Machine Learing Semester Project

NAME: SAYAB ABBASI

ID: 37182

Article NAME: Patient Diet Recommendation System through Machine Learning Model

DATASET NAME: Personalized Medical Diet Recommendations Dataset

In [600]:

```
import pandas as pd
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
from sklearn.multioutput import MultiOutputClassifier
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
```

EDA

In [601]:

```
df = pd.read_csv("/content/Personalized_Diet_Recommendations.csv")
```

In [602]:

```
df
```

Out[602]:

	Patient_ID	Age	Gender	Height_cm	Weight_kg	BMI	Chronic_Disease	Blood_Pressure_Systolic	Blood_Pressure_Diast
0	P00001	56	Other	163	66	24.84	NaN	175	
1	P00002	69	Female	171	114	38.99	NaN	155	
2	P00003	46	Female	172	119	40.22	NaN	137	
3	P00004	32	Female	197	118	30.41	NaN	148	
4	P00005	60	Female	156	109	44.79	Hypertension	160	
...	...	...	...	...	...	...	...	...	
4995	P04996	42	Female	172	99	33.46	NaN	115	
4996	P04997	39	Female	155	61	25.39	NaN	110	
4997	P04998	48	Female	165	61	22.41	Diabetes	113	
4998	P04999	34	Other	151	82	35.96	Heart Disease	105	
4999	P05000	72	Other	173	98	32.74	NaN	121	

5000 rows × 30 columns

In [603]:

```
df.head()
```

Out[603]:

	Patient_ID	Age	Gender	Height_cm	Weight_kg	BMI	Chronic_Disease	Blood_Pressure_Systolic	Blood_Pressure_Diastolic
0	P00001	56	Other	163	66	24.84	NaN	175	75
1	P00002	69	Female	171	114	38.99	NaN	155	72
2	P00003	46	Female	172	119	40.22	NaN	137	101
3	P00004	32	Female	197	118	30.41	NaN	148	91
4	P00005	60	Female	156	109	44.79	Hypertension	160	109

5 rows × 30 columns

In [604]:

```
df.tail()
```

Out[604]:

	Patient_ID	Age	Gender	Height_cm	Weight_kg	BMI	Chronic_Disease	Blood_Pressure_Systolic	Blood_Pressure_Diast
4995	P04996	42	Female	172	99	33.46	NaN	115	
4996	P04997	39	Female	155	61	25.39	NaN	110	
4997	P04998	48	Female	165	61	22.41	Diabetes	113	
4998	P04999	34	Other	151	82	35.96	Heart Disease	105	
4999	P05000	72	Other	173	98	32.74	NaN	121	

5 rows × 30 columns

In [605]:

```
df.describe()
```

Out[605]:

	Age	Height_cm	Weight_kg	BMI	Blood_Pressure_Systolic	Blood_Pressure_Diastolic	Cholesterol_Lev
count	5000.000000	5000.000000	5000.00000	5000.000000	5000.000000	5000.000000	5000.0000
mean	48.805600	174.244000	84.36620	28.353134	133.982400	89.735800	224.2978
std	17.906991	14.229173	20.18103	8.297745	26.216215	17.283025	42.9189
min	18.000000	150.000000	50.00000	12.630000	90.000000	60.000000	150.0000
25%	34.000000	162.000000	67.00000	21.850000	111.000000	75.000000	187.0000
50%	49.000000	174.000000	84.00000	27.640000	133.000000	90.000000	224.0000
75%	64.000000	186.000000	102.00000	33.812500	157.000000	105.000000	261.0000
max	79.000000	199.000000	119.00000	52.890000	179.000000	119.000000	299.0000

In [606]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 30 columns):
 #   Column                     Non-Null Count  Dtype
```

```
---  ------                     --------------  -----
 0   Patient_ID                 5000 non-null   object
 1   Age                        5000 non-null   int64
 2   Gender                     5000 non-null   object
 3   Height_cm                  5000 non-null   int64
 4   Weight_kg                  5000 non-null   int64
 5   BMI                        5000 non-null   float64
 6   Chronic_Disease            2957 non-null   object
 7   Blood_Pressure_Systolic    5000 non-null   int64
 8   Blood_Pressure_Diastolic   5000 non-null   int64
 9   Cholesterol_Level          5000 non-null   int64
 10  Blood_Sugar_Level          5000 non-null   int64
 11  Genetic_Risk_Factor        5000 non-null   object
 12  Allergies                  1503 non-null   object
 13  Daily_Steps                5000 non-null   int64
 14  Exercise_Frequency         5000 non-null   int64
 15  Sleep_Hours                5000 non-null   float64
 16  Alcohol_Consumption        5000 non-null   object
 17  Smoking_Habit              5000 non-null   object
 18  Dietary_Habits             5000 non-null   object
 19  Caloric_Intake             5000 non-null   int64
 20  Protein_Intake             5000 non-null   int64
 21  Carbohydrate_Intake        5000 non-null   int64
 22  Fat_Intake                 5000 non-null   int64
 23  Preferred_Cuisine          5000 non-null   object
 24  Food_Aversions             3775 non-null   object
 25  Recommended_Calories       5000 non-null   int64
 26  Recommended_Protein        5000 non-null   int64
 27  Recommended_Carbs          5000 non-null   int64
 28  Recommended_Fats           5000 non-null   int64
 29  Recommended_Meal_Plan      5000 non-null   object
dtypes: float64(2), int64(17), object(11)
memory usage: 1.1+ MB
```

In [607]:

```
df.isnull().sum()
```

Out[607]:

	0
Patient_ID	0
Age	0
Gender	0
Height_cm	0
Weight_kg	0
BMI	0
Chronic_Disease	2043
Blood_Pressure_Systolic	0
Blood_Pressure_Diastolic	0
Cholesterol_Level	0
Blood_Sugar_Level	0
Genetic_Risk_Factor	0
Allergies	3497
Daily_Steps	0
Exercise_Frequency	0
Sleep_Hours	0
Alcohol_Consumption	0
Smoking_Habit	0
Dietary_Habits	0

	0
Caloric_Intake	0
Protein_Intake	0
Carbohydrate_Intake	0
Fat_Intake	0
Preferred_Cuisine	0
Food_Aversions	1225
Recommended_Calories	0
Recommended_Protein	0
Recommended_Carbs	0
Recommended_Fats	0
Recommended_Meal_Plan	0

dtype: int64

Data Set Summry

Type	Examples	Description
Demographics	`Age`, `Gender`, `Height_cm`, `Weight_kg`	**Basic patient info**
Health Indicators	`BMI`, `Blood_Pressure`, `Cholesterol_Level`, `Blood_Sugar_Level`	**Key medical metrics**
Medical History	`Chronic_Disease`, `Genetic_Risk_Factor`, `Allergies`	**Existing conditions and risks**
Lifestyle	`Daily_Steps`, `Exercise_Frequency`, `Sleep_Hours`, `Smoking_Habit`, `Alcohol_Consumption`	**Behavior patterns**
Dietary Intake	`Caloric_Intake`, `Protein_Intake`, `Carbohydrate_Intake`, `Fat_Intake`, `Preferred_Cuisine`, `Food_Aversions`	**Actual diet consumed**
Recommendations	`Recommended_Calories`, `Recommended_Protein`, `Recommended_Carbs`, `Recommended_Fats`, `Recommended_Meal_Plan`	**Personalized nutritional advice**

In [608]:

```
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])
```

```
Chronic_Disease    2043
Allergies          3497
Food_Aversions     1225
dtype: int64
```

In [609]:

```
desease_count = df['Chronic_Disease'].value_counts()
print(desease_count)
```

```
Chronic_Disease
Diabetes         1019
Heart Disease     749
Hypertension      693
Obesity           496
Name: count, dtype: int64
```

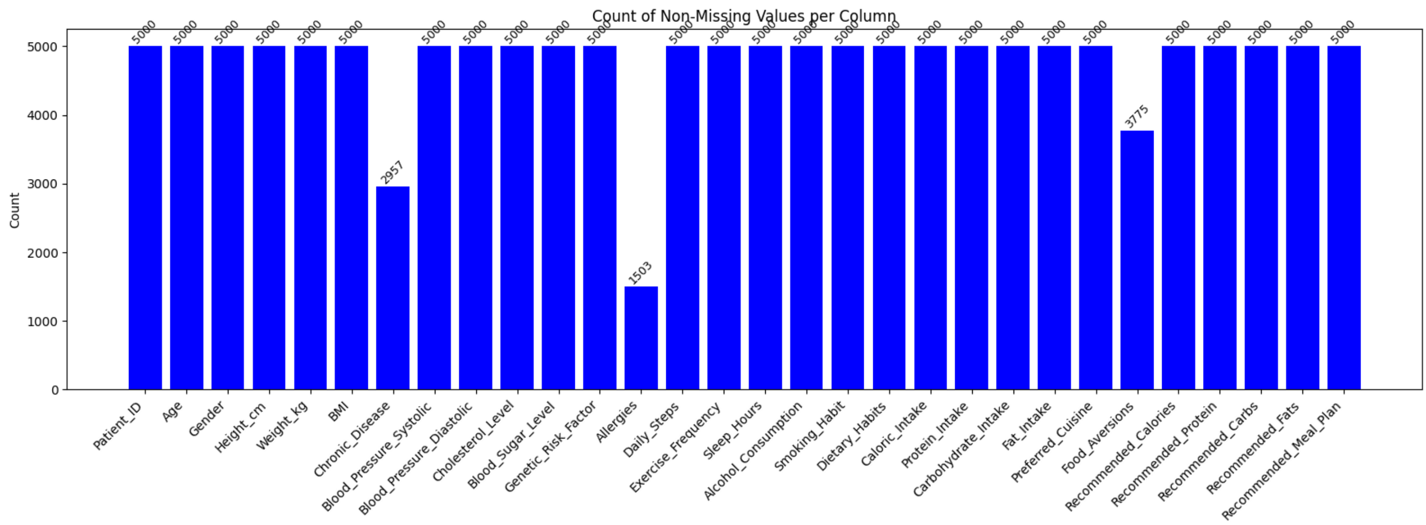
In [610]:

```
unique_dietary_habits = df['Dietary_Habits'].dropna().unique()

print("Unique Dietary Habits:")
for habit in sorted(unique_dietary_habits):
    print("-", habit)
```

```
Unique Dietary Habits:
- Keto
- Regular
- Vegan
- Vegetarian
```

In [611]:

```python
import pandas as pd
import matplotlib.pyplot as plt


non_missing_counts = df.notnull().sum()


plt.figure(figsize=(16, 6))
bars = plt.bar(non_missing_counts.index, non_missing_counts.values, color='blue')

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height, f'{int(height)}',
             ha='center', va='bottom', rotation=45, fontsize=9)

plt.xticks(rotation=45, ha='right')
plt.title("Count of Non-Missing Values per Column")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```



In [612]:

```python
df['Recommended_Meal_Plan'].value_counts()
```

Out[612]:

Recommended_Meal_Plan	count
Low-Fat Diet	1313
High-Protein Diet	1255
Balanced Diet	1250
Low-Carb Diet	1182

dtype: int64

In [616]:

```python
unique_dietary_habits = df['Dietary_Habits'].dropna().unique()
```

```
print("Unique Dietary Habits:")
for habit in sorted(unique_dietary_habits):
    print("-", habit)
```

```
Unique Dietary Habits:
- Keto
- Regular
- Vegan
- Vegetarian
```

Preprocessing

In [615]:

```
# Creating Smart_Diet from logic
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Calorie' if row['Caloric_Intake'] < 2000 else
                'High Protein' if row['Protein_Intake'] > 100 else
                'Balanced', axis=1)

# Map Smart_Diet to Recommended_Meal_Plan
df['Recommended_Meal_Plan'] = df['Smart_Diet'].map({
    'Low Calorie': 'Keto',
    'High Protein': 'Paleo',
    'Balanced': 'Mediterranean'
})
```

In [617]:

```
df['Recommended_Meal_Plan'].value_counts()
```

Out[617]:

	count
Recommended_Meal_Plan	
Paleo	2173
Keto	1732
Mediterranean	1095

dtype: int64

In [618]:

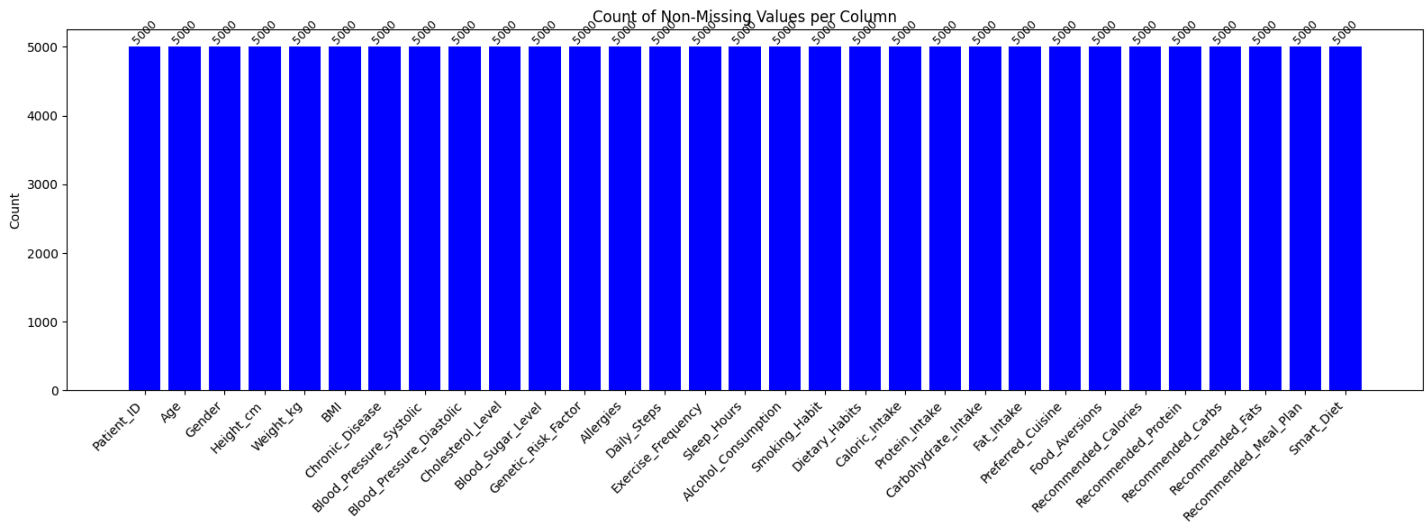
```
# Separating columns
num_cols = df.select_dtypes(include=['int64', 'float64']).columns
cat_cols = df.select_dtypes(include='object').columns

# Fill numeric columns with median
df[num_cols] = df[num_cols].apply(lambda x: x.fillna(x.median()))

# Fill categorical columns with mode
df[cat_cols] = df[cat_cols].apply(lambda x: x.fillna(x.mode()[0]))


print("Missing values filled: median for numeric, mode for categorical")


# df.dropna(inplace=True)
```

```
Missing values filled: median for numeric, mode for categorical
```

In [619]:

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

```
non_missing_counts = df.notnull().sum()


plt.figure(figsize=(16, 6))
bars = plt.bar(non_missing_counts.index, non_missing_counts.values, color='blue')

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height, f'{int(height)}',
             ha='center', va='bottom', rotation=45, fontsize=9)

plt.xticks(rotation=45, ha='right')
plt.title("Count of Non-Missing Values per Column")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```



In [670]:

```
# Encode categorical columns
cat_cols = df.select_dtypes(include='object').columns.difference(['Smart_Diet', 'Recomme
nded_Meal_Plan'])
encoders = {}
for col in cat_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    encoders[col] = le
```

In [671]:

```
# Encode target labels BEFORE train_test_split
le_sd = LabelEncoder()
le_rmp = LabelEncoder()

df['Smart_Diet'] = le_sd.fit_transform(df['Smart_Diet'])
df['Recommended_Meal_Plan'] = le_rmp.fit_transform(df['Recommended_Meal_Plan'])
```

In [672]:

```
# Define features and targets
X = df.drop(['Smart_Diet', 'Recommended_Meal_Plan'], axis=1)
y = df[['Smart_Diet', 'Recommended_Meal_Plan']].copy()
```

In [673]:

```
# Check class distributions
print(f"Smart_Diet distribution: {Counter(y['Smart_Diet'])}")
print(f"Recommended_Meal_Plan distribution: {Counter(y['Recommended_Meal_Plan'])}")
```

```
Smart_Diet distribution: Counter({1: 2173, 2: 1732, 0: 1095})
Recommended_Meal_Plan distribution: Counter({2: 2173, 0: 1732, 1: 1095})
```

In [674]:

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

In [675]:

```
# Scale numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [676]:

```
# Check for NaN values
if np.any(np.isnan(X_train_scaled)) or y_train.isna().any().any():
    raise ValueError("NaN values found in X_train_scaled or y_train")
```

In [677]:

```
# Before SMOTE Balance Smart_Diet
y_train_sd = y_train['Smart_Diet'].astype(int)
print(f"Before-SMOTE Smart_Diet distribution: {Counter(y_train_sd)}")
```

```
Before-SMOTE Smart_Diet distribution: Counter({1: 1738, 2: 1386, 0: 876})
```

In [678]:

```
sd_counts = Counter(y_train_sd)
min_class_samples = min(sd_counts.values())
k_neighbors_safe = max(1, min(min_class_samples - 1, 5))

smote = SMOTE(k_neighbors=k_neighbors_safe, random_state=42)
X_train_bal_sd, y_train_bal_sd = smote.fit_resample(X_train_scaled, y_train_sd)
y_train_bal_sd = y_train_bal_sd.astype(int)
print(f"After-SMOTE Smart_Diet distribution: {Counter(y_train_bal_sd)}")
```

```
After-SMOTE Smart_Diet distribution: Counter({1: 1738, 2: 1738, 0: 1738})
```

In [679]:

```
# Align Recommended_Meal_Plan with resampled Smart_Diet
original_indices = np.arange(len(X_train_scaled))
class_counts_before = Counter(y_train_sd)
class_counts_after = Counter(y_train_bal_sd)
sample_indices = []
for label in class_counts_after:
    original_mask = y_train_sd == label
    original_idx = original_indices[original_mask]
    n_original = class_counts_before[label]
    n_total = class_counts_after[label]
    sample_indices.extend(original_idx)
    if n_total > n_original:
        sample_indices.extend(np.repeat(original_idx[:1], n_total - n_original))
sample_indices = np.array(sample_indices)

y_train_bal = y_train.iloc[sample_indices].reset_index(drop=True)
y_train_bal['Smart_Diet'] = y_train_bal_sd
```

In [680]:

```
# SMOTE Balance Recommended_Meal_Plan
y_train_rmp = y_train_bal['Recommended_Meal_Plan'].astype(int)
if y_train_rmp.isna().any():
    raise ValueError("NaN values in y_train_bal['Recommended_Meal_Plan']")
print(f"Before-SMOTE Recommended_Meal_Plan distribution: {Counter(y_train_rmp)}")

rmp_counts = Counter(y_train_rmp)
min_class_samples = min(rmp_counts.values())
```

```
k_neighbors_safe = max(1, min(min_class_samples - 1, 5))

smote = SMOTE(k_neighbors=k_neighbors_safe, random_state=42)
X_train_bal, y_train_bal_rmp = smote.fit_resample(X_train_bal_sd, y_train_rmp)

y_train_bal_rmp = y_train_bal_rmp.astype(int)
print(f"After-SMOTE Recommended_Meal_Plan distribution: {Counter(y_train_bal_rmp)}")
```

```
Before-SMOTE Recommended_Meal_Plan distribution: Counter({2: 1738, 0: 1738, 1: 1738})
After-SMOTE Recommended_Meal_Plan distribution: Counter({2: 1738, 0: 1738, 1: 1738})
```

In [681]:

```
# Re-align Smart_Diet with resampled Recommended_Meal_Plan
original_indices = np.arange(len(y_train_bal))
class_counts_before = Counter(y_train_bal['Recommended_Meal_Plan'])
class_counts_after = Counter(y_train_bal_rmp)
sample_indices = []
for label in class_counts_after:
    original_mask = y_train_bal['Recommended_Meal_Plan'] == label
    original_idx = original_indices[original_mask]
    n_original = class_counts_before[label]
    n_total = class_counts_after[label]
    sample_indices.extend(original_idx)
    if n_total > n_original:
        sample_indices.extend(np.repeat(original_idx[:1], n_total - n_original))
sample_indices = np.array(sample_indices)
```

In [682]:

```
y_train_bal = y_train_bal.iloc[sample_indices].reset_index(drop=True)
y_train_bal['Recommended_Meal_Plan'] = y_train_bal_rmp
y_train_bal['Smart_Diet'] = y_train_bal['Smart_Diet'].astype(int)
print(f"y_train_bal dtypes: {y_train_bal.dtypes}")
```

```
y_train_bal dtypes: Smart_Diet               int64
Recommended_Meal_Plan    int64
dtype: object
```

In [683]:

```
# Train model
multi_model = MultiOutputClassifier(
    XGBClassifier(
        n_estimators=500,
        max_depth=7,
        learning_rate=0.05,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42,
        eval_metric='mlogloss'
    )
)
multi_model.fit(X_train_bal, y_train_bal)
```

Out[683]:

```
▸ MultiOutputClassifier
                    i ?

▸     estimator:
    XGBClassifier

  ▸   XGBClassifier
```

In [684]:

```
# Predict
y_pred = multi_model.predict(X_test_scaled)
```

In [685]:

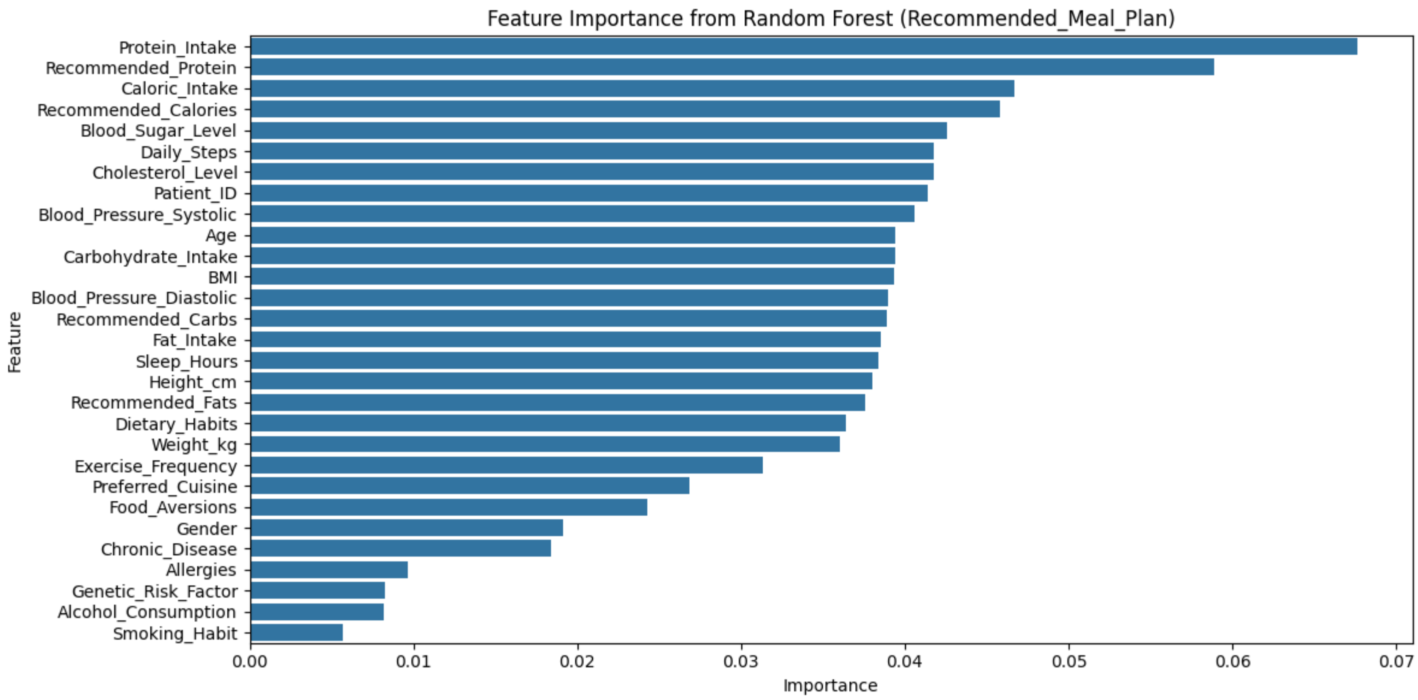
```
# Evaluate
y_pred_df = pd.DataFrame(y_pred, columns=['Smart_Diet', 'Recommended_Meal_Plan'])
y_pred_df['Smart_Diet'] = le_sd.inverse_transform(y_pred_df['Smart_Diet'])
y_pred_df['Recommended_Meal_Plan'] = le_rmp.inverse_transform(y_pred_df['Recommended_Meal
_Plan'])
y_test_decoded = y_test.copy()
y_test_decoded['Smart_Diet'] = le_sd.inverse_transform(y_test['Smart_Diet'])
y_test_decoded['Recommended_Meal_Plan'] = le_rmp.inverse_transform(y_test['Recommended_Me
al_Plan'])

for col in ['Smart_Diet', 'Recommended_Meal_Plan']:
    acc = accuracy_score(y_test_decoded[col], y_pred_df[col])
    print(f" {col} Accuracy: {acc:.4f}")
    print(f"Classification Report for {col}:\n")
    print(classification_report(y_test_decoded[col], y_pred_df[col], zero_division=0))
    print("-" * 40)
```

```
 Smart_Diet Accuracy: 0.9990
Classification Report for Smart_Diet:

              precision    recall  f1-score   support

    Balanced       1.00      1.00      1.00       219
High Protein       1.00      1.00      1.00       435
 Low Calorie       1.00      1.00      1.00       346

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000

----------------------------------------
 Recommended_Meal_Plan Accuracy: 0.4340
Classification Report for Recommended_Meal_Plan:

               precision    recall  f1-score   support

         Keto       0.39      0.54      0.45       346
Mediterranean       0.71      0.05      0.10       219
        Paleo       0.47      0.54      0.50       435

     accuracy                           0.43      1000
    macro avg       0.52      0.38      0.35      1000
 weighted avg       0.49      0.43      0.40      1000

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

In [637]:

```
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Train on Recommended_Meal_Plan balanced data
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_bal, y_train_bal_rmp)

# Get feature importances
importances = rf.feature_importances_

# Assuming X_train_bal comes from X_train_scaled with these columns:
feature_names = df.drop(columns=['Smart_Diet', 'Recommended_Meal_Plan']).columns

# Create DataFrame
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot
```

```
plt.figure(figsize=(12, 6))
sns.barplot(data=importance_df, x='Importance', y='Feature')
plt.title('Feature Importance from Random Forest (Recommended_Meal_Plan)')
plt.tight_layout()
plt.show()
```



In [638]:

```
# Imports
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
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_re
port
from imblearn.over_sampling import SMOTE

# Encode categorical features (excluding targets)
cat_cols = df.select_dtypes(include='object').columns.difference(['Smart_Diet', 'Recomme
nded_Meal_Plan'])
encoders = {}
for col in cat_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    encoders[col] = le

# Encode the target column (Recommended_Meal_Plan)
le_rmp = LabelEncoder()
df['Recommended_Meal_Plan'] = le_rmp.fit_transform(df['Recommended_Meal_Plan'])

# Feature and target separation
X = df.drop(columns=['Smart_Diet', 'Recommended_Meal_Plan'])
y = df['Recommended_Meal_Plan']

# Split the dataset before applying SMOTE
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, ran
dom_state=42)

# Apply SMOTE on training data only
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)
```

```python
# Scale the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_bal)
X_test_scaled = scaler.transform(X_test)

# Define classifiers
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "XGBoost": XGBClassifier(eval_metric='mlogloss', random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=1000, solver='lbfgs', random_stat
e=42)
}

# Train and evaluate each model
for name, model in models.items():
    print(f"\n{name}")

    # Train
    model.fit(X_train_scaled, y_train_bal)

    # Predict
    y_pred = model.predict(X_test_scaled)

    # Evaluation
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    print("Accuracy:", (acc*100))
    print("F1 Score:", f1)

    print("Classification Report:\n", classification_report(
        y_test, y_pred, target_names=le_rmp.classes_.astype(str)
    ))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    # plt.figure(figsize=(6, 4))
    # 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("--"*40)
```

```
Random Forest
Accuracy: 100.0
F1 Score: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       346
           1       1.00      1.00      1.00       219
           2       1.00      1.00      1.00       435

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000

--------------------------------------------------------------------------------

XGBoost
Accuracy: 99.7
F1 Score: 0.9969981513990142
Classification Report:
               precision    recall  f1-score   support

           0       0.99      1.00      1.00       346
           1       1.00      0.99      1.00       219
           2       1.00      1.00      1.00       435

    accuracy                           1.00      1000
```

```
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000

--------------------------------------------------------------------------------

Logistic Regression
Accuracy: 97.0
F1 Score: 0.9700353879941769
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.98      0.97       346
           1       0.96      0.96      0.96       219
           2       0.99      0.97      0.98       435

    accuracy                           0.97      1000
   macro avg       0.97      0.97      0.97      1000
weighted avg       0.97      0.97      0.97      1000

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

In [639]:

```python
from sklearn.model_selection import cross_val_score

for name, model in models.items():
    print(f"\n{name} (Cross-Validation Accuracy)")
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    print("CV Mean Accuracy:", scores.mean())
    print("CV Std Dev:", scores.std())
```

```
Random Forest (Cross-Validation Accuracy)
CV Mean Accuracy: 0.9996
CV Std Dev: 0.0004898979485566356

XGBoost (Cross-Validation Accuracy)
CV Mean Accuracy: 0.9982000000000001
CV Std Dev: 0.0009797958971132712

Logistic Regression (Cross-Validation Accuracy)
CV Mean Accuracy: 0.9456
CV Std Dev: 0.0032000000000000028
```

In [661]:

```python
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.metrics import accuracy_score, f1_score, confusion_matrix, classification_re
port
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectFromModel
from sklearn.impute import SimpleImputer

# Verify columns
print("Columns in DataFrame:", df.columns.tolist())

# Add noise to reduce deterministic mapping
np.random.seed(42)
df['Carbohydrate_Intake'] += np.random.normal(0, 20, df.shape[0])
df['Fat_Intake'] += np.random.normal(0, 15, df.shape[0])

# Redefine target with adjusted thresholds
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Carb' if row['Carbohydrate_Intake'] < 180 else
                'High Fat' if row['Fat_Intake'] > 90 else
```

```
                'Balanced', axis=1)
df['Recommended_Meal_Plan'] = df['Smart_Diet'].map({
    'Low Carb': 'Keto',
    'High Fat': 'Paleo',
    'Balanced': 'Mediterranean'
})
df = df.drop(columns=['Smart_Diet'])

# Drop non-predictive columns
columns_to_drop = [col for col in ['Patient_ID', 'BMI'] if col in df.columns]
df = df.drop(columns=columns_to_drop)

# Impute missing values before encoding
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()

# Check for highly correlated features
corr_matrix = df.select_dtypes(include=['int64', 'float64']).corr()
high_corr = [(i, j) for i in corr_matrix for j in corr_matrix if corr_matrix.loc[i, j] >
0.8 and i < j]
print("Highly correlated features:", high_corr)

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

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

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

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

# Step 8: Print class distribution
train_class_counts = pd.Series(y_train).value_counts()
print("Training set class distribution:", train_class_counts.to_dict())

# Apply SMOTE with corrected sampling strategy
smote = SMOTE(
    sampling_strategy={0: 1646, 1: 1646, 2: 1646},  # Balance classes
    random_state=42,
    k_neighbors=5
)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_bal)
X_test_scaled = scaler.transform(X_test)

# Feature selection using Random Forest
rf_for_selection = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=4
2)
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)
print("Selected features shape:", X_train_selected.shape)
selected_features = X.columns[selector.get_support()].tolist()
```

```
print("Selected features:", selected_features)
print("Feature importances:", dict(zip(X.columns, rf_for_selection.feature_importances_))
)

# Define tuned classifiers
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, 5.0, 10.0, 20.0],
            'solver': ['lbfgs', 'saga'],
            'class_weight': ['balanced']
        },
        cv=5,
        scoring='accuracy',
        n_jobs=-1
    )
}

# Train and evaluate models
for name, model in models.items():
    print(f"\n{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("Accuracy:", round(acc * 100, 2))
    print("F1 Score:", round(f1, 4))
    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='acc
uracy')
    print("Cross-Validation Accuracy (mean ± std):", f"{cv_scores.mean() * 100:.2f}% ± {
cv_scores.std() * 100:.2f}%")

    cm = confusion_matrix(y_test, y_pred)
    # Uncomment to visualize confusion matrix
    # plt.figure(figsize=(6, 4))
    # 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("--" * 40)
```

```
Columns in DataFrame: ['Age', 'Gender', 'Height_cm', 'Weight_kg', 'Chronic_Disease', 'Blo
od_Pressure_Systolic', 'Blood_Pressure_Diastolic', 'Cholesterol_Level', 'Blood_Sugar_Leve
l', 'Genetic_Risk_Factor', 'Allergies', 'Daily_Steps', 'Exercise_Frequency', 'Sleep_Hours
', 'Alcohol_Consumption', 'Smoking_Habit', 'Dietary_Habits', 'Caloric_Intake', 'Protein_I
ntake', 'Carbohydrate_Intake', 'Fat_Intake', 'Preferred_Cuisine', 'Food_Aversions', 'Reco
mmended_Calories', 'Recommended_Protein', 'Recommended_Carbs', 'Recommended_Fats', 'Recom
mended_Meal_Plan', 'Health_Score', 'Risk_Score']
Highly correlated features: [('Caloric_Intake', 'Recommended_Calories'), ('Protein_Intake
', 'Recommended_Protein')]
Class mappings: {0: 'Keto', 1: 'Mediterranean', 2: 'Paleo'}
Training set class distribution: {0: 1442, 1: 1335, 2: 1223}
Selected features shape: (4938, 3)
Selected features: ['Carbohydrate_Intake', 'Fat_Intake', 'Recommended_Carbs']
Feature importances: {'Age': np.float64(0.006472920471064673), 'Gender': np.float64(0.001
505672039095890 6), 'Height_cm': np.float64(0.006134038422741217), 'Weight_kg': np.float64
(0.006186817682737418), 'Chronic_Disease': np.float64(0.0021266824919182747), 'Blood_Pres
```

```
sure_Systolic': np.float64(0.006089905718036425), 'Blood_Pressure_Diastolic': np.float64
(0.005907394278162885), 'Cholesterol_Level': np.float64(0.006756306736900678), 'Blood_Sug
ar_Level': np.float64(0.006320871417388234), 'Genetic_Risk_Factor': np.float64(0.00099327
82699434515), 'Allergies': np.float64(0.001295282659855451), 'Daily_Steps': np.float64(0.
008152339381582602), 'Exercise_Frequency': np.float64(0.003309546987012298), 'Sleep_Hour
s': np.float64(0.006816640468274531), 'Alcohol_Consumption': np.float64(0.001137324145625
91), 'Smoking_Habit': np.float64(0.0007020585747328368), 'Dietary_Habits': np.float64(0.0
016952946844144708), 'Caloric_Intake': np.float64(0.007020656526342036), 'Protein_Intake'
: np.float64(0.005872363974178726), 'Carbohydrate_Intake': np.float64(0.49053369762880344
), 'Fat_Intake': np.float64(0.3540697713292837), 'Preferred_Cuisine': np.float64(0.002147
922030460854), 'Food_Aversions': np.float64(0.0014050186670294881), 'Recommended_Calories
': np.float64(0.006923747369427264), 'Recommended_Protein': np.float64(0.006638851609700l
12), 'Recommended_Carbs': np.float64(0.03827981802914785), 'Recommended_Fats': np.float64
(0.014598331375268224), 'Health_Score': np.float64(0.0009074470308711066), 'Risk_Score':
np.float64(0.0)}

XGBoost
Accuracy: 99.7
F1 Score: 0.997
Classification Report:
                precision    recall  f1-score   support

         Keto       1.00      1.00      1.00       361
Mediterranean       1.00      0.99      1.00       333
        Paleo       0.99      1.00      1.00       306

     accuracy                           1.00      1000
    macro avg       1.00      1.00      1.00      1000
 weighted avg       1.00      1.00      1.00      1000

Cross-Validation Accuracy (mean ± std): 99.76% ± 0.26%
--------------------------------------------------------------------------------

Logistic Regression
Accuracy: 98.7
F1 Score: 0.987
Classification Report:
                precision    recall  f1-score   support

         Keto       0.98      0.99      0.98       361
Mediterranean       1.00      0.98      0.99       333
        Paleo       0.99      0.98      0.98       306

     accuracy                           0.99      1000
    macro avg       0.99      0.99      0.99      1000
 weighted avg       0.99      0.99      0.99      1000

Cross-Validation Accuracy (mean ± std): 98.46% ± 0.71%
--------------------------------------------------------------------------------
```

In [641]:

```
print(df.columns.tolist())
```

```
['Age', 'Gender', 'Height_cm', 'Weight_kg', 'Chronic_Disease', 'Blood_Pressure_Systolic',
'Blood_Pressure_Diastolic', 'Cholesterol_Level', 'Blood_Sugar_Level', 'Genetic_Risk_Facto
r', 'Allergies', 'Daily_Steps', 'Exercise_Frequency', 'Sleep_Hours', 'Alcohol_Consumption
', 'Smoking_Habit', 'Dietary_Habits', 'Caloric_Intake', 'Protein_Intake', 'Carbohydrate_I
ntake', 'Fat_Intake', 'Preferred_Cuisine', 'Food_Aversions', 'Recommended_Calories', 'Rec
ommended_Protein', 'Recommended_Carbs', 'Recommended_Fats', 'Recommended_Meal_Plan']
```

In [ ]:

```
df.info()
```

In [662]:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
X_poly = poly.fit_transform(X)
```

In [663]:

```
from sklearn.feature_selection import SelectKBest, f_classif
selector = SelectKBest(score_func=f_classif, k=20)  # pick top 20 features
X_train_selected = selector.fit_transform(X_train_scaled, y_train_bal)
X_test_selected = selector.transform(X_test_scaled)
```

In [664]:

```
LogisticRegression(penalty='l1', solver='saga')
```

Out[664]:

```
▾           LogisticRegression
                                        i ?
LogisticRegression(penalty='l1', solver='saga')
```

In [665]:

```
# Train Logistic Regression GridSearchCV
logreg_model = models["Logistic Regression"]
logreg_model.fit(X_train_selected, y_train_bal)

# Extract the best model
best_logistic_regression = logreg_model.best_estimator_
```

In [646]:

```
xgb_model = XGBClassifier(
    max_depth=2, reg_lambda=6.0, reg_alpha=5.0, eval_metric='mlogloss', random_state=42
)
xgb_model.fit(X_train_selected, y_train_bal)
best_xgboost_model = xgb_model
```

APPLYING ENSEMBLE

In [647]:

```
from sklearn.ensemble import VotingClassifier

ensemble = VotingClassifier(estimators=[
    ('lr', best_logistic_regression),
    ('xgb', best_xgboost_model)
], voting='soft')

ensemble.fit(X_train_selected, y_train_bal)
y_pred = ensemble.predict(X_test_selected)
```

In [648]:

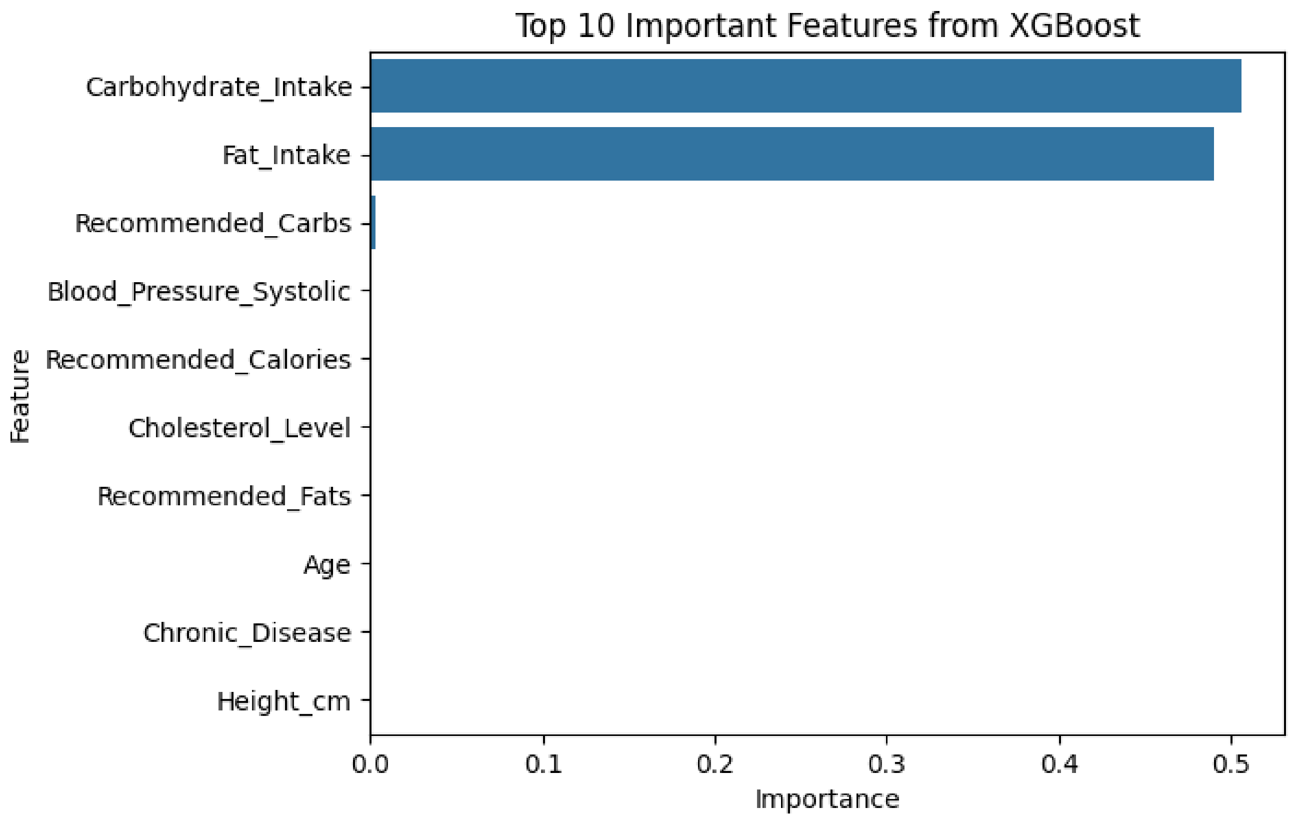
```
from sklearn.metrics import accuracy_score, f1_score, classification_report

acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')

print("Ensemble Accuracy:", acc * 100)
print("Ensemble F1 Score:", f1)
print("Classification Report:\n", classification_report(
    y_test, y_pred, target_names=le_rmp.classes_.astype(str)
))
```

```
Ensemble Accuracy: 99.3
Ensemble F1 Score: 0.9930006450254993
Classification Report:
                precision    recall  f1-score   support

         Keto       0.99      1.00      0.99       271
Mediterranean       1.00      0.99      0.99       393
        Paleo       0.99      1.00      1.00       336
```

```
    accuracy                           0.99      1000
   macro avg       0.99      0.99      0.99      1000
weighted avg       0.99      0.99      0.99      1000
```

In [649]:

```
importances = best_xgboost_model.feature_importances_
selected_features = selector.get_support(indices=True)
feature_names = X.columns[selected_features]

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

sns.barplot(data=importance_df.head(10), x='Importance', y='Feature')
plt.title("Top 10 Important Features from XGBoost")
plt.show()
```



In [650]:

```
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 sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_re
port
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectFromModel
from sklearn.impute import SimpleImputer

# Verify columns
print("Columns in DataFrame:", df.columns.tolist())

# Add noise to reduce deterministic mapping
np.random.seed(42)
df['Carbohydrate_Intake'] += np.random.normal(0, 25, df.shape[0])
```

```
df['Fat_Intake'] += np.random.normal(0, 20, df.shape[0])

# Redefine target with adjusted thresholds
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Carb' if row['Carbohydrate_Intake'] < 180 else
                'High Fat' if row['Fat_Intake'] > 90 else
                'Balanced', axis=1)
df['Recommended_Meal_Plan'] = df['Smart_Diet'].map({
    'Low Carb': 'Keto',
    'High Fat': 'Paleo',
    'Balanced': 'Mediterranean'
})
df = df.drop(columns=['Smart_Diet'])

# Drop non-predictive columns
columns_to_drop = [col for col in ['Patient_ID', 'BMI'] if col in df.columns]
df = df.drop(columns=columns_to_drop)

# Impute missing values before encoding
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()

#  Check for highly correlated features
corr_matrix = df.select_dtypes(include=['int64', 'float64']).corr()
high_corr = [(i, j) for i in corr_matrix for j in corr_matrix if corr_matrix.loc[i, j] >
0.8 and i < j]
print("Highly correlated features:", high_corr)

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

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

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

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

# Print class distribution
train_class_counts = pd.Series(y_train).value_counts()
print("Training set class distribution:", train_class_counts.to_dict())

#  Apply SMOTE with corrected sampling strategy
smote = SMOTE(
    sampling_strategy={0: 1646, 1: 1646, 2: 1646},  # Balance classes
    random_state=42,
    k_neighbors=5
)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_bal)
X_test_scaled = scaler.transform(X_test)

# Feature selection using Random Forest
rf_for_selection = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=4
2)
```

```
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)
print("Selected features shape:", X_train_selected.shape)
selected_features = X.columns[selector.get_support()].tolist()
print("Selected features:", selected_features)
print("Feature importances:", dict(zip(X.columns, rf_for_selection.feature_importances_))
)

#  Define Logistic Regression model
logistic_model = GridSearchCV(
    LogisticRegression(max_iter=2000, random_state=42),
    param_grid={
        'C': [0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0, 50.0],
        'solver': ['lbfgs', 'saga'],
        'class_weight': ['balanced']
    },
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

#  Train and evaluate Logistic Regression
print("\nLogistic Regression")

logistic_model.fit(X_train_selected, y_train_bal)
y_pred_logistic = logistic_model.predict(X_test_selected)

acc_logistic = accuracy_score(y_test, y_pred_logistic)
f1_logistic = f1_score(y_test, y_pred_logistic, average='weighted')

print("Accuracy:", round(acc_logistic * 100, 2))
print("F1 Score:", round(f1_logistic, 4))
print("Classification Report:\n", classification_report(y_test, y_pred_logistic, target_n
ames=le_rmp.classes_))

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

# Confusion matrix (optional visualization)
# cm_logistic = confusion_matrix(y_test, y_pred_logistic)
# plt.figure(figsize=(6, 4))
# sns.heatmap(cm_logistic, annot=True, fmt='d', cmap='Blues')
# plt.title('Confusion Matrix - Logistic Regression')
# plt.xlabel('Predicted')
# plt.ylabel('Actual')
# plt.tight_layout()
# plt.show()

print("--" * 40)
```

```
Columns in DataFrame: ['Age', 'Gender', 'Height_cm', 'Weight_kg', 'Chronic_Disease', 'Blo
od_Pressure_Systolic', 'Blood_Pressure_Diastolic', 'Cholesterol_Level', 'Blood_Sugar_Leve
l', 'Genetic_Risk_Factor', 'Allergies', 'Daily_Steps', 'Exercise_Frequency', 'Sleep_Hours
', 'Alcohol_Consumption', 'Smoking_Habit', 'Dietary_Habits', 'Caloric_Intake', 'Protein_I
ntake', 'Carbohydrate_Intake', 'Fat_Intake', 'Preferred_Cuisine', 'Food_Aversions', 'Reco
mmended_Calories', 'Recommended_Protein', 'Recommended_Carbs', 'Recommended_Fats', 'Recom
mended_Meal_Plan']
Highly correlated features: [('Caloric_Intake', 'Recommended_Calories'), ('Protein_Intake
', 'Recommended_Protein'), ('Carbohydrate_Intake', 'Recommended_Carbs')]
Class mappings: {0: 'Keto', 1: 'Mediterranean', 2: 'Paleo'}
Training set class distribution: {1: 1561, 2: 1338, 0: 1101}
Selected features shape: (4938, 4)
Selected features: ['Carbohydrate_Intake', 'Fat_Intake', 'Recommended_Carbs', 'Recommende
d_Fats']
Feature importances: {'Age': np.float64(0.0033317870577560423), 'Gender': np.float64(0.00
10690640246683368), 'Height_cm': np.float64(0.0038308378903513956), 'Weight_kg': np.float
64(0.0036213265621357106), 'Chronic_Disease': np.float64(0.0014716712717129576), 'Blood_P
ressure_Systolic': np.float64(0.003449709772963038), 'Blood_Pressure_Diastolic': np.float
```

```
64(0.0035102481767330493), 'Cholesterol_Level': np.float64(0.004117790072079343), 'Blood_
Sugar_Level': np.float64(0.0036430953213339693), 'Genetic_Risk_Factor': np.float64(0.0005
994480932218619), 'Allergies': np.float64(0.0007493635229322185), 'Daily_Steps': np.float
64(0.004671598568497702), 'Exercise_Frequency': np.float64(0.0020691623501544413), 'Sleep
_Hours': np.float64(0.0037825493985234253), 'Alcohol_Consumption': np.float64(0.000368022
2958298461), 'Smoking_Habit': np.float64(0.0004600590872574526), 'Dietary_Habits': np.flo
at64(0.0010962490231366424), 'Caloric_Intake': np.float64(0.004039968726301402), 'Protein
_Intake': np.float64(0.00411101495667810 6), 'Carbohydrate_Intake': np.float64(0.418994545
3060622), 'Fat_Intake': np.float64(0.3220070193850096), 'Preferred_Cuisine': np.float64(0
.0016386837182538207), 'Food_Aversions': np.float64(0.0011755135054509435), 'Recommended_
Calories': np.float64(0.0045798674076059 12), 'Recommended_Protein': np.float64(0.00403240
0842425556), 'Recommended_Carbs': np.float64(0.12515073083006217), 'Recommended_Fats': np
.float64(0.07242827283286286)}

Logistic Regression
Accuracy: 98.2
F1 Score: 0.982
Classification Report:
               precision    recall  f1-score   support

         Keto       0.96      0.97      0.97       275
Mediterranean       0.99      0.98      0.99       390
        Paleo       0.99      0.99      0.99       335

     accuracy                           0.98      1000
    macro avg       0.98      0.98      0.98      1000
 weighted avg       0.98      0.98      0.98      1000

Cross-Validation Accuracy (mean ± std): 98.52% ± 0.80%
--------------------------------------------------------------------------------
```

In [651]:

```
!pip install catboost
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.11/dist-packages (1.2.8
)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from
catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (fro
m catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packa
ges (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (f
rom catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from cat
boost) (1.15.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from ca
tboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catbo
ost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=0.24->catboost) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (f
rom pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages
(from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-package
s (from matplotlib->catboost) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (f
rom matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packag
es (from matplotlib->catboost) (4.58.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packag
es (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages
(from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from
matplotlib->catboost) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-package
s (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages
```

(from plotly->catboost) (9.1.2)

In [651]:

```

```

In [693]:

```
# ------------------ Imports ------------------
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_ma
trix


from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

from imblearn.over_sampling import SMOTE

# ------------------ Preprocessing ------------------

# Add Gaussian noise to reduce overfit
np.random.seed(42)
df['Carbohydrate_Intake'] += np.random.normal(0, 20, df.shape[0])
df['Fat_Intake'] += np.random.normal(0, 15, df.shape[0])

# Create target label
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Carb' if row['Carbohydrate_Intake'] < 180 else
                'High Fat' if row['Fat_Intake'] > 90 else
                'Balanced', axis=1)
df['Recommended_Meal_Plan'] = df['Smart_Diet'].map({
    'Low Carb': 'Keto',
    'High Fat': 'Paleo',
    'Balanced': 'Mediterranean'
})
df.drop(columns=['Smart_Diet'], inplace=True)

# Drop unnecessary columns
df.drop(columns=[col for col in ['Patient_ID', 'BMI'] if col in df.columns], inplace=Tru
e)

# 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()

# 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'])

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

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


# Print original class distribution
print("Original training class distribution:", Counter(y_train))

# Get max count among classes
max_count = max(Counter(y_train).values())

# Create sampling strategy to oversample minority classes to match max_count
sampling_strategy = {cls: max_count for cls in np.unique(y_train)}

# Apply SMOTE
smote = SMOTE(sampling_strategy=sampling_strategy, random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

print("After SMOTE:", Counter(y_train_bal))


# Add noise to X_train_bal
X_train_bal += np.random.normal(0, 0.01, X_train_bal.shape)

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

# Feature selection using RF
rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_train_scaled, y_train_bal)
selector = SelectFromModel(rf, threshold="1.0*mean", prefit=True)
X_train_sel = selector.transform(X_train_scaled)
X_test_sel = selector.transform(X_test_scaled)
selected_features = X.columns[selector.get_support()].tolist()
print("Selected Features:", selected_features)

# ------------------ Model Setup ------------------
models = {
    "Random Forest": RandomForestClassifier(
        n_estimators=50, max_depth=4, min_samples_split=100,
        min_samples_leaf=40, max_features='sqrt', random_state=42
    ),
    "XGBoost": XGBClassifier(
        max_depth=3, reg_lambda=25.0, reg_alpha=15.0, subsample=0.6,
        colsample_bytree=0.6, learning_rate=0.07, eval_metric='mlogloss',
        use_label_encoder=False, random_state=42
    ),
    "Logistic Regression": GridSearchCV(
        LogisticRegression(
            max_iter=1000, penalty='elasticnet', solver='saga',
            l1_ratio=0.5, random_state=42
        ),
        param_grid={'C': [0.1, 0.5, 1.0], 'class_weight': ['balanced']},
        cv=5, scoring='accuracy', n_jobs=-1
    ),
    "SVM": SVC(
        kernel='rbf', C=1.0, gamma='scale', class_weight='balanced', probability=True, r
andom_state=42
    ),
    "LightGBM": LGBMClassifier(
        num_leaves=20,
        max_depth=4,
        learning_rate=0.05,
        n_estimators=100,
```

```
        subsample=0.7,
        colsample_bytree=0.7,
        reg_lambda=5.0,
        reg_alpha=3.0,
        random_state=42,
        verbose=-1
    ),
    "CatBoost": CatBoostClassifier(
        iterations=100, depth=3, learning_rate=0.05,
        l2_leaf_reg=10, verbose=0, random_state=42
    )
}

# ------------------ Evaluation ------------------
for name, model in models.items():
    print(f"\nModel: {name}")
    model.fit(X_train_sel, y_train_bal)
    y_pred = model.predict(X_test_sel)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("Accuracy:", round(acc * 100, 2), "%")
    print("F1 Score:", round(f1, 4))
    print("Classification Report:\n", classification_report(y_test, y_pred, target_names
=le_rmp.classes_))

    cv = cross_val_score(model, X_train_sel, y_train_bal, cv=5, scoring='accuracy')
    print(f"CV Accuracy: {cv.mean() * 100:.2f}% ± {cv.std() * 100:.2f}%")
    print("-" * 80)
```

```
Original training class distribution: Counter({0: 1654, 1: 1232, 2: 1114})
After SMOTE: Counter({0: 1654, 2: 1654, 1: 1654})
Selected Features: ['Carbohydrate_Intake', 'Fat_Intake']

Model: Random Forest
Accuracy: 100.0 %
F1 Score: 1.0
Classification Report:
                precision    recall  f1-score   support

         Keto       1.00      1.00      1.00       413
Mediterranean       1.00      1.00      1.00       308
        Paleo       1.00      1.00      1.00       279

     accuracy                           1.00      1000
    macro avg       1.00      1.00      1.00      1000
 weighted avg       1.00      1.00      1.00      1000

CV Accuracy: 99.94% ± 0.08%
--------------------------------------------------------------------------------

Model: XGBoost
Accuracy: 99.6 %
F1 Score: 0.996
Classification Report:
                precision    recall  f1-score   support

         Keto       1.00      0.99      1.00       413
Mediterranean       0.99      1.00      1.00       308
        Paleo       1.00      1.00      1.00       279

     accuracy                           1.00      1000
    macro avg       1.00      1.00      1.00      1000
 weighted avg       1.00      1.00      1.00      1000

CV Accuracy: 99.74% ± 0.15%
--------------------------------------------------------------------------------

Model: Logistic Regression
Accuracy: 97.8 %
F1 Score: 0.978
```

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

         Keto       0.98      0.97      0.97       413
Mediterranean       0.97      0.99      0.98       308
        Paleo       0.99      0.99      0.99       279

     accuracy                           0.98      1000
    macro avg       0.98      0.98      0.98      1000
 weighted avg       0.98      0.98      0.98      1000

CV Accuracy: 97.86% ± 0.50%
--------------------------------------------------------------------------------

Model: SVM
Accuracy: 98.9 %
F1 Score: 0.989
Classification Report:
                precision    recall  f1-score   support

         Keto       1.00      0.98      0.99       413
Mediterranean       0.98      0.99      0.99       308
        Paleo       0.98      1.00      0.99       279

     accuracy                           0.99      1000
    macro avg       0.99      0.99      0.99      1000
 weighted avg       0.99      0.99      0.99      1000

CV Accuracy: 99.01% ± 0.20%
--------------------------------------------------------------------------------

Model: LightGBM
Accuracy: 99.4 %
F1 Score: 0.994
Classification Report:
                precision    recall  f1-score   support

         Keto       1.00      0.99      1.00       413
Mediterranean       0.98      1.00      0.99       308
        Paleo       1.00      0.99      0.99       279

     accuracy                           0.99      1000
    macro avg       0.99      0.99      0.99      1000
 weighted avg       0.99      0.99      0.99      1000

CV Accuracy: 99.76% ± 0.25%
--------------------------------------------------------------------------------

Model: CatBoost
Accuracy: 99.6 %
F1 Score: 0.996
Classification Report:
                precision    recall  f1-score   support

         Keto       1.00      0.99      1.00       413
Mediterranean       0.99      1.00      1.00       308
        Paleo       1.00      1.00      1.00       279

     accuracy                           1.00      1000
    macro avg       1.00      1.00      1.00      1000
 weighted avg       1.00      1.00      1.00      1000

CV Accuracy: 99.62% ± 0.20%
--------------------------------------------------------------------------------
```

In [653]:

```
# ------------------ Imports ------------------
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from collections import Counter
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_ma
trix

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from imblearn.over_sampling import SMOTE

# ------------------ Preprocessing ------------------

# Add Gaussian noise to reduce overfit
np.random.seed(42)
df['Carbohydrate_Intake'] += np.random.normal(0, 20, df.shape[0])
df['Fat_Intake'] += np.random.normal(0, 15, df.shape[0])

# Create target label
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Carb' if row['Carbohydrate_Intake'] < 180 else
                'High Fat' if row['Fat_Intake'] > 90 else
                'Balanced', axis=1)
df['Recommended_Meal_Plan'] = df['Smart_Diet'].map({
    'Low Carb': 'Keto',
    'High Fat': 'Paleo',
    'Balanced': 'Mediterranean'
})
df.drop(columns=['Smart_Diet'], inplace=True)

# Drop unnecessary columns
df.drop(columns=[col for col in ['Patient_ID', 'BMI'] if col in df.columns], inplace=Tru
e)

# 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()

# 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'])

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

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

# ------------------ SMOTE Balancing ------------------

print("Original training class distribution:", Counter(y_train))

# Get max count among classes
max_count = max(Counter(y_train).values())
```

```python
# Create sampling strategy to oversample minority classes
sampling_strategy = {cls: max_count for cls in np.unique(y_train)}

# Apply SMOTE
smote = SMOTE(sampling_strategy=sampling_strategy, random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

print("After SMOTE:", Counter(y_train_bal))

# Add noise to reduce model memorization
X_train_bal += np.random.normal(0, 0.01, X_train_bal.shape)

# ------------------ Scaling ------------------

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_bal)
X_test_scaled = scaler.transform(X_test)

# ------------------ Feature Selection ------------------

rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_train_scaled, y_train_bal)
selector = SelectFromModel(rf, threshold="1.0*mean", prefit=True)

X_train_sel = selector.transform(X_train_scaled)
X_test_sel = selector.transform(X_test_scaled)
selected_features = X.columns[selector.get_support()].tolist()
print("Selected Features:", selected_features)

# ------------------ Model Setup ------------------

models = {
    "SVM": SVC(
        kernel='rbf', C=1.0, gamma='scale', class_weight='balanced',
        probability=True, random_state=42
    ),
}

# ------------------ Evaluation ------------------

for name, model in models.items():
    print(f"\nModel: {name}")
    model.fit(X_train_sel, y_train_bal)
    y_pred = model.predict(X_test_sel)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("Accuracy:", round(acc * 100, 2), "%")
    print("F1 Score:", round(f1, 4))
    print("Classification Report:\n", classification_report(y_test, y_pred, target_names
=le_rmp.classes_))

    cv = cross_val_score(model, X_train_sel, y_train_bal, cv=5, scoring='accuracy')
    print(f"CV Accuracy: {cv.mean() * 100:.2f}% ± {cv.std() * 100:.2f}%")
    print("-" * 80)
```

```
Original training class distribution: Counter({1: 1481, 2: 1318, 0: 1201})
After SMOTE: Counter({0: 1481, 1: 1481, 2: 1481})
Selected Features: ['Carbohydrate_Intake', 'Fat_Intake', 'Recommended_Carbs']

Model: SVM
Accuracy: 98.4 %
F1 Score: 0.984
Classification Report:
                precision    recall  f1-score   support

         Keto       0.97      0.99      0.98       301
Mediterranean       1.00      0.97      0.98       370
        Paleo       0.98      0.99      0.99       329

     accuracy                           0.98      1000
```

```
    macro avg       0.98      0.98      0.98      1000
 weighted avg       0.98      0.98      0.98      1000

CV Accuracy: 98.38% ± 0.52%
--------------------------------------------------------------------------------
```

In [654]:

```python
# ------------------ Imports ------------------
import pandas as pd
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings("ignore")


# ------------------ Data Preprocessing ------------------

# Add slight noise to reduce overfit before modeling
np.random.seed(42)
df['Carbohydrate_Intake'] += np.random.normal(0, 10, df.shape[0])
df['Fat_Intake'] += np.random.normal(0, 8, df.shape[0])

# Generate target
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Carb' if row['Carbohydrate_Intake'] < 180 else
                'High Fat' if row['Fat_Intake'] > 90 else
                'Balanced', axis=1)
df['Recommended_Meal_Plan'] = df['Smart_Diet'].map({
    'Low Carb': 'Keto',
    'High Fat': 'Paleo',
    'Balanced': 'Mediterranean'
})
df.drop(columns=['Smart_Diet'], inplace=True)

# Drop irrelevant columns
df.drop(columns=[col for col in ['Patient_ID', 'BMI'] if col in df.columns], inplace=Tru
e)

# Fill 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()

# Encode categorical variables
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 variable
le_rmp = LabelEncoder()
df['Recommended_Meal_Plan'] = le_rmp.fit_transform(df['Recommended_Meal_Plan'])

# ------------------ Train-Test Split ------------------
X = df.drop(columns=['Recommended_Meal_Plan'])
y = df['Recommended_Meal_Plan']
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, ran
dom_state=42)
```

```python
# ------------------ SMOTE Balancing ------------------
print("Original class distribution:", Counter(y_train))
max_count = max(Counter(y_train).values())
smote = SMOTE(sampling_strategy={cls: max_count for cls in np.unique(y_train)}, random_s
tate=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)
print("After SMOTE:", Counter(y_train_bal))

# ------------------ Scaling ------------------
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_bal)
X_test_scaled = scaler.transform(X_test)

# ------------------ Feature Selection ------------------
rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_train_scaled, y_train_bal)
selector = SelectFromModel(rf, threshold="0.9*mean", prefit=True)  # slightly relaxed thr
eshold
X_train_sel = selector.transform(X_train_scaled)
X_test_sel = selector.transform(X_test_scaled)
selected_features = X.columns[selector.get_support()].tolist()
print("Selected Features:", selected_features)

# ------------------ Models ------------------
models = {
    "SVM": SVC(
        kernel='rbf', C=1.0, gamma='scale',
        class_weight='balanced', probability=True, random_state=42
    ),
    "LightGBM": LGBMClassifier(
        num_leaves=20,
        max_depth=4,
        learning_rate=0.05,
        n_estimators=100,
        subsample=0.7,
        colsample_bytree=0.7,
        reg_lambda=5.0,
        reg_alpha=3.0,
        random_state=42,
        verbose=-1  # <--- Suppresses all LightGBM internal training logs
)

}

# ------------------ Evaluation ------------------
for name, model in models.items():
    print(f"\n===== {name} Evaluation =====")
    model.fit(X_train_sel, y_train_bal)
    y_pred = model.predict(X_test_sel)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    report = classification_report(y_test, y_pred, target_names=le_rmp.classes_)

    print(f"Accuracy: {acc * 100:.2f}%")
    print(f"F1 Score: {f1:.4f}")
    print("Classification Report:")
    print(report)

    cv = cross_val_score(model, X_train_sel, y_train_bal, cv=5, scoring='accuracy')
    print(f"Cross-Val Accuracy: {cv.mean() * 100:.2f}% ± {cv.std() * 100:.2f}%")
    print("=" * 60)
```

```
Original class distribution: Counter({1: 1476, 2: 1298, 0: 1226})
After SMOTE: Counter({0: 1476, 1: 1476, 2: 1476})
Selected Features: ['Carbohydrate_Intake', 'Fat_Intake', 'Recommended_Carbs']

===== SVM Evaluation =====
Accuracy: 98.80%
F1 Score: 0.9880
Classification Report:
```

```
              precision    recall  f1-score   support

         Keto       0.98      0.99      0.99       306
Mediterranean       0.99      0.99      0.99       369
        Paleo       1.00      0.98      0.99       325

     accuracy                           0.99      1000
    macro avg       0.99      0.99      0.99      1000
 weighted avg       0.99      0.99      0.99      1000

Cross-Val Accuracy: 98.35% ± 0.32%
============================================================

===== LightGBM Evaluation =====
Accuracy: 99.70%
F1 Score: 0.9970
Classification Report:
              precision    recall  f1-score   support

         Keto       1.00      1.00      1.00       306
Mediterranean       0.99      1.00      1.00       369
        Paleo       1.00      0.99      1.00       325

     accuracy                           1.00      1000
    macro avg       1.00      1.00      1.00      1000
 weighted avg       1.00      1.00      1.00      1000

Cross-Val Accuracy: 99.59% ± 0.23%
============================================================
```

In [695]:

```
# ------------------ Imports ------------------
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_ma
trix

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from imblearn.over_sampling import SMOTE

# ------------------ Load & Add Noise ------------------
np.random.seed(42)
df['Carbohydrate_Intake'] += np.random.normal(0, 20, df.shape[0])
df['Fat_Intake'] += np.random.normal(0, 15, df.shape[0])

# ------------------ Create Targets ------------------
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Carb' if row['Carbohydrate_Intake'] < 180 else
                'High Fat' if row['Fat_Intake'] > 90 else
                'Balanced', axis=1)

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

# ------------------ 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()

# ------------------ Feature Engineering ------------------
def compute_health_score(row):
    score = 0
    if row['Blood_Pressure_Systolic'] >= 140 or row['Blood_Pressure_Diastolic'] >= 90:
        score += 2
    elif row['Blood_Pressure_Systolic'] <= 90 or row['Blood_Pressure_Diastolic'] <= 60:
        score += 1
    if row['Cholesterol_Level'] == 'High':
        score += 2
    if row['Blood_Sugar_Level'] == 'High':
        score += 2
    return score

def compute_risk_profile(row):
    score = 0
    if row['Chronic_Disease'] != 'None':
        score += 2
    if row['Genetic_Risk_Factor'] != 'None':
        score += 2
    if row['Smoking_Habit'] != 'Non-Smoker':
        score += 1
    if row['Alcohol_Consumption'] != 'None':
        score += 1
    if 'BMI' in df.columns and row['BMI'] > 30:
        score += 2
    return score

df['Health_Score'] = df.apply(compute_health_score, axis=1)
df['Risk_Score'] = df.apply(compute_risk_profile, axis=1)

# ------------------ Drop Unnecessary Columns ------------------
df.drop(columns=[col for col in ['Patient_ID', 'BMI'] if col in df.columns], inplace=Tru
e)

# ------------------ 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'])

# ------------------ Train-Test Split ------------------
X = df.drop(columns=['Recommended_Meal_Plan'])
y = df['Recommended_Meal_Plan']
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, ran
dom_state=42)

# ------------------ SMOTE Balancing ------------------
print("Original training class distribution:", Counter(y_train))

max_count = max(Counter(y_train).values())
sampling_strategy = {cls: max_count for cls in np.unique(y_train)}

smote = SMOTE(sampling_strategy=sampling_strategy, random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

print("After SMOTE:", Counter(y_train_bal))

# Add small noise to prevent memorization
X_train_bal += np.random.normal(0, 0.01, X_train_bal.shape)
```

```python
# ------------------ Scaling ------------------
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_bal)
X_test_scaled = scaler.transform(X_test)

# ------------------ Feature Selection ------------------
rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_train_scaled, y_train_bal)
selector = SelectFromModel(rf, threshold="1.0*mean", prefit=True)

X_train_sel = selector.transform(X_train_scaled)
X_test_sel = selector.transform(X_test_scaled)

selected_features = X.columns[selector.get_support()].tolist()
print("Selected Features:", selected_features)

# ------------------ Model Setup ------------------
models = {
    "SVM": SVC(
        kernel='rbf', C=1.0, gamma='scale', class_weight='balanced',
        probability=True, random_state=42
    ),
}

# ------------------ Evaluation ------------------
for name, model in models.items():
    print(f"\nModel: {name}")
    model.fit(X_train_sel, y_train_bal)
    y_pred = model.predict(X_test_sel)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("Accuracy:", round(acc * 100, 2), "%")
    print("F1 Score:", round(f1, 4))
    print("Classification Report:\n", classification_report(y_test, y_pred, target_names
=le_rmp.classes_))

    cv = cross_val_score(model, X_train_sel, y_train_bal, cv=5, scoring='accuracy')
    print(f"CV Accuracy: {cv.mean() * 100:.2f}% ± {cv.std() * 100:.2f}%")
    print("-" * 80)
```

```
Original training class distribution: Counter({0: 1677, 1: 1223, 2: 1100})
After SMOTE: Counter({1: 1677, 0: 1677, 2: 1677})
Selected Features: ['Carbohydrate_Intake', 'Fat_Intake']

Model: SVM
Accuracy: 98.8 %
F1 Score: 0.988
Classification Report:
               precision    recall  f1-score   support

         Keto       1.00      0.98      0.99       419
Mediterranean       0.98      1.00      0.99       306
        Paleo       0.99      0.99      0.99       275

     accuracy                           0.99      1000
    macro avg       0.99      0.99      0.99      1000
 weighted avg       0.99      0.99      0.99      1000

CV Accuracy: 98.77% ± 0.47%
--------------------------------------------------------------------------------
```

In [656]:

```python
# ------------------ Imports ------------------
import pandas as pd
import numpy as np
import warnings
from collections import Counter
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```python
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from imblearn.over_sampling import SMOTE

warnings.filterwarnings("ignore")

# ------------------- Step 1: Add Noise -------------------
np.random.seed(42)
df['Carbohydrate_Intake'] += np.random.normal(0, 20, df.shape[0])
df['Fat_Intake'] += np.random.normal(0, 15, df.shape[0])

# ------------------- Step 2: Generate Targets -------------------
df['Smart_Diet'] = df.apply(
    lambda row: 'Low Carb' if row['Carbohydrate_Intake'] < 180 else
                'High Fat' if row['Fat_Intake'] > 90 else
                'Balanced', axis=1)

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

# ------------------- Step 3: 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()

# ------------------- Step 4: Feature Engineering -------------------
def compute_health_score(row):
    score = 0
    if row['Blood_Pressure_Systolic'] >= 140 or row['Blood_Pressure_Diastolic'] >= 90:
        score += 2
    elif row['Blood_Pressure_Systolic'] <= 90 or row['Blood_Pressure_Diastolic'] <= 60:
        score += 1
    if row['Cholesterol_Level'] == 'High':
        score += 2
    if row['Blood_Sugar_Level'] == 'High':
        score += 2
    return score

def compute_risk_profile(row):
    score = 0
    if row['Chronic_Disease'] != 'None':
        score += 2
    if row['Genetic_Risk_Factor'] != 'None':
        score += 2
    if row['Smoking_Habit'] != 'Non-Smoker':
        score += 1
    if row['Alcohol_Consumption'] != 'None':
        score += 1
    if 'BMI' in df.columns and row['BMI'] > 30:
        score += 2
    return score

df['Health_Score'] = df.apply(compute_health_score, axis=1)
df['Risk_Score'] = df.apply(compute_risk_profile, axis=1)

# ------------------- Step 5: Drop Unnecessary -------------------
df.drop(columns=[col for col in ['Patient_ID', 'BMI'] if col in df.columns], inplace=Tru
e)

# ------------------- Step 6: Encode -------------------
cat_cols = df.select_dtypes(include='object').columns.difference(['Recommended_Meal_Plan'
])
for col in cat_cols:
    le = LabelEncoder()
```

```
    df[col] = le.fit_transform(df[col])

le_rmp = LabelEncoder()
df['Recommended_Meal_Plan'] = le_rmp.fit_transform(df['Recommended_Meal_Plan'])

# ------------------ Step 7: Split ------------------
X = df.drop(columns=['Recommended_Meal_Plan'])
y = df['Recommended_Meal_Plan']
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, ran
dom_state=42)

# ------------------ Step 8: Standard Scaling ------------------
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ------------------ Step 9: Train Models (Before SMOTE) ------------------
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "SVM": SVC(kernel='rbf', C=1.0, gamma='scale', probability=True, class_weight='balan
ced', random_state=42),
    "LightGBM": LGBMClassifier(random_state=42)
}

print("\n Results BEFORE SMOTE")
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    print(f"\nModel: {name}")
    print("Accuracy:", round(acc * 100, 2), "%")
    print("F1 Score:", round(f1, 4))
    print("-" * 50)

# ------------------ Step 10: Apply SMOTE ------------------
print("\nOriginal training distribution:", Counter(y_train))

smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)
print("After SMOTE distribution:", Counter(y_train_bal))

X_train_bal_scaled = scaler.fit_transform(X_train_bal)

# ------------------ Step 11: Train Models (After SMOTE) ------------------
print("\n Results AFTER SMOTE")
for name, model in models.items():
    model.fit(X_train_bal_scaled, y_train_bal)
    y_pred = model.predict(X_test_scaled)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    print(f"\nModel: {name}")
    print("Accuracy:", round(acc * 100, 2), "%")
    print("F1 Score:", round(f1, 4))
    print("-" * 50)
```

```
 Results BEFORE SMOTE

Model: Random Forest
Accuracy: 100.0 %
F1 Score: 1.0
--------------------------------------------------

Model: SVM
Accuracy: 92.1 %
F1 Score: 0.9209
--------------------------------------------------
```

```
Model: LightGBM
Accuracy: 99.7 %
F1 Score: 0.997
--------------------------------------------------

Original training distribution: Counter({1: 1396, 0: 1350, 2: 1254})
After SMOTE distribution: Counter({1: 1396, 0: 1396, 2: 1396})

 Results AFTER SMOTE

Model: Random Forest
Accuracy: 99.0 %
F1 Score: 0.99
--------------------------------------------------

Model: SVM
Accuracy: 92.0 %
F1 Score: 0.92
--------------------------------------------------

Model: LightGBM
Accuracy: 98.9 %
F1 Score: 0.989
--------------------------------------------------
```

In [656]:

```
```