Personalized Obesity Risk Analysis and Prediction Based on Demographic, Physical, and Lifestyle Data

Author: Wenqing Zhang & Xudong He

Team: Kun

1. Executive Summary

Obesity is a complex disease resulting from the interplay of physical characteristics, lifestyle habits, and genetic predispositions. It poses severe health risks to individuals while imposing a substantial economic burden on healthcare systems and social welfare. To address this challenge, this project aims to develop a predictive model that evaluates individual obesity risk based on dietary habits, physical attributes, and lifestyle data. By conducting thorough data analysis, the project seeks to identify the primary contributing factors and underlying causes of obesity, thereby enabling the formulation of effective early prevention strategies and personalized health recommendations to reduce obesity rates.

The project leverages existing datasets and incorporates machine learning techniques for classification, analysis, and visualization, providing comprehensive predictive and prescriptive insights focused on addressing obesity.

1.1 Decisions to be Impacted

The predictive model for obesity has the potential to influence critical decision-making in several key domains. Firstly, in public health, governments can use the model to accurately identify high-risk populations and design targeted health interventions. For instance, implementing community health education programs and promoting healthier dietary environments can significantly reduce obesity prevalence, easing the strain on public healthcare resources. Furthermore, insights derived from the model can support policymakers in setting priorities and allocating resources more effectively to create healthier living environments.Research has shown that machine learning models can effectively identify the relative importance of lifestyle factors contributing to obesity, providing actionable insights for targeted interventions [1].

Secondly, healthcare institutions can benefit greatly from the model. By utilizing its predictive capabilities, hospitals can develop personalized intervention plans for high-risk individuals. This precision-driven health management approach not only optimizes resource allocation but also enhances the effectiveness of interventions, reducing the costs associated with obesity-related treatments. The model serves as a valuable tool for hospitals to better address the complexities of obesity management.

Lastly, for individuals, the predictive model offers real-time feedback on their health status and lifestyle choices. By integrating with smart devices such as weight scales and health monitoring systems, users can monitor their progress, receive tailored recommendations, and make informed adjustments to their routines. This personalized approach fosters greater health awareness and supports users in achieving sustainable health goals.

1.2 Motivation

The motivations driving this project stem from the escalating severity of obesity and its multifaceted impact. Obesity is not only a significant contributor to chronic diseases but also a primary driver of increasing healthcare costs. This project aims to leverage data-driven solutions to tackle this pressing issue.

From a health perspective, obesity is a major risk factor for chronic diseases such as cardiovascular conditions and diabetes, with rising global prevalence. Developing a predictive model allows for early identification of high-risk individuals, enabling timely interventions that reduce the incidence of related health conditions. This capability is crucial for advancing global health management.

From an economic perspective, the costs associated with obesity treatment and management continue to account for a growing share of healthcare expenditures. Preventive measures, guided by the predictive model, are significantly more cost-effective than treatments. By providing actionable insights, the model empowers governments and healthcare institutions to implement interventions that lower healthcare costs and improve resource efficiency.

Furthermore, technological advancements have facilitated the implementation of this project. With the rapid development of machine learning and data science, the barriers to constructing complex predictive models have diminished. This creates new opportunities to address obesity challenges through innovative methods and tools.

1.3 Business Value

Our predictive model offers significant commercial value in improving individual health. By predicting health risk probabilities, the model alerts individuals to unhealthy lifestyle habits, encouraging them to adopt healthier routines. For example, the model monitors sleep quality, helping individuals enhance their sleep and boost metabolism to improve overall health. Additionally, the system analyzes and optimizes the nutritional value of each meal. By setting achievable and specific health goals, we aim to help each individual maintain a healthy weight and improve their quality of life.

The predictive model holds substantial value in aiding governments to reduce expenditures on public health improvement. Through early intervention and identification of potential obesity cases, the model can lower the cost of obesity prevention since preventive measures are generally less expensive than treatment. Governments can prioritize proactive actions such as health education, beneficial social programs, and policies that create healthier living environments for residents. Using data predictions from our model, governments can effectively reduce obesity prevalence in communities, easing the burden on healthcare systems.

Another critical commercial value of this predictive model lies in reducing research and treatment costs associated with obesity. By leveraging various machine learning models and algorithms, the model effectively analyzes cleaned data to construct a relationship network between dietary habits, physical conditions, and obesity risk. This network assists medical researchers in uncovering underlying patterns among these features, enabling them to identify potential research opportunities at the intersection of obesity studies with fields like nutrition and epidemiology. Such comprehensive analysis fosters innovation in medical research, supporting more effective obesity prevention strategies.

Summary

This dataset plays a critical role in analyzing obesity risks, providing a solid foundation for machine learning model development through its comprehensive coverage of demographic characteristics, physical attributes, and lifestyle variables, along with clearly labeled target variables. With its diverse and balanced feature set, the dataset supports various practical applications. For instance, the model can generate personalized health recommendations based on individual dietary habits and exercise frequencies, enabling users to optimize their lifestyles. The balanced distribution of data enhances the model's ability to learn from different obesity categories, ensuring predictive accuracy. Furthermore, the dataset demonstrates significant potential in optimizing healthcare resources and informing public policy. Hospitals can leverage the model's insights to design tailored intervention plans for high-risk individuals, while governments can use analytical results to implement more effective public health policies. Overall, this dataset not only offers robust scalability but also serves as a foundational resource for research in other fields, such as nutrition and epidemiology, showcasing immense potential for broader applications.

2. Data Preprocessing

Data preprocessing is a critical step in machine learning projects, ensuring that raw data is suitable for model development and analysis. In this section, we describe the detailed steps of preprocessing, including data cleaning, outlier detection, and feature encoding, based on an in-depth observation and analysis of the dataset.

2.1 Data Assets

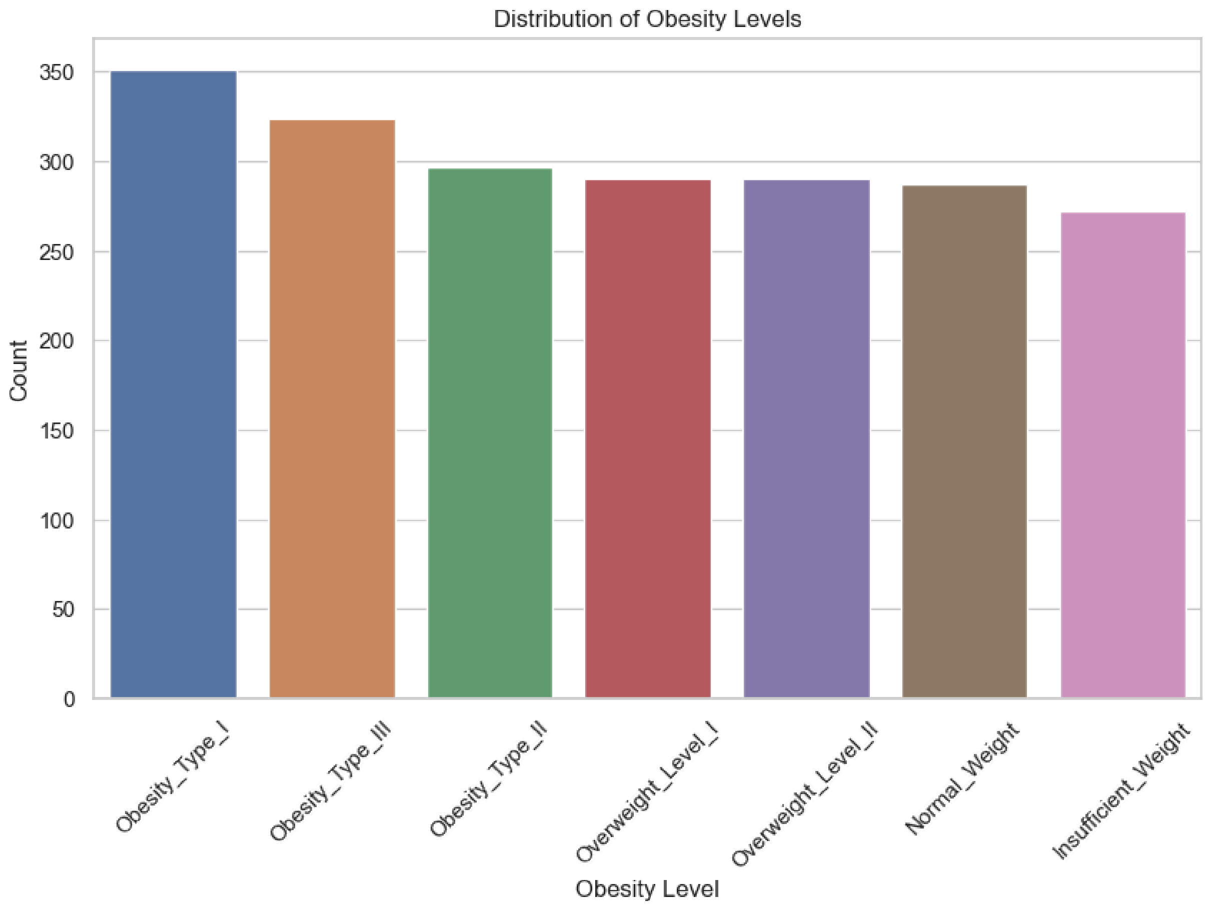
The dataset used in this project originates from Kaggle's Obesity Levels Dataset (link: Obesity Levels Dataset), which contains comprehensive individual features and behavioral data. It is an essential foundation for analyzing obesity risks. The dataset includes 2,111 records, covering individuals of different genders, ages, and lifestyles. With sufficient sample size and balanced data distribution, the dataset provides a robust basis for building predictive models. The target variable, "Obesity Level" (NObeyesdad), is categorized into seven classes, ranging from "Underweight" to "Morbid Obesity," clearly identifying the health status of individuals. Notably, the number of samples across different obesity levels is relatively balanced, with approximately 300 samples per category, ensuring the model learns features for each type comprehensively.

In [17]:
```
'''
Observe the official documentation of dataSet
'''
import pandas as pd
from scipy.io import arff
# Reading ARFF Files
file_path = '../DataSet/ObesityDataSet_raw_and_data_sinthetic.arff'
data, meta = arff.loadarff(file_path)

df = pd.DataFrame(data)
print("The shape of the dataset: ")
print(df.shape)
from IPython.display import display
print("The first five rows of the dataset:")
display(df.head())
```

```
The shape of the dataset:
(2111, 17)
The first five rows of the dataset:
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP
0	b'Female'	21.0	1.62	64.0	b'yes'	b'no'	2.0	3.0
1	b'Female'	21.0	1.52	56.0	b'yes'	b'no'	3.0	3.0
2	b'Male'	23.0	1.80	77.0	b'yes'	b'no'	2.0	3.0
3	b'Male'	27.0	1.80	87.0	b'no'	b'no'	3.0	3.0
4	b'Male'	22.0	1.78	89.8	b'no'	b'no'	2.0	1.0

In [18]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# String data in .arff files is usually in byte format, so it needs to be de
for col in df.select_dtypes(['object']).columns:
    df[col] = df[col].str.decode('utf-8')

# Make the chart background a white grid to facilitate data visualization ar
sns.set(style="whitegrid")

# Distribution analysis for the target variable
plt.figure(figsize=(10, 6))
# The order parameter sorts the target variable by category frequency
sns.countplot(data=df, x='NObeyesdad', order=df['NObeyesdad'].value_counts()
plt.title("Distribution of Obesity Levels")
plt.xticks(rotation=45)
plt.xlabel("Obesity Level")
plt.ylabel("Count")
plt.show()
```



The dataset comprises 17 feature variables, including demographic information, physical attributes, dietary habits, and lifestyle data, all of which are crucial for model development. For instance, gender and age variables can be used to analyze how obesity risks vary across different populations, as studies have shown that obesity risks increase significantly with age, and the influencing factors may differ between men and women. Height and weight variables directly relate to Body Mass Index (BMI), a critical indicator for evaluating obesity. During data preprocessing, BMI was calculated for each record and its strong correlation with obesity levels was verified. Furthermore, the dataset includes dietary habits such as eating speed, daily fruit and vegetable consumption, and frequency of high-calorie food intake. These variables are instrumental in identifying unhealthy eating patterns. Lifestyle variables like exercise frequency provide insights into physical activity levels, helping assess the relationship between exercise habits and obesity risks.

In [19]:
```
print("\n Dataset information: ")
print(df.info())
```

```
 Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Gender                          2111 non-null   object 
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object 
 5   FAVC                            2111 non-null   object 
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object 
 9   SMOKE                           2111 non-null   object 
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object 
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object 
 15  MTRANS                          2111 non-null   object 
 16  NObeyesdad                      2111 non-null   object 
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
None
```

The value of this dataset lies not only in offering a rich set of features for model development but also in its ability to support decision-making across multiple levels. For individual health management, the model can analyze dietary and exercise data to provide specific health recommendations, such as increasing exercise time or optimizing meal structures. Additionally, the model can issue warnings for high-risk individuals, prompting them to take measures to reduce obesity risks. In terms of healthcare resource allocation, the model serves as a tool for hospitals to efficiently identify high-risk patients, enabling them to develop personalized weight management plans and dynamically monitor health changes. By reducing unnecessary resource waste, this approach enhances the efficiency of healthcare resource utilization. Government agencies can also use data-driven insights to formulate health promotion policies, such as targeted dietary education for adolescents, and improve community health facilities to reduce overall obesity risks.

To ensure the dataset's usability for machine learning models, we performed comprehensive preprocessing, including cleaning missing values, handling outliers, and encoding categorical variables. Additionally, we introduced the BMI feature and normalized continuous variables (e.g., age and weight) to enhance model stability during training. These steps fully leveraged the dataset's potential, providing a reliable foundation for obesity risk analysis.

2.2 Data Observation and Analysis

2.2.1 **Correlation Between Variables**

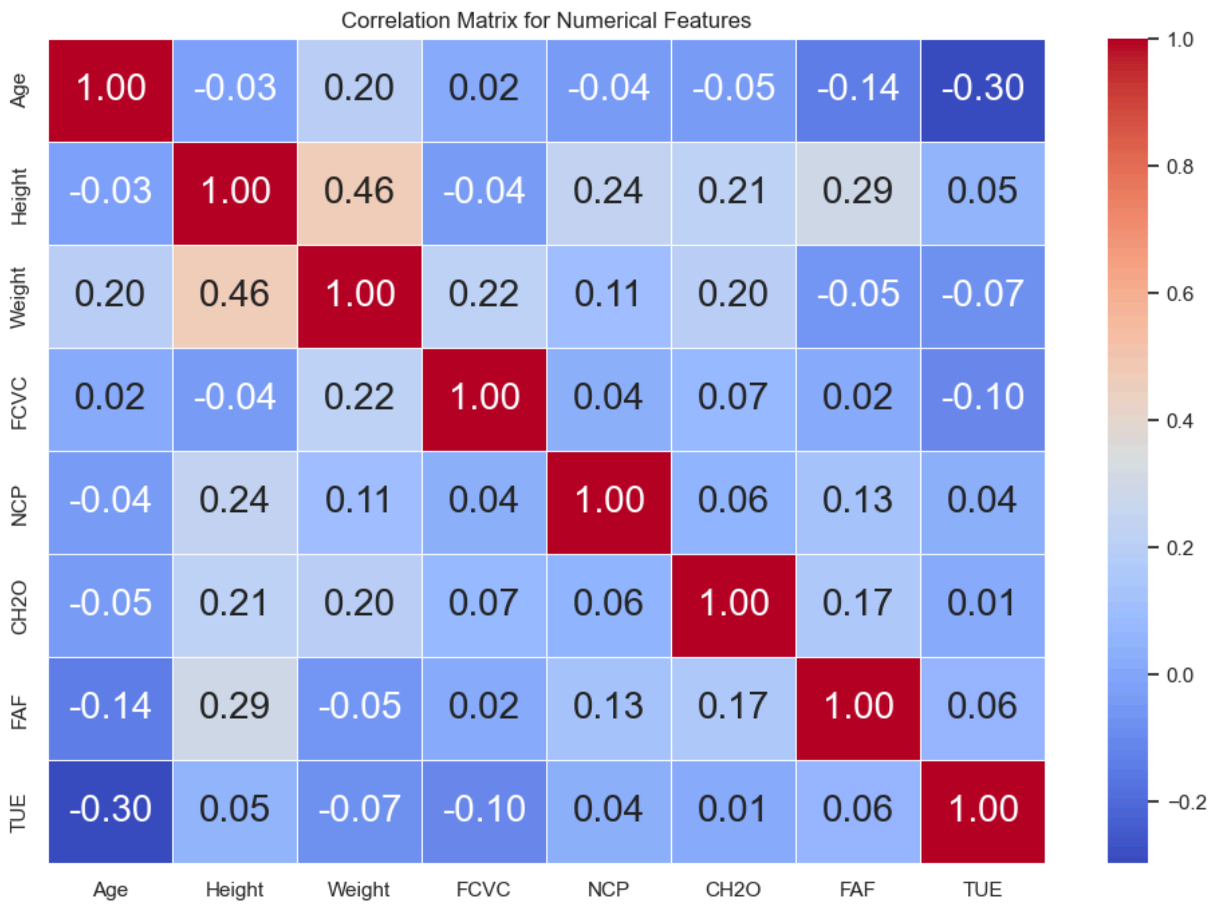
Before initiating preprocessing, we analyzed the correlation matrix to evaluate the relationships between numerical variables. The analysis revealed weak correlations among most variables. For instance, the correlation between Weight and Age is only 0.20, while that between Height and CH2O is 0.21, indicating weak linear relationships. Consequently, traditional linear models, such as logistic regression, may struggle to capture the complex interactions among these variables. Thus, we recommend using nonlinear models, such as tree-based methods (Random Forest, XGBoost) or neural networks, which are well-suited for capturing intricate interactions in datasets with weak linear correlations.

2.2.2 **Numerical Variables and BMI Feature Derivation**

The correlation between Weight and Height is observed to be 0.46 in the correlation matrix, indicating a moderate linear relationship. While weight and height individually provide valuable insights into an individual's physique, they are insufficient to fully capture the degree of obesity. To better quantify obesity risk, we derived a new feature, the Body Mass Index (BMI), using the formula:

$$ BMI = \frac{\text{Weight}}{\text{Height}^2} $$
BMI, as a direct indicator of obesity, integrates the effects of both weight and height, eliminating biases caused by differences in height. Specifically, BMI effectively reflects the combined influence of weight and height on obesity levels, making it a central variable for analyzing and predicting obesity risk. More importantly, BMI is widely recognized in medical and health domains, with its numerical range typically classified into categories such as normal weight, overweight, and obese, which enhances the interpretability of the target variable. Thus, BMI, as a derived feature, not only simplifies complex relationships between variables but also significantly strengthens the model's capability to understand obesity levels.

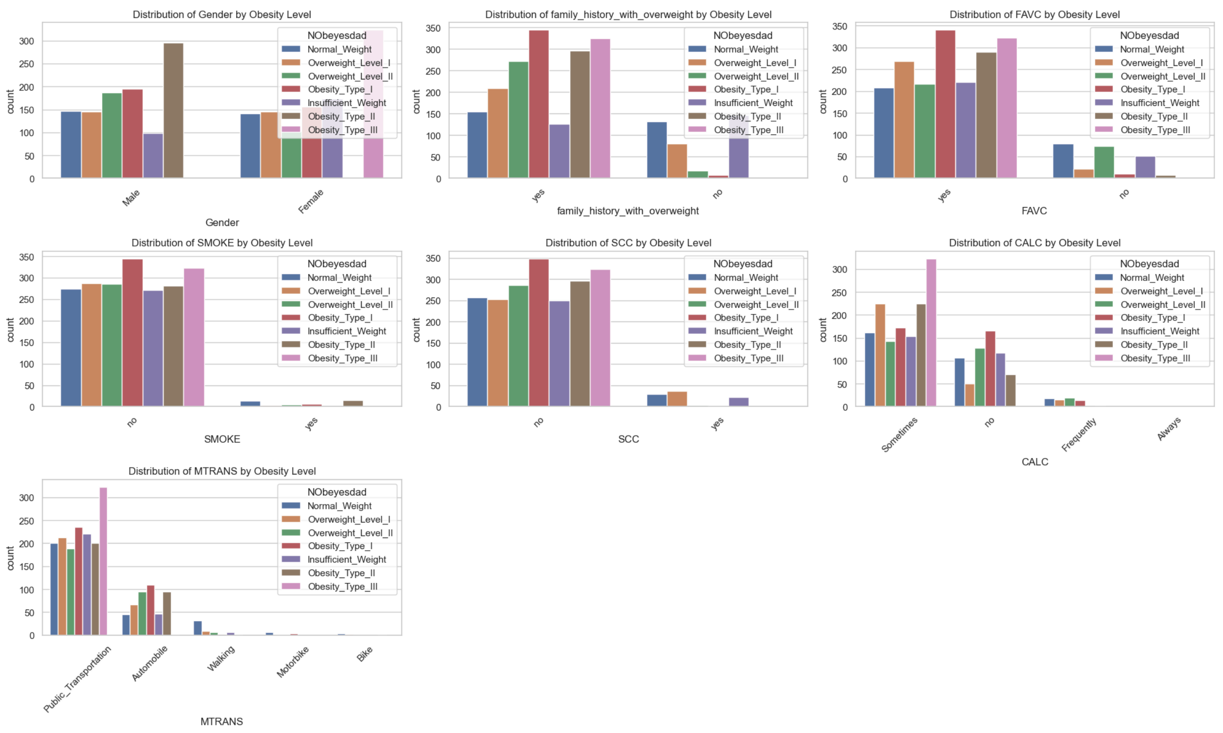
In [20]:
```
# Correlation analysis with a heatmap for numerical variables
numerical_features = df.select_dtypes(include=['float64']).columns
correlation_matrix = df[numerical_features].corr()

plt.figure(figsize=(12, 8))
# A heat map of the correlation matrix is created to visually display the co
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5,
plt.title("Correlation Matrix for Numerical Features")
plt.show()
```



2.2.3 **Key Variable Analysis**

Certain variables demonstrated strong associations with the target variable (NObeyesdad, obesity level) and were identified as critical for model training. For instance, individuals with a family history of obesity were prominently represented in the "Obesity_Type_I," "Obesity_Type_II," and "Obesity_Type_III" categories, highlighting a direct impact of genetic predisposition on obesity risk. Similarly, frequent high-calorie food consumers were predominantly found in the "Obesity_Type_III" category, indicating a strong relationship between dietary habits and obesity levels. Additionally, transportation mode emerged as another critical variable, with individuals relying on automobiles being more likely to fall into obese categories, whereas those who walk or use public transportation were predominantly in the normal or underweight categories. This suggests that transportation mode, as a proxy for daily activity levels, significantly influences obesity risk.

In [21]:
```
'''
The influence of independent variables on the dependent variable of obesity
'''
# Category variable analysis using count plots
categorical_features = ['Gender', 'family_history_with_overweight', 'FAVC',
plt.figure(figsize=(20, 12))
for i, col in enumerate(categorical_features, 1):
    plt.subplot(3, 3, i)
    sns.countplot(data=df, x=col, hue='NObeyesdad', order=df[col].value_cour
    plt.title(f"Distribution of {col} by Obesity Level")
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



2.2.4 **Categorical Variable Distribution and Encoding**

The distribution of categorical variables (e.g., Gender, family_history_with_overweight, FAVC) revealed significant distinctions across different obesity levels. These variables were encoded using One-Hot Encoding to transform them into a machine-readable format. One-Hot Encoding avoids assumptions of ordinal relationships among categories and preserves their independence, enabling the model to better interpret their contributions.

2.2.5 **Outlier Detection and Handling**

Outliers can negatively impact model performance and result interpretation, making their detection and handling a critical step in data preprocessing. Based on the dataset's characteristics, we employed three distinct methods for outlier detection and handling: the IQR method, the Z-Score method, and the Modified Z-Score method.

IQR Method (Interquartile Range)

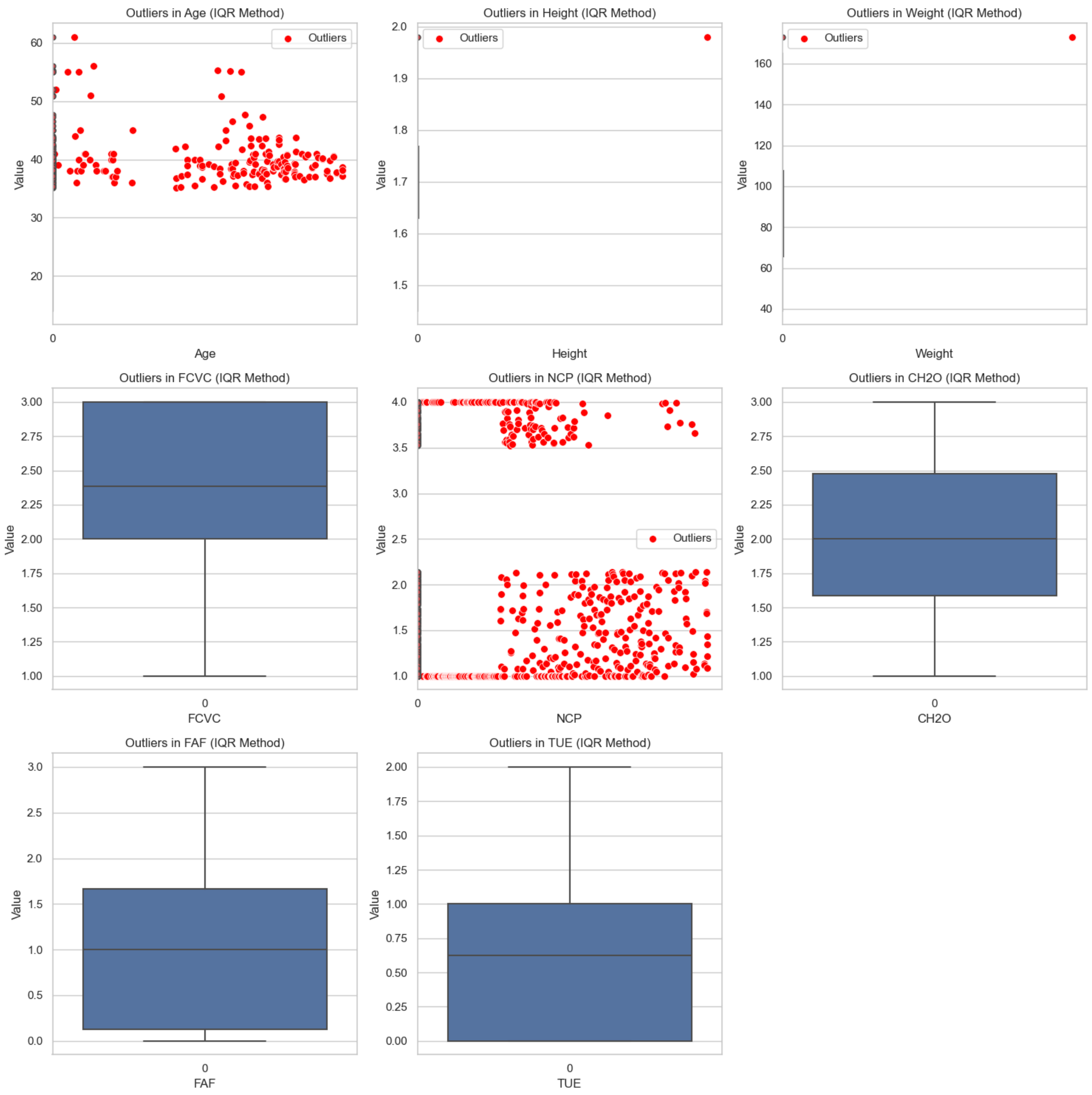
The IQR method is based on the 1st quartile (Q1) and the 3rd quartile (Q3) of a variable to detect outliers. By calculating the interquartile range (IQR = Q3 - Q1), upper and lower bounds are defined, and any data points falling outside this range are considered outliers. The bounds are calculated using the following formula: $$ \text{Lower Bound} = Q1 - 1.5 \times \text{IQR}, \quad \text{Upper Bound} = Q3 + 1.5 \times \text{IQR} $$ We visualized each variable using boxplots to highlight the distribution and mark outliers.

Outliers detected by this method were replaced with the median value of the corresponding variable, minimizing the impact of extreme values while preserving the overall trend of the data.

Z-Score Method
The Z-Score method identifies outliers based on standardized values, measuring the deviation of a data point from the mean. The Z-Score is calculated as: $$ Z\text{-Score} = \frac{\text{Value} - \text{Mean}}{\text{Standard Deviation}} $$ A data point is considered an outlier if its absolute Z-Score exceeds 3. This method standardizes the variables and provides a clear visualization of outliers within the distribution. Detected outliers were replaced with the mean value of the respective variable, smoothing the distribution and ensuring the model captures the overall variable trends.

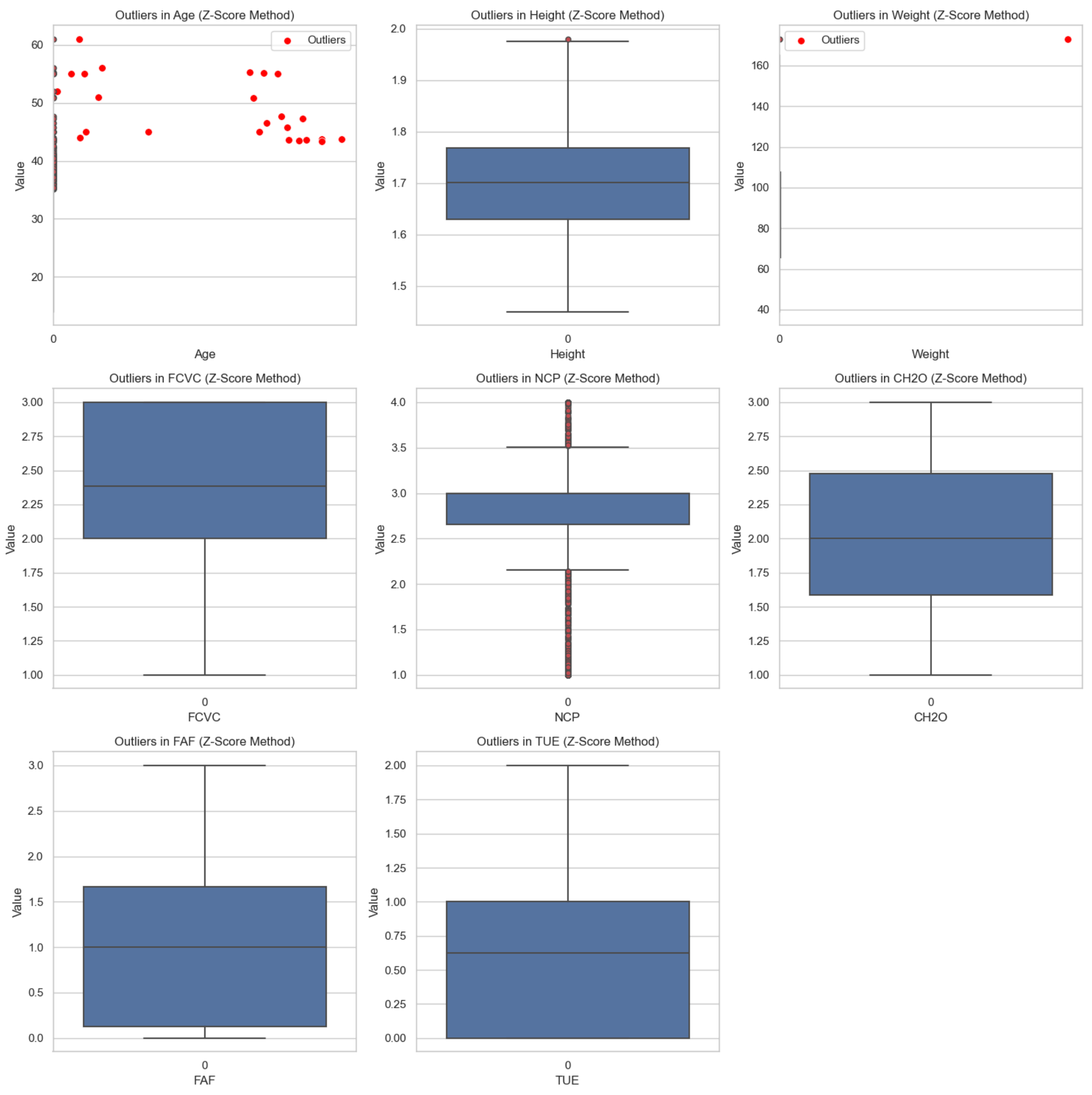
Modified Z-Score Method
The Modified Z-Score method is based on the median (Median) and the Median Absolute Deviation (MAD), making it more robust for skewed distributions with outliers. The formula for the Modified Z-Score is: $$ \text{Modified Z-Score} = 0.6745 \times \frac{\text{Value} - \text{Median}}{\text{MAD}} $$ A data point is flagged as an outlier if its Modified Z-Score exceeds an absolute value of 3.5. Outliers detected using this method were replaced with the median value of the variable, which preserves the structure of the data distribution while reducing the impact of extreme deviations.

In [24]:
```
'''
 Method1: IQR - Replace outliers with lower/upper bounds
'''
# Select numerical columns for analysis
numerical_cols = df.select_dtypes(include=['float64']).columns
df_numerical = df[numerical_cols].copy()

# Functions for outlier detection
def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return ((data < lower_bound) | (data > upper_bound))

# Function to plot boxplot for each numerical column with dynamic subplots
def plot_outliers_dynamic(data, outliers, title):
    num_cols = data.shape[1]
    num_rows = (num_cols + 2) // 3  # Calculate the number of rows needed fo

    plt.figure(figsize=(15, num_rows * 5))
    for i, col in enumerate(data.columns, 1):
        plt.subplot(num_rows, 3, i)
        sns.boxplot(data=data[col], flierprops=dict(markerfacecolor='r', mar
        outlier_points = data[outliers[col]][col]
        sns.scatterplot(x=outlier_points.index, y=outlier_points, color='rec
        plt.title(f'Outliers in {col} ({title})')
        plt.xlabel(col)
        plt.ylabel('Value')
    plt.tight_layout()
    plt.show()

# Method 1: IQR Method – Visualization and Handling
iqr_outliers = df_numerical.apply(detect_outliers_iqr)
plot_outliers_dynamic(df_numerical, iqr_outliers, 'IQR Method')

# Replace IQR outliers with median
df_iqr_cleaned = df_numerical.copy()
for col in df_numerical.columns:
    median = df_numerical[col].median()
    df_iqr_cleaned.loc[iqr_outliers[col], col] = median

# Display the first few rows of each cleaned dataset for comparison
print("Cleaned Data (IQR Method):")
print(df_iqr_cleaned.head())
```



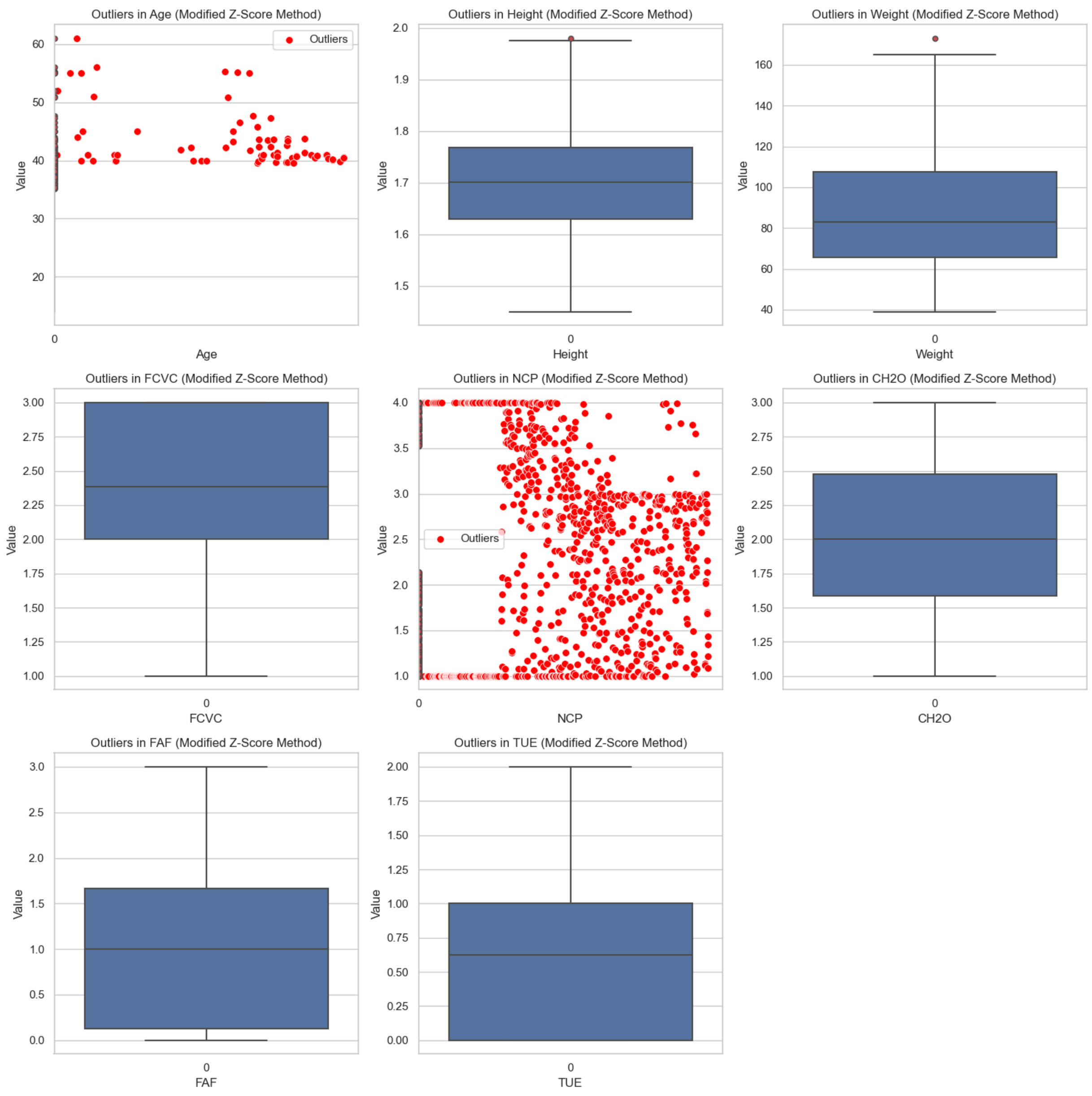
```
Cleaned Data (IQR Method):
    Age  Height  Weight  FCVC  NCP  CH2O  FAF  TUE
0  21.0    1.62    64.0   2.0  3.0   2.0  0.0  1.0
1  21.0    1.52    56.0   3.0  3.0   3.0  3.0  0.0
2  23.0    1.80    77.0   2.0  3.0   2.0  2.0  1.0
3  27.0    1.80    87.0   3.0  3.0   2.0  2.0  0.0
4  22.0    1.78    89.8   2.0  3.0   2.0  0.0  0.0
```

In [25]:
```
'''
Method 2: Z-Score - Outliers are replaced by the mean
'''
import numpy as np
def detect_outliers_zscore(data, threshold=3):
    mean = data.mean()
    std = data.std()
    z_score = (data - mean) / std
    return (np.abs(z_score) > threshold)

zscore_outliers = df_numerical.apply(detect_outliers_zscore)
plot_outliers_dynamic(df_numerical, zscore_outliers, 'Z-Score Method')

# Replace Z-Score outliers with mean
df_zscore_cleaned = df_numerical.copy()
for col in df_numerical.columns:
    mean = df_numerical[col].mean()
    df_zscore_cleaned.loc[zscore_outliers[col], col] = mean

print("\nCleaned Data (Z-Score Method):")
print(df_zscore_cleaned.head())
```



```
Cleaned Data (Z-Score Method):
    Age  Height  Weight  FCVC  NCP  CH2O  FAF  TUE
0  21.0    1.62    64.0   2.0  3.0   2.0  0.0  1.0
1  21.0    1.52    56.0   3.0  3.0   3.0  3.0  0.0
2  23.0    1.80    77.0   2.0  3.0   2.0  2.0  1.0
3  27.0    1.80    87.0   3.0  3.0   2.0  2.0  0.0
4  22.0    1.78    89.8   2.0  1.0   2.0  0.0  0.0
```

In [26]:
```
'''
Method 3: Modified Z-Score Method - Outliers are replaced by the median
'''
def detect_outliers_modified_zscore(data, threshold=3.5):
    median = data.median()
    mad = np.median(np.abs(data - median))
    modified_z_score = 0.6745 * (data - median) / mad
    return (np.abs(modified_z_score) > threshold)

modified_zscore_outliers = df_numerical.apply(detect_outliers_modified_zscor
plot_outliers_dynamic(df_numerical, modified_zscore_outliers, 'Modified Z-Sc
# Replace Modified Z-Score outliers with median
df_modified_zscore_cleaned = df_numerical.copy()
for col in df_numerical.columns:
    median = df_numerical[col].median()
    df_modified_zscore_cleaned.loc[modified_zscore_outliers[col], col] = mec

print("\nCleaned Data (Modified Z-Score Method):")
print(df_modified_zscore_cleaned.head())
```



```
Cleaned Data (Modified Z-Score Method):
    Age  Height  Weight  FCVC  NCP  CH2O  FAF  TUE
0  21.0    1.62    64.0   2.0  3.0   2.0  0.0  1.0
1  21.0    1.52    56.0   3.0  3.0   3.0  3.0  0.0
2  23.0    1.80    77.0   2.0  3.0   2.0  2.0  1.0
3  27.0    1.80    87.0   3.0  3.0   2.0  2.0  0.0
4  22.0    1.78    89.8   2.0  3.0   2.0  0.0  0.0
```

2.3 Data Cleaning

The goal of data cleaning is to handle missing values and ensure consistency in the dataset. The steps are as follows:

- **Handling Missing Values**:
    - For numerical variables (e.g., Weight, Height), mean imputation was applied to replace missing values with the mean of the variable. This approach reduces the impact of missing values on the data distribution while maintaining variable continuity.
    - For categorical variables (e.g., Transportation Mode, Gender), mode imputation was used to replace missing values with the most frequent category. This ensures the preservation of the category distribution.
    - After imputation, the dataset's completeness was verified using the `df.info()` method to ensure no missing values remained.
- **Consistency Check**:
    - We verified the filled dataset to ensure that the distributions of variables remained consistent with the original data.

In [27]:
```
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Handle missing values
# For numerical columns, replace missing values with the mean
num_imputer = SimpleImputer(strategy='mean')
df[df.select_dtypes(include=['float64']).columns] = num_imputer.fit_transfor

# For categorical columns, replace missing values with the most frequent val
cat_imputer = SimpleImputer(strategy='most_frequent')
df[df.select_dtypes(include=['object', 'category']).columns] = cat_imputer.f

# Display the result to verify
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Gender                          2111 non-null   object 
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object 
 5   FAVC                            2111 non-null   object 
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object 
 9   SMOKE                           2111 non-null   object 
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object 
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object 
 15  MTRANS                          2111 non-null   object 
 16  NObeyesdad                      2111 non-null   object 
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

2.4 **Outlier Detection and Handling**

Outliers can distort data distribution and negatively impact model performance. We used the IQR (Interquartile Range) method to detect and handle outliers, with the following steps:

- **Detection**:
  - The IQR method calculates the 1st quartile (Q1) and 3rd quartile (Q3) to define outlier boundaries. The formula is as follows: $$ \text{IQR} = Q3 - Q1, \quad \text{Lower Bound} = Q1 - 1.5 \times \text{IQR}, \quad \text{Upper Bound} = Q3 + 1.5 \times \text{IQR} $$
  - Any values outside these bounds were flagged as outliers.
- **Handling**:
  - Detected outliers were capped, with values exceeding the upper bound replaced by the upper bound and those below the lower bound replaced by the lower bound. This preserves the overall distribution while reducing the impact of extreme values.
- **Validation**:
  - After processing, the statistical summary of variables was checked using the `df.describe()` method to ensure reasonable distributions.

In [29]:
```
def cap_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[column] = df[column].apply(lambda x: lower_bound if x < lower_bound e

# Apply outlier capping for numerical columns
numerical_columns = df.select_dtypes(include=['float64']).columns
for col in numerical_columns:
    cap_outliers(df, col)

# Display information to verify outlier handling
df.describe()
```

Out[29]:

	Age	Height	Weight	FCVC	NCP	CH2O
count	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000
mean	23.910277	1.701676	86.584811	2.419043	2.835525	2.008011
std	5.277434	0.093299	26.187117	0.533927	0.400898	0.612953
min	14.000000	1.450000	39.000000	1.000000	2.146845	1.000000
25%	19.947192	1.630000	65.473343	2.000000	2.658738	1.584812
50%	22.777890	1.700499	83.000000	2.385502	3.000000	2.000000
75%	26.000000	1.768464	107.430682	3.000000	3.000000	2.477420
max	35.079212	1.976160	170.366691	3.000000	3.511893	3.000000

2.5 Data Encoding

To ensure categorical variables can be parsed by algorithms, we applied encoding techniques:

- **Target Variable Encoding**:
  - The target variable (NObeyesdad) was encoded using Label Encoding, mapping each category to a unique integer. This facilitated the use of classification models.
- **One-Hot Encoding of Categorical Variables**:
  - One-Hot Encoding was applied to the remaining categorical variables (e.g., Gender, family_history_with_overweight, FAVC). This approach avoids ordinal assumptions between categories and preserves their independence.
  - After encoding, the dataset's dimensionality increased significantly, but this ensured that the model could fully utilize the information within each category.
- **Post-Encoding Validation**:

- The success of encoding was verified using the `df_encoded.info()` method, ensuring all categorical variables were transformed into numerical forms.
- The `df_encoded.head()` method was used to check the results of One-Hot Encoding, ensuring proper mapping for each category.

In [30]:
```
# Calculate BMI and drop Weight and Height columns
df['BMI'] = df['Weight'] / (df['Height'] ** 2)
df = df.drop(['Weight', 'Height'], axis=1)
```

In [31]:
```
le = LabelEncoder()
df['NObeyesdad'] = le.fit_transform(df['NObeyesdad'])

# One-Hot Encode categorical features
df_encoded = pd.get_dummies(df, columns=['Gender', 'family_history_with_over
                                          'SMOKE', 'SCC', 'CALC', 'MTRANS'],

# Display information to verify transformations and encoding
df_encoded.info()
df_encoded.head()

# Prepare features and target for splitting
X = df_encoded.drop('NObeyesdad', axis=1)
y = df_encoded['NObeyesdad']
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 31 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   Age                                 2111 non-null   float64
 1   FCVC                                2111 non-null   float64
 2   NCP                                 2111 non-null   float64
 3   CH2O                                2111 non-null   float64
 4   FAF                                 2111 non-null   float64
 5   TUE                                 2111 non-null   float64
 6   NObeyesdad                          2111 non-null   int32  
 7   BMI                                 2111 non-null   float64
 8   Gender_Female                       2111 non-null   uint8  
 9   Gender_Male                         2111 non-null   uint8  
 10  family_history_with_overweight_no   2111 non-null   uint8  
 11  family_history_with_overweight_yes  2111 non-null   uint8  
 12  FAVC_no                             2111 non-null   uint8  
 13  FAVC_yes                            2111 non-null   uint8  
 14  CAEC_Always                         2111 non-null   uint8  
 15  CAEC_Frequently                     2111 non-null   uint8  
 16  CAEC_Sometimes                      2111 non-null   uint8  
 17  CAEC_no                             2111 non-null   uint8  
 18  SMOKE_no                            2111 non-null   uint8  
 19  SMOKE_yes                           2111 non-null   uint8  
 20  SCC_no                              2111 non-null   uint8  
 21  SCC_yes                             2111 non-null   uint8  
 22  CALC_Always                         2111 non-null   uint8  
 23  CALC_Frequently                     2111 non-null   uint8  
 24  CALC_Sometimes                      2111 non-null   uint8  
 25  CALC_no                             2111 non-null   uint8  
 26  MTRANS_Automobile                   2111 non-null   uint8  
 27  MTRANS_Bike                         2111 non-null   uint8  
 28  MTRANS_Motorbike                    2111 non-null   uint8  
 29  MTRANS_Public_Transportation        2111 non-null   uint8  
 30  MTRANS_Walking                      2111 non-null   uint8  
dtypes: float64(7), int32(1), uint8(23)
memory usage: 171.2 KB
```

Summary

Comprehensive implementation of data cleaning, outlier detection, and data encoding ensured the dataset's high quality and interpretability. Missing value imputation ensured completeness, IQR-based outlier handling stabilized data distribution, and feature engineering through BMI calculation and One-Hot Encoding enhanced model compatibility. These preprocessing steps laid a solid foundation for subsequent feature engineering and model development.

3. Data Preparation

In this project, multiple dataset versions were designed to explore the impact of different preprocessing methods on model performance. These versions include fundamental steps such as missing value handling, outlier detection, and encoding, while also incorporating advanced techniques like feature scaling, Principal Component Analysis (PCA), binning, and feature interaction. The goal is to provide optimized datasets for neural networks and random forest models, enabling the identification of the most effective preprocessing strategies.

3.1 Standard Version Dataset

The standard version dataset includes essential preprocessing steps such as missing value handling, outlier detection, BMI transformation, and encoding. For missing values, numerical variables (e.g., BMI) were imputed with their mean, while categorical variables (e.g., transportation mode) were imputed with their mode to ensure data completeness. Outliers were handled using the IQR method; for example, extreme age values beyond the normal range were capped to the upper or lower bound. Weight and Height were transformed into BMI, serving as the key feature for predicting obesity. The target variable (NObeyesdad) was encoded using Label Encoding, while other categorical variables (e.g., gender, dietary preferences) were expanded using One-Hot Encoding. This dataset serves as a baseline for both neural networks and random forest models.

All dataset versions were split into 80% training and 20% testing sets to ensure independent evaluation. A random seed (random_state=42) was used to maintain reproducibility and consistency across experiments.

In [16]:
```
X_train_std, X_test_std, y_train_std, y_test_std = train_test_split(X, y, te
```

3.2 Feature Scaling Version Dataset

Building on the standard version, the feature scaling version includes normalization of numerical variables using StandardScaler. This ensures that all numerical features (e.g., Age, BMI) are transformed into a standard normal distribution (mean=0, standard deviation=1). For example, while the original age data ranges from 0 to 80 and BMI ranges from 15 to 40, feature scaling ensures consistent feature magnitudes, preventing issues like uneven gradient updates during neural network training. This version is tailored for neural networks, where feature scaling is crucial for effective optimization.

In [17]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_sp
```

3.3 PCA Version Dataset

The PCA version dataset reduces dimensionality based on the scaled dataset, retaining essential information while eliminating redundancy. PCA selects the top 10 principal components, which explain 90% of the data variance. For instance, BMI and weight, being highly correlated, are combined into a single principal component, reducing feature overlap. This version is particularly suitable for neural networks in high-dimensional scenarios, as it mitigates overfitting and improves computational efficiency.

In [18]:
```
from sklearn.decomposition import PCA
# Define number of components for PCA - this can be adjusted based on desire
n_components = 10  # for example, reduce to 10 principal components

# Apply PCA on the scaled data (from Version 2)
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)

# Split PCA-transformed data into train and test sets
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y
```

3.4 Binned Version Dataset

The binned version dataset transforms continuous variables into discrete bins, enhancing feature stability and compatibility with tree-based models. Age and BMI, for example, were divided into 10 bins, with each sample assigned a corresponding bin label (e.g., 0 to 9). This transformation reduces the volatility of continuous variables while improving random forest models' performance. Post-binning, One-Hot Encoding was applied to ensure the model can leverage the new categorical features effectively.

In [19]:
```
binned_df = df_encoded.copy()
binned_df['Age_bin'] = pd.cut(binned_df['Age'], bins=10, labels=False)  # 10
binned_df['BMI_bin'] = pd.cut(binned_df['BMI'], bins=10, labels=False)  # 10
binned_df = binned_df.drop(['Age', 'BMI'], axis=1)  # Drop original Age and

X_binned = binned_df.drop('NObeyesdad', axis=1)
y_binned = binned_df['NObeyesdad']
X_train_binned, X_test_binned, y_train_binned, y_test_binned = train_test_sp
```

3.5 Feature Interaction Version Dataset

The feature interaction version dataset generates new interaction features to capture nonlinear relationships between variables. For instance, the product of Age and BMI (Age_BMI) represents the combined impact of age and body composition on obesity risk, where higher Age_BMI values may indicate greater risk. Another example is the interaction between daily fruit/vegetable intake (FCVC) and the number of main meals (NCP), represented as FCVC_NCP, which reflects dietary healthiness. Additionally, combining transportation mode (MTRANS) with weekly physical activity frequency (FAF) produces an interaction feature (MTRANS_FAF) that quantifies overall activity levels.

These features were normalized or encoded and validated through correlation analysis and significance testing to ensure their relevance. This dataset is effective for both neural networks and tree models, particularly when capturing complex patterns is critical.

In [20]:
```
interaction_df = X.copy()
interaction_df['Age_BMI'] = df['Age'] * df['BMI']  # Example interaction fea
interaction_df['FCVC_NCP'] = df['FCVC'] * df['NCP']  # Example interaction f

X_train_inter, X_test_inter, y_train_inter, y_test_inter = train_test_split(
```

Summary

By designing multiple dataset versions, we comprehensively explored how different preprocessing techniques impact model performance. The standard version serves as a baseline, while feature scaling and PCA improve neural network efficiency. Binning and feature interaction enhance feature representation for tree models. These datasets provide a robust foundation for subsequent model selection and performance evaluation.

In [21]:
```
datasets = {
    'standard': (X_train_std, X_test_std, y_train_std, y_test_std),
    'scaled': (X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled)
    'pca': (X_train_pca, X_test_pca, y_train_pca, y_test_pca),
    'binned': (X_train_binned, X_test_binned, y_train_binned, y_test_binned)
    'interaction': (X_train_inter, X_test_inter, y_train_inter, y_test_inter
}
```

4. Modeling approach

In this project, Random Forest (RF) and Artificial Neural Network (ANN) were selected as the primary models for obesity risk prediction. These models provide robust theoretical foundations and technical advantages, enabling the discovery of patterns in the data and offering strong support for decision-making.

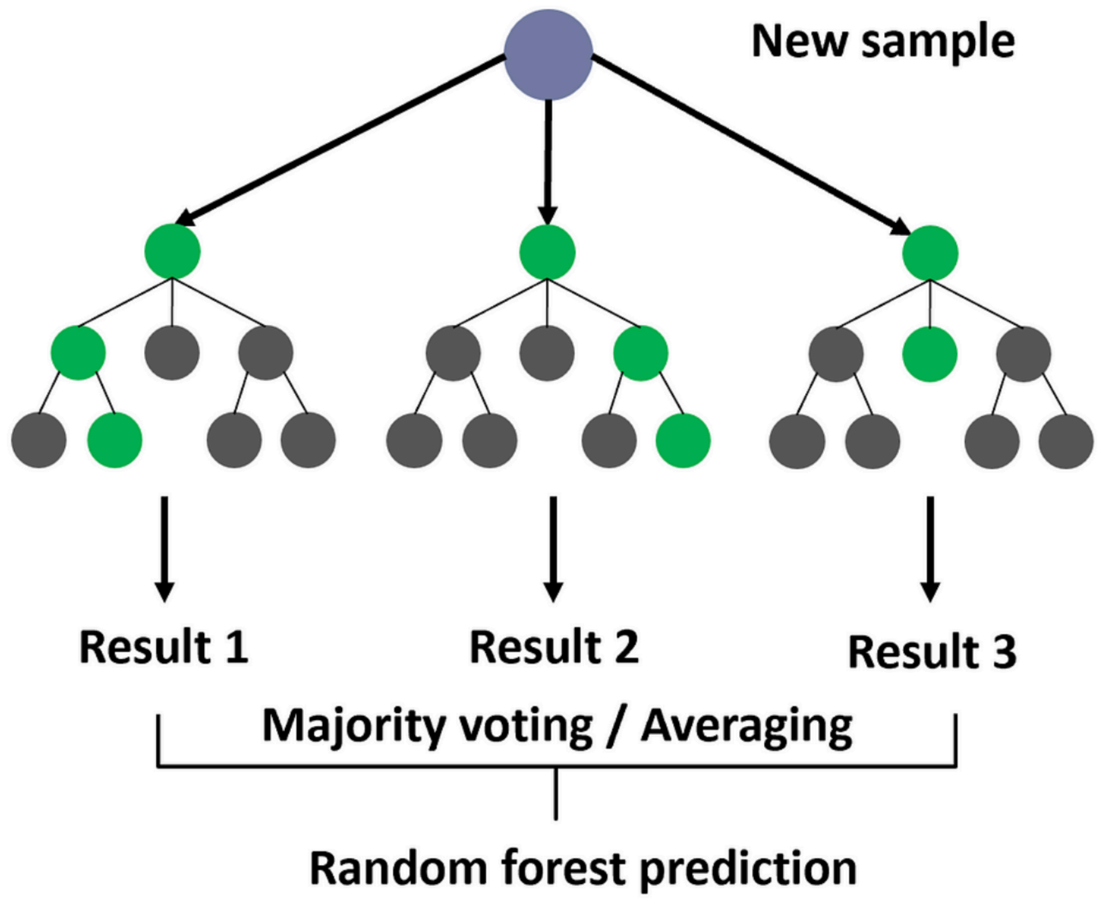
4.1 Model Selection Rationale

Random Forest (RF)
Random Forest is an ensemble learning model that enhances classification accuracy and robustness by aggregating predictions from multiple decision trees. Its core mathematical principles include Bootstrap Aggregation (Bagging) and feature randomness. In Bagging, each decision tree is trained independently on a subset of the training data generated through random sampling, which reduces the risk of overfitting and improves generalization. Additionally, at each split, a random subset of features is

selected to determine the optimal split point, further strengthening the model's robustness and generalization capabilities. The final prediction of the Random Forest is calculated as the weighted average of the predictions from all trees, represented mathematically as: $$ G = \sum_{i=1}^{N} \frac{1}{N} \cdot f_i(x), $$ where $f_i(x)$ is the prediction of the i-th tree, and G represents the aggregated prediction. RF excels in handling nonlinear relationships, missing values, and discrete features. It is particularly suitable for the standard and binned datasets, as it does not require feature scaling, ensuring both efficiency and interpretability 2 .

In [6]:
```
from IPython.display import HTML

base64_string_RF = """iVBORw0KGgoAAAANSUhEUgAABUQAAARaCAYAAACE13gIAAAMTGlDQ1
"""
HTML(f"""
<div style="text-align: center;">
    <img src="data:image/png;base64,{base64_string_RF}" alt="Random Forest S
    <p style="text-align: center; margin-top: 10px; font-size: 14px; line-he
        The diagram above depicts the structure of a Random Forest (RF) mode
        It builds multiple decision trees during training and aggregates the
        Each tree is trained on a random subset of the data (bagging), and a
    </p>
</div>
""")
```

Out[6]:



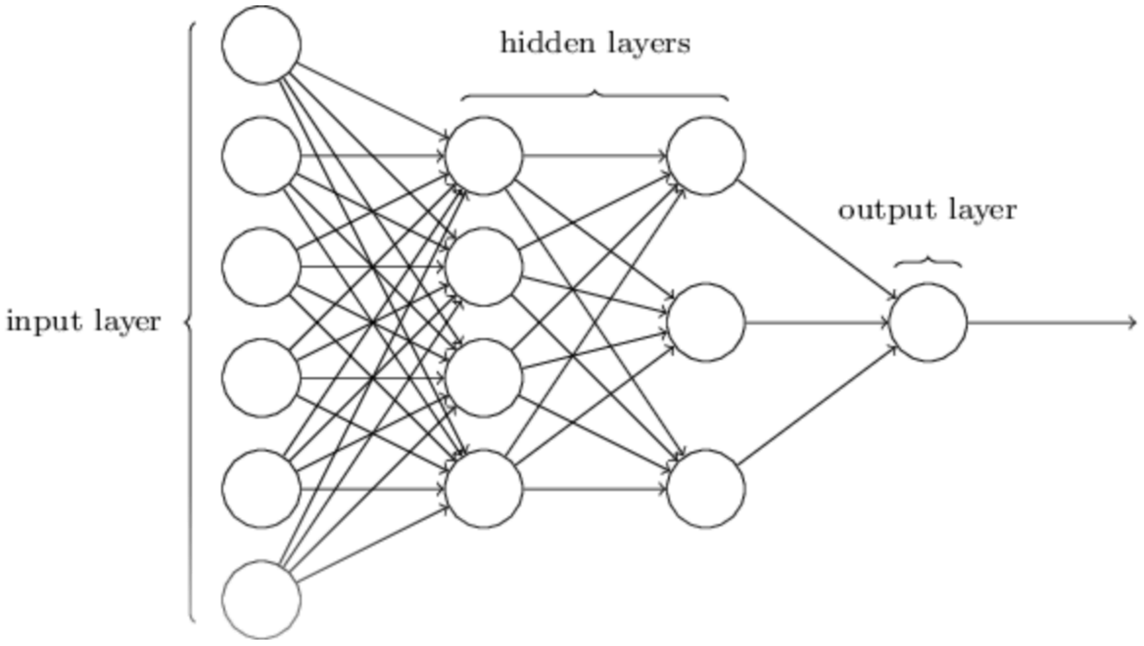
The diagram above depicts the structure of a Random Forest (RF) model, which is an ensemble learning method. It builds multiple decision trees during training and aggregates their outputs using majority voting (for classification) or averaging (for regression). Each tree is trained on a random subset of the data (bagging), and a random subset of features is selected at each split, making the model robust and reducing overfitting.

Artificial Neural Network (ANN)
Artificial Neural Network is a deep learning model inspired by the structure of biological neurons, capable of capturing complex nonlinear relationships through its multilayer architecture. Its core mathematical principles include Feedforward Networks and the Backpropagation Algorithm. In a feedforward network, input features are passed layer by layer through weighted sums and biases, followed by nonlinear transformations using activation functions such as ReLU. This process is mathematically expressed as: $$ h_i = \sigma(W_i \cdot x + b_i), $$ where W_i is the weight matrix, b_i is the bias, and σ is the activation function. This layer-wise propagation enables the network to learn deep patterns among the features. Furthermore, the Backpropagation Algorithm minimizes the loss function (e.g., categorical cross-entropy) using gradient descent to iteratively adjust weights and biases, optimizing the model toward a global solution. ANN is particularly effective in handling continuous features and high-dimensional data,

performing exceptionally well on scaled, PCA, and feature interaction datasets. These preprocessing steps align with the gradient-based optimization process, enabling ANN to efficiently learn complex nonlinear patterns 3 .

In [7]:
```
from IPython.display import HTML

base64_string_ANN = """
iVBORw0KGgoAAAANSUhEUgAABJwAAAKSCAYAAACX9MGWAAAMTGlDQ1BJQ0MgUHJvZmlsZQAASImV
"""
HTML(f"""
<div style="text-align: center;">
    <img src="data:image/png;base64,{base64_string_ANN}" alt="Artificial Neu
    <p style="text-align: center; margin-top: 10px; font-size: 14px; line-he
        The diagram above illustrates the structure of an Artificial Neural
        The input layer receives data features, which are processed through
        which is updated during backpropagation to minimize the loss functio
    </p>
</div>
""")
```

Out[7]:



The diagram above illustrates the structure of an Artificial Neural Network (ANN), which consists of three main components: the input layer, hidden layers, and the output layer. The input layer receives data features, which are processed through one or more hidden layers using activation functions (e.g., ReLU). Each connection is assigned a weight, which is updated during backpropagation to minimize the loss function. The final output layer provides the model's predictions (e.g., class probabilities for classification tasks).

4.2 Hyperparameter Tuning and Training

To achieve optimal performance, both RF and ANN underwent hyperparameter tuning, as detailed below:

Random Forest Hyperparameter Tuning

- **Parameter Settings**: Parameters such as the number of trees (`n_estimators`), maximum depth (`max_depth`), and minimum samples per split (`min_samples_split`) were tuned using RandomizedSearchCV. For example, the search space included $n\_estimators \in [50, 100, 200]$ and $max\_depth \in [None, 10, 20]$.
- **Cross-Validation**: A 5-fold cross-validation (KFold, k=5) was employed to evaluate each parameter combination's performance and reduce bias from single splits.
- **Model Saving**: The best RF model was saved in PKL format for subsequent analysis and comparison.

Artificial Neural Network Hyperparameter Tuning

- **Model Definition**: ANN models were dynamically created based on the number of hidden layers (`layers`) and neurons per layer (`neurons`), adapting to the characteristics of each dataset. For example, on the scaled dataset, a network with two hidden layers, each containing 32 neurons, was defined with ReLU activation.
- **Parameter Search**: Parameters such as the number of neurons (16, 32), layers (1, 2), batch size (10, 20), and epochs (50, 100) were explored. Each configuration was evaluated using cross-validation, and the one with the highest average accuracy was selected.
- **Model Saving**: The best ANN model was saved in H5 format for future deployment.

4.3 Training Results and Comparison

All training results, including test accuracy and optimal parameter configurations, were stored in a JSON file. These results provide structured data for systematic performance analysis and model selection.

In [20]:
```python
import numpy as np
import json
import joblib
from sklearn.model_selection import KFold, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# Create ANN model function with dynamic input shape
def create_ann_model(neurons, layers, input_shape, num_classes):
    model = Sequential()
    model.add(Dense(neurons, input_dim=input_shape, activation='relu'))
    for _ in range(layers - 1):
        model.add(Dense(neurons, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics
    return model

# Hyperparameter tuning function
def hyperparameter_tuning(X_train, y_train, model_type, num_classes, version
    kf = KFold(n_splits=5, shuffle=True, random_state=42)

    X_train = X_train.astype(np.float32)
    y_train = y_train.astype(np.int32)

    if model_type == "RF":
        # params = {
        #     'n_estimators': [50],
        #     'max_depth': [10],
        #     'min_samples_split': [2]
        # }
        params = {
            'n_estimators': [50, 100, 200],
            'max_depth': [None, 10, 20],
            'min_samples_split': [2, 5]
        }
        rf_model = RandomForestClassifier(random_state=42)
        random_search_rf = RandomizedSearchCV(
            rf_model, param_distributions=params, n_iter=1,
            cv=kf, scoring='accuracy', n_jobs=-1
        )
        random_search_rf.fit(X_train, y_train)
        best_rf_model = random_search_rf.best_estimator_
        best_rf_params = random_search_rf.best_params_

        # Save model as PKL
        joblib.dump(best_rf_model, f'RF_best_model_{version}.pkl')

        return best_rf_model, best_rf_params

    elif model_type == "ANN":
        y_train = to_categorical(y_train, num_classes=num_classes)
        input_shape = X_train.shape[1]
        best_score = 0
        best_params = None

        # param_grid = {
        #     'neurons': [16],
        #     'layers': [2],
        #     'epochs': [50],
        #     'batch_size': [20]
        # }
        param_grid = {
            'neurons': [16, 32],
            'layers': [1, 2],
            'epochs': [50, 100],
            'batch_size': [10, 20]
        }
        for neurons in param_grid['neurons']:
            for layers in param_grid['layers']:
                for epochs in param_grid['epochs']:
                    for batch_size in param_grid['batch_size']:
                        scores = []
                        for train_idx, test_idx in kf.split(X_train):
                            X_train_fold, X_test_fold = X_train[train_idx],
                            y_train_fold, y_test_fold = y_train[train_idx],

                            model = create_ann_model(neurons, layers, input_
                            model.fit(
                                X_train_fold, y_train_fold,
                                epochs=epochs, batch_size=batch_size, verbos
                            )
                            score = model.evaluate(X_test_fold, y_test_fold,
                            scores.append(score)

                        mean_score = np.mean(scores)
                        if mean_score > best_score:
                            best_score = mean_score
                            best_params = {
                                'neurons': neurons,
                                'layers': layers,
                                'epochs': epochs,
                                'batch_size': batch_size
                            }

        best_ann_model = create_ann_model(best_params['neurons'], best_param
        best_ann_model.fit(X_train, y_train,
                           epochs=best_params['epochs'],
                           batch_size=best_params['batch_size'],
                           verbose=0)

        # Save model as H5
        best_ann_model.save(f'ANN_best_model_{version}.h5')

        return best_ann_model, best_params

    else:
        raise ValueError("Model type must be 'RF' or 'ANN'")

# Define datasets as numpy arrays
datasets = {
    'standard': (X_train_std.values.astype(np.float32), X_test_std.values.as
    'scaled': (X_train_scaled.astype(np.float32), X_test_scaled.astype(np.fl
    'pca': (X_train_pca.astype(np.float32), X_test_pca.astype(np.float32), y
    'binned': (X_train_binned.values.astype(np.float32), X_test_binned.value
    'interaction': (X_train_inter.values.astype(np.float32), X_test_inter.va
}

# Train models for each dataset
results = {}
num_classes = len(np.unique(y))

for version, (X_train, X_test, y_train, y_test) in datasets.items():
    result = {'version': version}

    if version in ['standard', 'binned']:
        best_rf, best_rf_params = hyperparameter_tuning(X_train, y_train, "R
        rf_accuracy = best_rf.score(X_test, y_test)
        result['RF'] = {'accuracy': rf_accuracy, 'params': best_rf_params}

    if version in ['scaled', 'pca', 'interaction']:
        best_ann, best_ann_params = hyperparameter_tuning(X_train, y_train,
        y_test_cat = to_categorical(y_test, num_classes=num_classes)
        ann_accuracy = best_ann.evaluate(X_test, y_test_cat, verbose=0)[1]
        result['ANN'] = {'accuracy': ann_accuracy, 'params': best_ann_params

    results[version] = result

# Save results to JSON (without model objects)
serializable_results = {}
for version, result in results.items():
    serializable_results[version] = {}
    for model_type, metrics in result.items():
        if isinstance(metrics, dict):
            serializable_results[version][model_type] = {
                'accuracy': metrics['accuracy'],
                'params': metrics['params']
            }

with open('model_results.json', 'w') as f:
    json.dump(serializable_results, f)

# Display results
serializable_results
```

Out[20]:
```
{'standard': {'RF': {'accuracy': 0.966903073286052,
   'params': {'n_estimators': 200, 'min_samples_split': 2, 'max_depth': 2
0}}},
 'scaled': {'ANN': {'accuracy': 0.9432623982429504,
   'params': {'neurons': 32, 'layers': 2, 'epochs': 100, 'batch_size': 1
0}}},
 'pca': {'ANN': {'accuracy': 0.8250591158866882,
   'params': {'neurons': 32, 'layers': 2, 'epochs': 100, 'batch_size': 2
0}}},
 'binned': {'RF': {'accuracy': 0.9196217494089834,
   'params': {'n_estimators': 100,
    'min_samples_split': 2,
    'max_depth': None}}},
 'interaction': {'ANN': {'accuracy': 0.8888888955116272,
   'params': {'neurons': 32, 'layers': 2, 'epochs': 50, 'batch_size': 1
0}}}}
```

Summary

Through model selection and hyperparameter tuning, we maximized the strengths of RF and ANN. RF effectively handled nonlinear relationships and discrete features, excelling on standard and binned datasets. ANN leveraged its deep network structure and optimization algorithms to capture complex patterns, performing exceptionally on scaled, PCA, and feature interaction datasets. This comprehensive modeling approach provides robust technical support for obesity risk prediction.

5. Model Performance Evaluation

To systematically evaluate the performance of Random Forest (RF) and Artificial Neural Network (ANN) across different dataset versions, we used a range of metrics, including Accuracy, Precision, Recall, F1 Score, and AUC-ROC curves. Additionally, confusion matrix visualization provided an intuitive analysis of classification accuracy and potential misclassification patterns.

5.1 **Evaluation Methodology**

We designed a comprehensive evaluation framework to ensure the reliability and accuracy of performance analysis:

1. **Classification Metrics**: Accuracy evaluates overall correctness, Precision assesses the accuracy of positive predictions, Recall measures the proportion of true positives correctly identified, and F1 Score combines Precision and Recall, making it suitable for imbalanced datasets.
2. **Confusion Matrix**: The confusion matrix highlights the alignment between true labels and predicted labels. Heatmap visualization facilitates the identification of well-classified and misclassified categories.
3. **AUC-ROC Curve**: For multiclass problems, a One-vs-Rest (OVR) strategy was employed to compute ROC curves and AUC values for each class, measuring the model's discriminative ability across classes.

5.2 **Evaluation Results**

Standard Version
On the standard dataset, Random Forest performed the best with hyperparameters `n_estimators=200`, `min_samples_split=2`, and `max_depth=20`. The model achieved an accuracy of **96.69%** and an F1 Score of **0.9670**. The confusion matrix showed minor misclassifications, particularly in the "Obesity Type II" category, demonstrating strong overall classification capability.

Feature Scaling Version
On the scaled dataset, ANN achieved optimal performance with hyperparameters `neurons=32`, `layers=2`, `epochs=100`, and `batch_size=10`. The model achieved an accuracy of **94.33%** and an F1 Score of **0.9429**. While the model performed consistently well in categories like "Underweight" and "Normal weight," slight errors were observed in "Obesity Type II." The results highlight ANN's compatibility with scaled data.

PCA Version

On the PCA-reduced dataset, ANN performance dropped slightly, with an accuracy of **82.51%** and an F1 Score of **0.8233**. Although dimensionality reduction minimized feature redundancy, it also led to information loss, increasing misclassifications in categories like “Overweight Level I” and “Obesity Type III.” Optimal hyperparameters were `neurons=32`, `layers=2`, `epochs=100`, and `batch_size=20`.

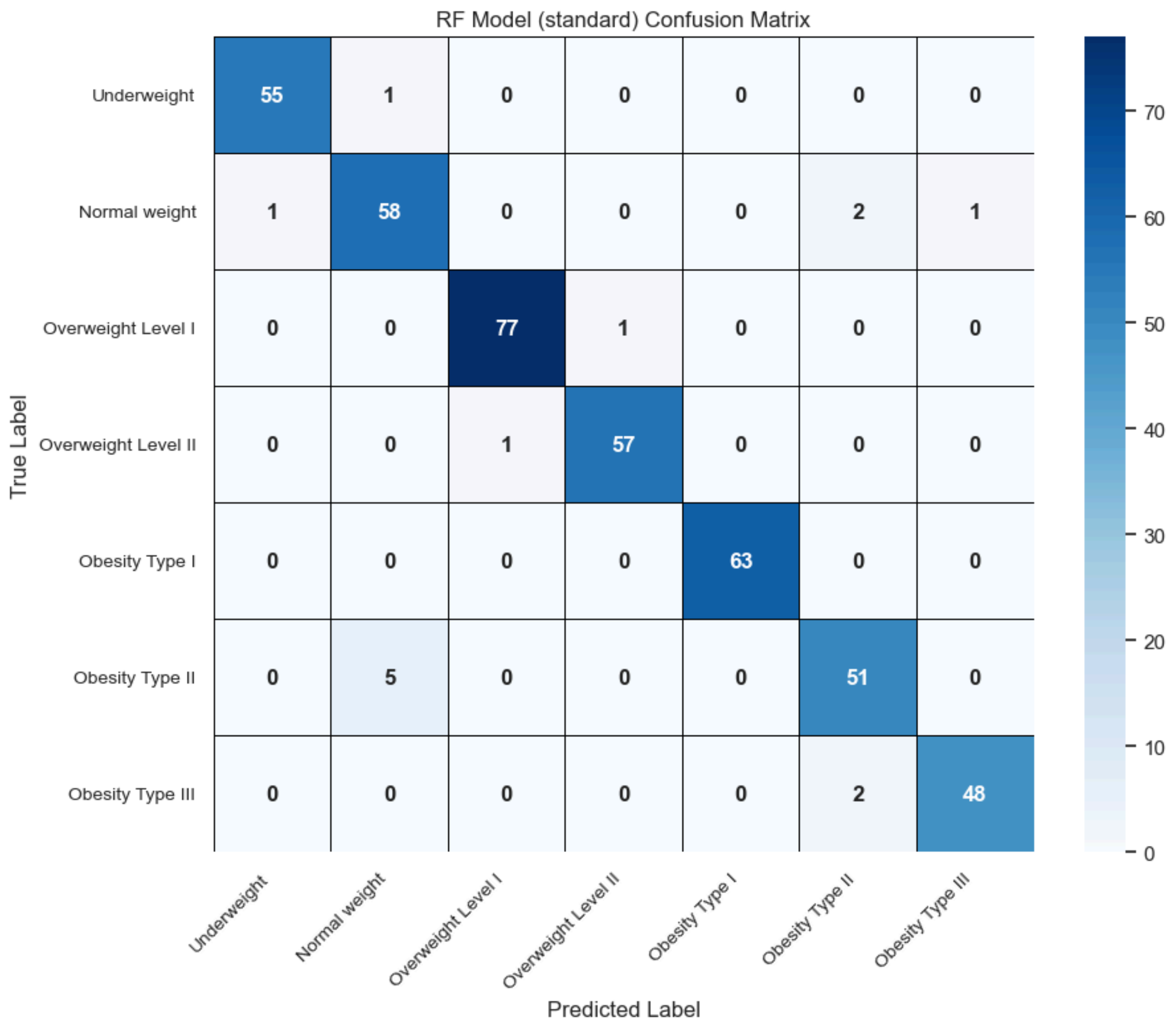
Binned Version

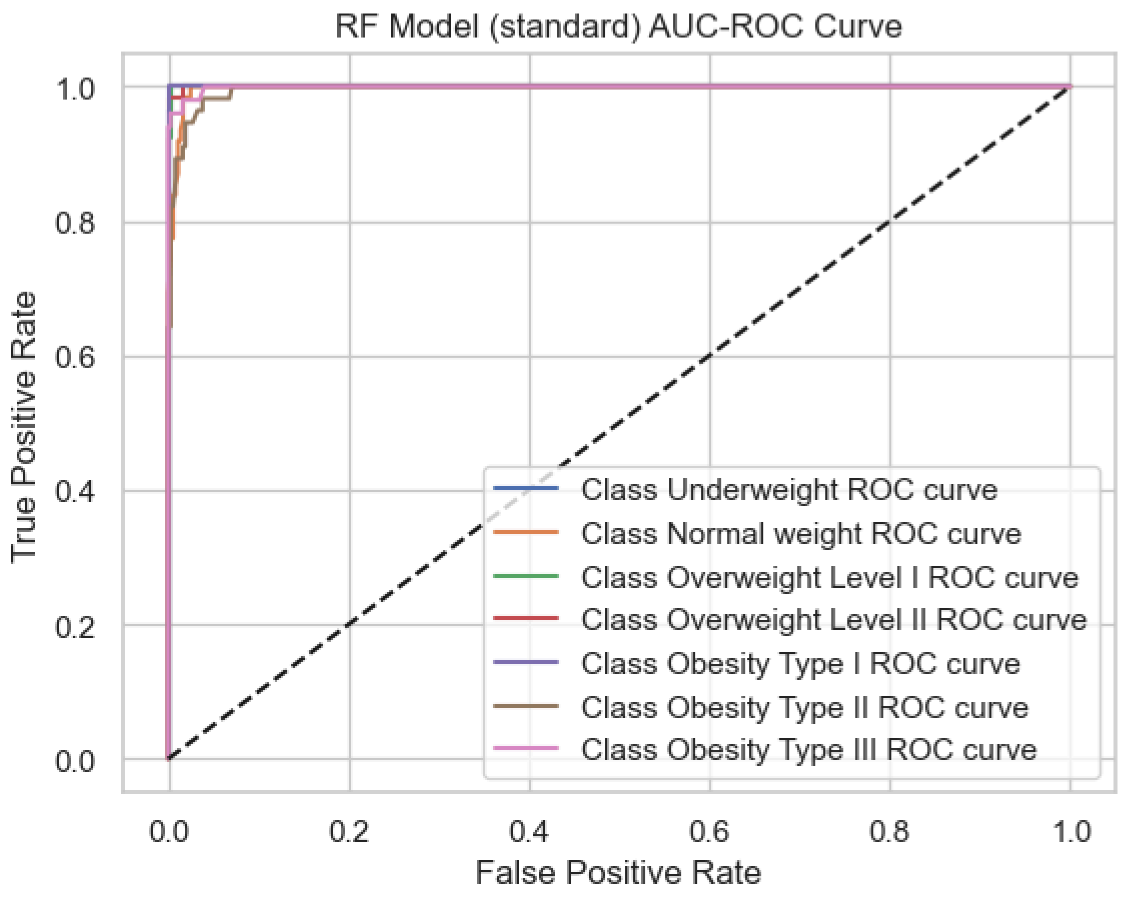
For the binned dataset, Random Forest achieved an accuracy of **91.96%** and an F1 Score of **0.9191** with hyperparameters `n_estimators=100`, `min_samples_split=2`, and `max_depth=None`. The model adapted well to discretized features, with strong performance across most categories, though higher misclassification rates were noted in the “Normal weight” category.

Feature Interaction Version

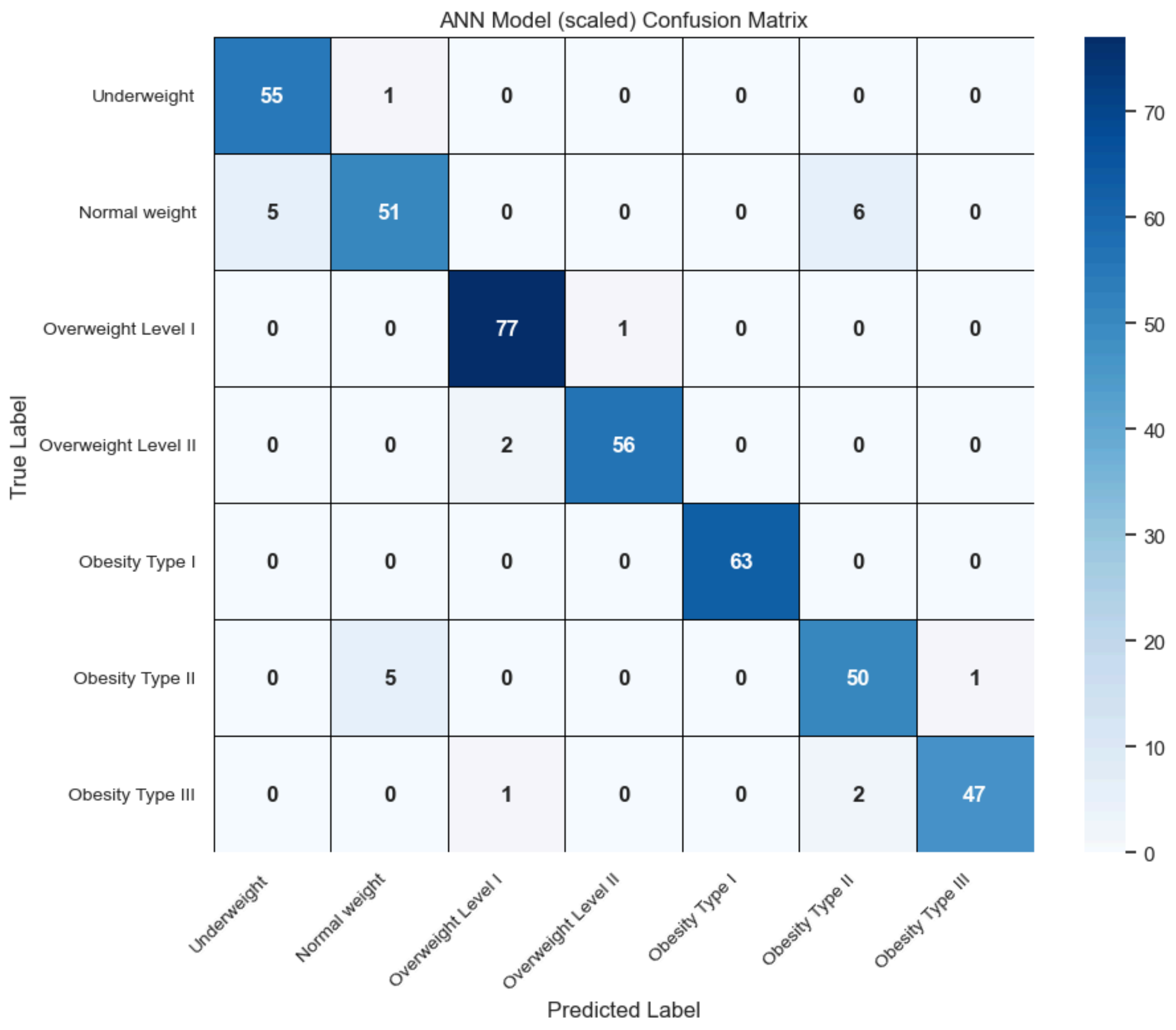
On the feature interaction dataset, ANN outperformed other versions with an accuracy of **88.89%** and an F1 Score of **0.8865**. Interaction features (e.g., Age_BMI and FCVC_NCP) helped capture complex nonlinear relationships, although some difficulty remained in the “Overweight Level I” category.

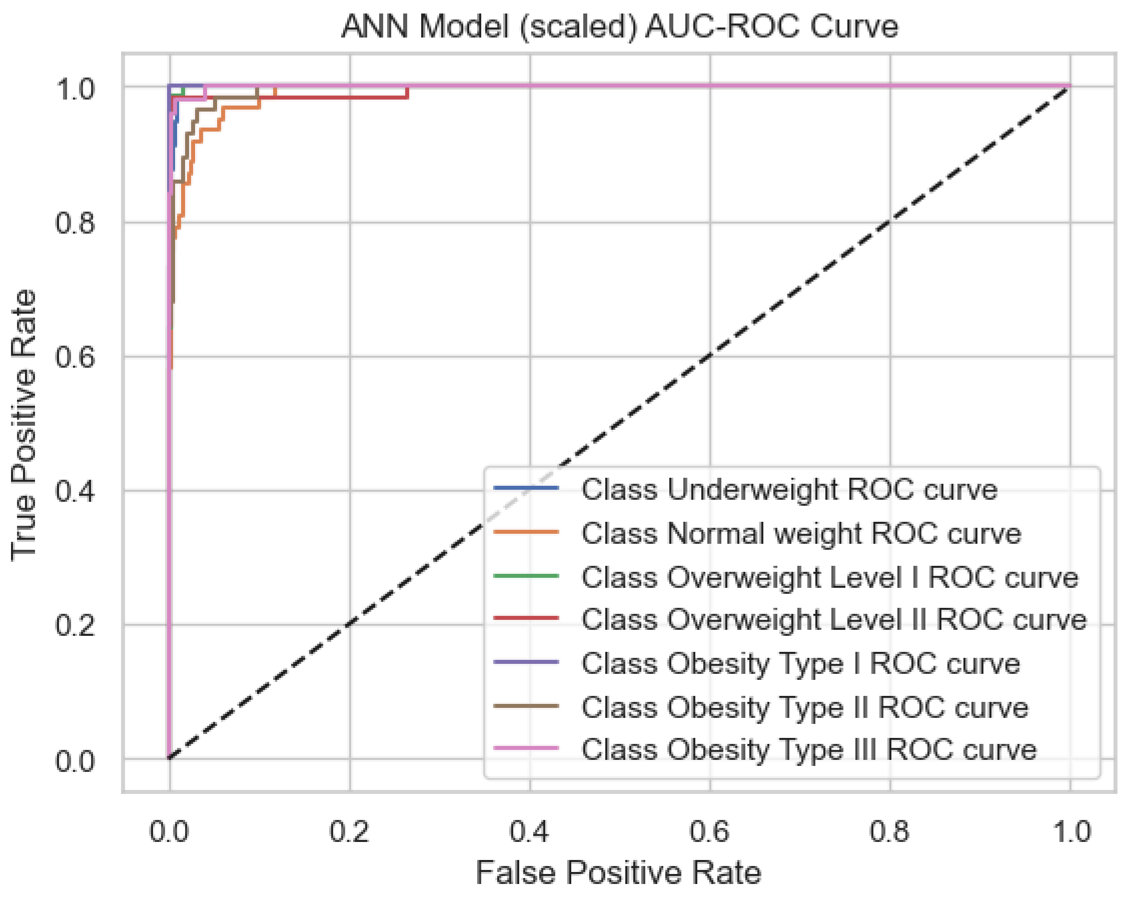
In [26]:
```
import numpy as np
import json
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.preprocessing import label_binarize
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
import tensorflow as tf

# Define the mapping from numeric labels to obesity types
label_mapping = {
    0: "Underweight",
    1: "Normal weight",
    2: "Overweight Level I",
    3: "Overweight Level II",
    4: "Obesity Type I",
    5: "Obesity Type II",
    6: "Obesity Type III"
}

# Function to plot confusion matrix with labels
def plot_confusion_matrix_with_labels(conf_matrix, title='Confusion Matrix',
    if label_mapping:
        labels = [label_mapping[i] for i in range(len(label_mapping))]
    else:
        labels = [str(i) for i in range(conf_matrix.shape[0])]

    plt.figure(figsize=(10, 8))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=True,
                xticklabels=labels, yticklabels=labels,
                annot_kws={"size": 12, "weight": "bold"}, linewidths=0.5, li
    plt.title(title)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.xticks(rotation=45, ha='right', fontsize=10)
    plt.yticks(rotation=0, fontsize=10)
    plt.show()

# Suppress AutoGraph warnings globally
@tf.autograph.experimental.do_not_convert
def model_predict(model, data):
    return model.predict(data)

# Define evaluation function
def evaluate_model(y_true, y_pred, y_proba, num_classes, model_name, best_pa
    # Output best model parameters
    print(f"{model_name} Best Parameters: {best_params}")

    # Calculate accuracy, precision, recall, F1 score
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    # Print evaluation results
    print(f"{model_name} Evaluation Results:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    # Confusion matrix
    conf_matrix = confusion_matrix(y_true, y_pred)
    print("Confusion Matrix:")
    print(conf_matrix)

    # Plot confusion matrix as a heatmap with labels
    plot_confusion_matrix_with_labels(conf_matrix, title=f'{model_name} Conf

    # AUC-ROC curve
    if num_classes > 2:  # Multiclass case
        y_true_bin = label_binarize(y_true, classes=np.arange(num_classes))
        auc = roc_auc_score(y_true_bin, y_proba, average='weighted', multi_c

        # Plot multiclass ROC curve
        fpr = dict()
        tpr = dict()
        plt.figure()
        for i in range(num_classes):
            fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_proba[:, i])
            plt.plot(fpr[i], tpr[i], label=f'Class {label_mapping[i]} ROC cu
    else:  # Binary case
        auc = roc_auc_score(y_true, y_proba[:, 1])
        fpr, tpr, _ = roc_curve(y_true, y_proba[:, 1])
        plt.figure()
        plt.plot(fpr, tpr, label=f'AUC-ROC (area = {auc:.2f})')

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'{model_name} AUC-ROC Curve')
    plt.legend(loc='lower right')
    plt.show()

# Load results file
with open('model_results.json', 'r') as f:
    results = json.load(f)

# Load and evaluate the best model for each dataset
for version, result in results.items():
    print(f"\nEvaluating Dataset: {version}")

    X_test = None
    y_test = None

    # Select the corresponding test set based on the version
    if version == 'standard':
        X_test, y_test = X_test_std.values, y_test_std.values
    elif version == 'scaled':
        X_test, y_test = X_test_scaled, y_test_scaled
    elif version == 'pca':
        X_test, y_test = X_test_pca, y_test_pca
    elif version == 'binned':
        X_test, y_test = X_test_binned.values, y_test_binned.values
    elif version == 'interaction':
        X_test, y_test = X_test_inter.values, y_test_inter.values

    # Ensure X_test is a NumPy array with the correct dtype
    X_test = np.array(X_test, dtype=np.float32)
    num_classes = len(np.unique(y_test))

    # Evaluate Random Forest model
    if 'RF' in result:
        best_rf_model = joblib.load(f'RF_best_model_{version}.pkl')
        best_rf_params = result['RF']['params']
        y_pred_rf = best_rf_model.predict(X_test)
        y_proba_rf = best_rf_model.predict_proba(X_test)
        evaluate_model(y_test, y_pred_rf, y_proba_rf, num_classes, f'RF Mode

    # Evaluate ANN model
    if 'ANN' in result:
        best_ann_model = load_model(f'ANN_best_model_{version}.h5')

        best_ann_params = result['ANN']['params']
        y_test_cat = to_categorical(y_test, num_classes=num_classes)
        y_proba_ann = model_predict(best_ann_model, X_test)
        y_pred_ann = np.argmax(y_proba_ann, axis=1)
        evaluate_model(y_test, y_pred_ann, y_proba_ann, num_classes, f'ANN M
Evaluating Dataset: standard
RF Model (standard) Best Parameters: {'n_estimators': 200, 'min_samples_spli
t': 2, 'max_depth': 20}
RF Model (standard) Evaluation Results:
Accuracy: 0.9669
Precision: 0.9671
Recall: 0.9669
F1 Score: 0.9670
Confusion Matrix:
[[55  1  0  0  0  0  0]
 [ 1 58  0  0  0  2  1]
 [ 0  0 77  1  0  0  0]
 [ 0  0  1 57  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0  5  0  0  0 51  0]
 [ 0  0  0  0  0  2 48]]
```



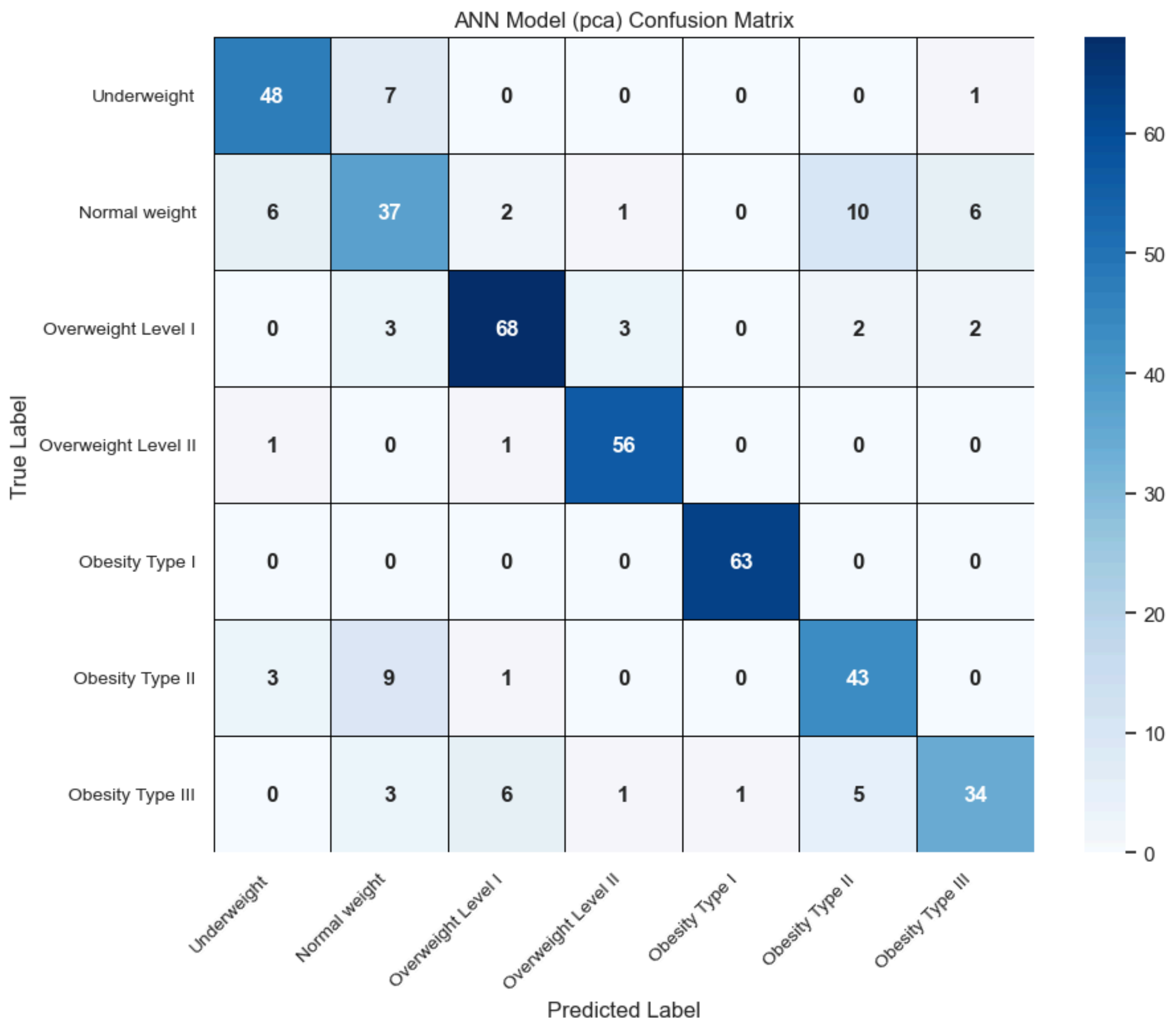


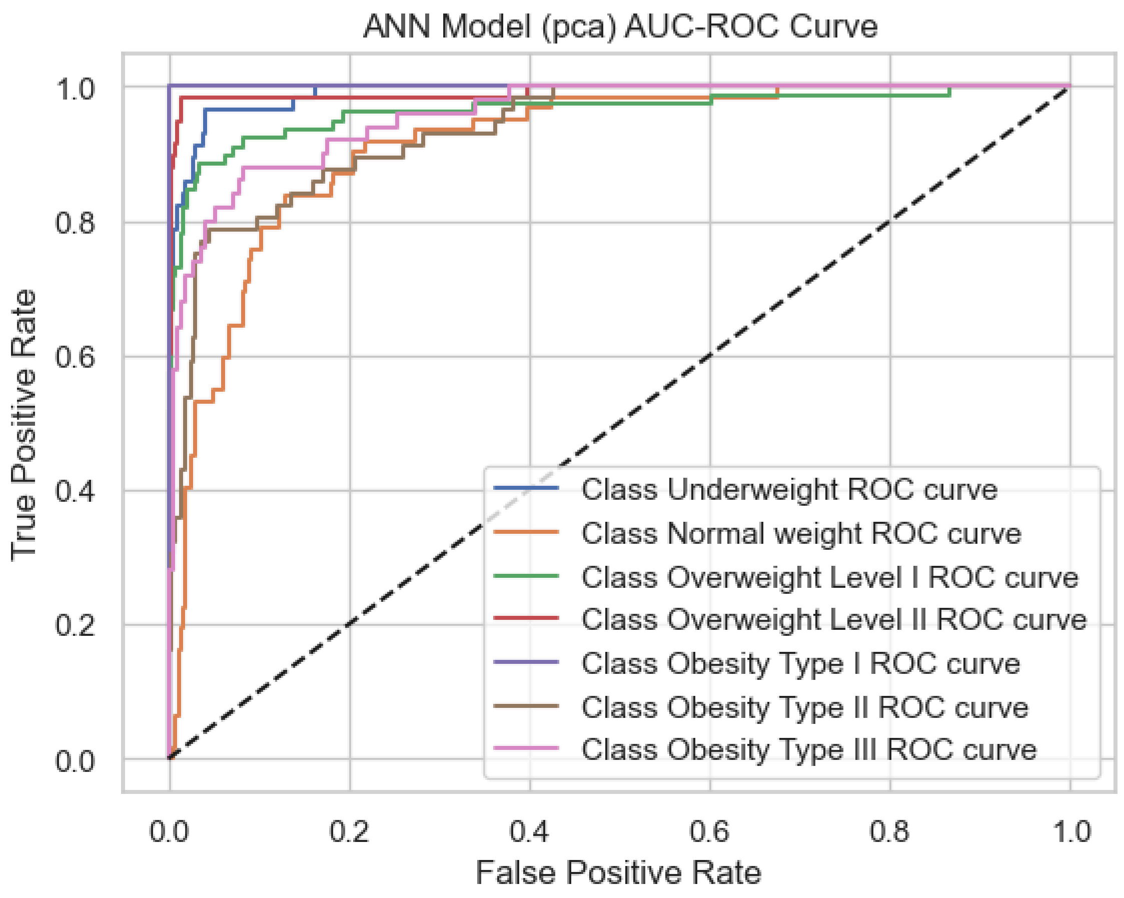
```
Evaluating Dataset: scaled
ANN Model (scaled) Best Parameters: {'neurons': 32, 'layers': 2, 'epochs': 1
00, 'batch_size': 10}
ANN Model (scaled) Evaluation Results:
Accuracy: 0.9433
Precision: 0.9435
Recall: 0.9433
F1 Score: 0.9429
Confusion Matrix:
[[55  1  0  0  0  0  0]
 [ 5 51  0  0  0  6  0]
 [ 0  0 77  1  0  0  0]
 [ 0  0  2 56  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0  5  0  0  0 50  1]
 [ 0  0  1  0  0  2 47]]
```


ANN Model (scaled) Confusion Matrix
True Label
Predicted Label
Underweight
Normal weight
Overweight Level I
Overweight Level II
Obesity Type I
Obesity Type II
Obesity Type III
55 1 0 0 0 0 0
5 51 0 0 0 6 0
0 0 77 1 0 0 0
0 0 2 56 0 0 0
0 0 0 0 63 0 0
0 5 0 0 0 50 1
0 0 1 0 0 2 47
70
60
50
40
30
20
10
0

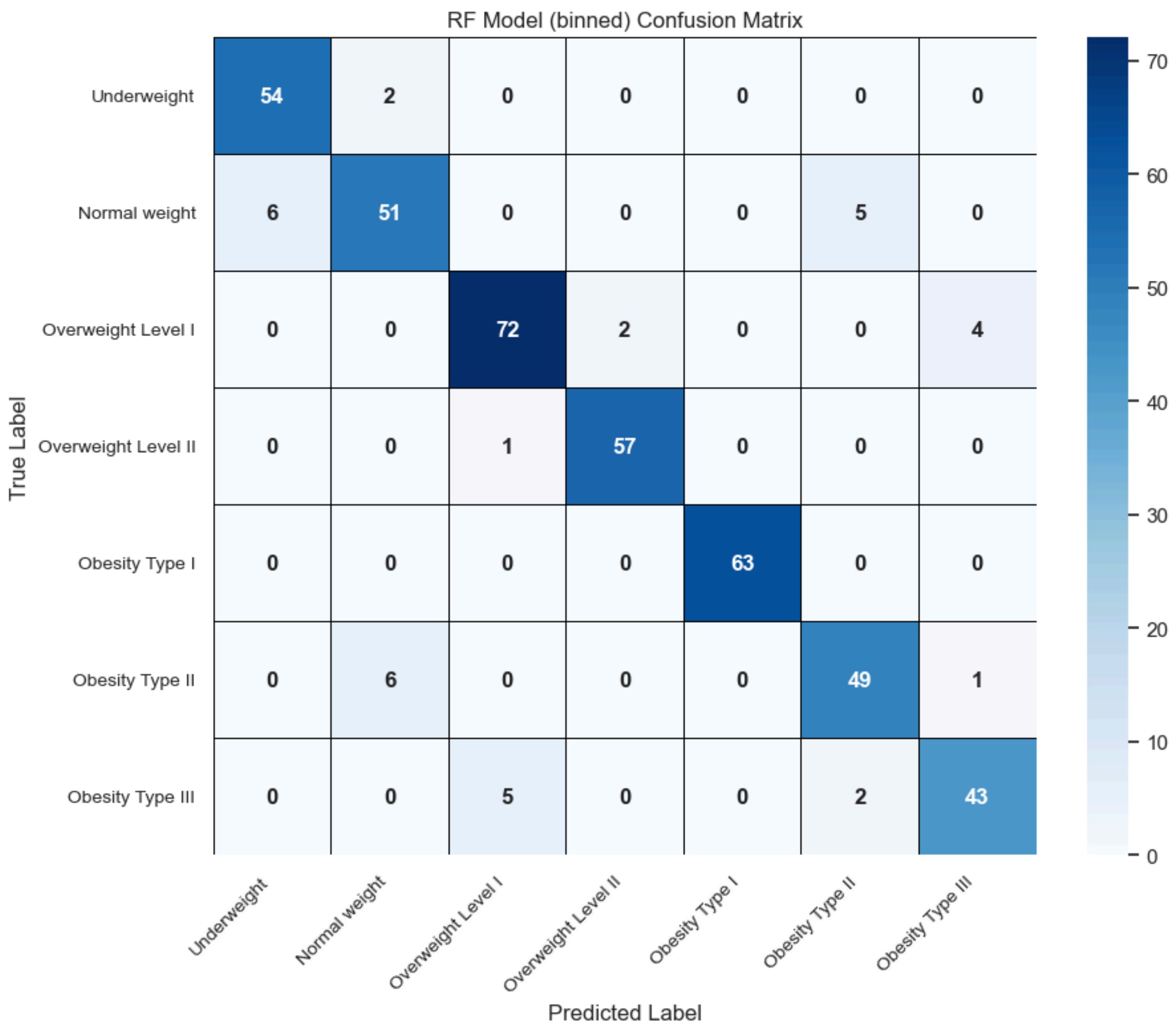


```
Evaluating Dataset: pca
ANN Model (pca) Best Parameters: {'neurons': 32, 'layers': 2, 'epochs': 100,
'batch_size': 20}
ANN Model (pca) Evaluation Results:
Accuracy: 0.8251
Precision: 0.8231
Recall: 0.8251
F1 Score: 0.8233
Confusion Matrix:
[[48  7  0  0  0  0  1]
 [ 6 37  2  1  0 10  6]
 [ 0  3 68  3  0  2  2]
 [ 1  0  1 56  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 3  9  1  0  0 43  0]
 [ 0  3  6  1  1  5 34]]
```

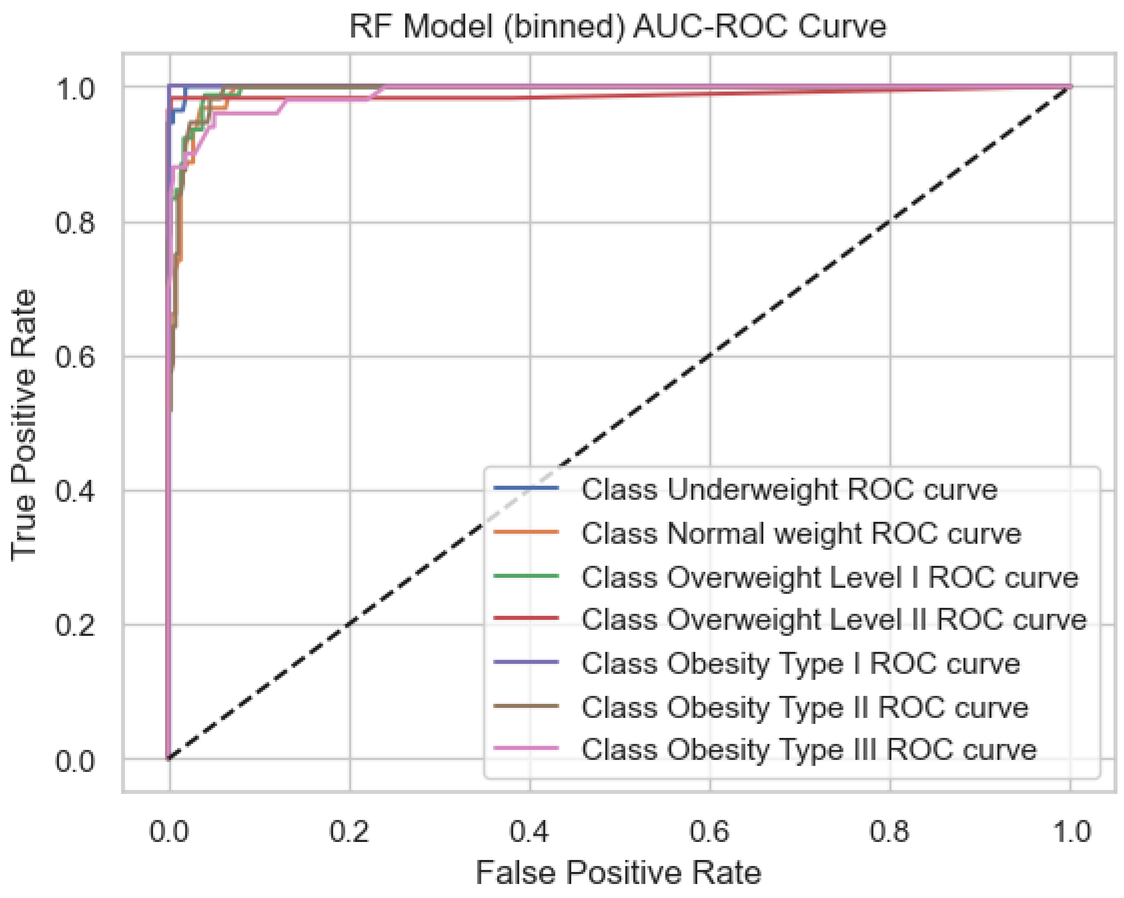

ANN Model (pca) Confusion Matrix
True Label
Predicted Label
Underweight
Normal weight
Overweight Level I
Overweight Level II
Obesity Type I
Obesity Type II
Obesity Type III
48 7 0 0 0 0 1
6 37 2 1 0 10 6
0 3 68 3 0 2 2
1 0 1 56 0 0 0
0 0 0 0 63 0 0
3 9 1 0 0 43 0
0 3 6 1 1 5 34
60
50
40
30
20
10
0



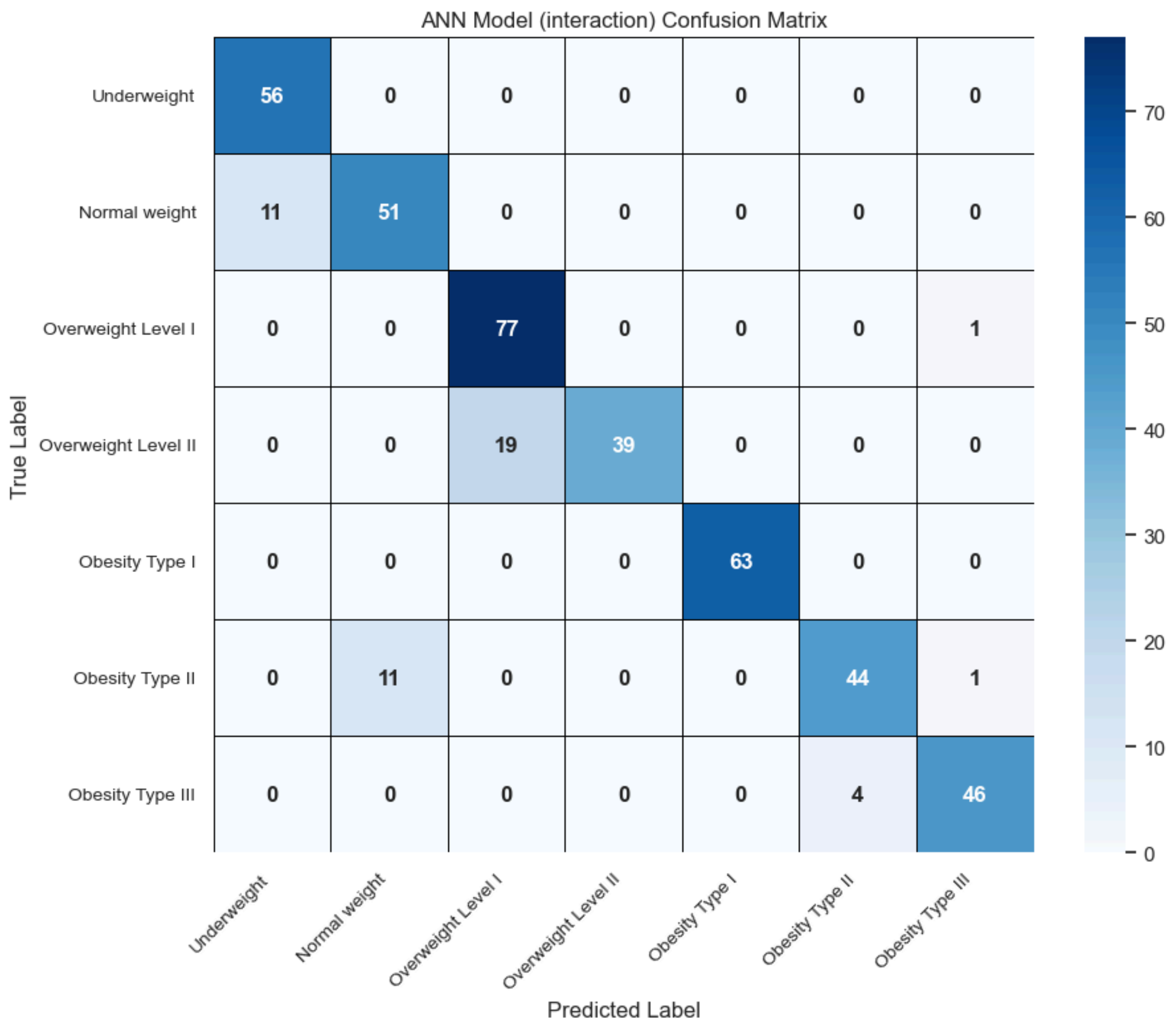
```
Evaluating Dataset: binned
RF Model (binned) Best Parameters: {'n_estimators': 100, 'min_samples_spli
t': 2, 'max_depth': None}
RF Model (binned) Evaluation Results:
Accuracy: 0.9196
Precision: 0.9192
Recall: 0.9196
F1 Score: 0.9191
Confusion Matrix:
[[54  2  0  0  0  0  0]
 [ 6 51  0  0  0  5  0]
 [ 0  0 72  2  0  0  4]
 [ 0  0  1 57  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0  6  0  0  0 49  1]
 [ 0  0  5  0  0  2 43]]
```



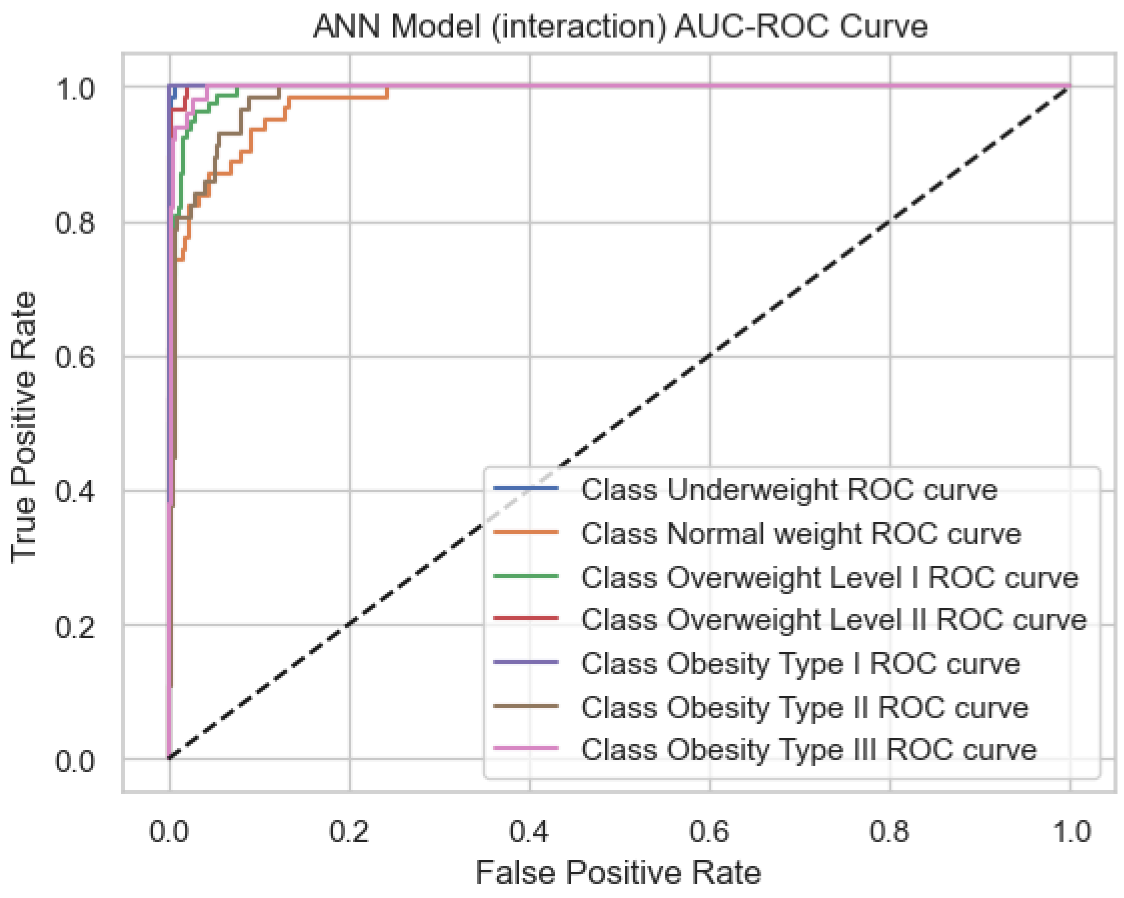
	Underweight	Normal weight	Overweight Level I	Overweight Level II	Obesity Type I	Obesity Type II	Obesity Type III
Underweight	54	2	0	0	0	0	0
Normal weight	6	51	0	0	0	5	0
Overweight Level I	0	0	72	2	0	0	4
Overweight Level II	0	0	1	57	0	0	0
Obesity Type I	0	0	0	0	63	0	0
Obesity Type II	0	6	0	0	0	49	1
Obesity Type III	0	0	5	0	0	2	43



```
Evaluating Dataset: interaction
ANN Model (interaction) Best Parameters: {'neurons': 32, 'layers': 2, 'epoch
s': 50, 'batch_size': 10}
ANN Model (interaction) Evaluation Results:
Accuracy: 0.8889
Precision: 0.8998
Recall: 0.8889
F1 Score: 0.8865
Confusion Matrix:
[[56  0  0  0  0  0  0]
 [11 51  0  0  0  0  0]
 [ 0  0 77  0  0  0  1]
 [ 0  0 19 39  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0 11  0  0  0 44  1]
 [ 0  0  0  0  0  4 46]]
```



True \ Predicted	Underweight	Normal weight	Overweight Level I	Overweight Level II	Obesity Type I	Obesity Type II	Obesity Type III
Underweight	56	0	0	0	0	0	0
Normal weight	11	51	0	0	0	0	0
Overweight Level I	0	0	77	0	0	0	1
Overweight Level II	0	0	19	39	0	0	0
Obesity Type I	0	0	0	0	63	0	0
Obesity Type II	0	11	0	0	0	44	1
Obesity Type III	0	0	0	0	0	4	46



5.3 **Experiment Summary**

This experiment analyzed the impact of various data preprocessing techniques and model hyperparameter configurations on obesity risk prediction. Five dataset versions were tested: standard, feature scaling, PCA-reduced, binned, and feature interaction. Random Forest (RF) achieved the best performance on the standard dataset with hyperparameters `n_estimators=200`, `max_depth=20`, and `min_samples_split=2`, attaining an accuracy of **96.69%**. This result highlights RF's robustness in capturing nonlinear relationships in raw features. Meanwhile, advanced preprocessing methods, such as feature scaling and interaction terms, enhanced the Artificial Neural Network (ANN), which achieved **88.89%** accuracy on the feature interaction dataset, demonstrating its adaptability to complex data patterns.

Random Forest (RF) performed exceptionally well on raw and binned datasets due to its robustness to unscaled features and ability to directly handle discrete inputs. RF also supports feature importance analysis, offering strong interpretability and practical insights. However, it struggled with datasets containing complex nonlinear relationships, such as the feature interaction version.

Artificial Neural Network (ANN) excelled on preprocessed datasets, leveraging its multilayer structure to model intricate relationships. Feature scaling and PCA significantly improved ANN’s performance, highlighting its sensitivity to input quality. However, ANN requires extensive computation and lacks RF’s interpretability.

Misclassification Analysis revealed that errors primarily occurred between "neighboring classes," such as "Overweight Level I" and "Overweight Level II," suggesting that feature differentiation or class boundaries could be improved.

Conclusion and Recommendations:
Model selection depends on task requirements. RF is ideal for scenarios prioritizing efficiency and interpretability, while ANN suits tasks involving high-dimensional, complex data. Future efforts could combine RF’s interpretability and ANN’s modeling power through ensemble or hybrid methods to enhance overall predictive performance.

6. Result and insights

Through an in-depth analysis and modeling of the obesity prediction dataset, this project has achieved several key findings that directly support the research objectives while providing valuable insights for future health management and policymaking.

First, the data exploration phase revealed significant driving factors behind obesity risk. Using data visualization, we observed that dietary habits and physical activity levels have a strong correlation with obesity. For instance, individuals who frequently consume high-calorie fast food and lack sufficient physical activity are significantly more likely to be classified as "obese" or "severely obese" compared to those with a healthier lifestyle. These findings not only provide a theoretical basis for feature selection in modeling but also offer actionable directions for public health policymakers, such as promoting the construction of community fitness facilities and implementing dietary education in schools.

Second, the evaluation results demonstrated that the Random Forest (RF) model outperformed other approaches on the standardized dataset (`standard`). After hyperparameter tuning, the optimal parameters were identified as 200 trees (`n_estimators`), a maximum depth (`max_depth`) of 20, and a minimum sample split (`min_samples_split`) of 2. On the test set, the RF model achieved an accuracy of **96.69%**, with precision, recall, and F1 scores all close to **96.7%**. The confusion matrix further revealed that while the model effectively classified "normal weight" and "slightly overweight" categories, there was some misclassification between "overweight" and "mild obesity." This indicates that future optimization could focus on adjusting class weights or introducing additional features to improve the model's ability to differentiate between adjacent categories.

Third, the experiments with Artificial Neural Networks (ANN) validated its potential in capturing complex nonlinear relationships. Although the overall accuracy of the ANN model was slightly lower than that of the RF model, it performed notably well on datasets with high-dimensional features, such as Principal Component Analysis (PCA) and interaction features. This highlights the suitability of deep learning techniques for modeling intricate relationships between variables, offering promising directions for future research on feature complexity.

Finally, the project demonstrated the practical value of predictive modeling. The models can identify high-risk individuals early and integrate with smart devices for real-time health monitoring. For example, users can adjust their daily dietary and exercise plans based on real-time feedback, while healthcare institutions can leverage predictive results to design more effective health education and intervention programs, ultimately reducing public healthcare costs related to obesity. This data-driven capability underscores the potential of machine learning models to enhance both individual health management and the scientific rigor of public health policies.

In summary, these findings validate the potential of machine learning models in obesity prediction and provide novel perspectives for personalized health management and intervention.

7. Conclusions

This project centered on obesity prediction, successfully demonstrating the applicability of data-driven techniques in public health management. The research process and results led to the following key conclusions:

Obesity is a multifactorial issue driven by dietary habits, physical activity, and body characteristics. These findings emphasize the importance of multidimensional intervention strategies. For example, community fitness programs, dietary guidance, and educational campaigns can significantly mitigate the impact of obesity. These insights are not only statistically significant but also practically valuable in addressing the challenges posed by obesity.

The Random Forest model showcased the efficiency of traditional machine learning algorithms in handling structured data, achieving an accuracy of **96.69%** on standardized datasets. This highlights the effectiveness of proper feature selection and parameter tuning in achieving superior performance for classification tasks. At the same time, the potential of ANN models in capturing complex features offers a promising avenue for future research, particularly in analyzing high-dimensional and nonlinear datasets.

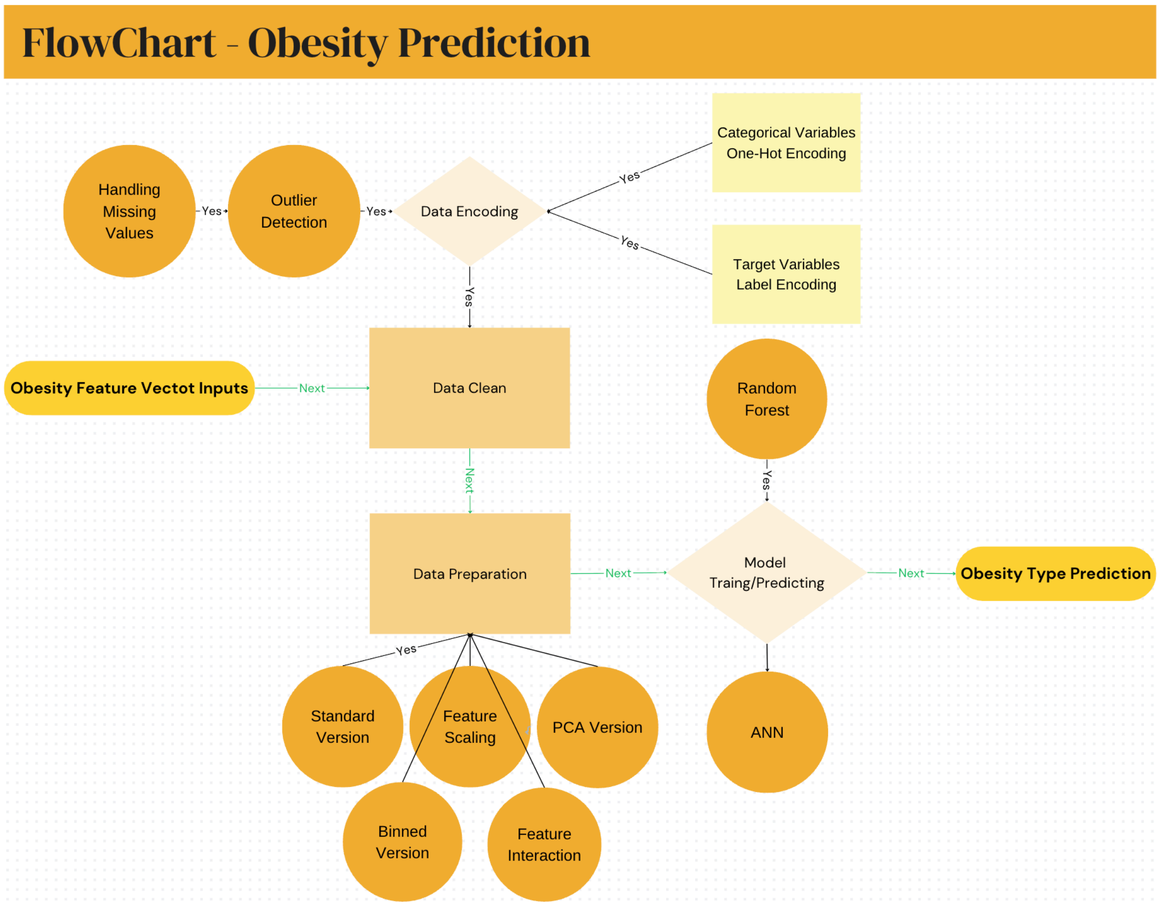
The study also underscored the wide-ranging applicability of predictive models in real-world scenarios. By integrating with smart devices, these models can assist individuals in real-time weight and dietary monitoring while providing healthcare institutions with efficient tools for health management. Additionally, governments can utilize predictive results to optimize resource allocation and policy implementation, effectively addressing the public health challenges posed by obesity.

Future research could focus on enhancing the diversity of data and improving model generalization. For instance, incorporating variables related to socioeconomic status and living environment could provide a more comprehensive understanding of the multidimensional causes of obesity. Meanwhile, exploring more advanced deep learning techniques, such as multitask learning and adaptive network architectures, could further improve prediction accuracy. Moreover, interdisciplinary collaboration with fields like nutrition and epidemiology could yield deeper insights into health management, contributing significantly to addressing obesity as a global public health challenge.

Through these efforts, we anticipate that this project will provide a solid technical foundation for obesity research and inspire new directions in personalized health management and policy optimization.

In [11]:
```
from IPython.display import HTML

base64_string_FL = """
iVBORw0KGgoAAAANSUhEUgAAByoAAAWUCAYAAACeAIF6AAAMTGlDQ1BJQ0MgUHJvZmlsZQAASImV
"""
HTML(f"""
<div style="text-align: center;">
    <img src="data:image/png;base64,{base64_string_FL}" alt="Random Forest S
    <p style="text-align: center; margin-top: 10px; font-size: 14px; line-he
        This flowchart outlines the process of handling missing values, dete
    </p>
</div>
""")
```

Out[11]:



This flowchart outlines the process of handling missing values, detecting outliers, and encoding variables during the data cleaning phase. Subsequently, multiple data preparation strategies were applied, including standardization, feature scaling, PCA transformation, feature interaction, and binned versions. Two models, Random Forest (RF) and Artificial Neural Networks (ANN), were trained and evaluated, culminating in obesity type prediction with actionable insights for health management.

Reference

[1] Sun, Z., Yuan, Y., Farrahi, V., et al. (2024). *Using interpretable machine learning methods to identify the relative importance of lifestyle factors for overweight and obesity in adults: pooled evidence from CHNS and NHANES*. BMC Public Health, 24, Article 3034. Retrieved from https://bmcpublichealth.biomedcentral.com/articles/10.1186/s12889-024-20510-z

[2] Brownlee, J. (2020, December 3). *Bagging and Random Forest Ensemble Algorithms for Machine Learning*. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/

[3] Li, J., Cheng, J., Shi, J., & Huang, F. (2012). *Brief Introduction of Back Propagation (BP) Neural Network Algorithm and Its Improvement*. In D. Jin & S. Lin (Eds.), *Advances in Computer Science and Information Engineering* (Vol. 169, pp. 553–558). Springer, Berlin, Heidelberg. Retrieved from https://doi.org/10.1007/978-3-642-30223-7_87