```
  # Install if needed
!pip install pandas matplotlib seaborn --quiet

# Import packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# For better visuals
sns.set(style="whitegrid")
```

```
# Load the gym data
gym_df = pd.read_csv('/content/gym_members_exercise_tracking.csv')

# Load the food data (change name if needed)
food_df = pd.read_csv('/content/daily_food_nutrition_dataset.csv')  # or use the exact file name
```

```
# View the first few rows of each
print("Gym Data:")
display(gym_df.head())

print("\nFood Data:")
display(food_df.head())
```

```
Gym Data:
```

	Age	Gender	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Workout_Type	Fat_Percentage	W
0	56	Male	88.3	1.71	180	157	60	1.69	1313.0	Yoga	12.6	
1	46	Female	74.9	1.53	179	151	66	1.30	883.0	HIIT	33.9	
2	32	Female	68.1	1.66	167	122	54	1.11	677.0	Cardio	33.4	
3	25	Male	53.2	1.70	190	164	56	0.59	532.0	Strength	28.8	
4	38	Male	46.1	1.79	188	158	68	0.64	556.0	Strength	29.2	

```
Food Data:
```

	Date	User_ID	Food_Item	Category	Calories (kcal)	Protein (g)	Carbohydrates (g)	Fat (g)	Fiber (g)	Sugars (g)	Sodium (mg)	Cholesterol (mg)	Meal_Type	Wat
0	2024-09-11	496	Eggs	Meat	173	42.4	83.7	1.5	1.5	12.7	752	125	Lunch	
1	2024-12-17	201	Apple	Fruits	66	39.2	13.8	3.2	2.6	12.2	680	97	Lunch	
2	2024-06-09	776	Chicken Breast	Meat	226	27.1	79.1	25.8	3.2	44.7	295	157	Breakfast	
3	2024-08-27	112	Banana	Fruits	116	43.4	47.1	16.1	6.5	44.1	307	13	Snack	
4	2024-07-28	622	Banana	Fruits	500	33.9	75.8	47.0	7.8	19.4	358	148	Lunch	

```
# ================================
# 👉 Gym Exercise Data Overview
# ================================
print("🔍 Gym Dataset Overview")
```

```
print("\n➡️ First 5 Rows:")
display(gym_df.head())

print("\n➡️ Last 5 Rows:")
display(gym_df.tail())

print("\n➡️ Column Info:")
gym_df.info()

print("\n➡️ Summary Statistics:")
display(gym_df.describe(include='all'))
```

🔍 Gym Dataset Overview

➡️ First 5 Rows:

	Age	Gender	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Workout_Type	Fat_Percentage	W
0	56	Male	88.3	1.71	180	157	60	1.69	1313.0	Yoga	12.6	
1	46	Female	74.9	1.53	179	151	66	1.30	883.0	HIIT	33.9	
2	32	Female	68.1	1.66	167	122	54	1.11	677.0	Cardio	33.4	
3	25	Male	53.2	1.70	190	164	56	0.59	532.0	Strength	28.8	
4	38	Male	46.1	1.79	188	158	68	0.64	556.0	Strength	29.2	

➡️ Last 5 Rows:

	Age	Gender	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Workout_Type	Fat_Percentage
968	24	Male	87.1	1.74	187	158	67	1.57	1364.0	Strength	10.0
969	25	Male	66.6	1.61	184	166	56	1.38	1260.0	Strength	25.0
970	59	Female	60.4	1.76	194	120	53	1.72	929.0	Cardio	18.8
971	32	Male	126.4	1.83	198	146	62	1.10	883.0	HIIT	28.2
972	46	Male	88.7	1.63	166	146	66	0.75	542.0	Strength	28.8

➡️ Column Info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 973 entries, 0 to 972
Data columns (total 15 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Age                            973 non-null    int64  
 1   Gender                         973 non-null    object 
 2   Weight (kg)                    973 non-null    float64
 3   Height (m)                     973 non-null    float64
 4   Max_BPM                        973 non-null    int64  
```

```
 5   Avg_BPM                        973 non-null    int64
 6   Resting_BPM                    973 non-null    int64
 7   Session_Duration (hours)       973 non-null    float64
 8   Calories_Burned                973 non-null    float64
 9   Workout_Type                   973 non-null    object
 10  Fat_Percentage                 973 non-null    float64
 11  Water_Intake (liters)          973 non-null    float64
 12  Workout_Frequency (days/week)  973 non-null    int64
 13  Experience_Level               973 non-null    int64
 14  BMI                            973 non-null    float64
dtypes: float64(7), int64(6), object(2)
memory usage: 114.2+ KB

➡️ Summary Statistics:
```

	Age	Gender	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Workout_
count	973.000000	973	973.000000	973.00000	973.000000	973.000000	973.000000	973.000000	973.000000	
unique	NaN	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
top	NaN	Male	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Str
freq	NaN	511	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
mean	38.683453	NaN	73.854676	1.72258	179.883864	143.766701	62.223022	1.256423	905.422405	
std	12.180928	NaN	21.207500	0.12772	11.525686	14.345101	7.327060	0.343033	272.641516	
min	18.000000	NaN	40.000000	1.50000	160.000000	120.000000	50.000000	0.500000	303.000000	
25%	28.000000	NaN	58.100000	1.62000	170.000000	131.000000	56.000000	1.040000	720.000000	
50%	40.000000	NaN	70.000000	1.71000	180.000000	143.000000	62.000000	1.260000	893.000000	
75%	49.000000	NaN	86.000000	1.80000	190.000000	156.000000	68.000000	1.460000	1076.000000	
max	59.000000	NaN	129.900000	2.00000	199.000000	169.000000	74.000000	2.000000	1783.000000	

```
# ===============================
# 👉 Food & Nutrition Data Overview
# ===============================
print("🥗 Food & Nutrition Dataset Overview")

print("\n➡️ First 5 Rows:")
display(food_df.head())

print("\n➡️ Last 5 Rows:")
display(food_df.tail())

print("\n➡️ Column Info:")
food_df.info()

print("\n➡️ Summary Statistics:")
display(food_df.describe(include='all'))
```

🥗 Food & Nutrition Dataset Overview

➡️ First 5 Rows:

	Date	User_ID	Food_Item	Category	Calories (kcal)	Protein (g)	Carbohydrates (g)	Fat (g)	Fiber (g)	Sugars (g)	Sodium (mg)	Cholesterol (mg)	Meal_Type	Wat
0	2024-09-11	496	Eggs	Meat	173	42.4	83.7	1.5	1.5	12.7	752	125	Lunch	
1	2024-12-17	201	Apple	Fruits	66	39.2	13.8	3.2	2.6	12.2	680	97	Lunch	
2	2024-06-09	776	Chicken Breast	Meat	226	27.1	79.1	25.8	3.2	44.7	295	157	Breakfast	
3	2024-08-27	112	Banana	Fruits	116	43.4	47.1	16.1	6.5	44.1	307	13	Snack	
4	2024-07-28	622	Banana	Fruits	500	33.9	75.8	47.0	7.8	19.4	358	148	Lunch	

➡️ Last 5 Rows:

	Date	User_ID	Food_Item	Category	Calories (kcal)	Protein (g)	Carbohydrates (g)	Fat (g)	Fiber (g)	Sugars (g)	Sodium (mg)	Cholesterol (mg)	Meal_Type
9995	2024-09-18	455	Salmon	Meat	346	29.8	55.6	4.6	1.7	0.5	976	87	Breakfast
9996	2024-12-13	913	Grapes	Fruits	174	22.9	54.9	32.1	2.5	5.9	255	56	Lunch
9997	2024-01-31	943	Strawberry	Fruits	63	36.5	23.8	21.6	0.8	48.9	757	63	Snack
9998	2024-09-28	571	Spinach	Vegetables	564	26.2	58.9	11.9	3.3	43.0	482	33	Breakfast
9999	2024-09-07	33	Banana	Fruits	442	20.9	27.3	29.6	9.9	30.9	919	22	Dinner

➡️ Column Info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Date               10000 non-null  object 
 1   User_ID            10000 non-null  int64  
 2   Food_Item          10000 non-null  object 
 3   Category           10000 non-null  object 
 4   Calories (kcal)    10000 non-null  int64  
 5   Protein (g)        10000 non-null  float64
 6   Carbohydrates (g)  10000 non-null  float64
 7   Fat (g)            10000 non-null  float64
 8   Fiber (g)          10000 non-null  float64
 9   Sugars (g)         10000 non-null  float64
 10  Sodium (mg)        10000 non-null  int64  
 11  Cholesterol (mg)   10000 non-null  int64  
 12  Meal_Type          10000 non-null  object 
 13  Water_Intake (ml)  10000 non-null  int64  
dtypes: float64(5), int64(5), object(4)
memory usage: 1.1+ MB

➡️ Summary Statistics:
```

	Date	User_ID	Food_Item	Category	Calories (kcal)	Protein (g)	Carbohydrates (g)	Fat (g)	Fiber (g)	Sugars (g
count	10000	10000.000000	10000	10000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.0000(
unique	366	NaN	35	7	NaN	NaN	NaN	NaN	NaN	Na
top	2024-05-20	NaN	Milk	Dairy	NaN	NaN	NaN	NaN	NaN	Na
freq	45	NaN	311	1460	NaN	NaN	NaN	NaN	NaN	Na
mean	NaN	498.706300	NaN	NaN	327.693900	25.523050	52.568550	25.43735	4.986940	25.05257
std	NaN	289.123477	NaN	NaN	158.194716	14.131993	27.387152	14.14532	2.864984	14.4806(
min	NaN	1.000000	NaN	NaN	50.000000	1.000000	5.000000	1.00000	0.000000	0.0000(
25%	NaN	245.000000	NaN	NaN	190.000000	13.200000	28.800000	13.30000	2.500000	12.5000(

50%	NaN	492.000000	NaN	NaN	328.000000	25.500000	52.800000	25.30000	5.000000	25.00000
75%	NaN	748.000000	NaN	NaN	464.000000	37.700000	76.400000	37.60000	7.500000	37.70000
max	NaN	1000.000000	NaN	NaN	600.000000	50.000000	100.000000	50.00000	10.000000	50.00000

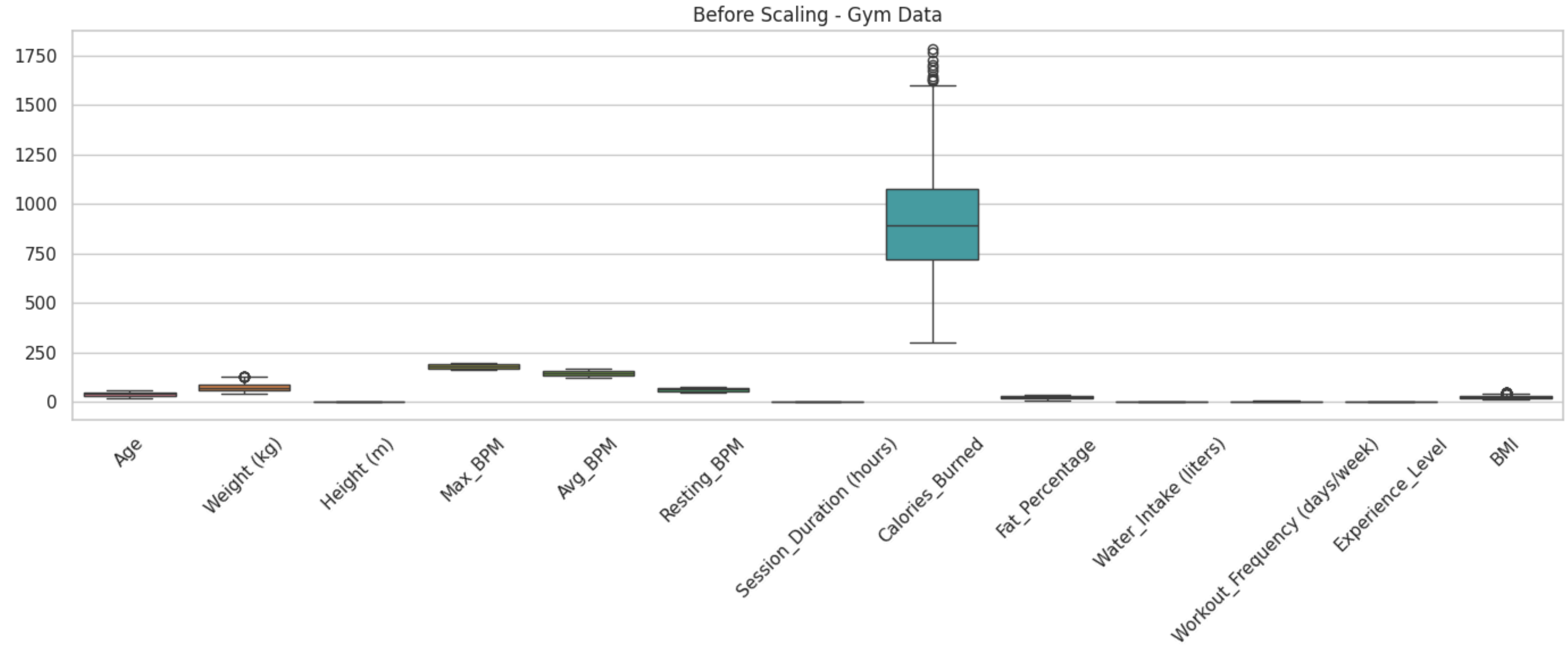
```
print("Missing values in Gym Data:")
print(gym_df.isnull().sum())

print("Missing values in Food Data:")
print(food_df.isnull().sum())
```

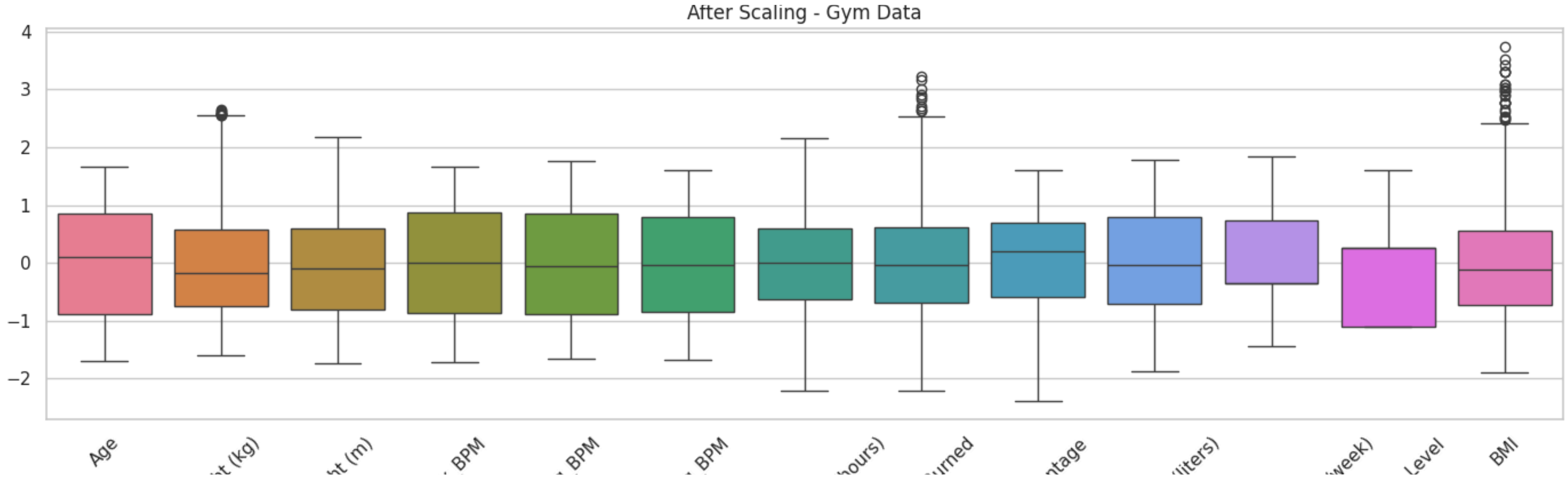
```
Missing values in Gym Data:
Age                              0
Gender                           0
Weight (kg)                      0
Height (m)                       0
Max_BPM                          0
Avg_BPM                          0
Resting_BPM                      0
Session_Duration (hours)         0
Calories_Burned                  0
Workout_Type                     0
Fat_Percentage                   0
Water_Intake (liters)            0
Workout_Frequency (days/week)    0
Experience_Level                 0
BMI                              0
dtype: int64
Missing values in Food Data:
Date                    0
User_ID                 0
Food_Item               0
Category                0
Calories (kcal)         0
Protein (g)             0
Carbohydrates (g)       0
Fat (g)                 0
Fiber (g)               0
Sugars (g)              0
Sodium (mg)             0
Cholesterol (mg)        0
Meal_Type               0
Water_Intake (ml)       0
dtype: int64
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df_gym = pd.read_csv("gym_members_exercise_tracking.csv")

# Step 1: Check and Handle Missing Values
print("Missing Values:\n", df_gym.isnull().sum())

# Step 2: Encode Categorical Columns
df_gym_processed = df_gym.copy()
categorical_cols = ['Gender', 'Workout_Type']
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df_gym_processed[col] = le.fit_transform(df_gym_processed[col])
    label_encoders[col] = le

# Step 3: Feature Scaling
numerical_cols = ['Age', 'Weight (kg)', 'Height (m)', 'Max_BPM', 'Avg_BPM', 'Resting_BPM',
                  'Session_Duration (hours)', 'Calories_Burned', 'Fat_Percentage',
                  'Water_Intake (liters)', 'Workout_Frequency (days/week)', 'Experience_Level', 'BMI']

scaler = StandardScaler()
df_gym_processed[numerical_cols] = scaler.fit_transform(df_gym_processed[numerical_cols])

# Step 4: Visualization
plt.figure(figsize=(14, 6))
plt.title("Before Scaling - Gym Data")
sns.boxplot(data=df_gym[numerical_cols])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
plt.figure(figsize=(14, 6))
plt.title("After Scaling - Gym Data")
sns.boxplot(data=df_gym_processed[numerical_cols])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Missing Values:
 Age                             0
Gender                          0
Weight (kg)                     0
Height (m)                      0
Max_BPM                         0
Avg_BPM                         0
Resting_BPM                     0
Session_Duration (hours)        0
Calories_Burned                 0
Workout_Type                    0
Fat_Percentage                  0
Water_Intake (liters)           0
Workout_Frequency (days/week)   0
Experience_Level                0
BMI                             0
dtype: int64
```

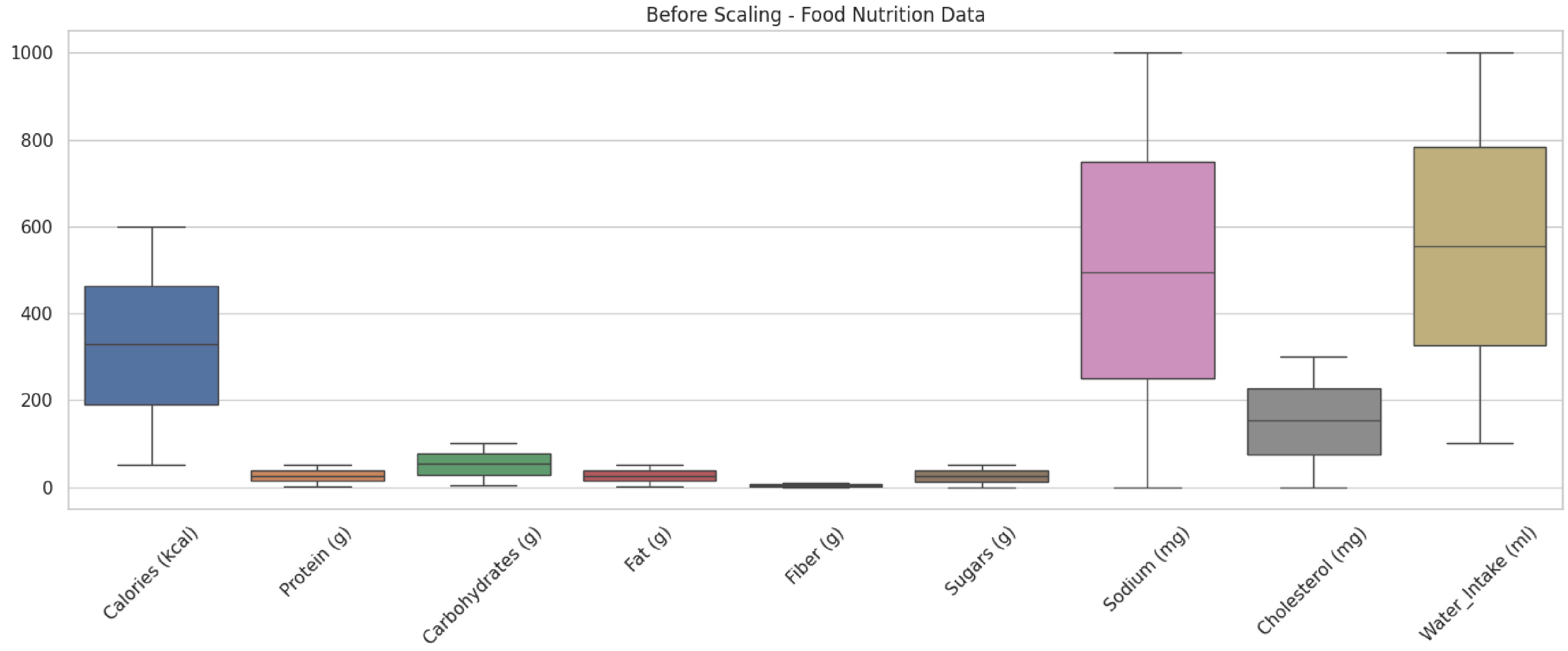




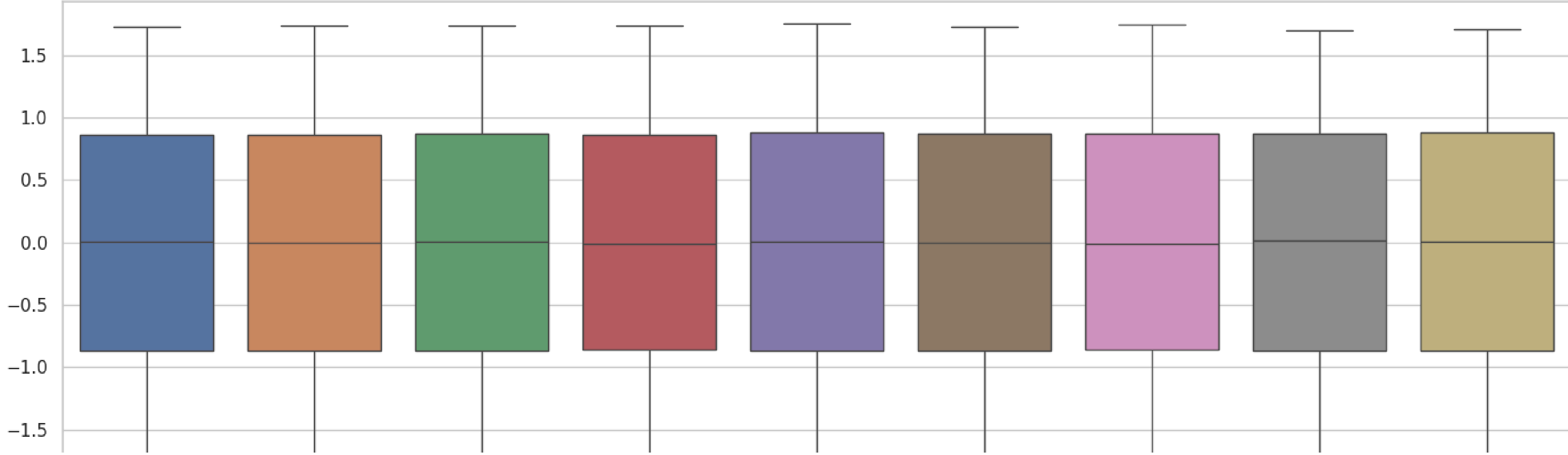

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df_food = pd.read_csv("daily_food_nutrition_dataset.csv")

# Step 1: Check and Handle Missing Values
print("Missing Values:\n", df_food.isnull().sum())

# Step 2: Encode Categorical Columns
df_food_processed = df_food.copy()
categorical_cols = ['Food_Item', 'Category', 'Meal_Type']
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df_food_processed[col] = le.fit_transform(df_food_processed[col])
    label_encoders[col] = le
```

```
# Step 3: Feature Scaling for Numeric Columns
numerical_cols = ['Calories (kcal)', 'Protein (g)', 'Carbohydrates (g)', 'Fat (g)',
                  'Fiber (g)', 'Sugars (g)', 'Sodium (mg)', 'Cholesterol (mg)', 'Water_Intake (ml)']

scaler = StandardScaler()
df_food_processed[numerical_cols] = scaler.fit_transform(df_food_processed[numerical_cols])

# Step 4: Visualization
plt.figure(figsize=(14, 6))
plt.title("Before Scaling - Food Nutrition Data")
sns.boxplot(data=df_food[numerical_cols])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.figure(figsize=(14, 6))
plt.title("After Scaling - Food Nutrition Data")
sns.boxplot(data=df_food_processed[numerical_cols])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Missing Values:
 Date                 0
User_ID              0
Food_Item            0
Category             0
Calories (kcal)      0
Protein (g)          0
Carbohydrates (g)    0
Fat (g)              0
Fiber (g)            0
Sugars (g)           0
Sodium (mg)          0
Cholesterol (mg)     0
Meal_Type            0
Water_Intake (ml)    0
dtype: int64
```



After Scaling - Food Nutrition Data




```
import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from collections import Counter

# Let's assume 'Workout_Type' is your target column
# Load your already preprocessed and encoded dataset
df = pd.read_csv('/content/gym_members_exercise_tracking.csv')

# Encode categorical variables
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Workout_Type'] = le.fit_transform(df['Workout_Type'])  # Target
df['Gender'] = le.fit_transform(df['Gender'])              # Input

# Define features (X) and target (y)
X = df.drop('Workout_Type', axis=1)
y = df['Workout_Type']

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

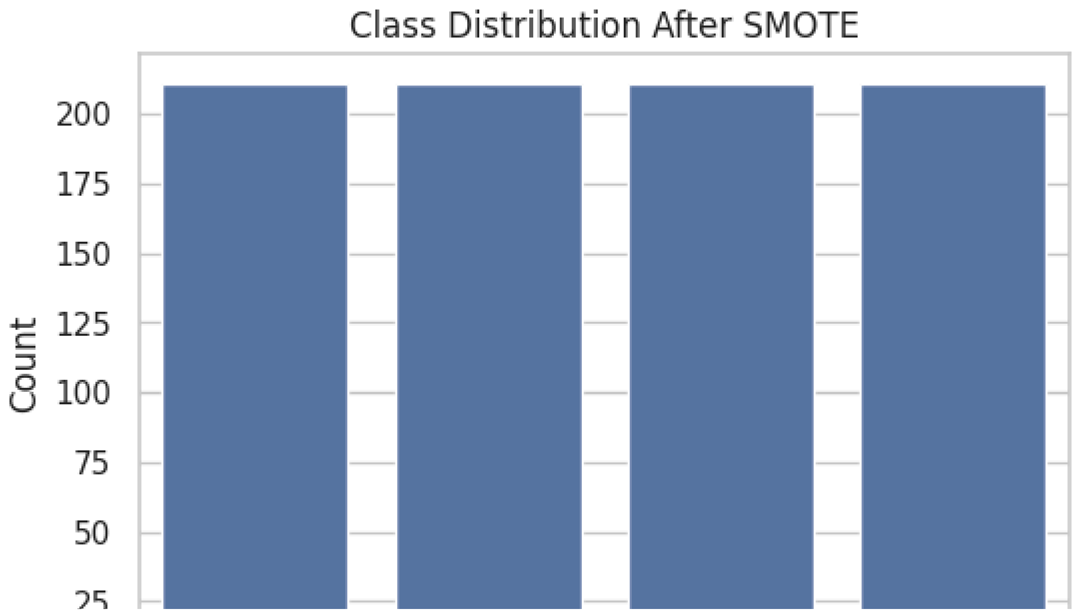
```
# Before balancing
print("Before SMOTE:", Counter(y_train))

# Apply SMOTE to training set
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# After balancing
print("After SMOTE:", Counter(y_train_balanced))
```

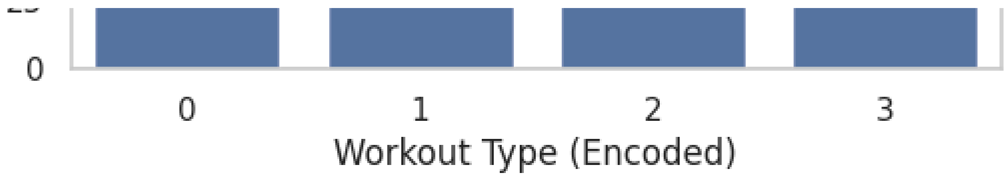
```
Before SMOTE: Counter({2: 211, 0: 209, 3: 179, 1: 179})
After SMOTE: Counter({3: 211, 1: 211, 0: 211, 2: 211})
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from collections import Counter

# Load dataset
df = pd.read_csv("gym_members_exercise_tracking.csv")

# Label Encoding for categorical variables
le = LabelEncoder()
df['Workout_Type'] = le.fit_transform(df['Workout_Type'])  # Target
df['Gender'] = le.fit_transform(df['Gender'])              # Input

# Define features and target
X = df.drop('Workout_Type', axis=1)
y = df['Workout_Type']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Plot class distribution BEFORE SMOTE
plt.figure(figsize=(6, 4))
sns.countplot(x=y_train)
plt.title('Class Distribution Before SMOTE')
plt.xlabel('Workout Type (Encoded)')
plt.ylabel('Count')
plt.show()

# Apply SMOTE to balance the training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# Plot class distribution AFTER SMOTE
plt.figure(figsize=(6, 4))
sns.countplot(x=y_train_balanced)
plt.title('Class Distribution After SMOTE')
plt.xlabel('Workout Type (Encoded)')
plt.ylabel('Count')
plt.show()
```


Class Distribution Before SMOTE
Count
200
175
150
125
100
75
50
25
0
0
1
2
3
Workout Type (Encoded)

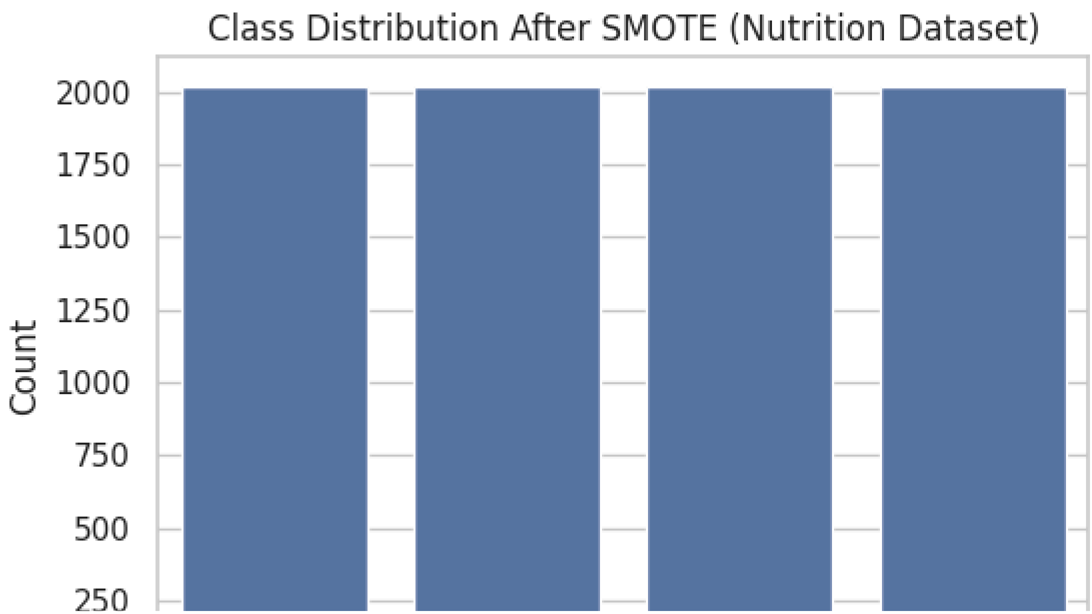

Class Distribution After SMOTE
Count
200
175
150
125
100
75
50
25




```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder, StandardScaler
from collections import Counter

# Load dataset
df = pd.read_csv('/content/daily_food_nutrition_dataset.csv')

# Encode categorical variables
le = LabelEncoder()
df['Meal_Type'] = le.fit_transform(df['Meal_Type'])     # Target
df['Food_Item'] = le.fit_transform(df['Food_Item'])     # Input
df['Category'] = le.fit_transform(df['Category'])       # Input

# Optional: scale numerical features for better model performance
# Correcting the numerical column names to match the dataset
numerical_cols = ['Calories (kcal)', 'Protein (g)', 'Carbohydrates (g)', 'Fat (g)',
                  'Fiber (g)', 'Sugars (g)', 'Sodium (mg)'] # Adjusted based on previous successful code block

scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Define features and target
# Assuming 'Water_Intake (ml)' is also a numerical column to be included as a feature,
# even if not scaled in this block
# Exclude the 'Date' column as it's not numerical and not relevant for SMOTE
X = df.drop(['Meal_Type', 'Date'], axis=1)
y = df['Meal_Type']
```

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Plot class distribution BEFORE SMOTE
plt.figure(figsize=(6, 4))
sns.countplot(x=y_train)
plt.title('Class Distribution Before SMOTE (Nutrition Dataset)')
plt.xlabel('Meal Type (Encoded)')
plt.ylabel('Count')
plt.show()

# Apply SMOTE to balance the training set
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# Plot class distribution AFTER SMOTE
plt.figure(figsize=(6, 4))
sns.countplot(x=y_train_balanced)
plt.title('Class Distribution After SMOTE (Nutrition Dataset)')
plt.xlabel('Meal Type (Encoded)')
plt.ylabel('Count')
plt.show()
```


Class Distribution Before SMOTE (Nutrition Dataset)
Count
Meal Type (Encoded)
0
1
2
3
0
250
500
750
1000
1250
1500
1750
2000


Class Distribution After SMOTE (Nutrition Dataset)
Count
250
500
750
1000
1250
1500
1750
2000


```
# Import necessary library
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Assuming 'df' is your DataFrame loaded with the data
# And assuming you have already run the previous steps to load the data

# --- Add the following code after you fit_transform the 'Workout_Type' column ---

# Example of fitting LabelEncoder (replace with your actual data loading and encoding)
# df = pd.read_csv('/content/gym_members_exercise_tracking.csv')
# le = LabelEncoder()
# df['Workout_Type_Encoded'] = le.fit_transform(df['Workout_Type']) # Use a new column name to keep original


# To demonstrate, let's assume you have run the following line as in your original code block:
df = pd.read_csv('/content/gym_members_exercise_tracking.csv')
le = LabelEncoder()
df['Workout_Type'] = le.fit_transform(df['Workout_Type'])  # The encoding happens here

# Now, get the mapping
# The 'classes_' attribute of the fitted LabelEncoder contains the original unique values
original_classes = le.classes_

# The mapping is implicit based on the index of 'classes_'
# For example, original_classes[0] corresponds to encoded label 0
# original_classes[1] corresponds to encoded label 1, and so on.

# Create a dictionary to clearly show the mapping
encoding_mapping = dict(zip(le.transform(original_classes), original_classes))
```

```
# Print the mapping
print("Mapping of Encoded Labels to Original Workout Types:")
for encoded_label, original_class in encoding_mapping.items():
    print(f"{encoded_label}: {original_class}")

# --- End of the code to distinguish ---

# You can then continue with the rest of your original code (splitting, SMOTE, etc.)
# X = df.drop('Workout_Type', axis=1)
# y = df['Workout_Type']
# ... rest of your code
```

```
Mapping of Encoded Labels to Original Workout Types:
0: Cardio
1: HIIT
2: Strength
3: Yoga
```

```
# Import necessary libraries
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Assuming df_food is your DataFrame loaded with the food data
# And assuming you have already run the previous steps to load and process the food data

# --- Add the following code after you fit_transform the categorical columns for food data ---

# Example of loading and encoding (replace with your actual code block)
# df_food_processed = df_food.copy()
# categorical_cols = ['Food_Item', 'Category', 'Meal_Type']
# label_encoders = {} # Assuming you stored encoders here

# For demonstration, let's recreate the encoding step you had:
df_food = pd.read_csv("daily_food_nutrition_dataset.csv") # Load data again if needed or use your existing df_food
df_food_processed = df_food.copy()
categorical_cols = ['Food_Item', 'Category', 'Meal_Type']
label_encoders = {} # Dictionary to store encoders
```

```
for col in categorical_cols:
    le = LabelEncoder()
    df_food_processed[col] = le.fit_transform(df_food_processed[col])
    label_encoders[col] = le # Store the fitted encoder for this column

# --- End of example encoding block ---

# Now, print the mapping for each encoded column
print("\nMapping for Encoded Nutrition Data Columns:")

for col in categorical_cols:
    le = label_encoders[col] # Get the fitted encoder for this column
    original_classes = le.classes_ # Get the original unique values
    encoding_mapping = dict(zip(le.transform(original_classes), original_classes))

    print(f"\n--- Mapping for '{col}' ---")
    for encoded_label, original_class in encoding_mapping.items():
        print(f"{encoded_label}: {original_class}")

# --- End of the code to distinguish nutrition data encodings ---

# You can then continue with the rest of your original code (scaling, visualization, SMOTE if applicable)
# ... rest of your code for food data processing
```

```
Mapping for Encoded Nutrition Data Columns:

--- Mapping for 'Food_Item' ---
0: Apple
1: Banana
2: Beef Steak
3: Bread
4: Broccoli
5: Butter
6: Carrot
7: Cheese
8: Chicken Breast
9: Chips
```

```
10: Chocolate
11: Coffee
12: Cookies
13: Eggs
14: Grapes
15: Green Tea
16: Milk
17: Milkshake
18: Nuts
19: Oats
20: Orange
21: Orange Juice
22: Paneer
23: Pasta
24: Popcorn
25: Pork Chop
26: Potato
27: Quinoa
28: Rice
29: Salmon
30: Spinach
31: Strawberry
32: Tomato
33: Water
34: Yogurt

--- Mapping for 'Category' ---
0: Beverages
1: Dairy
2: Fruits
3: Grains
4: Meat
5: Snacks
6: Vegetables

--- Mapping for 'Meal_Type' ---
0: Breakfast
1: Dinner
2: Lunch
3: Snack
```

```
import pandas as pd

# Load the dataset
data = pd.read_csv('/content/daily_food_nutrition_dataset.csv')

# Display first few rows
print(data.head())

# Check column names
print(data.columns)
```

```
         Date  User_ID       Food_Item Category  Calories (kcal)  Protein (g)  \
0  2024-09-11      496            Eggs     Meat              173         42.4
1  2024-12-17      201           Apple   Fruits               66         39.2
2  2024-06-09      776  Chicken Breast     Meat              226         27.1
3  2024-08-27      112          Banana   Fruits              116         43.4
4  2024-07-28      622          Banana   Fruits              500         33.9

   Carbohydrates (g)  Fat (g)  Fiber (g)  Sugars (g)  Sodium (mg)  \
0               83.7      1.5        1.5        12.7          752
1               13.8      3.2        2.6        12.2          680
2               79.1     25.8        3.2        44.7          295
3               47.1     16.1        6.5        44.1          307
4               75.8     47.0        7.8        19.4          358

   Cholesterol (mg)  Meal_Type  Water_Intake (ml)
0               125      Lunch                478
1                97      Lunch                466
2               157  Breakfast                635
3                13      Snack                379
4               148      Lunch                471
Index(['Date', 'User_ID', 'Food_Item', 'Category', 'Calories (kcal)',
       'Protein (g)', 'Carbohydrates (g)', 'Fat (g)', 'Fiber (g)',
       'Sugars (g)', 'Sodium (mg)', 'Cholesterol (mg)', 'Meal_Type',
       'Water_Intake (ml)'],
      dtype='object')
```

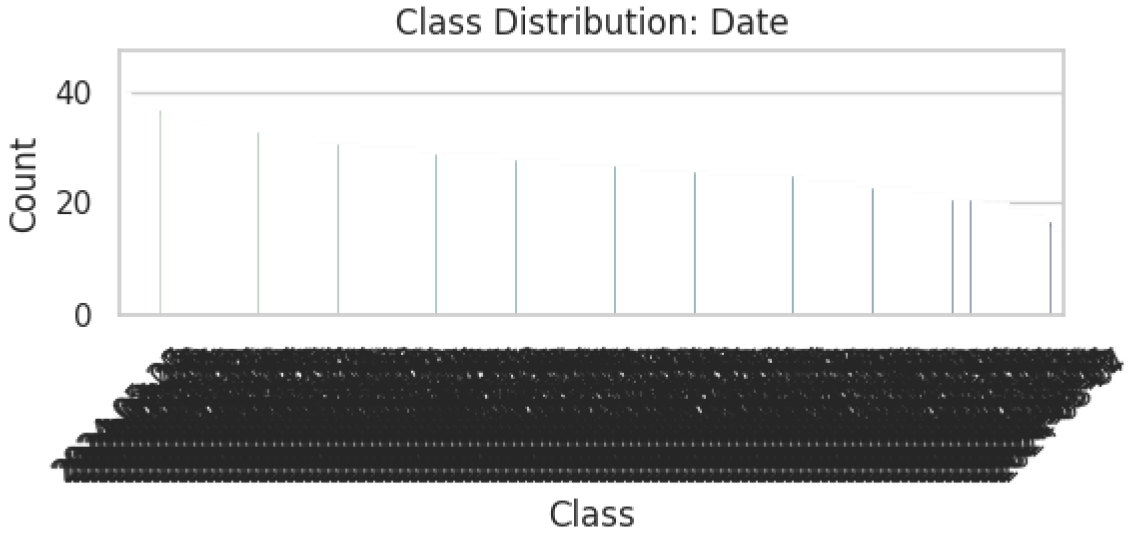
```
# Separate features (X) and labels (y)
X = data.drop('Meal_Type', axis=1)
y = data['Meal_Type']


print("Original class distribution:\n", y.value_counts())
```

```
Original class distribution:
 Meal_Type
Breakfast    2559
Dinner       2503
Lunch        2487
Snack        2451
Name: count, dtype: int64
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
food_data_path = '/content/daily_food_nutrition_dataset.csv'  # <- Replace with your actual path
df = pd.read_csv(food_data_path)

# Set visual style
sns.set(style="whitegrid")

print(" 🔹 Dataset Shape:", df.shape)
print(" 🔹 Columns:", df.columns.tolist())

# Threshold for imbalance detection
imbalance_threshold = 1.5

# Track imbalance results
imbalance_summary = []

# Loop through columns
for col in df.columns:
```

```
        unique_vals = df[col].nunique()

        # Only analyze potential categorical columns (low unique values or strings, excluding IDs)
        if df[col].dtype == 'object' or unique_vals <= 10 and col not in ['User_ID', 'Date']:
            counts = df[col].value_counts()
            top = counts.iloc[0]
            rest = counts.iloc[1:].sum()
            ratio = top / (rest + 1e-5)

            # Display class distribution
            print(f"\n📊 Column: {col}")
            print(counts)
            print(f"🔸 Imbalance Ratio = {top} / ({rest}) = {ratio:.2f} (Threshold: {imbalance_threshold})")

            # Bar plot
            plt.figure(figsize=(6, 3))
            sns.barplot(x=counts.index.astype(str), y=counts.values, palette='crest')
            plt.title(f'Class Distribution: {col}')
            plt.xlabel('Class')
            plt.ylabel('Count')
            plt.xticks(rotation=45)
            plt.tight_layout()
            plt.show()

            # Mark result
            if ratio > imbalance_threshold:
                imbalance_summary.append((col, '⚠️ Imbalanced', round(ratio, 2)))
            else:
                imbalance_summary.append((col, '✅ Balanced', round(ratio, 2)))

    # Summary
    print("\n🔍 Summary of Imbalance Check (Threshold: 1.5):")
    for col, status, ratio in imbalance_summary:
        print(f"{col}: {status} (Ratio: {ratio})")
```

```
🔹 Dataset Shape: (10000, 14)
🔹 Columns: ['Date', 'User_ID', 'Food_Item', 'Category', 'Calories (kcal)', 'Protein (g)', 'Carbohydrates (g)', 'Fat (g)', 'Fib

📊 Column: Date
Date
2024-05-20    45
2024-02-07    41
2024-08-25    40
2024-03-19    40
2024-06-22    39
              ..
2024-04-19    16
2024-02-01    15
2024-05-02    15
2024-03-22    13
2024-08-07    13
Name: count, Length: 366, dtype: int64
🔸 Imbalance Ratio = 45 / (9955) = 0.00 (Threshold: 1.5)
/tmp/ipython-input-75-22789147.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=counts.index.astype(str), y=counts.values, palette='crest')
```

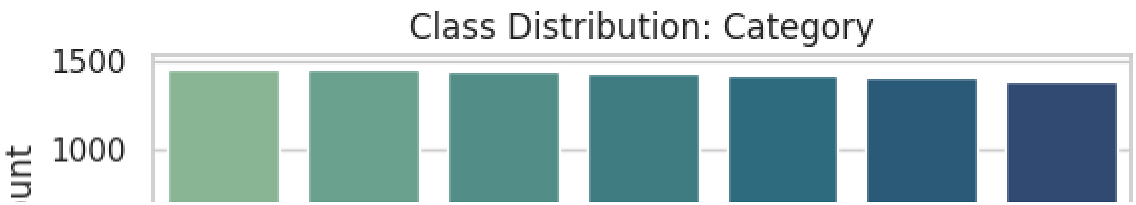


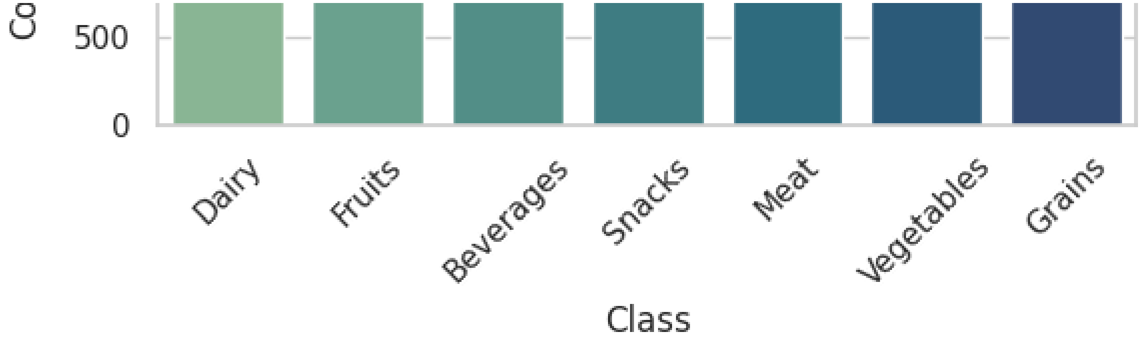

```
📊 Column: Food_Item
```

```
📊 Column: Food_Item
Food_Item
Milk              311
Orange            308
Pork Chop         307
Orange Juice      302
Carrot            301
Apple             299
Chocolate         299
Yogurt            298
Milkshake         297
Spinach           295
Quinoa            294
Coffee            292
Butter            292
Chicken Breast    290
Cookies           287
Cheese            286
Popcorn           285
Grapes            284
Banana            283
Rice              283
Chips             282
Beef Steak        282
Broccoli          280
Strawberry        279
Nuts              279
Green Tea         278
Potato            276
Water             276
Paneer            273
Oats              271
Salmon            270
Eggs              269
Pasta             269
Bread             267
Tomato            256
Name: count, dtype: int64
🔸 Imbalance Ratio = 311 / (9689) = 0.03 (Threshold: 1.5)
/tmp/ipython-input-75-22789147.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set
```

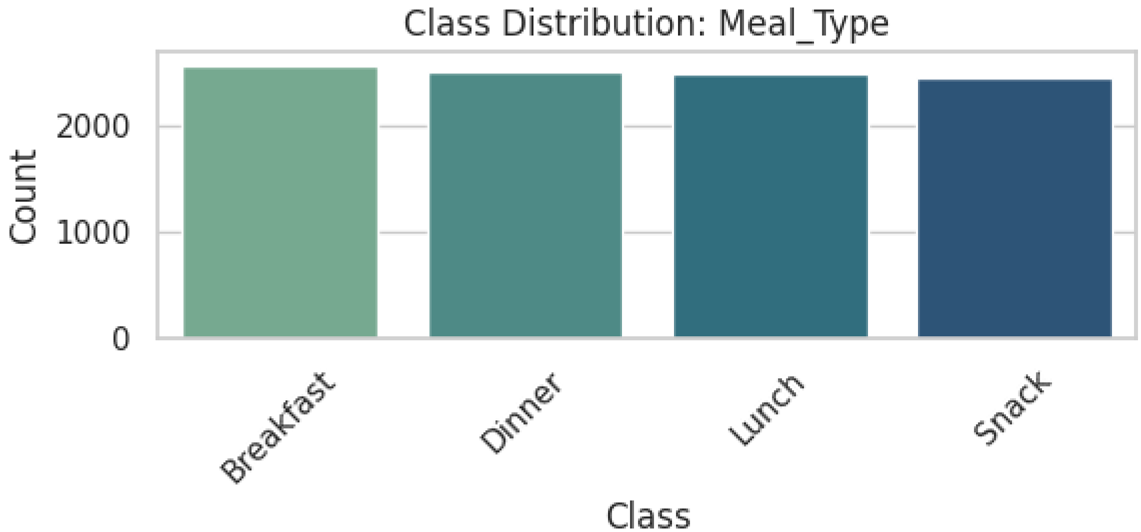
```
  sns.barplot(x=counts.index.astype(str), y=counts.values, palette='crest')
```


```
📊 Column: Category
Category
Dairy         1460
Fruits        1453
Beverages     1445
Snacks        1432
Meat          1418
Vegetables    1408
Grains        1384
Name: count, dtype: int64
🔸 Imbalance Ratio = 1460 / (8540) = 0.17 (Threshold: 1.5)
/tmp/ipython-input-75-22789147.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=counts.index.astype(str), y=counts.values, palette='crest')
```






```
📊 Column: Meal_Type
Meal_Type
Breakfast    2559
Dinner       2503
Lunch        2487
Snack        2451
Name: count, dtype: int64
🔸 Imbalance Ratio = 2559 / (7441) = 0.34 (Threshold: 1.5)
/tmp/ipython-input-75-22789147.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=counts.index.astype(str), y=counts.values, palette='crest')
```



```
🔍 Summary of Imbalance Check (Threshold: 1.5):
Date: ✅ Balanced (Ratio: 0.0)
Food_Item: ✅ Balanced (Ratio: 0.03)
Category: ✅ Balanced (Ratio: 0.17)
Meal_Type: ✅ Balanced (Ratio: 0.34)
```

```
# Load your dataset
import pandas as pd

df = pd.read_csv('/content/daily_food_nutrition_dataset.csv')

# Create artificial imbalance: Keep all 'Breakfast', drop most of other classes
imbalanced_df = pd.concat([
    df[df['Meal_Type'] == 'Breakfast'],
    df[df['Meal_Type'] != 'Breakfast'].sample(frac=0.2, random_state=42)  # Keep only 20% of other classes
])

# Check result
print(imbalanced_df['Meal_Type'].value_counts())
```

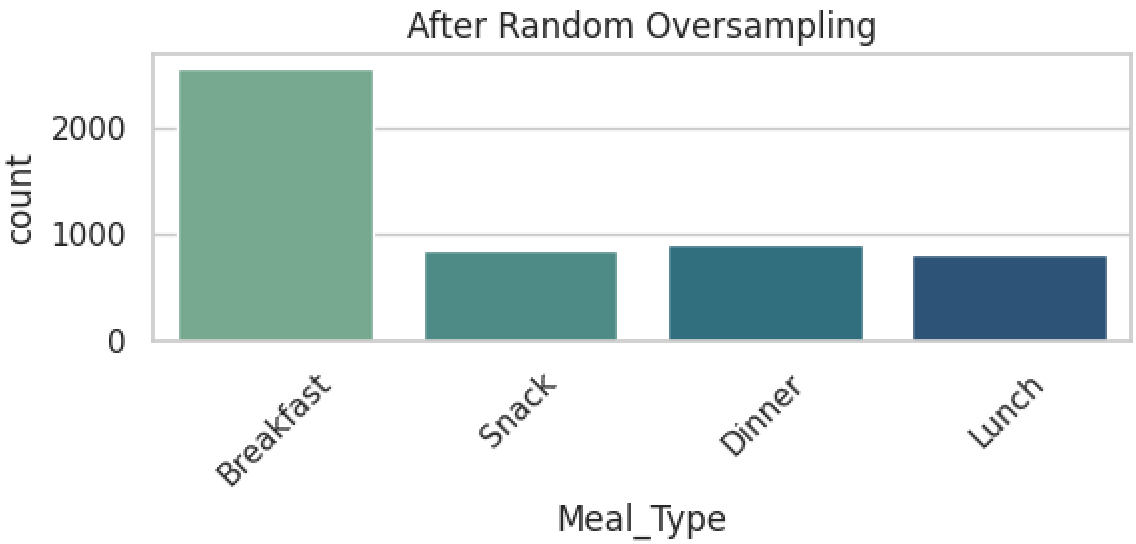
```
Meal_Type
Breakfast    2559
Dinner        512
Snack         493
Lunch         483
Name: count, dtype: int64
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Import resample function
from sklearn.utils import resample

# Assuming imbalanced_df is already created from the previous step
# Create artificial imbalance: Keep all 'Breakfast', drop most of other classes
# This block is copied from the previous cell to ensure imbalanced_df exists
df = pd.read_csv('/content/daily_food_nutrition_dataset.csv')

imbalanced_df = pd.concat([
    df[df['Meal_Type'] == 'Breakfast'],
    df[df['Meal_Type'] != 'Breakfast'].sample(frac=0.2, random_state=42)  # Keep only 20% of other classes
])
```

```
# Check result
print("Class distribution BEFORE oversampling:\n", imbalanced_df['Meal_Type'].value_counts())


# Separate majority and minority classes
df_major = imbalanced_df[imbalanced_df['Meal_Type'] == 'Breakfast'] # Assuming 'Breakfast' is the majority
# Identify the minority classes
minority_classes = imbalanced_df['Meal_Type'].value_counts().index.tolist()
minority_classes.remove('Breakfast')

# Concatenate all minority classes into a single DataFrame
df_minor = imbalanced_df[imbalanced_df['Meal_Type'].isin(minority_classes)]


# Oversample minority classes to match the number of majority class samples
# We oversample the entire df_minor block
df_minority_oversampled = resample(
    df_minor,
    replace=True,          # sample with replacement
    n_samples=len(df_major), # to match majority class size
    random_state=42        # reproducible results
)

# Combine majority class with oversampled minority class
balanced_oversampled = pd.concat([df_major, df_minority_oversampled])

# Display new class counts
print("\nClass distribution AFTER Random Oversampling:")
print(balanced_oversampled['Meal_Type'].value_counts())


# Plot distribution
plt.figure(figsize=(6, 3))
sns.countplot(x='Meal_Type', data=balanced_oversampled, palette='crest')
plt.title("After Random Oversampling")
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()

# New imbalance ratio
new_counts = balanced_oversampled['Meal_Type'].value_counts()
# Calculate ratio of the largest class to the sum of all other classes
top_count = new_counts.iloc[0]
rest_counts_sum = new_counts.iloc[1:].sum()
new_ratio = top_count / (rest_counts_sum + 1e-5) # Add small epsilon to avoid division by zero
print(f"✅ Oversampled Imbalance Ratio: {new_ratio:.2f}")
```

```
Class distribution BEFORE oversampling:
 Meal_Type
Breakfast    2559
Dinner        512
Snack         493
Lunch         483
Name: count, dtype: int64

Class distribution AFTER Random Oversampling:
Meal_Type
Breakfast    2559
Dinner        899
Snack         848
Lunch         812
Name: count, dtype: int64
/tmp/ipython-input-77-3907191551.py:50: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x='Meal_Type', data=balanced_oversampled, palette='crest')
```



✅ Oversampled Imbalance Ratio: 1.00

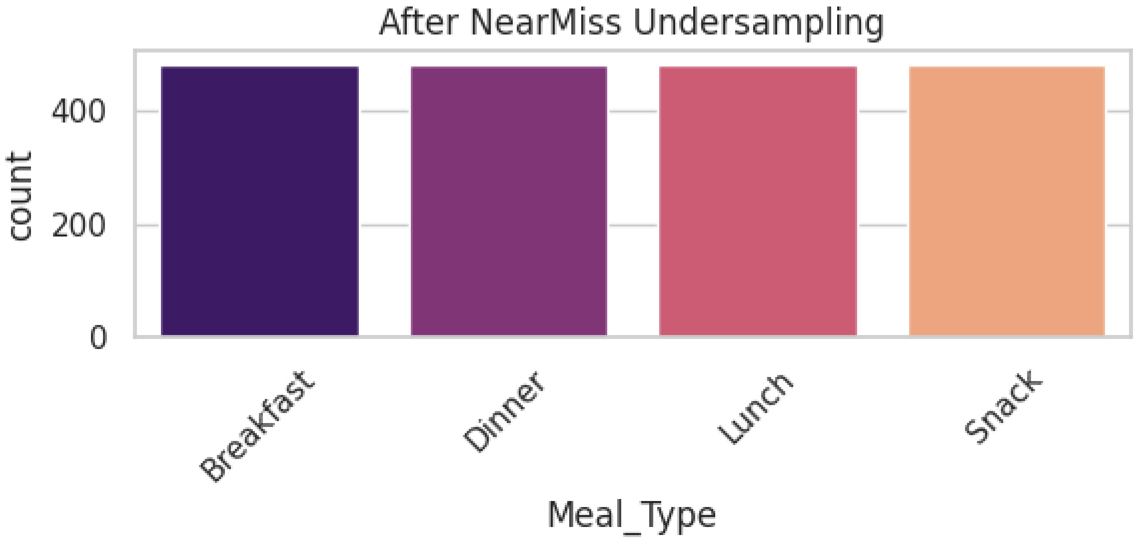
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Encode Meal_Type as target
df_encoded = imbalanced_df.dropna(subset=['Meal_Type'])  # Drop rows with missing target
X = df_encoded.drop(columns=['Meal_Type'])
X = pd.get_dummies(X.select_dtypes(include=['number']), drop_first=True)
y = LabelEncoder().fit_transform(df_encoded['Meal_Type'])

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

# Train model with class_weight='balanced'
model = LogisticRegression(class_weight='balanced', max_iter=1000)
model.fit(X_train, y_train)

print("✅ Logistic Regression trained with class weights to handle imbalance (no resampling).")
```

```
✅ Logistic Regression trained with class weights to handle imbalance (no resampling).
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (sta
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
from imblearn.under_sampling import NearMiss
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Define the target column name
target_col = 'Meal_Type'
```

```
# Prepare features and target
# Use the imbalanced_df created in previous steps
df_encoded = imbalanced_df.dropna(subset=[target_col])
X = pd.get_dummies(df_encoded.select_dtypes(include='number'), drop_first=True)
y = df_encoded[target_col]

# Apply NearMiss
nm = NearMiss(version=1)
X_res, y_res = nm.fit_resample(X, y)

# Convert to DataFrame for plotting
nm_df = pd.DataFrame(X_res, columns=X.columns) # Preserve column names
nm_df[target_col] = y_res

print("\n✅ Class counts AFTER NearMiss Undersampling:")
print(nm_df[target_col].value_counts())

plt.figure(figsize=(6, 3))
sns.countplot(x=target_col, data=nm_df, palette='magma')
plt.title("After NearMiss Undersampling")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
✅ Class counts AFTER NearMiss Undersampling:
Meal_Type
Breakfast    483
Dinner       483
Lunch        483
Snack        483
Name: count, dtype: int64
/tmp/ipython-input-80-3851979435.py:27: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x=target_col, data=nm_df, palette='magma')
```




```
import pandas as pd

# Load datasets
food_df = pd.read_csv('/content/daily_food_nutrition_dataset.csv')
gym_df = pd.read_csv('/content/gym_members_exercise_tracking.csv')

# Clean column names
food_df.columns = food_df.columns.str.strip()
```

```
gym_df.columns = gym_df.columns.str.strip()



# Reset index for both
food_df.reset_index(drop=True, inplace=True)
gym_df.reset_index(drop=True, inplace=True)

# Merge side-by-side
merged_df = pd.concat([gym_df, food_df], axis=1)

print("✅ Merged dataset shape:", merged_df.shape)
merged_df.head()
```

```
✅ Merged dataset shape: (10000, 29)
```

	Age	Gender	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Workout_Type	...	Calories (kcal)	P
0	56.0	Male	88.3	1.71	180.0	157.0	60.0	1.69	1313.0	Yoga	...	173	
1	46.0	Female	74.9	1.53	179.0	151.0	66.0	1.30	883.0	HIIT	...	66	
2	32.0	Female	68.1	1.66	167.0	122.0	54.0	1.11	677.0	Cardio	...	226	
3	25.0	Male	53.2	1.70	190.0	164.0	56.0	0.59	532.0	Strength	...	116	
4	38.0	Male	46.1	1.79	188.0	158.0	68.0	0.64	556.0	Strength	...	500	

5 rows × 29 columns

```
# Binary Target for 'Went to Gym' — assume if 'Session_Duration' > 0
merged_df['Went_To_Gym'] = merged_df['Session_Duration (hours)'].apply(lambda x: 1 if x > 0 else 0)

# Optional: Clean nulls
merged_df.dropna(subset=['Workout_Type', 'Calories_Burned', 'Session_Duration (hours)'], inplace=True)
```

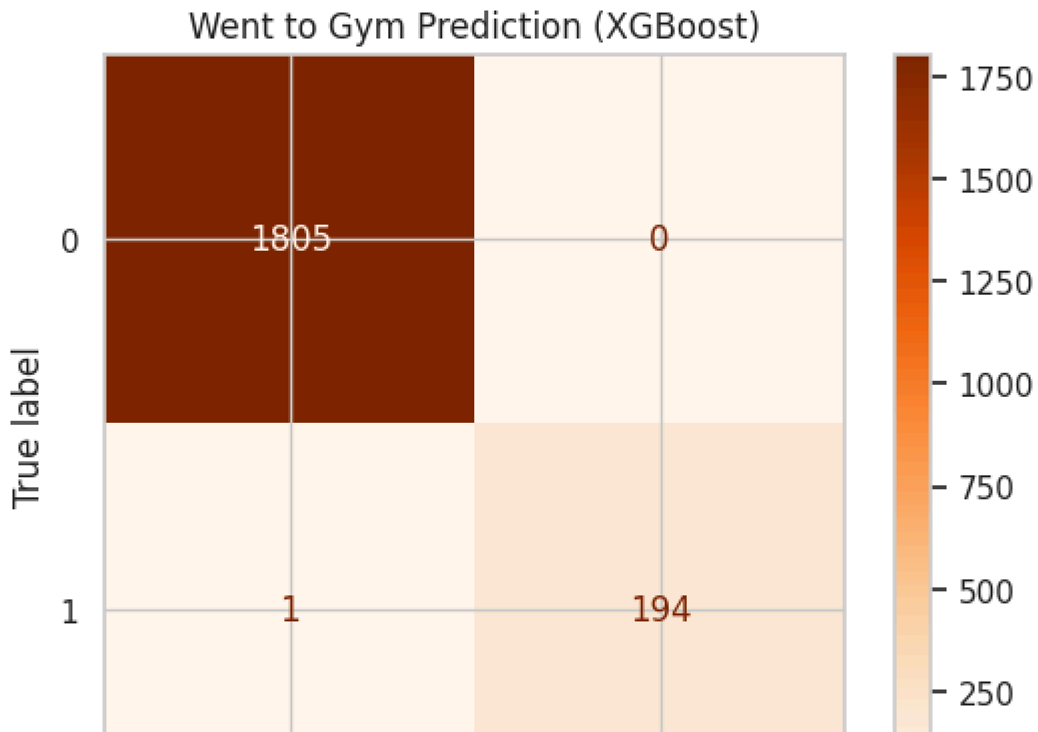
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, ConfusionMatrixDisplay
import seaborn as sns
from xgboost import XGBClassifier

# Load datasets
food_df = pd.read_csv('/content/daily_food_nutrition_dataset.csv')
gym_df = pd.read_csv('/content/gym_members_exercise_tracking.csv')

# Clean column names
food_df.columns = food_df.columns.str.strip()
gym_df.columns = gym_df.columns.str.strip()

# Reset index for both
food_df.reset_index(drop=True, inplace=True)
gym_df.reset_index(drop=True, inplace=True)

# Merge side-by-side
merged_df = pd.concat([gym_df, food_df], axis=1)

print("✅ Merged dataset shape:", merged_df.shape)
merged_df.head()

# --- Added Diagnostic Step ---
print("\n🔹 Value counts for 'Session_Duration (hours)' AFTER merge:")
print(merged_df['Session_Duration (hours)'].value_counts(dropna=False))

print("\n🔹 Number of NaN values in 'Session_Duration (hours)' AFTER merge:")
print(merged_df['Session_Duration (hours)'].isnull().sum())

# Calculate 'Went_To_Gym' after checking the base data
merged_df['Went_To_Gym'] = merged_df['Session_Duration (hours)'].apply(lambda x: 1 if pd.notna(x) and x > 0

# --- Added Diagnostic Step ---
print("\n🔹 Value counts for 'Went_To_Gym' AFTER calculation:")
```

```
    print(merged_df['Went_To_Gym'].value_counts(dropna=False))
    # --- End of Diagnostic Step ---

    # Now, proceed only if Went_To_Gym has more than one class
    if merged_df['Went_To_Gym'].nunique() < 2:
        print("\nError: The target variable 'Went_To_Gym' still contains only one unique class after calculatio
        print("Please inspect the 'Session_Duration (hours)' column in the merged data.")
        print("If the original gym data contained 0s or NaNs for non-gym days, the merge may have lost them.")
    else:
        # Impute NaN values in the feature columns with 0
        X = merged_df[['Calories (kcal)', 'Fat (g)', 'Water_Intake (ml)', 'BMI']].fillna(0)
        y = merged_df['Went_To_Gym']

        # --- Added Diagnostic Step ---
        print("\nMissing values in features after imputation:")
        print(X.isnull().sum())
        # --- End of Diagnostic Step ---

        # Check class distribution before splitting
        print("\nClass counts in target variable BEFORE splitting (after calculation):")
        print(y.value_counts())

        # Split the data
        X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

        # --- Added Diagnostic Step ---
        print("\nClass counts in y_train before training:")
        print(y_train.value_counts())
        # --- End of Diagnostic Step ---

        # Apply XGBoost Classifier
        xgb_clf = XGBClassifier(
            random_state=42,
            use_label_encoder=False,
            eval_metric='logloss'  # Suitable for binary classification
        )

        # Fit the model (no GridSearchCV for simplicity, but can be added)
```

```
    xgb_clf.fit(X_train, y_train)
    y_pred = xgb_clf.predict(X_test)

    # Evaluate the model
    print("\n🔹 Gym Day Prediction Accuracy (XGBoost):", accuracy_score(y_test, y_pred))
    print("\n🔹 XGBoost Classification Report:\n", classification_report(y_test, y_pred))

    # Display Confusion Matrix
    ConfusionMatrixDisplay.from_estimator(xgb_clf, X_test, y_test, cmap="Oranges")
    plt.title("Went to Gym Prediction (XGBoost)")
    plt.show()

    # --- Feature Importance (Optional but useful) ---
    feature_importance = pd.DataFrame({
        'Feature': X.columns,
        'Importance': xgb_clf.feature_importances_
    }).sort_values(by='Importance', ascending=False)
    print("\n🔹 Feature Importance (XGBoost):\n", feature_importance)
```

```
✅ Merged dataset shape: (10000, 29)

🔹 Value counts for 'Session_Duration (hours)' AFTER merge:
Session_Duration (hours)
NaN     9027
1.13      20
1.03      20
1.37      20
1.08      19
        ...
0.78       1
1.53       1
0.50       1
1.68       1
2.00       1
Name: count, Length: 148, dtype: int64

🔹 Number of NaN values in 'Session_Duration (hours)' AFTER merge:
9027

🔹 Value counts for 'Went_To_Gym' AFTER calculation:
Went_To_Gym
0    9027
1     973
Name: count, dtype: int64

Missing values in features after imputation:
Calories (kcal)     0
Fat (g)             0
Water_Intake (ml)   0
BMI                 0
dtype: int64

Class counts in target variable BEFORE splitting (after calculation):
Went_To_Gym
0    9027
1     973
Name: count, dtype: int64

Class counts in y_train before training:
Went_To_Gym
```

```
0    7222
1     778
Name: count, dtype: int64

🔹 Gym Day Prediction Accuracy (XGBoost): 0.9995

🔹 XGBoost Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1805
           1       1.00      0.99      1.00       195

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000

/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [08:45:54] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```




```
🔹 Feature Importance (XGBoost):
             Feature  Importance
3                BMI    0.996984
2  Water_Intake (ml)    0.001579
0    Calories (kcal)    0.000721
1            Fat (g)    0.000717
```

Start coding or generate with AI.

```
merged_df['Went_To_Gym'] = merged_df['Session_Duration (hours)'].apply(lambda x: 1 if x > 0 else 0)



# Check distribution
print("Went_To_Gym class counts:\n", merged_df['Went_To_Gym'].value_counts())
```

```
Went_To_Gym class counts:
 Went_To_Gym
1    973
Name: count, dtype: int64
```

```
print("Class counts in Went_To_Gym:\n", merged_df['Went_To_Gym'].value_counts())
```

```
Class counts in Went_To_Gym:
 Went_To_Gym
1    973
Name: count, dtype: int64
```

```
merged_df.dropna(subset=['Session_Duration (hours)', 'Calories (kcal)', 'BMI'], inplace=True)
```

```
import pandas as pd

# Load datasets (user will upload these manually in Colab)
food_df = pd.read_csv('/content/daily_food_nutrition_dataset.csv')
gym_df = pd.read_csv('/content/gym_members_exercise_tracking.csv')

# Print column names for verification
print("✅ Nutrition dataset columns:", food_df.columns.tolist())
print("✅ Gym dataset columns:", gym_df.columns.tolist())
```

```
# Attempt to find a common key to merge on
common_keys = set(food_df.columns).intersection(set(gym_df.columns))
print("🔍 Common columns for potential merging:", common_keys)

# If no good common key, add an artificial ID to merge on row index
food_df['Merge_ID'] = food_df.index
gym_df['Merge_ID'] = gym_df.index

# Merge the datasets on the artificial ID
merged_df = pd.merge(food_df, gym_df, on='Merge_ID', how='inner')
print("✅ Merged dataset shape:", merged_df.shape)

# Save merged dataset to CSV for next steps
merged_df.to_csv('/content/merged_dataset.csv', index=False)
print("📁 Saved merged dataset to '/content/merged_dataset.csv'")
```

```
✅ Nutrition dataset columns: ['Date', 'User_ID', 'Food_Item', 'Category', 'Calories (kcal)', 'Protein (g)', 'Carbohydrates (g)
✅ Gym dataset columns: ['Age', 'Gender', 'Weight (kg)', 'Height (m)', 'Max_BPM', 'Avg_BPM', 'Resting_BPM', 'Session_Duration (
🔍 Common columns for potential merging: set()
✅ Merged dataset shape: (973, 30)
📁 Saved merged dataset to '/content/merged_dataset.csv'
```

Start coding or generate with AI.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report
```

```
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectFromModel
from sklearn.impute import SimpleImputer

# Load dataset
df = pd.read_csv('/content/merged_dataset.csv')
print("🧾 Column names:\n", df.columns.tolist())

# Analyze distributions
print("\nCarbohydrates (g) distribution:\n", df['Carbohydrates (g)'].describe())
print("\nFat (g) distribution:\n", df['Fat (g)'].describe())

# Add Gaussian noise
np.random.seed(42)
df['Carbohydrates (g)'] += np.random.normal(0, 50, df.shape[0])
df['Fat (g)'] += np.random.normal(0, 30, df.shape[0])

# Define Smart_Diet with quantile-based thresholds
carb_low = df['Carbohydrates (g)'].quantile(0.33)
fat_high = df['Fat (g)'].quantile(0.66)
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Carb' if row['Carbohydrates (g)'] < carb_low else
                'High Fat' if row['Fat (g)'] > fat_high else
                'Balanced', axis=1)

# Map to Recommended_Meal_Plan
df['Recommended_Meal_Plan'] = df['Smart_Diet'].map({
    'Low Carb': 'Keto',
    'High Fat': 'Paleo',
    'Balanced': 'Mediterranean'
})
df = df.drop(columns=['Smart_Diet'])

# Verify class distribution
print("\nRecommended_Meal_Plan distribution:\n", df['Recommended_Meal_Plan'].value_counts())

# Drop unneeded columns
columns_to_drop = [col for col in ['Patient_ID', 'BMI'] if col in df.columns]
```

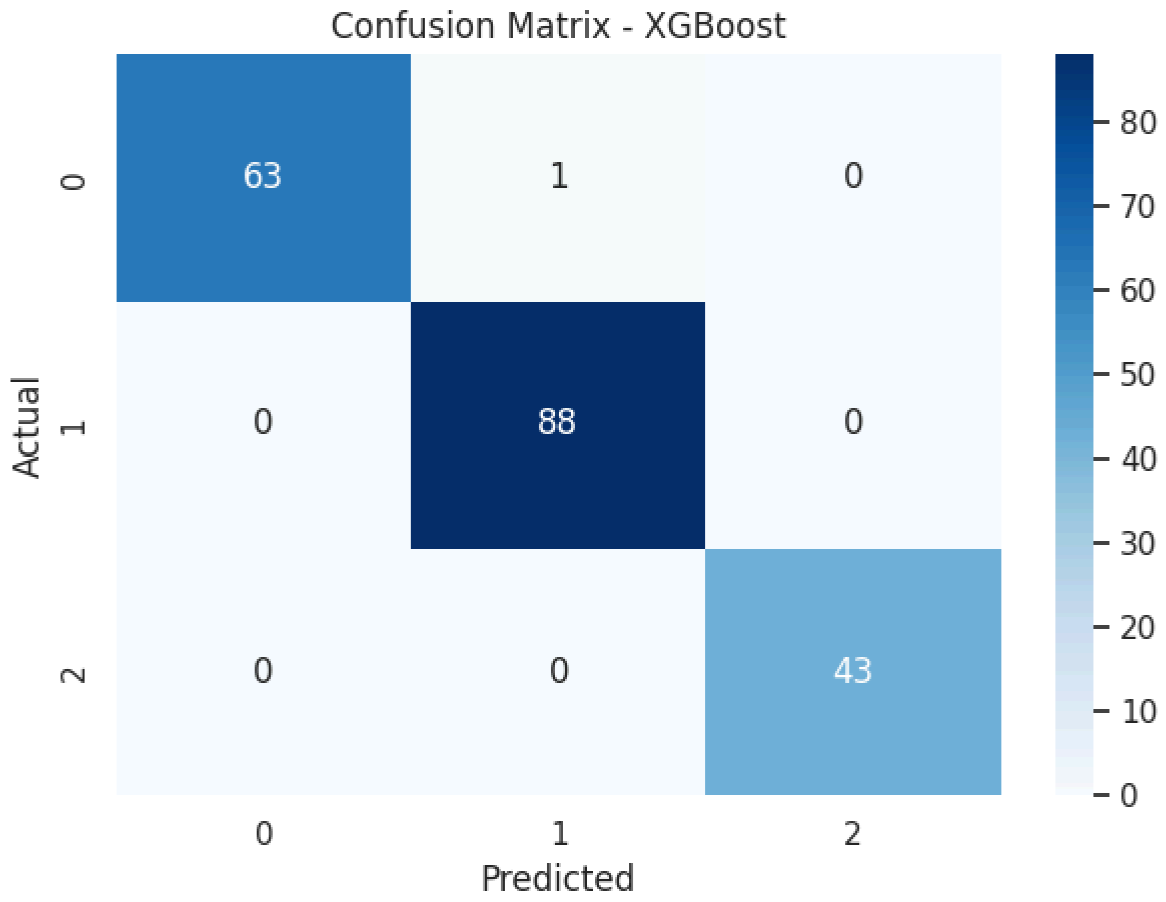
```
    df = df.drop(columns=columns_to_drop)

    # Handle missing values
    imputer = SimpleImputer(strategy='constant', fill_value='Unknown')
    for col in ['Chronic_Disease', 'Allergies', 'Food_Aversions']:
        if col in df.columns and df[col].dtype == 'object':
            df[col] = imputer.fit_transform(df[[col]]).ravel()

    # Label encode categorical features
    cat_cols = df.select_dtypes(include='object').columns.difference(['Recommended_Meal_Plan'])
    encoders = {}
    for col in cat_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        encoders[col] = le

    # Encode target
    le_rmp = LabelEncoder()
    df['Recommended_Meal_Plan'] = le_rmp.fit_transform(df['Recommended_Meal_Plan'])
    print("\nClass mappings:", dict(zip(range(len(le_rmp.classes_)), le_rmp.classes_)))

    # Define features and target
    X = df.drop(columns=['Recommended_Meal_Plan'])
    y = df['Recommended_Meal_Plan']

    # Split dataset
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
    print("\nTraining set class distribution:", pd.Series(y_train).value_counts().to_dict())

    # Balance dataset using SMOTE
    smote = SMOTE(sampling_strategy='auto', random_state=42, k_neighbors=5)
    X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

    # Feature scaling
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train_bal)
    X_test_scaled = scaler.transform(X_test)
```

```
    # Feature selection
    rf_for_selection = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
    rf_for_selection.fit(X_train_scaled, y_train_bal)
    selector = SelectFromModel(rf_for_selection, prefit=True, threshold="0.5*mean")
    X_train_selected = selector.transform(X_train_scaled)
    X_test_selected = selector.transform(X_test_scaled)
    selected_features = X.columns[selector.get_support()].tolist()
    print("\nSelected features:", selected_features)

    # Define models
    models = {
        "XGBoost": XGBClassifier(max_depth=3, reg_lambda=12.0, reg_alpha=10.0,
                                 subsample=0.7, colsample_bytree=0.7,
                                 eval_metric='mlogloss', random_state=42),
        "Logistic Regression": GridSearchCV(
            LogisticRegression(max_iter=2000, random_state=42),
            param_grid={'C': [0.01, 0.1, 0.5, 1.0, 2.0], 'solver': ['lbfgs', 'saga'],
                        'class_weight': ['balanced']},
            cv=5, scoring='accuracy', n_jobs=-1),
        "SVM": GridSearchCV(
            SVC(probability=True),
            param_grid={'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']},
            cv=5, scoring='accuracy', n_jobs=-1)
    }

    # Train and evaluate models
    for name, model in models.items():
        print(f"\nTraining {name}...")
        model.fit(X_train_selected, y_train_bal)
        y_pred = model.predict(X_test_selected)

        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')
        print(f"Accuracy: {acc*100:.2f}%")
        print(f"F1 Score: {f1:.4f}")
        print("Classification Report:\n", classification_report(y_test, y_pred, target_names=le_rmp.classes_))

        cv_scores = cross_val_score(model, X_train_selected, y_train_bal, cv=5, scoring='accuracy')
```

```
    cv_scores = cross_val_score(model, X_train_selected, y_train_bal, cv=5, scoring='accuracy')
    print(f"Cross-Validation Accuracy: {cv_scores.mean()*100:.2f}% ± {cv_scores.std()*100:.2f}%")

    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.tight_layout()
    plt.show()
    print("-" * 80)
```

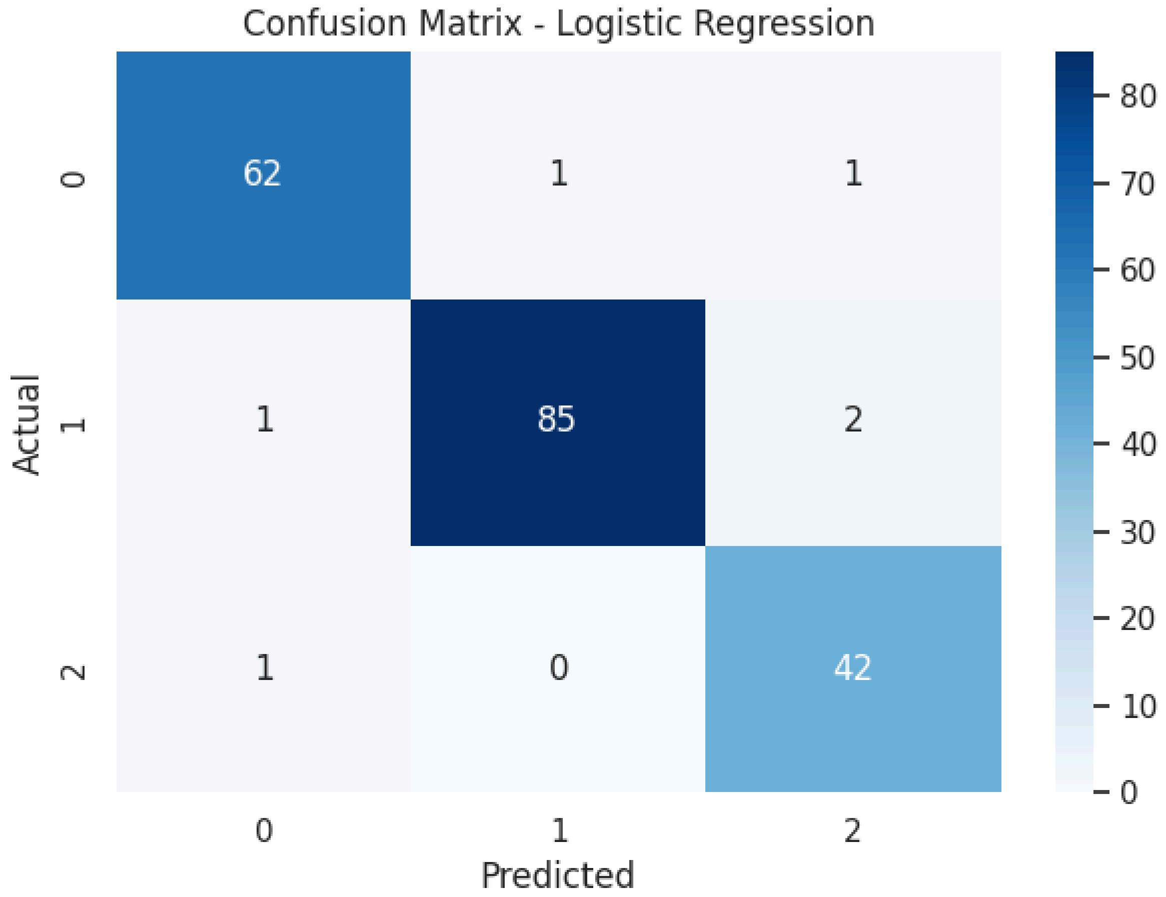
```
📃 Column names:
 ['Date', 'User_ID', 'Food_Item', 'Category', 'Calories (kcal)', 'Protein (g)', 'Carbohydrates (g)', 'Fat (g)', 'Fiber (g)', 'Su

Carbohydrates (g) distribution:
 count    973.000000
mean      50.909661
std       27.303617
min        5.100000
25%       26.200000
50%       51.300000
75%       75.200000
max      100.000000
Name: Carbohydrates (g), dtype: float64

Fat (g) distribution:
 count    973.000000
mean      25.271326
std       13.904356
min        1.000000
25%       13.400000
50%       25.800000
75%       37.000000
max       50.000000
Name: Fat (g), dtype: float64

Recommended_Meal_Plan distribution:
 Recommended_Meal_Plan
Mediterranean    436
Keto             321
Paleo            216
Name: count, dtype: int64

Class mappings: {0: 'Keto', 1: 'Mediterranean', 2: 'Paleo'}

Training set class distribution: {1: 348, 0: 257, 2: 173}

Selected features: ['Carbohydrates (g)', 'Fat (g)']

Training XGBoost...
Accuracy: 99.49%
F1 Score: 0.9949
```

```
Classification Report:
               precision    recall  f1-score   support

        Keto       1.00      0.98      0.99        64
Mediterranean      0.99      1.00      0.99        88
       Paleo       1.00      1.00      1.00        43

    accuracy                           0.99       195
   macro avg       1.00      0.99      1.00       195
weighted avg       0.99      0.99      0.99       195

Cross-Validation Accuracy: 99.71% ± 0.38%
```

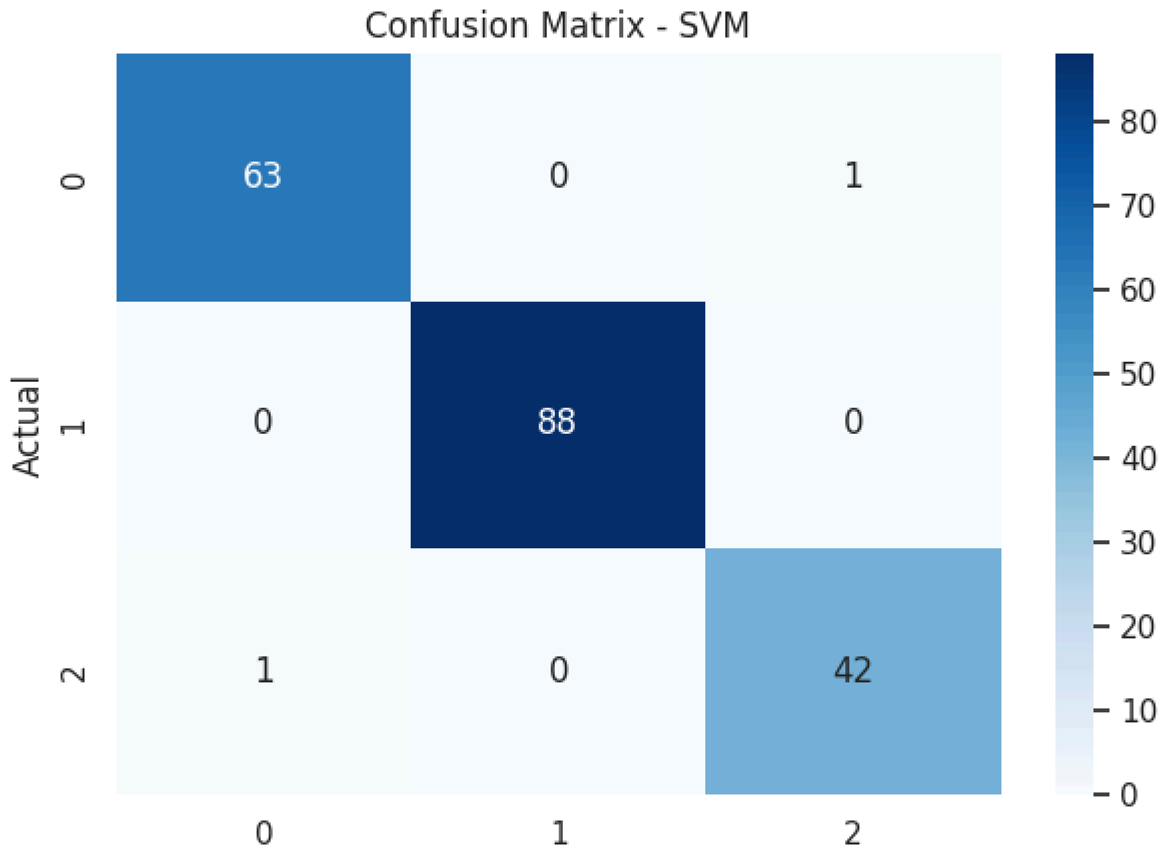



```
--------------------------------------------------------------------------------

Training Logistic Regression...
```

```
Training Logistic Regression...
Accuracy: 96.92%
F1 Score: 0.9693
Classification Report:
                precision    recall  f1-score   support

         Keto       0.97      0.97      0.97        64
Mediterranean       0.99      0.97      0.98        88
        Paleo       0.93      0.98      0.95        43

     accuracy                           0.97       195
    macro avg       0.96      0.97      0.97       195
 weighted avg       0.97      0.97      0.97       195

Cross-Validation Accuracy: 97.41% ± 0.65%
```



```
--------------------------------------------------------------------------------

Training SVM...
Accuracy: 98.97%
F1 Score: 0.9897
Classification Report:
                precision    recall  f1-score   support

        Keto       0.98      0.98      0.98        64
Mediterranean       1.00      1.00      1.00        88
       Paleo       0.98      0.98      0.98        43

    accuracy                           0.99       195
   macro avg       0.99      0.99      0.99       195
weighted avg       0.99      0.99      0.99       195

Cross-Validation Accuracy: 99.04% ± 0.80%
```



Predicted

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, ConfusionMatrixDisplay
import seaborn as sns

# Load datasets
food_df = pd.read_csv('/content/daily_food_nutrition_dataset.csv')
gym_df = pd.read_csv('/content/gym_members_exercise_tracking.csv')

# Clean column names (Good step, keep this)
food_df.columns = food_df.columns.str.strip()
gym_df.columns = gym_df.columns.str.strip()

# %%
# Reset index for both (Necessary for concat(axis=1) to work row-wise)
food_df.reset_index(drop=True, inplace=True)
gym_df.reset_index(drop=True, inplace=True)

# Merge side-by-side (This is the point to check)
merged_df = pd.concat([gym_df, food_df], axis=1)

print("✅ Merged dataset shape:", merged_df.shape)
merged_df.head()

# --- Added Diagnostic Step ---
print("\n 🔹 Value counts for 'Session_Duration (hours)' AFTER merge:")
print(merged_df['Session_Duration (hours)'].value_counts(dropna=False))

print("\n 🔹 Number of NaN values in 'Session_Duration (hours)' AFTER merge:")
print(merged_df['Session_Duration (hours)'].isnull().sum())

# Calculate 'Went_To_Gym' *after* checking the base data
merged_df['Went_To_Gym'] = merged_df['Session_Duration (hours)'].apply(lambda x: 1 if pd.notna(x) and x > 0 else 0) # Added pd.notna

# --- Added Diagnostic Step ---
```

```
    print("\n 🔹 Value counts for 'Went_To_Gym' AFTER calculation:")
    print(merged_df['Went_To_Gym'].value_counts(dropna=False))
    # --- End of Diagnostic Step ---

    # Now, proceed *only if* Went_To_Gym has more than one class
    if merged_df['Went_To_Gym'].nunique() < 2:
        print("\nError: The target variable 'Went_To_Gym' still contains only one unique class after calculation.")
        print("Please inspect the 'Session_Duration (hours)' column in the merged data.")
        print("If the original gym data contained 0s or NaNs for non-gym days, the merge may have lost them.")
    else:
        # Impute NaN values in the feature columns with 0 as requested
        X = merged_df[['Calories (kcal)', 'Fat (g)', 'Water_Intake (ml)', 'BMI']].fillna(0)
        y = merged_df['Went_To_Gym']

        # --- Added Diagnostic Step (optional, but good practice) ---
        print("\nMissing values in features after imputation:")
        print(X.isnull().sum())
        # --- End of Diagnostic Step ---

        # Check class distribution before splitting (should match the print above)
        print("\nClass counts in target variable BEFORE splitting (after calculation):")
        print(y.value_counts())


        X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

        # --- Added Diagnostic Step ---
        print("\nClass counts in y_train before training:")
        print(y_train.value_counts())
        # --- End of Diagnostic Step ---

        logreg = LogisticRegression(max_iter=1000)
        logreg.fit(X_train, y_train)
        y_pred = logreg.predict(X_test)

        print("\n 🔹 Gym Day Prediction Accuracy:", accuracy_score(y_test, y_pred))
        print(classification_report(y_test, y_pred))
```