Classification

```
Require: Dataset with numeric features $X$ and multiclass target $y$
Ensure: Comparison of model accuracy and weighted F1-score with and without outliers
  Compute mean vector $\mu$ and covariance matrix $\Sigma$ of $X$
  Calculate Mahalanobis distance for each row: $D_M = \sqrt{(x-\mu)^T\Sigma^{-1}(x-\mu)}$
  Determine threshold from Chi-square distribution at significance level $\alpha$
  Identify outliers where $D_M >$ threshold
  Separate original dataset into:
  Full data (all rows)
  Cleaned data (rows without outliers)

  Model A: Train RandomForestClassifier on full data
  Model B: Train RandomForestClassifier on cleaned data

  Evaluate both models on the same test set
  Compare accuracy and weighted F1-score
  if Model B metrics > Model A metrics then
      Removing outliers improved model performance
  else
      Outliers did not significantly affect performance
  end if
```

To detect the outlier we have used the following algorithm

From the model we can interpret that removing these outliers has a negligible effect on both accuracy and weighted F1-score.In this dataset, multivariate outliers do not significantly impact the performance of the RandomForest classifier for the multiclass target. Therefore, retaining or removing them does not materially affect predictive performance.

## 3 Feature Engineering

Even if the dataset is relatively clean, feature engineering is an essential step in machine learning. Properly engineered features can help models capture underlying patterns more effectively, reduce bias, improve convergence, and sometimes improve interpretability. Additionally, new features can provide more meaningful information to the model than raw data alone, even when no missing values or outliers exist.

### 3.1 Creation of New Features

BMI is a useful addition because it provides a normalized measure of body fat relative to height and weight, which directly relates to obesity risk. While weight and height alone give absolute values, BMI captures the proportional relationship, allowing the model to better distinguish between underweight, normal, overweight, and obese individuals. This derived feature can reveal patterns that may not be evident from the raw weight and height features separately.

$$\text{BMI} = \frac{\text{Weight (kg)}}{(\text{Height (m)})^2} \tag{1}$$

### 3.2 Encoding Categorical Features

Machine learning models require numerical inputs, so categorical features must be converted into numeric form. This process, known as **encoding**, ensures that the model can interpret and process categorical information effectively.

- **Binary / Label Encoding:** Used for ordinal categories where the order carries meaning (e.g., `CAEC`: no ¡ Sometimes ¡ Frequently ¡ Always). This preserves the natural ordering of the categories.
- **One-Hot Encoding (OHE):** Used for nominal categories without inherent order (e.g., `MTRANS`: Car, Bike, Walking). Each category is represented as a separate binary feature to prevent the model from assuming any ordinal relationship.

### 3.3 Scaling and Normalization

Scaling ensures that all features contribute equally to the model training. Common scaling techniques include:

- **StandardScaler** StandardScaler standardizes features by removing the mean and scaling to unit variance:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma} \tag{2}$$

  where $\mu$ is the mean and $\sigma$ is the standard deviation of the feature.

- **RobustScaler** RobustScaler scales features using the median and interquartile range (IQR) to reduce the effect of outliers:

$$X_{\text{robust}} = \frac{X - \text{Median}(X)}{\text{IQR}(X)} \tag{3}$$

In our dataset, RobustScaler is not strictly necessary because the outliers have been already handled using Box-Cox transformations and Isolation Forest. Therefore, standard scaling (StandardScaler) is sufficient to normalize features.

## 4 Feature Selection

### 4.1 Correlation Analysis

The correlation matrix represents the **Pearson correlation coefficients** between numerical features. The coefficient $r$ ranges from $-1$ to 1:

- $r = 1$ indicates a perfect positive correlation (both features increase together).
- $r = 0$ indicates no linear correlation.
- $r = -1$ indicates a perfect negative correlation (one feature increases while the other decreases).

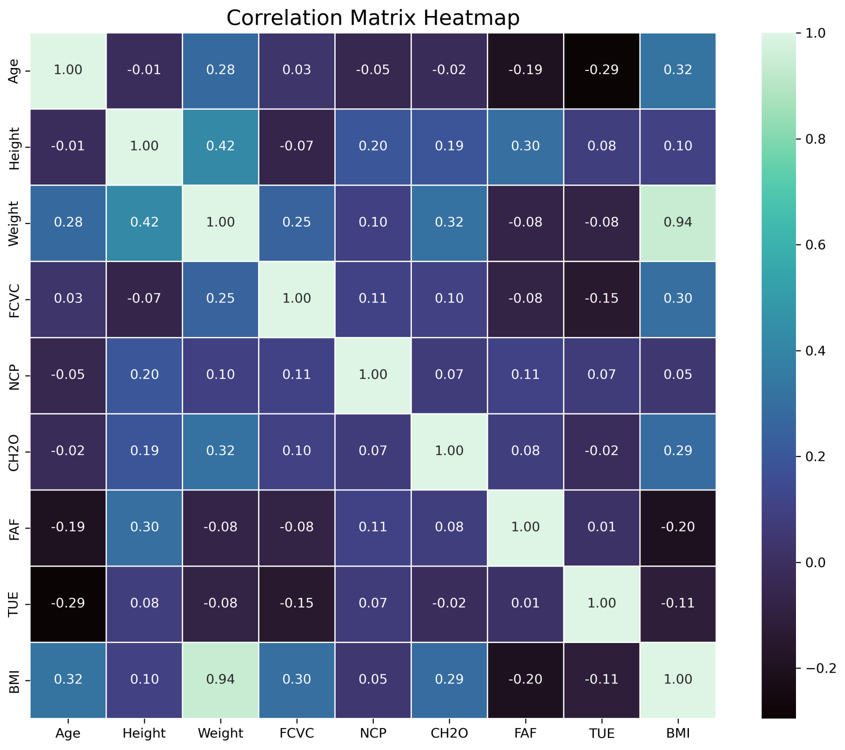


Figure 4: Correlation heatmap of numerical features. Darker shades indicate stronger correlation.

**Interpretation of values:**

- **BMI and Weight:** $r = 0.94$, indicating a very strong positive correlation. This is expected since BMI is directly derived from weight and height.
- **Weight and Height:** $r = 0.42$, moderate positive correlation.
- **Age and BMI:** $r = 0.32$, mild positive correlation.
- Other feature pairs generally have weak correlations ($|r| < 0.3$), indicating low linear relationships.

**Key observations:**

- **Highly correlated features:** BMI and Weight. Including both may be redundant for some models; however, since BMI captures non-linear combination, it may still provide useful information.
- **Weakly correlated features:** Most other features, such as NCP, FAF, and TUE, show weak correlations, suggesting they carry unique information and should be retained.
- **Positive vs. negative relations:**
  - Positive correlations: e.g., Weight & BMI, Weight & FCVC.
  - Negative correlations: e.g., FAF & BMI, TUE & BMI (slight negative).

Highly correlated pairs may be candidates for dimensionality reduction or dropping one to reduce redundancy, but weakly correlated features should generally be kept as they add unique information.

### 4.2 Feature Contribution Analysis Using Logistic Regression

To understand which features contribute most to predicting the weight category, we trained a logistic regression model on the dataset. We included all numeric and categorical features, added the derived **BMI** feature, and encoded categorical variables using label encoding for binary features and one-hot encoding for multiclass features. Scaling was applied to numeric features to ensure coefficients are comparable. By examining the magnitude of the coefficients, we can quantify the contribution of each feature to the prediction.

Table 4: Feature Contributions (Weights) from Logistic Regression

| **Feature** | **Weight** |
|---|---|
| Weight | 5.4208 |
| BMI | 5.3752 |
| Gender | 2.0424 |
| MTRANS_Public_Transportation | 1.0635 |
| FCVC | 0.8714 |
| family_history_with_overweight | 0.7762 |
| CALC_no | 0.7648 |
| CALC_Sometimes | 0.6775 |
| CAEC_Frequently | 0.6507 |
| CAEC_no | 0.4886 |
| MTRANS_Walking | 0.4740 |
| SCC | 0.4239 |
| Age | 0.4224 |
| CAEC_Sometimes | 0.4123 |
| NCP | 0.3919 |
| FAVC | 0.3718 |
| MTRANS_Motorbike | 0.3590 |
| Height | 0.2896 |
| SMOKE | 0.2523 |
| CH2O | 0.2456 |
| MTRANS_Bike | 0.2324 |
| FAF | 0.2153 |
| TUE | 0.1254 |

We can conclude that:

- **Most important features:** Weight (5.42) and BMI (5.38) are the strongest predictors of the weight category. Gender (2.04) also contributes noticeably.
- **Moderate contributors:** Lifestyle and activity features such as MTRANS_Public_Transportation, FCVC, CAEC, CALC, and family history contribute moderately.
- **Least important features:** Variables like Height, SMOKE, CH2O, FAF, TUE have minimal effect on prediction.

### 4.3 Effect of SMOTE on Model Performance

SMOTE (**Synthetic Minority Over-sampling Technique**) is a method used to address class imbalance in datasets. It generates synthetic examples of minority classes by interpolating between existing samples, rather than simply duplicating them. This helps classifiers to better learn decision boundaries for underrepresented classes. Although our dataset has seven weight categories, the distribution is slightly imbalanced, as shown below. To ensure that minority classes are not ignored by the model, we applied SMOTE to the training data and compared model performance on original and SMOTE-augmented data.

To evaluate the effect of class imbalance on model performance, we applied SMOTE to the training data. The workflow is summarized in Algorithm 3.

From the results, we observe that:

Table 5: Frequency and Percentage of Each WeightCategory Class

| **WeightCategory** | **Count** | **Percentage (%)** |
|---|---|---|
| Obesity_Type_III | 2983 | 19.20 |
| Obesity_Type_II | 2403 | 15.47 |
| Normal_Weight | 2345 | 15.10 |
| Obesity_Type_I | 2207 | 14.21 |
| Overweight_Level_II | 1881 | 12.11 |
| Insufficient_Weight | 1870 | 12.04 |
| Overweight_Level_I | 1844 | 11.87 |

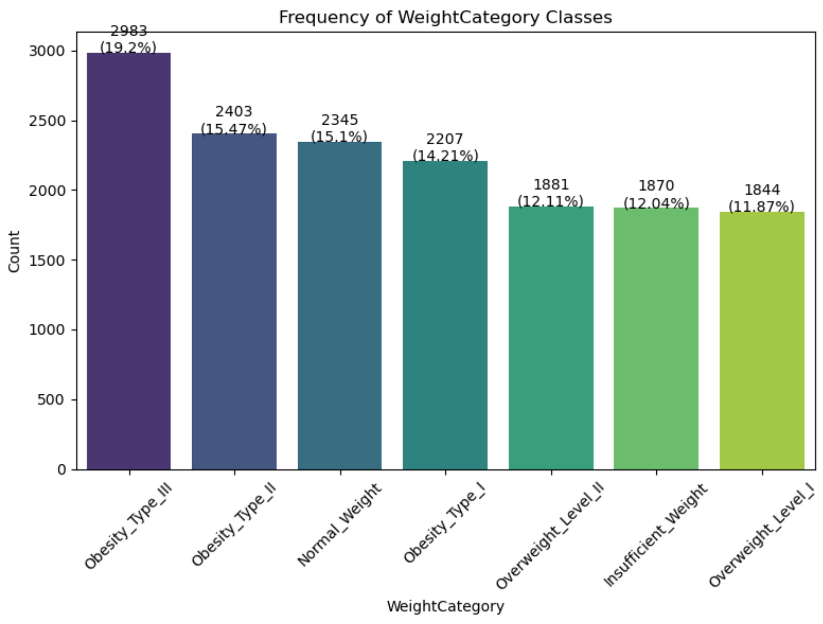


Figure 5: Frequency plot of WeightCategory classes

Table 6: Model Performance Comparison

| **Model** | **Accuracy** | **Weighted F1-score** |
|---|---|---|
| Original Data | 0.8983 | 0.8975 |
| SMOTE Data | 0.8973 | 0.8966 |

- The accuracy and weighted F1-score are very similar for both original and SMOTE-augmented data.
- SMOTE did **not significantly improve performance** for this dataset.
- This is consistent with the class distribution: the dataset is only mildly imbalanced (largest class 19.2%, smallest 11.87%), so standard classifiers already perform well.
- SMOTE is more useful for datasets with highly imbalanced classes, especially when minority classes are severely underrepresented.

## 5 Model Selection

As per the project guidelines, we were asked to develop our model using one or more of the following algorithms: Logistic Regression, Decision Tree, Random Forest, Bagging, AdaBoost, Gradient Boost, XGBoost and KNN. Each algorithm has distinct strengths and is suited for specific types of data and learning scenarios. Table 7 talks about when each algorith is best suited and what is the key advantage of using that algorithm.

**Algorithm 3** Effect of SMOTE on Model Performance

```
Require: Dataset with features $X$ and target $y$
Ensure: Comparison of model accuracy and weighted F1-score with and without SMOTE
  Preprocess numeric features and scale
  Encode categorical features (binary with label encoding, multiclass with one-hot encoding)
  Split the dataset into training and test sets (stratified by $y$)

  Model A: Train on original data
  Train Random Forest on original training data
  Evaluate on test set
  Record Accuracy and Weighted F1-score

  Model B: Train on SMOTE data
  Apply SMOTE to training data
  Train Random Forest on SMOTE-augmented data
  Evaluate on same test set
  Record Accuracy and Weighted F1-score

  if Weighted F1-score of Model B > Model A then
      Conclude SMOTE improved performance
  else
      Conclude SMOTE did not significantly improve performance
  end if
```

Table 7: Comparison of Machine Learning Algorithms

| **Algorithm** | **Best Suited For** | **Key Advantage** |
|---|---|---|
| **Logistic Regression** | Linearly separable data | Interpretable and simple baseline |
| **Decision Tree** | Non-linear, mixed-type data | Handles categorical/numeric data well |
| **Random Forest** | Large, noisy data | Reduces overfitting and variance |
| **Bagging** | High-variance models | Stabilizes and improves accuracy |
| **AdaBoost** | Clean, balanced data | Sequentially corrects weak learners |
| **Gradient Boost** | Complex, moderately large data | Learns non-linear relationships effectively |
| **XGBoost** | Large, structured data | High performance, robust, regularized |
| **K-Nearest Neighbors (KNN)** | Small to medium datasets with clear proximity patterns | Simple, non-parametric, captures local structure |

Before applying any algorithm, we have created a function of pre-processing the data based on our observations from data pre-processing. Refer 4.

**Algorithm 4** Preprocessing Dataset for Obesity Prediction

```
Input: Dataset $df$ with features including Height, Weight, and target column $target\_col$ (default: WeightCategory)
Output: Preprocessed features $X$, encoded target $y$, numeric columns $num\_cols$, categorical columns $cat\_cols$, target label encoder $le\_target$
 1: Copy the original dataset: $df\_processed \leftarrow df$
 2: Compute BMI if Height and Weight exist:
      Convert Height to meters if needed
      $BMI \leftarrow Weight/(Height\_m^2)$
 3: Drop temporary Height column
 4: if $target\_col$ exists in $df\_processed$ then
 5:     Encode target using LabelEncoder: $y \leftarrow le\_target.fit\_transform(df\_processed[target\_col])$
 6:     Drop $target\_col$ and 'id' columns from $df\_processed$
 7: else
 8:     $y \leftarrow$ None, $le\_target \leftarrow$ None
 9: end if
10: Identify numeric columns: $num\_cols \leftarrow$ numeric columns of $df\_processed$
11: Identify categorical columns: $cat\_cols \leftarrow$ categorical columns of $df\_processed$
12: return $df\_processed, y, num\_cols, cat\_cols, le\_target$
```

Different algorithms that are used for this project are:

1. We implemented a Bagging ensemble using XGBoost as the base estimator to predict the `WeightCategory`.
    - The Bagging ensemble consisted of 23 XGBoost models, each trained on 80% of the data and 80% of the features with bootstrapping enabled.
    - The dataset was split into 80% training and 20% validation sets, and the model achieved a validation accuracy of 90.77%, with strong per-class performance as shown in the confusion matrix and classification report.
    - Refer Algo 5
2. We implemented an XGBoost pipeline with preprocessing to predict the `WeightCategory`.
    - The dataset was split into 80% training and 20% validation sets. The model achieved a validation accuracy of 90.77%, with strong per-class performance as shown in the confusion matrix and classification report.
    - Refer Algo 6
3. We implemented a Voting classifier combining KNN and XGBoost to predict the `WeightCategory`.
    - The dataset was split into 80% training and 20% validation sets. The model achieved a validation accuracy of 90.44%, with strong per-class performance as shown in the confusion matrix and classification report.
    - Refer Algo 9
4. We implemented a Voting classifier combining Logistic Regression, KNN, and XGBoost to predict the `WeightCategory`.
    - The dataset was split into 80% training and 20% validation sets. The model achieved a validation accuracy of 90.41%, with strong per-class performance as shown in the confusion matrix and classification report.
    - Refer Algo 8
5. We implemented Bagging ensembles using XGBoost and AdaBoost as base estimators to predict the `WeightCategory`.
    - Bagged XGBoost used 15 base XGBoost models ($n\_estimators$ = 700, $max\_depth$ = 2, $learning\_rate$ = 0.05), while Bagged AdaBoost used 15 AdaBoost models ($n\_estimators$ = 300, $learning\_rate$ = 0.5). Both were trained with 80% of samples and features with bootstrapping enabled.

- The dataset was split into 80% training and 20% validation sets. The validation accuracy achieved was 90.02% for Bagged XGBoost and slightly lower for Bagged AdaBoost.
- Refer Algo 7

6. We implemented a hybrid ensemble combining 15 XGBoost pipelines and 15 Logistic Regression pipelines with soft voting to predict the `WeightCategory`.
   - Each XGBoost pipeline used $n\_estimators = 300$, $max\_depth = 2$, $learning\_rate = 0.05$, subsample=0.8, and colsample_bytree=0.8. Logistic Regression pipelines used max_iter=3000 with multinomial loss.
   - The dataset was split into 80% training and 20% validation sets. The validation accuracy achieved was 88.54%.
   - Refer Algo 10
7. We implemented a Bagging ensemble using LGBMClassifier as the base estimator to predict the `WeightCategory`.
   - The ensemble consisted of 20 LGBM models ($n\_estimators = 900$, $max\_depth = 3$, $learning\_rate = 0.1$), trained with 80% of samples and features using bootstrapping.
   - The dataset was split into 80% training and 20% validation sets. The validation accuracy achieved was 90.73%, with strong per-class performance as shown in the confusion matrix and classification report.
   - Refer Algo 11

### 5.1 Model Selection Algorithms

Various model selction algorithms are shown in algo: 5,6,7,8,9,10,11.

## 6 Hyperparameter Tuning

Hyperparameter tuning is the process of selecting the optimal set of parameters that govern the learning behavior of a machine learning model. Unlike model parameters, hyperparameters are not learned directly from the data but are predefined before training. Proper tuning of these hyperparameters is essential to achieve high model accuracy, avoid overfitting, and improve generalization.

### 6.1 Approaches to Hyperparameter Tuning

Various strategies can be used for tuning hyperparameters:

- **Manual Search:** Parameters are adjusted manually using prior knowledge or trial and error. While simple, this method is time-consuming and may overlook optimal combinations.
- **Grid Search:** Grid Search exhaustively explores all possible combinations of predefined hyperparameter values. Using `GridSearchCV` in `scikit-learn`, cross-validation is applied for each configuration to ensure robustness. This guarantees the best combination is found, but it can be computationally expensive for large parameter spaces.
- **Random Search:** Random combinations of parameters are sampled instead of checking all possible configurations. Although less exhaustive, it is faster and can yield good results with sufficient iterations.
- **Bayesian Optimization / Optuna:** Optuna is a modern, efficient framework that uses Bayesian optimization and pruning techniques. It dynamically narrows the search space based on past trial results, focusing on promising parameter regions. This allows faster convergence toward optimal configurations compared to grid or random search.

### 6.2 Tuning Results

The following tables summarize the results of hyperparameter tuning for different models. Each model was trained with varying configurations, and the version yielding the highest accuracy was selected for final use.

**Algorithm 5** Bagging Ensemble with XGBoost

```
Require: Preprocessed dataset $X$, target $y$
Ensure: Trained Bagging-XGBoost pipeline
1: Split dataset into training/validation (80% / 20%)
2: Base XGBoost: $n\_estimators = 900$, $max\_depth = 5$, $learning\_rate = 0.3$, $subsample = 0.8$, $colsample\_bytree = 0.8$
3: Bagging: 23 estimators, max samples 0.8, max features 0.8, bootstrap=True
4: Combine into pipeline, train, evaluate
5: Save trained pipeline
```

**Algorithm 6** XGBoost Pipeline with Preprocessing

```
Require: Preprocessed dataset $X$, target $y$
Ensure: Trained XGBoost pipeline
1: Split dataset into training/validation (80% / 20%)
2: XGBoost: $n\_estimators = 700$, $max\_depth = 3$, $learning\_rate = 0.1$, $subsample = 0.8$, $colsample\_bytree = 0.8$
3: Combine preprocessing + XGBoost into pipeline, train, evaluate
4: Retrain on full dataset, save pipeline
```

**Algorithm 7** Bagging Ensembles: XGBoost + AdaBoost

```
Require: Dataset $X$, target $y$
Ensure: Trained Bagged-XGBoost and Bagged-AdaBoost pipelines
1: Split dataset into training/validation (80% / 20%)
2: Base models: XGBoost (n=700, depth=2), AdaBoost (n=300, lr=0.5)
3: Wrap each in Bagging: 15 estimators, max samples=0.8, max features=0.8, bootstrap=True
4: Combine preprocessing + Bagging, train, evaluate
5: Save trained pipelines
```

**Algorithm 8** Voting Classifier LR + KNN + XGBoost

```
Require: Preprocessed dataset $X$, target $y$
Ensure: Trained Voting-LR-KNN-XGBoost pipeline
1: Split dataset into training/validation (80% / 20%)
2: Base models: XGBoost (n=700, depth=3), KNN (n=11), LogisticRegression (max_iter=2100)
3: VotingClassifier with hard voting, weights [6,2,2]
4: Combine preprocessing + Voting, train, evaluate
5: Retrain on full dataset
```

**Algorithm 9** Voting Classifier KNN + XGBoost

```
Require: Preprocessed dataset $X$, target $y$
Ensure: Trained Voting-KNN-XGBoost pipeline
1: Split dataset into training/validation (80% / 20%)
2: Base models: XGBoost (n=700, depth=3), KNN (n=11, weights='distance')
3: VotingClassifier with soft voting, weights [1,3]
4: Combine preprocessing + Voting, train, evaluate
5: Retrain on full dataset, save pipeline
```

**Algorithm 10** Hybrid Bagging + Voting: 15 XGB + 15 LR

```
Require: Dataset $X$, target $y$
Ensure: Trained hybrid Voting ensemble
1: Split dataset into training/validation (80% / 20%)
2: Build 15 XGBoost + 15 Logistic Regression pipelines
3: Combine into VotingClassifier with soft voting
4: Train ensemble, evaluate, retrain on full dataset
```

**Algorithm 11** Bagging Ensemble with LGBMClassifier

```
Require: Dataset $X$, target $y$
Ensure: Trained Bagging-LGBM pipeline
1: Split dataset into training/validation (80% / 20%)
2: Base LGBM: n=900, depth=3, lr=0.1, subsample=0.8, colsample_bytree=0.8
3: Wrap in Bagging: 20 estimators, max samples=0.8, max features=0.8, bootstrap=True
4: Combine preprocessing + Bagging, train, evaluate
5: Retrain on full dataset
```

Table 8: XGBoost + Bagging Hyperparameter Tuning

| **n_estimators** | **max_depth** | **learning_rate** | **Accuracy (%)** |
|---|---|---|---|
| 900 | 5 | 0.3 | 90.97 |
| 700 | 3 | 0.3 | 90.34 |
| 1000 | 5 | 0.1 | 90.54 |
| 700 | 7 | 0.3 | 90.41 |

Table 9: XGBoost Hyperparameter Tuning

| **n_estimators** | **max_depth** | **learning_rate** | **Accuracy (%)** |
|---|---|---|---|
| 700 | 3 | 0.1 | 90.79 |
| 700 | 5 | 0.1 | 89.93 |
| 900 | 5 | 0.1 | 89.93 |

### 6.3 Observations

From the results, it can be seen that:

- Lower learning rates (0.05–0.1) consistently improved model stability and accuracy.
- Increasing the number of estimators improved performance up to a certain point but with diminishing returns.
- The final XGBoost + bagging model achieved the highest individual performance with 90.97% accuracy.

## 7 Results and Evaluation

Initially, XGBoost was used as a standalone model, which achieved strong baseline performance with high accuracy and stable predictions. For further improvements in the results , XGBoost was combined with Adaptive Boosting, which helped the model focus on harder-to-predict samples, which slightly increased the performance. Finally, XGBoost was combined with Bagging, which aggregated multiple base learners, which helped to reduce variance and improve generalization. This approach achieved the highest accuracy of 91.212%, outperforming all previous model configurations. When we focus on best results of the models discussed above we find the following results as tabulated in table 10

Table 10: Summary of Hyperparameter Tuning Results

| **Model** | **n_estimators** | **max_depth** | **learning_rate** | **Accuracy (%)** |
|---|---|---|---|---|
| **XGBoost + Bagging** | 900 | 5 | 0.3 | 90.97 |
| **XGBoost (Final)** | 700 | 3 | 0.1 | 90.79 |
| **Base Model (XGB + AdaBoost)** | 700 | 4 | 0.05 | 89.90 |

The results indicate that combining boosting and bagging techniques can enhance predictive performance for obesity classification tasks.Together, they provide a balance between bias and variance, leading to more robust predictions and better generalization on unseen data. Additionally, the chosen hyperparameters (moderate tree depth, sufficient number of estimators, and an appropriate learning rate) allow the model to capture complex patterns in the data without overfitting.