Report: Multi-Class Obesity Classification

Rahul Raman
Aayank Singhai

October 26, 2025

Abstract

This report breaks down the full process we used for a multi-class obesity classification project. We started with a deep dive into the data (**Exploratory Data Analysis - EDA**) using the training dataset to find insights and check for problems. After that, we built machine learning models. The main model is an **XGBoost Classifier**, which we tuned using **Optuna** and trained with a 10-fold **Stratified K-Fold** strategy to make it solid. We also built a **Random Forest** model using **GridSearchCV** as a comparison. This document covers everything from data cleaning and feature engineering to model training and evaluation.

1 Introduction

The main goal here was to accurately predict someone's obesity status based on their lifestyle and personal attributes. The target variable, `NObeyesdad`, represents multiple classes of weight status, making this a multi-class classification problem.

We were given a `train.csv` and a `test.csv`. This report focuses on the analysis and modeling performed primarily using the `train.csv` data. First, we'll cover the data quality checks and the Exploratory Data Analysis (EDA) performed on the training set. Then, we'll walk through how we built and evaluated models on this training data to eventually make predictions for the unseen test set.

2 Data Pre-Analysis and Quality Check

First things first, we had to check the raw `train.csv` dataset (15,533 rows, 18 columns including the target) to see if there were any obvious problems before diving deeper.

- **Data Types:** The dataset consists of 8 numerical features (float64), 8 categorical features (object), 1 integer ID column (int64), and 1 target column (object).
- **Missing Values:** A check using `isnull().sum()` confirmed that the dataset is complete. **Status: No missing values found.**
- **Duplicate Rows:** A check using `duplicated().sum()` was performed. **Status: No duplicate rows found.**

It turns out the data was in great shape. We didn't need to fill in any missing values or remove duplicate rows. The basic stats (via `describe()`) showed that all the numbers were in reasonable ranges. Categorical data analysis revealed that `Male` (7783) was slightly more frequent than

Female, family_history_with_overweight was predominantly 'yes' (12696), FAVC (high caloric food consumption) was mostly 'yes' (14184), CAEC (food between meals) was mostly 'Sometimes' (13126), and Public_Transportation was the most common transport method (12470).

3 Feature Engineering and Pre-processing

To help and understand the data better, we did two main things.

3.1 Attribute Renaming

The original column names were a bit cryptic (like FAVC), so we renamed them to be more descriptive (like High_Caloric_Food_Consumption) for clarity. This was applied to both train and test sets for consistency.

3.2 Feature Engineering: BMI

We engineered a new feature for **Body Mass Index (BMI)**, since it's obviously a critical factor for classifying obesity. This was added to both train and test sets.

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

3.3 Preprocessing for Modeling

To get the data ready for the models, we used a standard preprocessing approach. These steps are essential to convert the raw data into a format that machine learning models can understand.

1. **Target Encoding:** The target column NObeyesdad in the training set was text (e.g., 'Overweight_Level_I'). We used LabelEncoder to convert these text labels into integers (0, 1, 2, etc.), since models require numerical outputs.

2. **Feature Encoding:** Similarly, categorical features like Gender ('Male'/'Female') in both train and test sets were converted into numbers. We used pd.get_dummies (One-Hot Encoding) for this, which creates new binary columns (e.g., Gender_Male) that the model can use.

3. **Scaling:** Numerical features like 'Age' and 'Weight' exist on very different scales. StandardScaler was used to transform them (fitting on the training data, transforming both train and test) so they all have a similar mean and variance. This helps the model learn more effectively and prevents one feature from dominating the others.

4. **Alignment:** After One-Hot Encoding, the train and test sets might have a different number of columns (if one set was missing a category). We explicitly aligned them to ensure both datasets had the exact same columns in the same order, filling any missing ones with 0. This is crucial for making predictions on the test set.

4 Exploratory Data Analysis (EDA)

This analysis was performed on the train.csv data.

4.1 Univariate Analysis

4.1.1 Numerical Features

Looking at the histograms for the numerical features (Figure 1), a few things stood out:

- **Age:** The distribution is right-skewed; most people are in the 20-25 age range.
- **Height & Weight:** Both show somewhat bimodal distributions (two peaks), which often happens when you have a balanced mix of genders.
- **Lifestyle:** Most individuals report around 3 main meals (`Number_of_Main_Meals`), about 2 liters of water (`Daily_Water_Intake`), and low physical activity (`Physical_Activity_Frequency` shows spikes near 0). `Vegetable_Consumption_Frequency` peaks around 2 and 3.

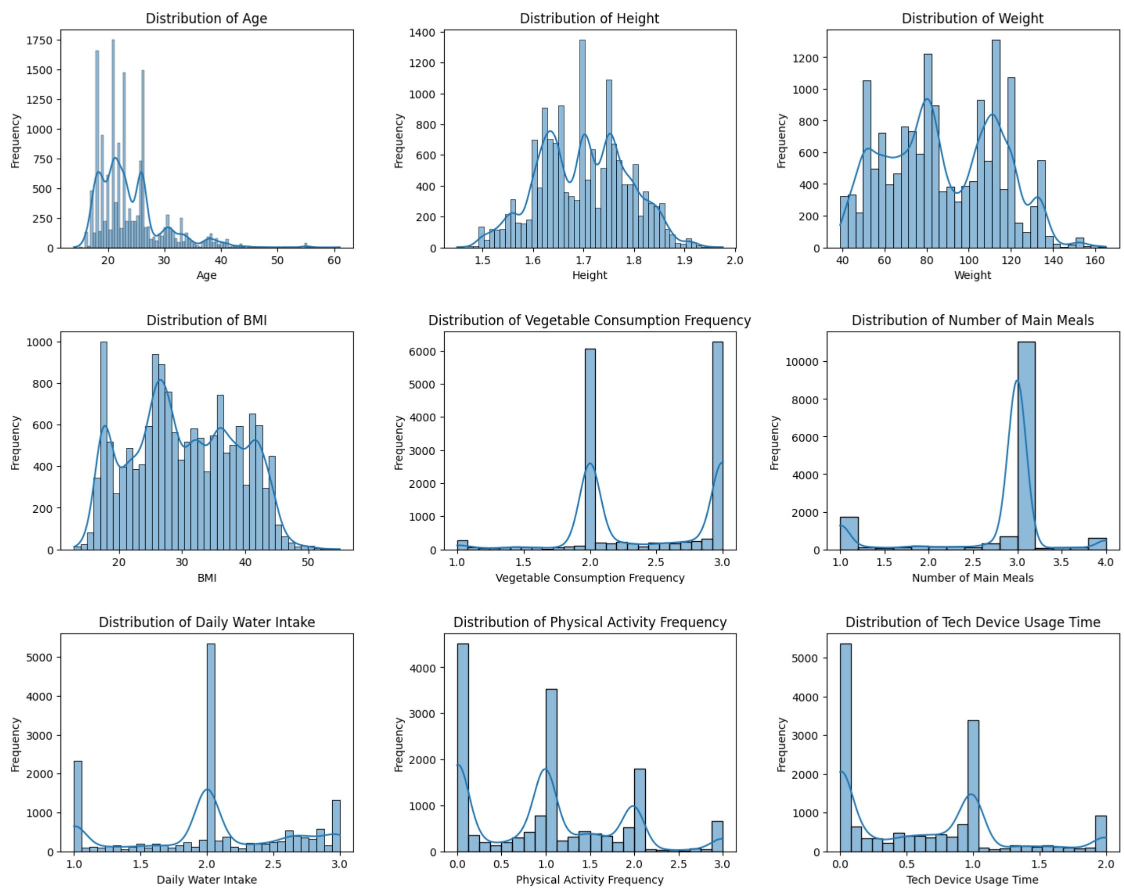

Figure 1: Distributions of Numerical Features (including engineered BMI).

4.1.2 Categorical Features

For the categorical data, the count plots (Figure 2) showed:

- **High Prevalence:** `Family_History_Overweight` ('yes') and `High_Caloric_Food_Consumption` ('yes') are overwhelmingly common.

- **Imbalance:** Very few people identify as `SMOKE` ('yes') or practice `Calorie_Consumption_Monitoring` ('yes').
- **Transport:** `Public_Transportation` is the most dominant mode.

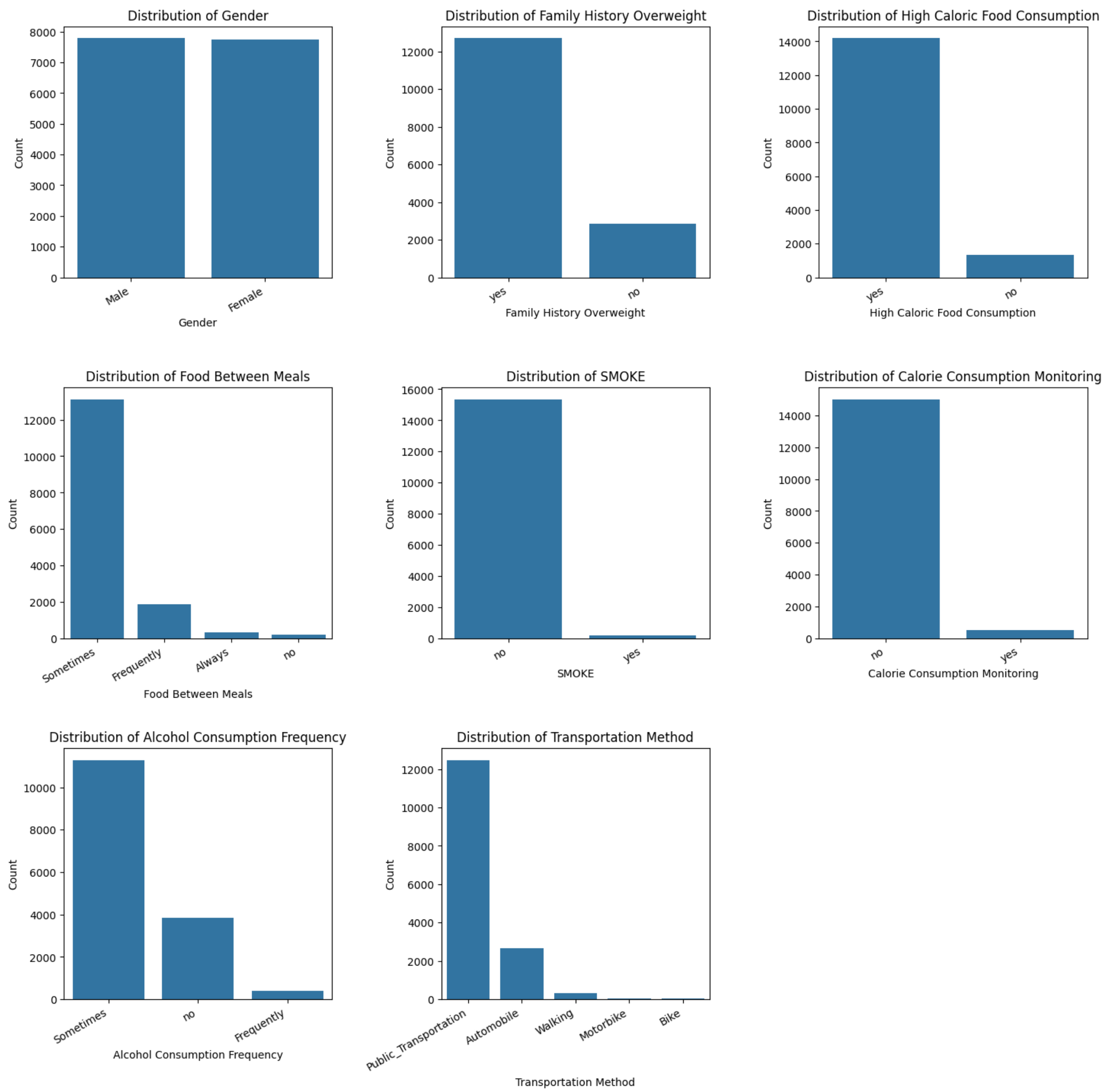

Figure 2: Distributions of Categorical Features.

4.2 Bivariate Analysis

4.2.1 Numerical-Numerical Correlation

The correlation heatmap (Figure 3) shows how the numerical features relate to each other. As you'd expect, `BMI` has a very strong positive (**0.94**) correlation with `Weight` and a low (**0.10**) one with `Height`. `Weight` and `Height` themselves have a moderate positive correlation (0.42). Most other features are weakly correlated, which is good—it means they're all providing independent information to the model.

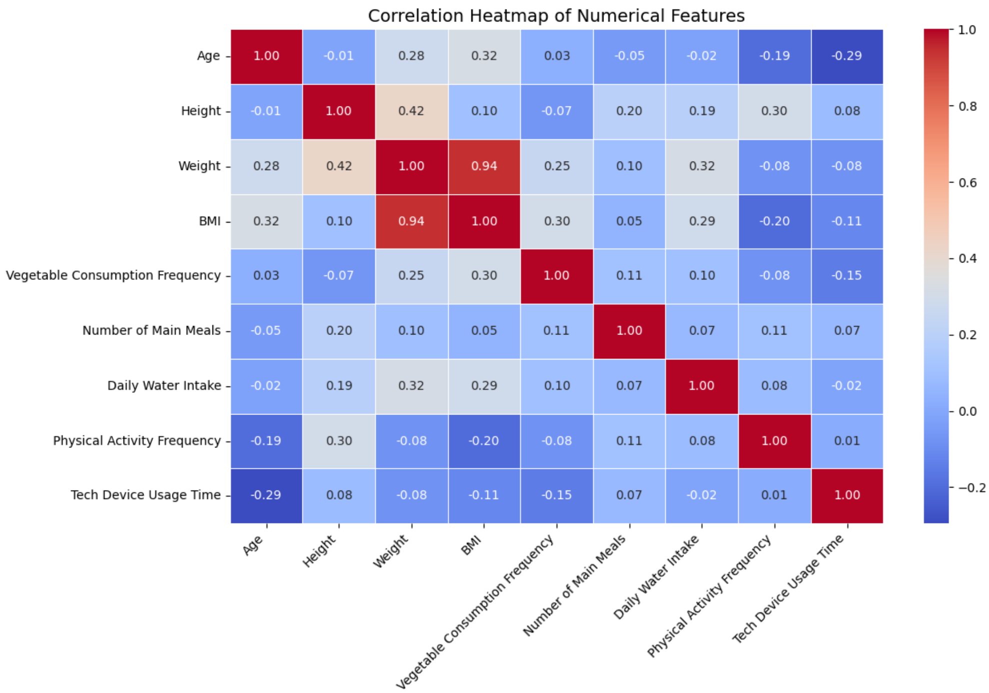

Figure 3: Correlation Heatmap of Numerical Features.

4.2.2 Categorical vs. Numerical

The box plots (Figure 4) were great for spotting relationships:

- **Gender vs. Biometrics:** Males show higher median `Height` and `Weight`, but the median `BMI` is slightly higher for Females in this training data.
- **History vs. BMI:** Individuals *with* a family history of overweight have a visibly higher median `BMI`.
- **Transport vs. Activity:** `Walking`, `Motorbike`, and `Bike` transportation methods correlate with significantly higher physical activity compared to Public Transport or Automobile.

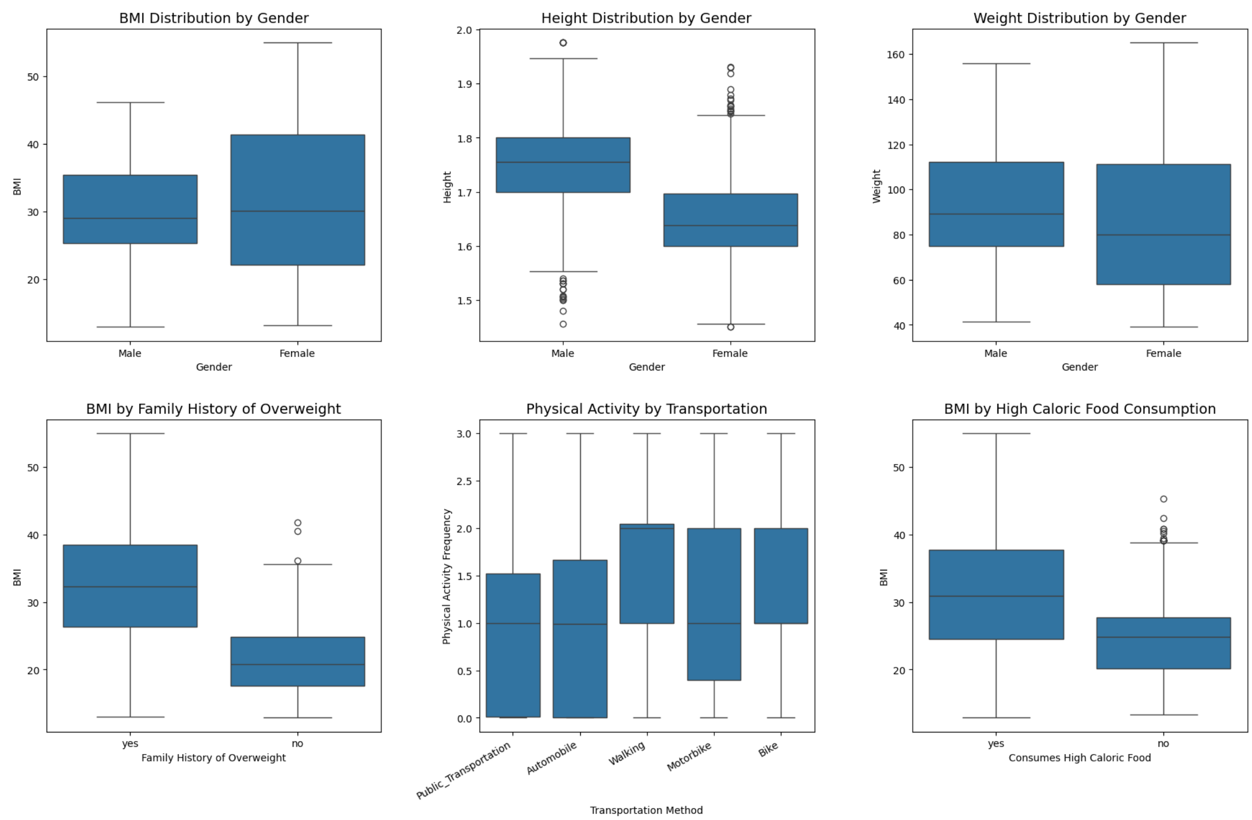

Figure 4: Bivariate Analysis: Categorical vs. Numerical Features.

4.2.3 Categorical vs. Categorical

Finally, the grouped count plots (Figure 5) showed a strong link between `Family_History_Overweight` ('yes') and `High_Caloric_Food_Consumption` ('yes'), which suggests a connection between family habits and personal ones. Both genders show a high tendency towards high caloric food consumption.

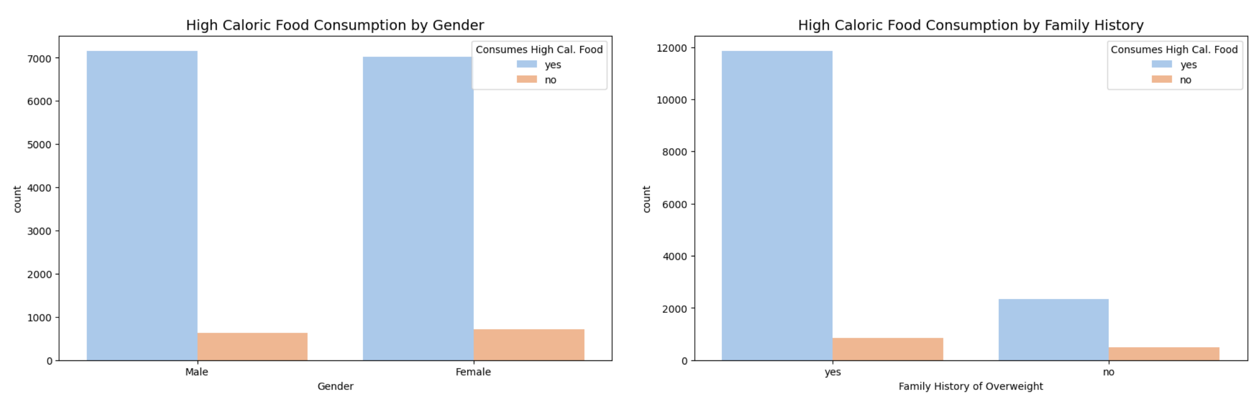

Figure 5: Bivariate Analysis: Categorical vs. Categorical Features.

5 Modeling Methodology

We set up a solid model to try and get the best accuracy possible using the `train.csv` data.

5.1 Model Choice: XGBoost

We chose the **XGBoost Classifier** (`XGBClassifier`) as our main model. It's a go-to for this kind of structured, tabular data, and for good reason:

- **Performance:** It's famous for winning data science competitions on tabular data, just like this dataset.
- **Regularization:** It has built-in features (`reg_alpha` and `reg_lambda`) to prevent overfitting, which is a common problem.
- **Handles Complexity:** It's great at finding complex, non-linear relationships and feature interactions automatically.
- **Speed:** It's optimized for performance and can use all CPU cores, which makes tuning and training much faster.
- **Tunability:** It has a lot of hyperparameters to tweak, which makes it perfect for an auto-tuning library like Optuna.

5.2 Hyperparameter Tuning with Optuna

Finding the best "settings" (hyperparameters) for a model is crucial. Manually guessing them is inefficient. We used **Optuna**, a modern tuning library, to automate this. Optuna performs a "smart search": it runs a trial with one set of parameters, sees how well it does, and then uses that information to make a more educated guess for the next trial. This is much faster than just testing random combinations.

We let it run for 35 trials to find the combination that gave the best 5-fold cross-validated accuracy. The search space included:

- `n_estimators`: [500, 2500]
- `max_depth`: [3, 10]
- `learning_rate`: [0.01, 0.3] (log scale)
- `subsample`, `colsample_bytree`: [0.5, 1.0]
- `gamma`, `reg_alpha`, `reg_lambda`, `min_child_weight`:[[0, 5], [0, 5], [0, 5],[1, 10]]

5.3 Final Training with 10-Fold Ensembling

Once Optuna found the best parameters, we used them to train the final model. We used a **10-fold Stratified K-Fold** strategy to make the model more stable and reliable.

1. The training data was split into 10 "folds."
2. We trained 10 separate XGBoost models. Each model was trained on 9 folds and validated on the 1 fold left out.

3. The final prediction for the test set was an **average** of the predictions from all 10 models. This is usually more accurate than just one model.
4. We collected the validation predictions (Out-of-Fold, or OOF) to get a single, trustworthy CV accuracy score.

5.4 Alternative Model: Random Forest with GridSearchCV

Just to have a comparison, we also trained a **Random Forest Classifier**. For this one, we used **GridSearchCV**, which is a more traditional tuning method. It performs an "exhaustive" or "brute-force" search: it simply tries every single combination of parameters we define in a grid.

The grid search ran a 5-fold cross-validation on the 80% training split, searching over parameters such as:

- `n_estimators`: [200, 400]
- `max_depth`: [10, 15]
- `min_samples_split`: [2, 5]
- `min_samples_leaf`: [1, 2]
- `max_features`: ["sqrt", "log2"]

This model served as a strong baseline to see if the more complex XGBoost setup was actually worth it.

6 Results

Both models actually did really well on this task.

6.1 XGBoost + Optuna (Primary Model)

The Optuna tuning for XGBoost came up with these parameters (or something close to them):

```
Best Parameters Found:
{'n_estimators': 1811, 'max_depth': 6,
 'learning_rate': 0.028, 'subsample': 0.55,
 'colsample_bytree': 0.51, 'gamma': 1.03, ...}
```

After training on all 10 folds using the full training data, the final model achieved a:

Final Mean Out-of-Fold CV Accuracy: 0.9148

The aggregated test set predictions were saved to `/mnt/drive/MyDrive/Obesity Dataset/submission_optuna_xg`

6.2 Random Forest + GridSearchCV (Baseline Model)

The Random Forest model was first tuned with `GridSearchCV` on an 80/20 split of the training data. The best parameters found were then used to train the final model using 5-fold cross-validation on the full training data. This final 5-fold CV-trained Random Forest model achieved a:

Final Mean Out-of-Fold CV Accuracy: 0.8942

The final test set predictions from this model were saved to `/mnt/drive/MyDrive/Obesity Dataset/submission_r`
The performance of this tuned model serves as a robust baseline, which confirmed that the XGBoost pipeline was the superior approach for this task.

7 Conclusion

Overall, this project was a success. We built a complete model to classify obesity, starting with really high-quality, clean data. The EDA helped uncover some clear links, like family history and BMI.

The main strategy of using **XGBoost** with **Optuna** tuning and **K-Fold** ensembling paid off, hitting a final accuracy of over 91%. The baseline **Random Forest** (tuned with 'GridSearchCV') also did great with over 89% accuracy, confirming that tree-based models were the right way to go for this dataset. This model is a really strong starting point for this problem.

Notebook Link: link   Github Link: link

A Analysis Script

```
# Analysis part
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from google.colab import drive

# Suppress warnings for cleaner output
warnings.filterwarnings("ignore")

# --- Mount Drive ---
# This part is for Google Colab.
drive.mount('/mnt/drive')
try:
    # Load train.csv instead of test.csv
    df_train = pd.read_csv('/mnt/drive/MyDrive/Obesity Dataset/train.csv')
    print(f"    Data Loaded: Shape: {df_train.shape}") # Use df_train.shape
except FileNotFoundError:
    # Fallback for local environment if drive mount fails or isn't used
    try:
        # Load train.csv locally
        df_train = pd.read_csv('train.csv')
        print(f"    Data Loaded locally: Shape: {df_train.shape}") # Use df_train.shape
    except FileNotFoundError:
        print("Error: train.csv not found in Google Drive path or local folder.")
        exit()

# ======================================================
# === 1. PRE-EDA: DATA QUALITY & STRUCTURE CHECKS ===
# ======================================================

print("\n--- 1. Data Types and Non-Null Counts ---")
# Use .info() to check data types and get a quick count of non-null values
df_train.info() # Use df_train

print("\n--- 2. Missing Value Check (Explicit Count) ---")
# Summing up all null values for each column
missing_values = df_train.isnull().sum() # Use df_train
print("Missing values per column:")
print(missing_values[missing_values > 0] if missing_values.sum() > 0 else "No missing values found.")
if missing_values.sum() == 0:
    print("\ n   Status: No missing values found.")
else:
    print(f"\  n     Status: Found a total of {missing_values.sum()} missing values.")

print("\n--- 3. Duplicate Rows Check ---")
duplicate_rows = df_train.duplicated().sum() # Use df_train
print(f"Total duplicate rows: {duplicate_rows}")
if duplicate_rows == 0:
    print("\ n   Status: No duplicate rows found.")
else:
    print(f"\  n     Status: Found {duplicate_rows} duplicate rows.")

print("\n--- 4. Numerical Feature Statistics ---")
# .describe() on numerical columns
print(df_train.describe()) # Use df_train

print("\n--- 5. Categorical Feature Statistics ---")
# .describe() on 'object' type columns (including the target 'NObeyesdad' if it's object type)
print(df_train.describe(include=['object'])) # Use df_train

print("\  n     Pre-EDA checks complete. Starting main EDA...")
```

```
# ======================================================
# === 2. MAIN EDA: ANALYSIS & VISUALIZATION ===
# ======================================================

# --- 2a. Rename Attributes for Clarity ---
column_rename_map = {
    'family_history_with_overweight': 'Family_History_Overweight',
    'FAVC': 'High_Caloric_Food_Consumption',
    'FCVC': 'Vegetable_Consumption_Frequency',
    'NCP': 'Number_of_Main_Meals',
    'CAEC': 'Food_Between_Meals',
    'CH2O': 'Daily_Water_Intake',
    'SCC': 'Calorie_Consumption_Monitoring',
    'FAF': 'Physical_Activity_Frequency',
    'TUE': 'Tech_Device_Usage_Time',
    'CALC': 'Alcohol_Consumption_Frequency',
    'MTRANS': 'Transportation_Method'
    # NObeyesdad is the target, usually not renamed here unless desired
}
df_train.rename(columns=column_rename_map, inplace=True) # Use df_train
print("\ n   Attributes renamed for better readability.")


# --- 2b. Feature Engineering: BMI ---
df_train['BMI'] = df_train['Weight'] / (df_train['Height'] ** 2) # Use df_train
print("    Created 'BMI' feature.")

# --- 2c. Define Column Types with New Names ---
# Note: Ensure these lists don't include the target variable 'NObeyesdad'
numerical_cols = [
    'Age', 'Height', 'Weight', 'BMI',
    'Vegetable_Consumption_Frequency',
    'Number_of_Main_Meals',
    'Daily_Water_Intake',
    'Physical_Activity_Frequency',
    'Tech_Device_Usage_Time'
]

categorical_cols = [
    'Gender',
    'Family_History_Overweight',
    'High_Caloric_Food_Consumption',
    'Food_Between_Meals',
    'SMOKE',
    'Calorie_Consumption_Monitoring',
    'Alcohol_Consumption_Frequency',
    'Transportation_Method'
]
# Define target column if needed for specific plots later
target_col = 'NObeyesdad'

print("    Defined numerical and categorical columns with new names.")


# --- 2d. Univariate Analysis: Numerical Features ---
print("\  n     Generating histograms for numerical features...")
plt.figure(figsize=(15, 12))
for i, col in enumerate(numerical_cols):
    plt.subplot(3, 3, i + 1)
    sns.histplot(df_train[col], kde=True) # Use df_train
    plt.title(f'Distribution of {col.replace("_", " ")}', fontsize=12)
    plt.xlabel(col.replace("_", " "))
    plt.ylabel('Frequency')
plt.tight_layout(pad=3.0)
# Save with the name matching the report \includegraphics command
plt.savefig("Univariate Non Categorical.png")
print("    Saved Univariate Non Categorical.png")
```

```python


# --- 2e. Univariate Analysis: Categorical Features ---
print("\n    Generating bar charts for categorical features...")
plt.figure(figsize=(15, 15))
for i, col in enumerate(categorical_cols):
    plt.subplot(3, 3, i + 1)
    # Ensure df_train[col] exists and is categorical before plotting
    if col in df_train.columns:
        order = df_train[col].value_counts().index
        sns.countplot(data=df_train, x=col, order=order) # Use df_train
        plt.title(f'Distribution of {col.replace("_", " ")}', fontsize=12)
        plt.xlabel(col.replace("_", " "))
        plt.ylabel('Count')
        plt.xticks(rotation=30, ha='right') # Rotate labels for readability
    else:
        print(f"Warning: Column '{col}' not found in df_train.")
plt.tight_layout(pad=3.0)
# Save with the name matching the report \includegraphics command
plt.savefig("Univariate Categorical.png")
print("    Saved Univariate Categorical.png")


# --- 2f. Bivariate Analysis: Numerical Correlation ---
print("\n    Generating correlation heatmap...")
plt.figure(figsize=(12, 8))
# Create clean labels for the heatmap axes
renamed_numerical_labels = [label.replace("_", " ") for label in numerical_cols]
# Ensure all columns exist before calculating correlation
valid_numerical_cols = [col for col in numerical_cols if col in df_train.columns]
corr_matrix = df_train[valid_numerical_cols].corr() # Use df_train

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5,
            xticklabels=[label.replace("_", " ") for label in valid_numerical_cols], # Use valid labels
            yticklabels=[label.replace("_", " ") for label in valid_numerical_cols]) # Use valid labels

plt.title('Correlation Heatmap of Numerical Features', fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
# Save with the name matching the report \includegraphics command
plt.savefig("Correlation.png")
print("    Saved Correlation.png")


# --- 2g. Bivariate Analysis: Categorical vs. Numerical ---
print("\n    Generating Bivariate Box Plots (Categorical vs. Numerical)...")
plt.figure(figsize=(18, 12))

# Plot 1: Gender vs. BMI
plt.subplot(2, 3, 1)
sns.boxplot(data=df_train, x='Gender', y='BMI') # Use df_train
plt.title('BMI Distribution by Gender', fontsize=14)

# Plot 2: Gender vs. Height
plt.subplot(2, 3, 2)
sns.boxplot(data=df_train, x='Gender', y='Height') # Use df_train
plt.title('Height Distribution by Gender', fontsize=14)

# Plot 3: Gender vs. Weight
plt.subplot(2, 3, 3)
sns.boxplot(data=df_train, x='Gender', y='Weight') # Use df_train
plt.title('Weight Distribution by Gender', fontsize=14)

# Plot 4: Family History vs. BMI
plt.subplot(2, 3, 4)
```

```python
sns.boxplot(data=df_train, x='Family_History_Overweight', y='BMI') # Use df_train
plt.title('BMI by Family History of Overweight', fontsize=14)
plt.xlabel('Family History of Overweight')

# Plot 5: Transportation Method vs. Physical Activity
plt.subplot(2, 3, 5)
sns.boxplot(data=df_train, x='Transportation_Method', y='Physical_Activity_Frequency') # Use df_train
plt.title('Physical Activity by Transportation', fontsize=14)
plt.xlabel('Transportation Method')
plt.ylabel('Physical Activity Frequency')
plt.xticks(rotation=30, ha='right')

# Plot 6: High Caloric Food Consumption vs. BMI
plt.subplot(2, 3, 6)
sns.boxplot(data=df_train, x='High_Caloric_Food_Consumption', y='BMI') # Use df_train
plt.title('BMI by High Caloric Food Consumption', fontsize=14)
plt.xlabel('Consumes High Caloric Food')

plt.tight_layout(pad=3.0)
# Save with the name matching the report \includegraphics command
plt.savefig("Bivariate non Categorical.png")
print("    Saved Bivariate non Categorical.png")


# --- 2h. Bivariate Analysis: Categorical vs. Categorical ---
print("\  n     Generating Bivariate Count Plots (Categorical vs. Categorical)...")
plt.figure(figsize=(18, 6))

# Plot 1: Gender vs. High Caloric Food Consumption
plt.subplot(1, 2, 1)
sns.countplot(data=df_train, x='Gender', hue='High_Caloric_Food_Consumption', palette='pastel') # Use df_train
plt.title('High Caloric Food Consumption by Gender', fontsize=14)
plt.legend(title='Consumes High Cal. Food')

# Plot 2: Family History vs. High Caloric Food Consumption
plt.subplot(1, 2, 2)
sns.countplot(data=df_train, x='Family_History_Overweight', hue='High_Caloric_Food_Consumption', palette='pastel') # Use df_train
plt.title('High Caloric Food Consumption by Family History', fontsize=14)
plt.xlabel('Family History of Overweight')
plt.legend(title='Consumes High Cal. Food')

plt.tight_layout(pad=3.0)
# Save with the name matching the report \includegraphics command
plt.savefig("Bivariate Categorical.png")
print("    Saved Bivariate Categorical.png")

# --- Optional: Analysis involving the target variable ---
# Example: Distribution of the target variable
if target_col in df_train.columns:
    print(f"\  n     Generating count plot for the target variable '{target_col}'...")
    plt.figure(figsize=(10, 6))
    sns.countplot(data=df_train, y=target_col, order = df_train[target_col].value_counts().index) # Use df_train
    plt.title(f'Distribution of Obesity Levels ({target_col})')
    plt.xlabel('Count')
    plt.ylabel('Obesity Level')
    plt.tight_layout()
    plt.savefig("train_target_distribution.png")
    print("    Saved train_target_distribution.png")

    # Example: BMI distribution per Obesity Level
    print(f"\  n     Generating box plot for BMI vs '{target_col}'...")
    plt.figure(figsize=(12, 8))
    sns.boxplot(data=df_train, x='BMI', y=target_col, order=sorted(df_train[target_col].unique())) # Use df_train
```

```python
    plt.title(f'BMI Distribution by Obesity Level ({target_col})')
    plt.xlabel('BMI')
    plt.ylabel('Obesity Level')
    plt.tight_layout()
    plt.savefig("train_bmi_vs_target.png")
    print("    Saved train_bmi_vs_target.png")


print("\  n     All EDA tasks complete. Check the saved .png files for all plots.")
```

Listing 1: Full EDA and Analysis Script (using train.csv)

B Modeling Script 1: XGBoost + Optuna

```python
# ==============================================
#        Obesity Classification     Optuna + XGBoost + K-Fold + Save Predictions (0.91294)
# ==============================================
!pip install optuna --quiet
import pandas as pd
import numpy as np
import optuna
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from google.colab import drive
import warnings
warnings.filterwarnings("ignore")

# === 1       Mount Drive & Load Data ===
drive.mount('/mnt/drive')
train_df = pd.read_csv('/mnt/drive/MyDrive/Obesity Dataset/train.csv')
test_df = pd.read_csv('/mnt/drive/MyDrive/Obesity Dataset/test.csv')

print(f"    Data Loaded: Train shape: {train_df.shape}, Test shape: {test_df.shape}")

# === 2       Identify Target Column Automatically ===
TARGET = "NObeyesdad"
if TARGET not in train_df.columns:
    possible_targets = [col for col in train_df.columns if col not in test_df.columns and col.lower() != "id"]
    if len(possible_targets) == 1:
        TARGET = possible_targets[0]
    else:
        print("       Possible target columns found:")
        print(possible_targets)
        raise KeyError("Target column not found     please verify manually.")
print(f"    Target column detected: {TARGET}")

# === 3       Prepare Train/Test Data ===
X = train_df.drop(columns=["id", TARGET])
y = train_df[TARGET]
test_ids = test_df["id"].copy() # Make sure to copy IDs before dropping

# Encode target labels
le = LabelEncoder()
y_enc = le.fit_transform(y)

# One-hot encode categorical features
cat_cols = X.select_dtypes(include=["object", "category"]).columns.tolist()
X_enc = pd.get_dummies(X, columns=cat_cols)
test_enc = pd.get_dummies(test_df.drop(columns=["id"]), columns=cat_cols) # Drop ID from test set

# Align train/test features
X_aligned, test_aligned = X_enc.align(test_enc, join="left", axis=1, fill_value=0)

# Scale numeric features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_aligned)
test_scaled = scaler.transform(test_aligned)

# === 4       Optuna Hyperparameter Tuning ===
def objective(trial):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 500,2500),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3, log=True),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
```

```
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "gamma": trial.suggest_float("gamma", 0, 5),
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 10),
        "reg_alpha": trial.suggest_float("reg_alpha", 0.0, 5.0),
        "reg_lambda": trial.suggest_float("reg_lambda", 0.0, 5.0),
        "objective": "multi:softprob",
        "num_class": len(le.classes_),
        "eval_metric": "mlogloss",
        "n_jobs": -1,
        "tree_method": "hist",
        "random_state": 42,
        "verbosity": 0
    }

    model = XGBClassifier(**params)
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    scores = cross_val_score(model, X_scaled, y_enc, cv=cv, scoring="accuracy", n_jobs=-1)
    return scores.mean()

print("       Starting Optuna hyperparameter tuning...")
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=35, timeout=3600)

print("\ n   Best Parameters Found:")
print(study.best_params)
print(f"Best CV Accuracy: {study.best_value:.4f}")

# === 5       Final Training with Best Params + K-Fold ===
best_params = study.best_params
best_params.update({
    "objective": "multi:softprob",
    "num_class": len(le.classes_),
    "eval_metric": "mlogloss",
    "n_jobs": -1,
    "tree_method": "hist",
    "random_state": 42,
})

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
oof_preds = np.zeros(len(X_scaled))
test_preds_agg = np.zeros((len(test_scaled), len(le.classes_)))

print("\  n     Starting Final 10-Fold Training...")
for fold, (train_idx, val_idx) in enumerate(skf.split(X_scaled, y_enc)):
    model = XGBClassifier(**best_params)
    model.fit(X_scaled[train_idx], y_enc[train_idx])
    preds_val = np.argmax(model.predict_proba(X_scaled[val_idx]), axis=1)
    oof_preds[val_idx] = preds_val
    test_preds_agg += model.predict_proba(test_scaled) / skf.n_splits
    acc = accuracy_score(y_enc[val_idx], preds_val)
    print(f"Fold {fold+1} Accuracy: {acc:.4f}")

final_acc = accuracy_score(y_enc, oof_preds)
print(f"\  n     Final Mean CV Accuracy: {final_acc:.4f}")

# === 6       Final Predictions + Save to CSV ===
final_test_preds = np.argmax(test_preds_agg, axis=1)
pred_labels = le.inverse_transform(final_test_preds.astype(int))

submission = pd.DataFrame({
    "id": test_ids,
    TARGET: pred_labels
})

save_path = "/mnt/drive/MyDrive/Obesity Dataset/submission_optuna_xgb.csv"
submission.to_csv(save_path, index=False)
print(f"\ n   Saved final predictions to: {save_path}")
```

```
print(submission.head())
```

Listing 2: ML Pipeline Script 1 (XGBoost + Optuna)

C Modeling Script 2: Random Forest + GridSearchCV

```

# ==============================================
#  Obesity Classification     Random Forest + GridSearchCV
# ==============================================

import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
from google.colab import drive
import warnings
warnings.filterwarnings("ignore")

# === 1 Mount Drive & Load Data ===
drive.mount('/mnt/drive')
train_df = pd.read_csv('/mnt/drive/MyDrive/Obesity Dataset/train.csv')
test_df = pd.read_csv('/mnt/drive/MyDrive/Obesity Dataset/test.csv')
sample_df = pd.read_csv('/mnt/drive/MyDrive/Obesity Dataset/sample_submission.csv')

print(f" Data Loaded: Train shape: {train_df.shape}, Test shape: {test_df.shape}")

# === 2 Preserve Test IDs ===
test_ids = test_df["id"].copy()

# === 3 Prepare Train/Test Data ===
TARGET = "WeightCategory"
X = train_df.drop(columns=["id", TARGET])
y = train_df[TARGET]

# Encode target
le = LabelEncoder()
y_enc = le.fit_transform(y)

# One-hot encode categorical features
cat_cols = X.select_dtypes(include=["object", "category"]).columns.tolist()
X_enc = pd.get_dummies(X, columns=cat_cols)
test_enc = pd.get_dummies(test_df.drop(columns=["id"]), columns=cat_cols)

# Align train/test features
X_aligned, test_aligned = X_enc.align(test_enc, join="left", axis=1, fill_value=0)

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_aligned)
test_scaled = scaler.transform(test_aligned)

# === 4 Train/Validation Split ===
X_tr, X_val, y_tr, y_val = train_test_split(
    X_scaled, y_enc, test_size=0.2, random_state=42, stratify=y_enc
)

# === 5 Define Random Forest + Expanded GridSearchCV ===
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

param_grid = {
    "n_estimators": [200, 400],
    "max_depth": [10, 15, None],
    "min_samples_split": [2, 5],
    "min_samples_leaf": [1, 2],
    "max_features": ["sqrt", "log2"],
    "bootstrap": [True],
    "criterion": ["gini"],
    "class_weight": [None, "balanced"]
```

```
}


grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=5,                   # 5-fold cross validation
    scoring="accuracy",
    n_jobs=-1,
    verbose=2
)

print("\n Running Expanded GridSearchCV...")
grid_search.fit(X_tr, y_tr)

print("\n Best parameters found:")
print(grid_search.best_params_)
print(f" Best CV Accuracy: {grid_search.best_score_:.4f}")

# Use best model
best_rf = grid_search.best_estimator_

# === 6 Evaluate on Validation Set ===
val_pred = best_rf.predict(X_val)
print(f"\n Validation Accuracy: {accuracy_score(y_val, val_pred):.4f}")
print("\nClassification Report:")
print(classification_report(y_val, val_pred, target_names=le.classes_))

# === 7 Retrain on Full Data ===
best_rf.fit(X_scaled, y_enc)

# === 8 Predict on Test Set ===
test_pred = best_rf.predict(test_scaled)
pred_labels = le.inverse_transform(test_pred)

# === 9 Build & Save Submission ===
submission = pd.DataFrame({
    "id": test_ids,
    TARGET: pred_labels
})

# Reorder columns to match sample submission if required
if set(sample_df["id"]) == set(submission["id"]):
    submission = sample_df[["id"]].merge(submission, on="id", how="left")

save_path = "/mnt/drive/MyDrive/Obesity Dataset/submission_rf_grid_expanded.csv"
submission.to_csv(save_path, index=False)
print(f"\ n   Saved submission file to: {save_path}")
print(submission.head())
```

Listing 3: ML Pipeline Script 2 (Random Forest + GridSearchCV)