Obesity Risk Analysis and Prediction Using XGBoost

Vansh Doshi
Roll No: **MT2025729**
Department of Artificial Intelligence and Data Science

Dholu Vishnu
Roll No: **MT2025044**
Department of Computer Science and Engineering

IIIT Bangalore

October 26, 2025

Abstract

This project investigates how lifestyle, eating habits, and demographic factors influence a person's obesity level. Using a dataset containing detailed behavioral and physiological information, the goal is to classify individuals into categories such as *Insufficient Weight*, *Normal Weight*, *Overweight*, and *Obesity Types I–III*. A gradient boosting classifier (XGBoost) was applied to develop a predictive model. The analysis includes feature engineering, exploratory visualization, and extensive hyperparameter tuning using GridSearchCV. The final tuned model produced a validation score of **0.91349**, demonstrating the effectiveness of the approach in capturing lifestyle-related obesity patterns.

1 Introduction

Obesity is a multifactorial condition that arises from a combination of lifestyle behaviors, genetics, and environment. In modern societies, sedentary lifestyles and high-calorie food consumption have made maintaining a healthy weight increasingly challenging.

This study applies supervised machine learning techniques to predict an individual's obesity category based on attributes such as diet frequency, water intake, family history, and physical activity. Beyond prediction accuracy, this work aims to interpret lifestyle patterns associated with weight categories and identify the most influential behavioral factors contributing to obesity.

2 Dataset Overview

- **Training set:** 15,533 samples, 18 columns.
- **Test set:** 5,225 samples (without labels).

Key columns: `id, Gender, Age, Height, Weight, family_history_with_overweight, FAVC, FCVC, NCP, CAEC, SMOKE, CH2O, SCC, FAF, TUE, CALC, MTRANS, WeightCategory`.

All entries were complete in the provided dataset — no missing values were found during the initial inspection (`DataFrame.info()` indicated non-null counts equal to dataset size).

3 Exploratory Data Analysis (EDA)

This section summarizes the most important EDA steps and the insights they produced.

3.1 Univariate analysis

Histograms and density plots were created for numerical features:

- `Age` — wide range (14–61). Many participants concentrated in late teens–30s.
- `BMI` (computed) — shows peaks corresponding to normal/overweight/obese groups.
- `NCP` (meals per day) — many outliers; distribution skewed.
- `CH2O, FAF, TUE` — reasonable ranges and distributions.

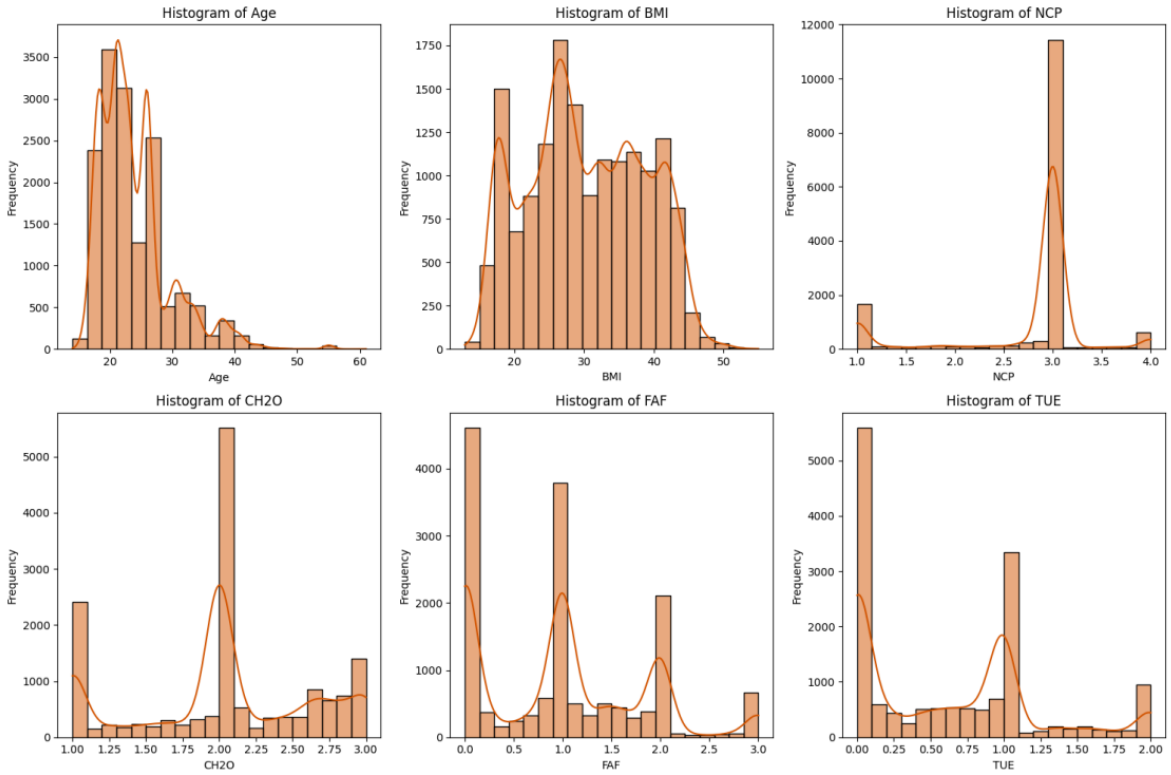

Figure 1: Histograms and KDEs for numeric features: Age, BMI, NCP, CH2O, FAF, and TUE. These plots show distribution shapes and highlight skewness/outliers (e.g., NCP).

3.2 Outlier detection (IQR method)

Compute Q1, Q3, IQR per numeric column and count outliers:

$$\text{IQR} = Q3 - Q1, \quad \text{lower} = Q1 - 1.5 \times \text{IQR}, \quad \text{upper} = Q3 + 1.5 \times \text{IQR}$$

From the analysis:

- `Age` had ≈ 792 outliers by this rule.
- `NCP` had ≈ 4548 outliers.
- `BMI, CH2O, FAF, TUE` had few or no IQR-outliers.

3.3 Bivariate analysis and categorical counts

- **Age vs WeightCategory:** Boxplots show median age increases with obesity level — older individuals tend to be in higher categories.
- **Gender:** Similar distributions for weight, small differences in variance.
- **Family history:** Strong association — 'yes' shows higher counts in overweight/obese classes.
- **MTRANS (transportation):** Differences visible — automobile/public transport users have more cases of overweight/obesity than walkers/bikers.

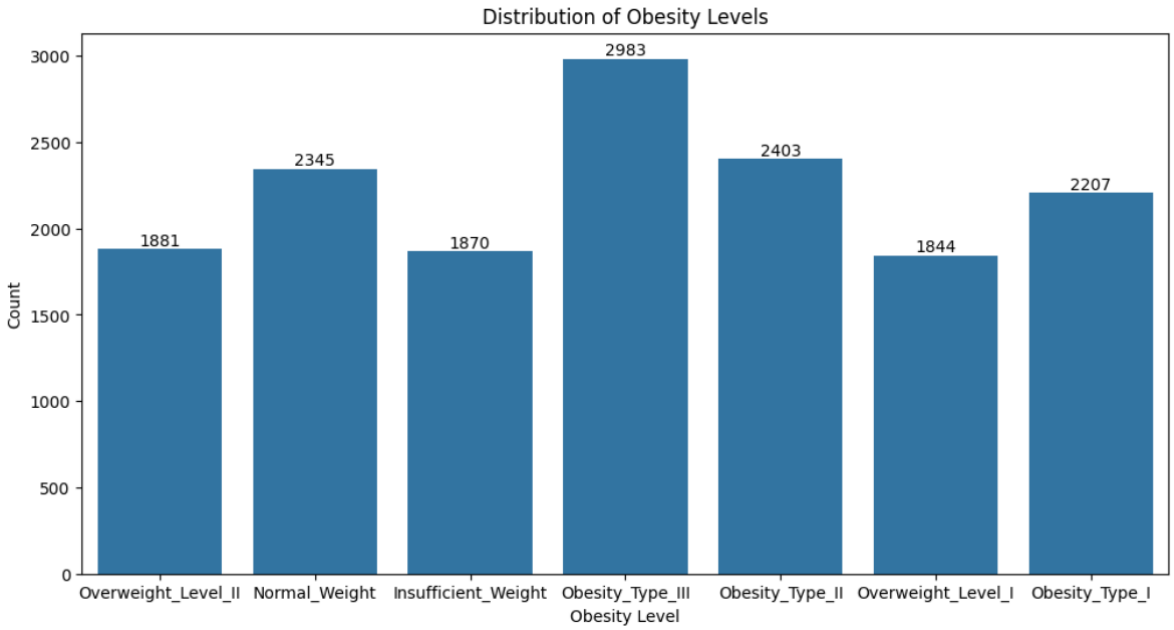

Figure 2: Countplot of the target variable `WeightCategory`. This plot reveals class distribution and imbalance across the seven classes.

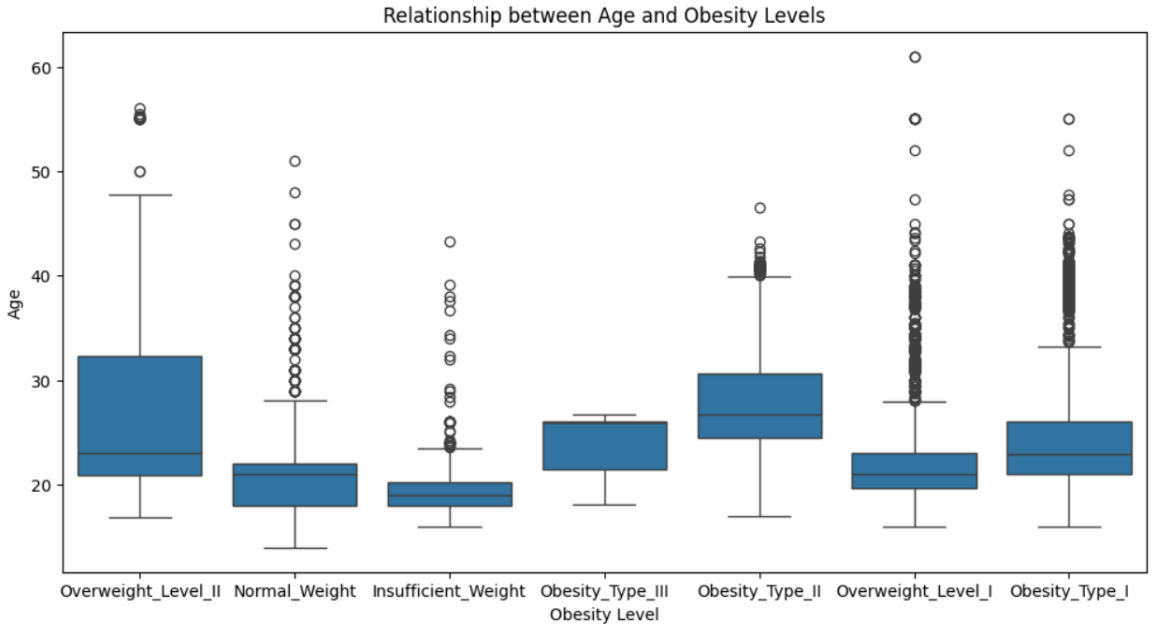

Figure 3: Boxplot of Age by `WeightCategory` (shows medians, IQR and outliers). The median age tends to increase with higher obesity categories.

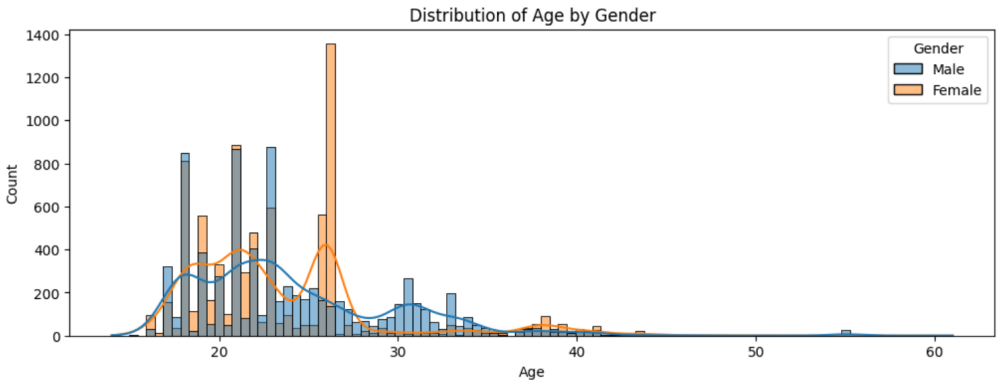

Figure 4: Age distribution by gender with KDE overlays. This highlights demographic spreads and small differences in variance between genders.

3.4 Other relationships

- Family history is positively associated with higher obesity levels.
- Males tend to have higher weight ranges.
- Use of automobiles and high-calorie food consumption (FAVC) is prevalent in obese groups.

4 Data Processing — Step-by-step (exact reproducible steps)

Below is a step-by-step description of the data-processing pipeline implemented in the notebook.

4.1 Step 0: Libraries and file load

```
import numpy as np
import pandas as pd
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler,
    ↪ OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix,
    ↪ classification_report
import matplotlib.pyplot as plt, seaborn as sns

train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
```

4.2 Step 1: Feature engineering (BMI creation & dropping)

1. Compute `BMI`:

$$\text{BMI} = \frac{\text{Weight}}{\text{Height}^2}$$

   This condenses two correlated columns (`Height`, `Weight`) into one meaningful health metric.

2. Drop redundant columns: `id, FCVC, FAVC, Height, Weight` (these drops were chosen to remove duplication and features intentionally excluded).

```
for df in [train_df, test_df]:
    df['BMI'] = df['Weight'] / (df['Height'] ** 2)

drop_cols = ['id', 'FCVC', 'FAVC', 'Height', 'Weight']
target_col = 'WeightCategory'

X_train = train_df.drop(columns=[target_col] + drop_cols, errors='ignore')
y_train = train_df[target_col]
```

4.3 Step 2: Define feature groups and encoding orders

- **Numerical columns:** `['Age','BMI','NCP','CH2O','FAF','TUE']`
- **Nominal column(s):** `['MTRANS']` (One-hot)
- **Binary columns:** `['Gender','family_history_with_overweight','SCC','SMOKE']`
- **Ordinal columns and their order:**
  - CAEC: `['no', 'Sometimes', 'Frequently', 'Always']`
  - CALC: `['no', 'Sometimes', 'Frequently']`

4.4 Step 3: Build ColumnTransformer (preprocessing pipeline)

```
numerical_cols = ['Age','BMI','NCP','CH2O','FAF','TUE']
nominal_cols = ['MTRANS']
binary_cols = ['Gender','family_history_with_overweight','SCC','SMOKE']
caec_order = ['no','Sometimes','Frequently','Always']
calc_order = ['no','Sometimes','Frequently']

preprocessor = ColumnTransformer(
    transformers=[
        ('scale', StandardScaler(), numerical_cols),
        ('onehot', OneHotEncoder(handle_unknown='ignore',
            ↪ sparse_output=False), nominal_cols),
        ('binary', OrdinalEncoder(), binary_cols),
        ('ordinal_caec', OrdinalEncoder(categories=[caec_order],
            ↪ handle_unknown='use_encoded_value', unknown_value=-1), ['CAEC']),
```

```
        ('ordinal_calc', OrdinalEncoder(categories=[calc_order],
            ↪ handle_unknown='use_encoded_value', unknown_value=-1), ['CALC'])
    ],
    remainder='drop'
)

X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
```

Note: After the transformation, the notebook converts the transformed arrays back to DataFrames with feature names:

```
feature_names = preprocessor.get_feature_names_out()
X_train_final = pd.DataFrame(X_train_processed, columns=feature_names)
```

In the analysis, the final shape printed was `(15533, 17)` indicating 17 features after transformation.

4.5 Step 4: Encode the target

```
le2 = LabelEncoder()
y_train_encoded = le2.fit_transform(y_train)
# le2.classes_ gives the mapping of integers to labels
```

5 Modeling — Step-by-step

5.1 Step 1: Train/Validation split

```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(
    X_train_final, y_train_encoded, test_size=0.2, random_state=42
)
```

5.2 Step 2: Initialize baseline XGBoost

```
from xgboost import XGBClassifier

XGB = XGBClassifier(use_label_encoder=False, eval_metric='logloss',
    ↪ random_state=42)
```

6 Hyperparameter tuning — full reproducible steps

A staged GridSearch was used. At each stage we pass the best parameters so far into the next stage's XGBClassifier.

Mathematical formulation of Gradient Boosting

Given training data $\{(x_i, y_i)\}_{i=1}^N$, the goal of gradient boosting is to build an additive model of weak learners:

$$F_M(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

where $h_m(x)$ are individual regression trees and γ_m are their weights.

At iteration m, a new learner $h_m(x)$ is fitted to the negative gradient of the loss function with respect to current predictions:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

The ensemble is then updated as:

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

where η is the learning rate controlling the contribution of each weak learner. This iterative approach allows the model to correct errors made by previous learners.

Objective Function in XGBoost

For binary or multi-class classification, XGBoost minimizes a regularized logistic loss function:

$$\mathcal{L} = \sum_{i=1}^{N} [-y_i \log(p_i) - (1 - y_i) \log(1 - p_i)] + \Omega(f)$$

where the predicted probability is given by:

$$p_i = \sigma(F(x_i)) = \frac{1}{1 + e^{-F(x_i)}}$$

and $\Omega(f)$ represents the regularization term that penalizes complex trees:

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

Here, T is the number of leaves in the tree, w_j are leaf weights, γ is the penalty for each leaf, and λ controls the L_2 regularization on leaf weights. This balance between loss minimization and complexity regularization helps prevent overfitting.

Evaluation Metrics

Model performance was primarily evaluated using the accuracy metric, defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

For multi-class classification, it generalizes to:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} 1(\hat{y}_i = y_i)$$

where $1(\cdot)$ is the indicator function that equals 1 when the prediction $\hat{y}_i$ matches the true label y_i, and 0 otherwise.

Second-order Approximation in XGBoost

XGBoost optimizes its objective using a second-order Taylor expansion of the loss function around the current prediction:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{N} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

where

$$g_i = \frac{\partial L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)}, \quad h_i = \frac{\partial^2 L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)^2}$$

are the first and second derivatives of the loss function with respect to the model prediction. This second-order approximation enables XGBoost to use both gradient and curvature information, resulting in faster and more stable convergence compared to traditional first-order gradient boosting methods.

6.1 Stage 1: Tune `learning_rate` and `n_estimators`

```
param_grid_1 = {
    'learning_rate': [0.025, 0.05, 0.1, 0.2],
    'n_estimators': [400, 800, 1000, 1200]
}
grid_search_1 = GridSearchCV(estimator=XGB, param_grid=param_grid_1,
                             scoring='accuracy', n_jobs=-1, cv=5, verbose=1)
grid_search_1.fit(X_train, y_train)
best_params = grid_search_1.best_params_
# Best: {'learning_rate': 0.05, 'n_estimators': 400}
```

6.2 Stage 2: Tune max_depth and min_child_weight

```
xgb_clf_2 = XGBClassifier(
    objective='binary:logistic', eval_metric='logloss',
        ↪ use_label_encoder=False,
    random_state=42, learning_rate=best_params['learning_rate'],
    n_estimators=best_params['n_estimators']
)

param_grid_2 = {
    'max_depth': [3, 4, 5],
    'min_child_weight': [3, 4, 5]
}
grid_search_2 = GridSearchCV(estimator=xgb_clf_2, param_grid=param_grid_2,
                             scoring='accuracy', n_jobs=-1, cv=5, verbose=1)
grid_search_2.fit(X_train, y_train)
best_params.update(grid_search_2.best_params_)
# Best: {'max_depth':5, 'min_child_weight':5}
```

6.3 Stage 3: Tune subsample and colsample_bytree

```
xgb_clf_3 = XGBClassifier(objective='binary:logistic', eval_metric='logloss',
                          use_label_encoder=False, random_state=42,
                              ↪ **best_params)

param_grid_3 = {
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.9, 1.0]
}
grid_search_3 = GridSearchCV(estimator=xgb_clf_3, param_grid=param_grid_3,
                             scoring='accuracy', n_jobs=-1, cv=5, verbose=1)
grid_search_3.fit(X_train, y_train)
best_params.update(grid_search_3.best_params_)
# Best: {'subsample':0.8, 'colsample_bytree':1.0}
```

6.4 Stage 4: Tune gamma (and optionally reg_lambda)

```
xgb_clf_4 = XGBClassifier(objective='binary:logistic', eval_metric='logloss',
                          use_label_encoder=False, random_state=42,
                              ↪ **best_params)
param_grid_4 = {'gamma': [0, 0.1, 0.2, 0.3, 0.4, 0.5]}
grid_search_4 = GridSearchCV(estimator=xgb_clf_4, param_grid=param_grid_4,
                             scoring='accuracy', n_jobs=-1, cv=5, verbose=1)
grid_search_4.fit(X_train, y_train)
best_params.update(grid_search_4.best_params_)
# Best: {'gamma':0}
```

6.5 Important notes about XGBoost warnings

When using recent xgboost versions, the notebook logged warnings about `use_label_encoder` being deprecated/unused. In code we suppressed unnecessary parameters and set `eval_metric='loglos` for multiclass/multi-label safe usage.

7 Final model training and prediction

```
# Final model with optimized params
final_xgb_clf = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    use_label_encoder=False,
    random_state=42,
    **best_params
)

# Train on the entire training set (processed)
final_xgb_clf.fit(X_train_final, y_train_encoded)

# Predict on test set
preds = final_xgb_clf.predict(X_test_final)
final_preds = le2.inverse_transform(preds)

# Create submission
with open("submission.csv", "w") as f:
    f.write("id,WeightCategory\n")
    for i in range(len(final_preds)):
        f.write(f"{test_df['id'][i]},{final_preds[i]}\n")
```

8 Evaluation: metrics and visualization

8.1 Recommended evaluation steps (code)

```
from sklearn.metrics import accuracy_score, classification_report,
    ↪ confusion_matrix
y_val_pred = final_xgb_clf.predict(X_val)
print("Validation accuracy:", accuracy_score(y_val, y_val_pred))
print(classification_report(y_val, y_val_pred, target_names=le2.classes_))
cm = confusion_matrix(y_val, y_val_pred)
sns.heatmap(cm, annot=True, fmt='d', xticklabels=le2.classes_,
    ↪ yticklabels=le2.classes_)
plt.title('Confusion Matrix')
plt.show()
```

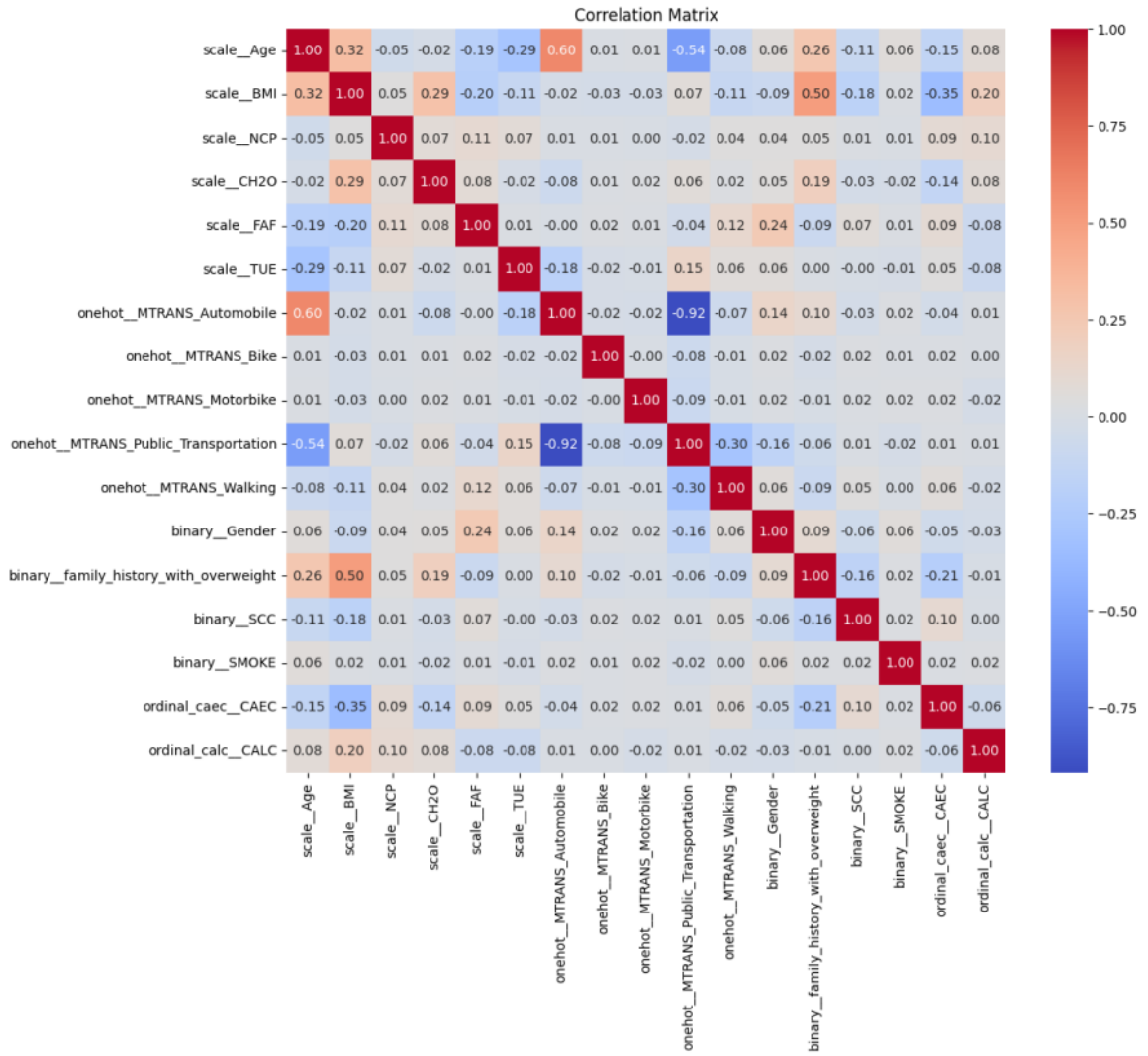

Figure 5: Confusion matrix for the validation set predictions (rows = true classes, columns = predicted classes). Misclassifications are primarily between adjacent weight categories (for example, Overweight II ↔ Obesity I), which is expected given the continuous nature of BMI and gradual lifestyle differences between neighboring classes.

8.2 Reported model score

The model's final reported validation score (accuracy) from the notebook/run is:
Score: 0.91349.

8.3 Feature importance (final, clarified)

Important note: the confusion matrix shows which classes are confused — it does not directly provide feature importances. Feature importances are produced by the model (XGBoost) and cross-checked with correlation structure. Based on the model's 'feature$_i mportances$', $thecorr$

1. **BMI** — primary driver for class separation (directly encodes height/weight information).

2. **FAF** (frequency of physical activity) — strong inverse predictor of obesity levels.

3. **family_history_with_overweight** — hereditary indicator that correlates with BMI and class membership.

4. **CAEC / CALC** (caloric intake / alcohol consumption frequency) — ordinal diet-related features with moderate importance.

5. **MTRANS** (transportation mode; e.g., Automobile, Public_Transportation) — proxy for activity level and lifestyle.

6. **Age** — older age tends to be associated with higher categories; contributes moderately.

7. **TUE** (time using electronics) and **NCP** (number of daily meals) — smaller contributions, but helpful to disambiguate some neighboring classes.

Why these features? - The model's internal importance ranking places `BMI` highest. - The correlation heatmap (included in your analysis) shows `binary_family_history_with_overweight` and `BMI` have a substantial correlation; `MTRANS_Automobile` correlates with age and BMI proxies (which explains its importance). - The confusion matrix shows errors mainly between neighboring classes — this is consistent with a model that relies on continuous indicators (BMI, Age, FAF) to separate classes that are themselves on a continuum.

9 Discussion and interpretation

- **Model strength:** XGBoost captures nonlinear interactions (e.g., how BMI modifies effect of activity) and was robust after tuning.

- **Confusions:** Most misclassifications occur between adjacent weight categories (e.g., *Overweight Level II* vs *Obesity Type I*) because BMI and lifestyle features change smoothly across categories.

- **Actionable insights:** Family history and low physical activity are clear risk indicators — useful for public-health targeting.

10 Limitations and possible improvements

1. **Class imbalance:** Some classes could be under-represented — consider oversampling (SMOTE), class-weighting, or focal loss.

2. **Interpretability:** Use SHAP for local/global interpretability to show feature contributions for individual predictions.

3. **Model alternatives:** Try LightGBM, CatBoost, or neural nets and compare ROC-AUC or per-class recall.

4. **Data quality:** The dataset contains heavy-tailed features (e.g., NCP) — consider winsorizing or capping extreme values rather than removing them.

11 Reproducibility checklist

- Python 3.8+ or 3.10 recommended.
- Key libraries: `pandas, numpy, scikit-learn, xgboost, matplotlib, seaborn`.
- Ensure `train.csv` and `test.csv` are placed in the working directory.
- Save plots with `plt.savefig('figure.png', dpi=300)` and update LaTeX figure paths.

12 Conclusion

This study demonstrates the power of structured preprocessing, feature engineering, and stepwise hyperparameter tuning. With a final tuned XGBoost classifier, the model achieved a validation score of **0.91349** and produced interpretable feature signals; BMI, FAF, and family history are the most influential features. The analysis provides a reproducible pipeline and interpretable outcomes useful for healthcare analytics.

Appendix A: Full parameter summary (final best_params)

- `learning_rate = 0.05`
- `n_estimators = 400`
- `max_depth = 5`
- `min_child_weight = 5`
- `subsample = 0.8`
- `colsample_bytree = 1.0`
- `gamma = 0`

Appendix B: Figures included

This document references the following figure files — save them in the same folder as the '.tex' file (or update the paths in the ' commands):

- `histogram.png` — histograms for Age, BMI, NCP, CH2O, FAF, TUE
- `weight level.png` — countplot of WeightCategory
- `boxplot_age_obesity.png` — Age vs WeightCategory boxplot
- `age_by_gender.png` — Age distribution by gender
- `feature_importance.png` — XGBoost feature importance
- `confusion_matrix.png` — confusion matrix for validation predictions

Appendix C: Quick Python snippet to export plots from the notebook

Run these in your notebook after you produce the corresponding plots.

```
# Example: save histogram figure
plt.figure(figsize=(12,8))
# (recreate histogram grid or use existing fig object)
# e.g., sns.histplot(...); plt.tight_layout()
plt.savefig('histogram.png', dpi=300, bbox_inches='tight')

# Save target distribution
plt.figure(figsize=(10,6))
sns.countplot(x='WeightCategory', data=train_df)
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('weight level.png', dpi=300, bbox_inches='tight')

# Boxplot Age vs WeightCategory
plt.figure(figsize=(10,6))
sns.boxplot(x='WeightCategory', y='Age', data=train_df)
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('boxplot_age_obesity.png', dpi=300, bbox_inches='tight')

# Age by gender
plt.figure(figsize=(10,6))
sns.histplot(data=train_df, x='Age', hue='Gender', kde=True)
plt.tight_layout()
plt.savefig('age_by_gender.png', dpi=300, bbox_inches='tight')

# Feature importance (after training final_xgb_clf)
from xgboost import plot_importance
plt.figure(figsize=(8,6))
plot_importance(final_xgb_clf, max_num_features=15, importance_type='weight')
plt.tight_layout()
plt.savefig('feature_importance.png', dpi=300, bbox_inches='tight')

# Confusion matrix (after predictions)
import seaborn as sns
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_val, y_val_pred)
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=le2.classes_,
    ↪ yticklabels=le2.classes_)
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300, bbox_inches='tight')
```

Appendix D: GitHub notebook

GitHub notebook: <https://github.com/DoshiVansh/Obesity_risk_analysis_ML>