Multi-Class Prediction of Obesity Risk

Accuracy: 0.913 Mean CV Accuracy: 0.913 AUC: 0.99

(EXPLORATORY DATA ANALYSIS + CLASSIFICATION & PREDICTION with HYPERPARAMETER Tuning)

1. **Logistic Regression**
2. **K-Nearest Neighbours**
3. **Support Vector Machines**
4. **Gaussian Naïve Bayes**
5. **Decision Tree**
6. **Random Forest**
7. **Gradient Boosting**
8. **XGBoost**
9. **CatBoost**
10. **LightGBM**

Importing the Libraries and Loading the dataset

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import chi2
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

warnings.filterwarnings("ignore", category=FutureWarning)


import warnings
warnings.filterwarnings('ignore')

import optuna
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBc
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
```

In [2]:
```
df1 = pd.read_csv('ObesityDataSet.csv')
```

In [3]:
```
df=df1.copy()
```

In [4]:
```
#Import the dataset
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
```

Basic Exploratory Data Analysis

In [5]:
```
df.shape
```

Out[5]:
```
(2111, 17)
```

In [6]:
```
df_train.shape
```

Out[6]: (20758, 18)

In [7]:
```
df_test.shape
```

Out[7]: (13840, 17)

In [8]:
```
df.head()
```

Out[8]:

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

In [9]:
```
df.info()
```

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

In [10]:
```
df.describe()
```

Out[10]:

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	
count	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	21
mean	24.312600	1.701677	86.586058	2.419043	2.685628	2.008011	1.010298	
std	6.345968	0.093305	26.191172	0.533927	0.778039	0.612953	0.850592	
min	14.000000	1.450000	39.000000	1.000000	1.000000	1.000000	0.000000	
25%	19.947192	1.630000	65.473343	2.000000	2.658738	1.584812	0.124505	
50%	22.777890	1.700499	83.000000	2.385502	3.000000	2.000000	1.000000	
75%	26.000000	1.768464	107.430682	3.000000	3.000000	2.477420	1.666678	
max	61.000000	1.980000	173.000000	3.000000	4.000000	3.000000	3.000000	

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
```

The attributes are:

FAVC: Frequent consumption of high caloric food

FCVC: Frequency of consumption of vegetables

NCP: Number of main meals

CAEC: Consumption of food between meals

CH20: Consumption of water daily

CALC: Consumption of alcohol

SCC: Calories consumption monitoring

FAF: Physical activity frequency

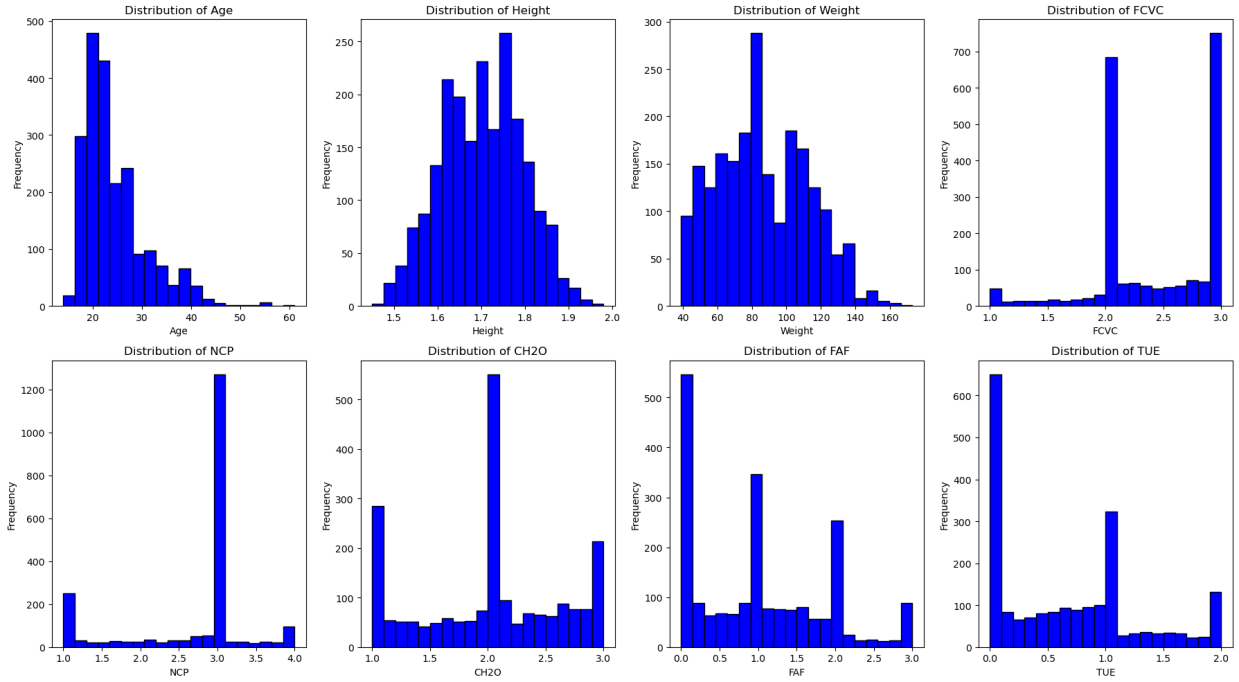
TUE: Time using technology devices

MTRANS: Transportation used

Visualizing the data

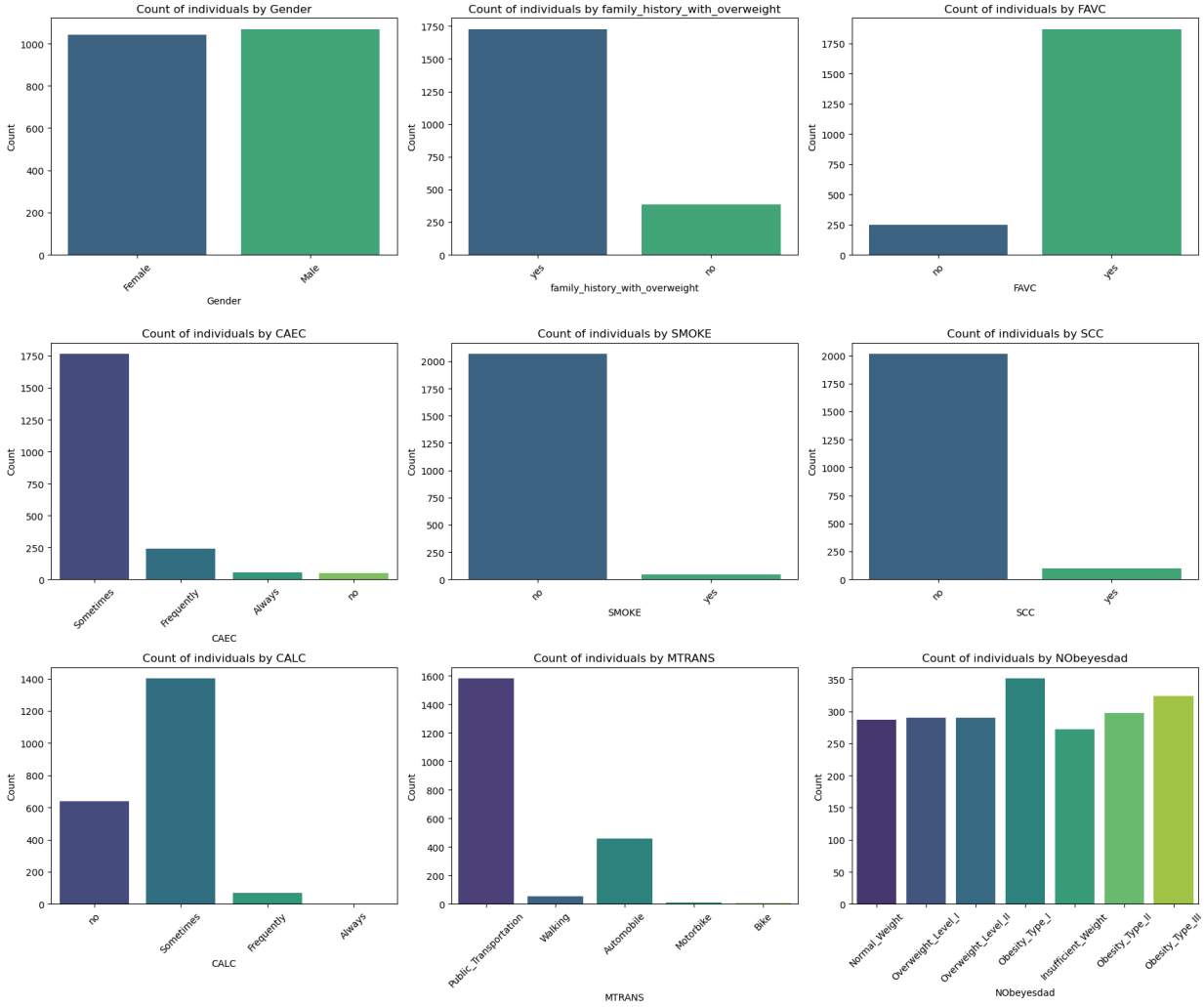
Numerical Features

In [12]:
```
numerical_features = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(18, 10))
axes = axes.flatten()
for i, feature in enumerate(numerical_features):
    axes[i].hist(df[feature], bins=20, color='blue', edgecolor='black')
    axes[i].set_title(f'Distribution of {feature}')
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Frequency')

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



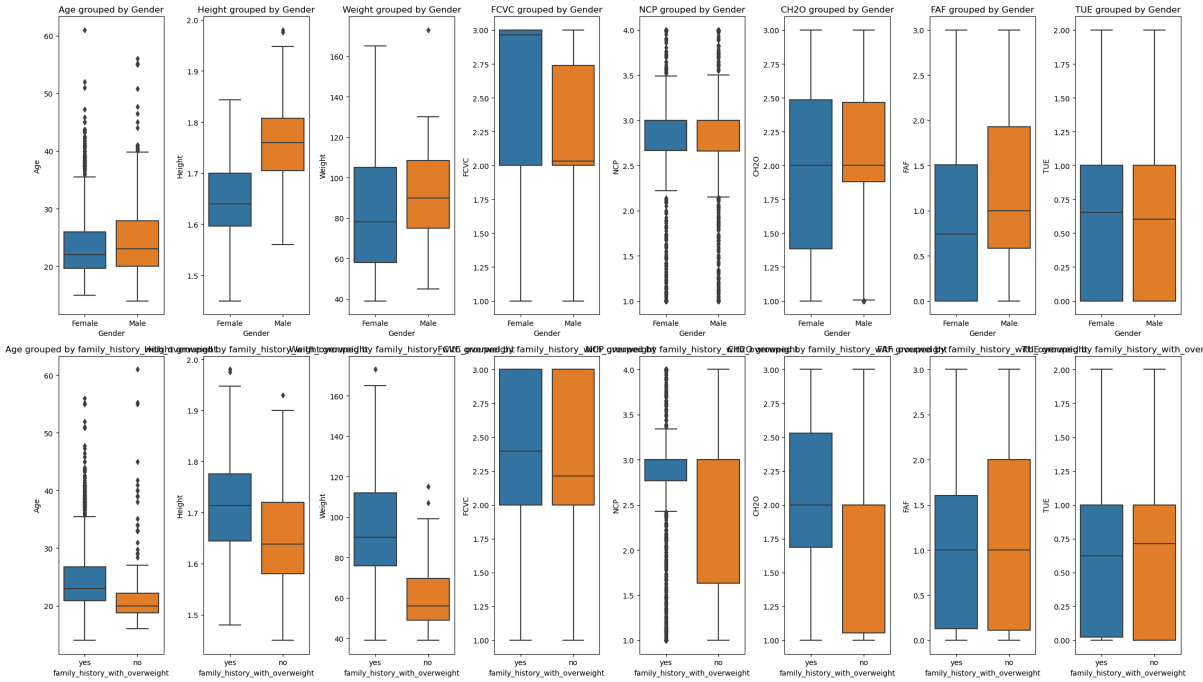
Categorical Features

In [13]:
```
categorical_features = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'S
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(18, 15))
axes = axes.flatten()
for i, feature in enumerate(categorical_features):
    sns.countplot(x=feature, data=df, ax=axes[i], palette='viridis')
    axes[i].set_title(f'Count of individuals by {feature}')
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Count')
    axes[i].tick_params(axis='x', rotation=45)

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

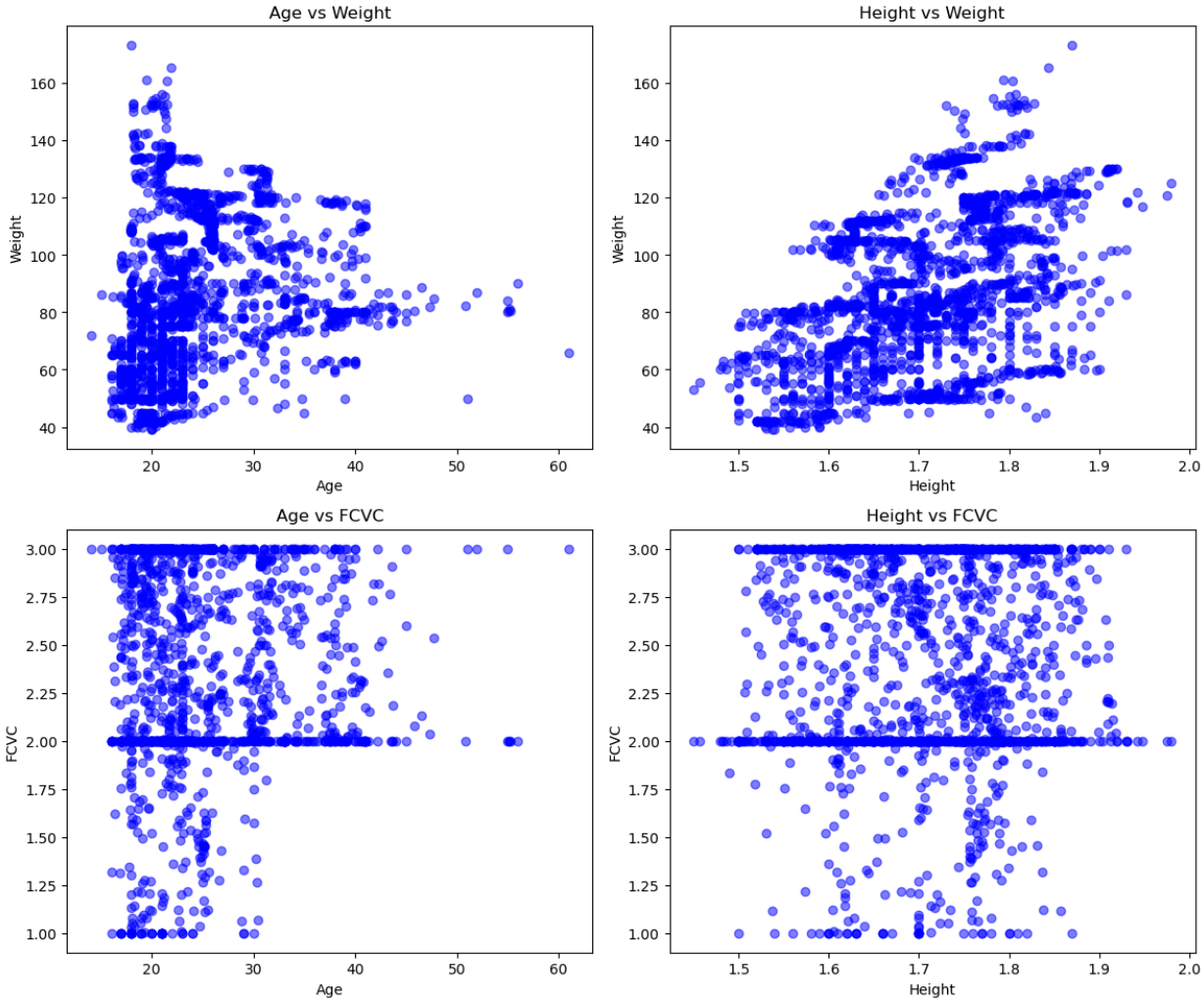


In [14]:

```
categorical_variables = ['Gender', 'family_history_with_overweight']
numerical_features = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']
fig, axes = plt.subplots(nrows=len(categorical_variables), ncols=len(numerical_feature
for i, cat_var in enumerate(categorical_variables):
    for j, num_var in enumerate(numerical_features):
        sns.boxplot(x=cat_var, y=num_var, data=df, ax=axes[i, j])
        axes[i, j].set_title(f'{num_var} grouped by {cat_var}')
        axes[i, j].set_xlabel(cat_var)
        axes[i, j].set_ylabel(num_var)

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



In [15]:
```
numerical_pairs = [('Age', 'Weight'), ('Height', 'Weight'), ('Age', 'FCVC'), ('Height'
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
axes = axes.flatten()
for i, (x_var, y_var) in enumerate(numerical_pairs):
    axes[i].scatter(df[x_var], df[y_var], alpha=0.5, color='blue')
    axes[i].set_title(f'{x_var} vs {y_var}')
    axes[i].set_xlabel(x_var)
    axes[i].set_ylabel(y_var)
plt.tight_layout()
plt.show()
```



In [16]:

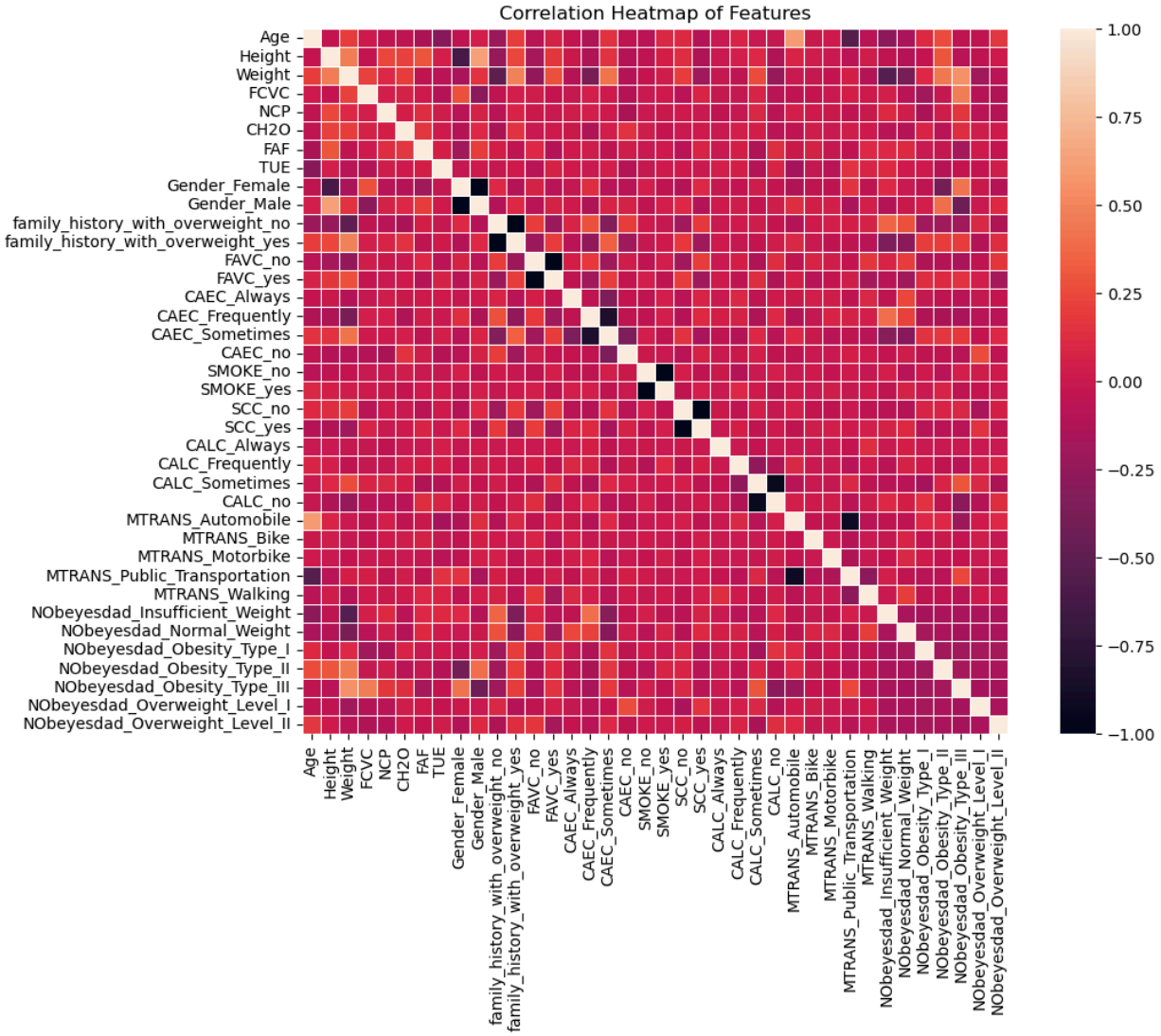
```
categorical_variables = ['family_history_with_overweight', 'FAVC', 'SMOKE', 'SCC', 'CA
num_plots = len(categorical_variables)
num_cols = 4
num_rows = -(-num_plots // num_cols)
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(18, 5*num_rows))
axes = axes.flatten()

for i, cat_var in enumerate(categorical_variables):
    category_counts = df[cat_var].value_counts()
    labels = category_counts.index
    sizes = category_counts.values
    axes[i].pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
    axes[i].set_title(f'Proportion of {cat_var}')
    axes[i].axis('equal')
for j in range(num_plots, num_cols*num_rows):
    fig.delaxes(axes[j])

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

In [17]:
```
numerical_columns = df.select_dtypes(include=['float64']).columns
non_numerical_columns = df.select_dtypes(include=['object']).columns
print("Non-Numeric Columns:")
print(non_numerical_columns)
```

```
Non-Numeric Columns:
Index(['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE',
       'SCC', 'CALC', 'MTRANS', 'NObeyesdad'],
      dtype='object')
```

In [18]:
```
encoded_data = pd.get_dummies(df, columns=non_numerical_columns)
correlation_matrix = encoded_data.corr()
```

In [19]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, linewidths=.5, square=True)
plt.title('Correlation Heatmap of Features')
plt.show()
```



In [20]:
```
print(correlation_matrix)
```

	Age	Height	Weight	FCVC	\
Age	1.000000	-0.025958	0.202560	0.016291	
Height	-0.025958	1.000000	0.463136	-0.038121	
Weight	0.202560	0.463136	1.000000	0.216125	
FCVC	0.016291	-0.038121	0.216125	1.000000	
NCP	-0.043944	0.243672	0.107469	0.042216	
CH2O	-0.045304	0.213376	0.200575	0.068461	
FAF	-0.144938	0.294709	-0.051436	0.019939	
TUE	-0.296931	0.051912	-0.071561	-0.101135	
Gender_Female	-0.048394	-0.618466	-0.161668	0.274505	
Gender_Male	0.048394	0.618466	0.161668	-0.274505	
family_history_with_overweight_no	-0.205725	-0.247684	-0.496820	-0.040372	
family_history_with_overweight_yes	0.205725	0.247684	0.496820	0.040372	
FAVC_no	-0.063902	-0.178364	-0.272300	0.027283	
FAVC_yes	0.063902	0.178364	0.272300	-0.027283	
CAEC_Always	-0.031292	0.004580	-0.094966	-0.018204	
CAEC_Frequently	-0.122175	-0.122365	-0.380660	0.048188	
CAEC_Sometimes	0.143999	0.143035	0.411804	0.009318	
CAEC_no	-0.061795	-0.095625	-0.106260	-0.103902	
SMOKE_no	-0.091987	-0.055499	-0.025746	-0.014320	
SMOKE_yes	0.091987	0.055499	0.025746	0.014320	
SCC_no	0.116283	0.133753	0.201906	-0.071852	
SCC_yes	-0.116283	-0.133753	-0.201906	0.071852	
CALC_Always	-0.011367	-0.000391	-0.017947	-0.017090	
CALC_Frequently	0.082566	0.045518	-0.041778	-0.018655	
CALC_Sometimes	-0.012483	0.107070	0.259599	0.082804	
CALC_no	-0.018803	-0.127828	-0.249828	-0.077072	
MTRANS_Automobile	0.604683	0.077851	-0.013619	-0.061049	
MTRANS_Bike	0.003652	0.028996	-0.021745	-0.029843	
MTRANS_Motorbike	0.020286	-0.010471	-0.037300	-0.007512	
MTRANS_Public_Transportation	-0.554448	-0.089684	0.059332	0.057966	
MTRANS_Walking	-0.063064	0.036951	-0.100805	0.013991	
NObeyesdad_Insufficient_Weight	-0.274559	-0.043538	-0.538726	0.044486	
NObeyesdad_Normal_Weight	-0.160927	-0.106700	-0.370099	-0.062828	
NObeyesdad_Obesity_Type_I	0.110675	-0.037693	0.107174	-0.194922	
NObeyesdad_Obesity_Type_II	0.250082	0.304150	0.443793	-0.021042	
NObeyesdad_Obesity_Type_III	-0.054835	-0.064443	0.558662	0.463421	
NObeyesdad_Overweight_Level_I	-0.056291	-0.059212	-0.187748	-0.115437	
NObeyesdad_Overweight_Level_II	0.168847	0.008857	-0.068593	-0.118467	

	NCP	CH2O	FAF	TUE	\
Age	-0.043944	-0.045304	-0.144938	-0.296931	
Height	0.243672	0.213376	0.294709	0.051912	
Weight	0.107469	0.200575	-0.051436	-0.071561	
FCVC	0.042216	0.068461	0.019939	-0.101135	
NCP	1.000000	0.057088	0.129504	0.036326	
CH2O	0.057088	1.000000	0.167236	0.011965	
FAF	0.129504	0.167236	1.000000	0.058562	
TUE	0.036326	0.011965	0.058562	1.000000	
Gender_Female	-0.067600	-0.107930	-0.189607	-0.017269	
Gender_Male	0.067600	0.107930	0.189607	0.017269	
family_history_with_overweight_no	-0.071370	-0.147437	0.056673	-0.022943	
family_history_with_overweight_yes	0.071370	0.147437	-0.056673	0.022943	
FAVC_no	0.007000	-0.009719	0.107995	-0.068417	
FAVC_yes	-0.007000	0.009719	-0.107995	0.068417	
CAEC_Always	0.025931	0.002843	0.022981	0.020557	
CAEC_Frequently	0.047811	-0.139352	0.033413	-0.002213	
CAEC_Sometimes	0.008587	0.052773	-0.048546	0.039202	
CAEC_no	-0.146328	0.159008	0.024298	-0.110870	
SMOKE_no	-0.007811	0.031995	-0.011216	-0.017613	

SMOKE_yes	0.007811	-0.031995	0.011216	0.017613
SCC_no	0.015624	-0.008036	-0.074221	0.010928
SCC_yes	-0.015624	0.008036	0.074221	-0.010928
CALC_Always	-0.047176	-0.000285	-0.000264	0.047995
CALC_Frequently	-0.019126	0.049442	0.054739	0.060964
CALC_Sometimes	0.099280	0.062251	-0.136138	-0.102871
CALC_no	-0.092407	-0.083272	0.118678	0.079755
MTRANS_Automobile	0.053424	-0.046237	0.002347	-0.165111
MTRANS_Bike	0.023312	0.026139	0.067129	-0.021725
MTRANS_Motorbike	0.003875	-0.000946	-0.016351	-0.067402
MTRANS_Public_Transportation	-0.056849	0.037504	-0.048751	0.143597
MTRANS_Walking	0.006502	0.008307	0.108927	0.073404
NObeyesdad_Insufficient_Weight	0.113111	-0.085809	0.108464	0.114718
NObeyesdad_Normal_Weight	0.027052	-0.102168	0.110591	0.011789
NObeyesdad_Obesity_Type_I	-0.145691	0.075940	-0.012367	0.013847
NObeyesdad_Obesity_Type_II	0.030653	-0.086071	-0.018291	-0.094833
NObeyesdad_Obesity_Type_III	0.172090	0.139303	-0.172988	-0.037240
NObeyesdad_Overweight_Level_I	-0.093069	0.033025	0.021820	-0.029415
NObeyesdad_Overweight_Level_II	-0.097527	0.011150	-0.024508	0.025833

	Gender_Female	Gender_Male	...	\
Age	-0.048394	0.048394	...	
Height	-0.618466	0.618466	...	
Weight	-0.161668	0.161668	...	
FCVC	0.274505	-0.274505	...	
NCP	-0.067600	0.067600	...	
CH2O	-0.107930	0.107930	...	
FAF	-0.189607	0.189607	...	
TUE	-0.017269	0.017269	...	
Gender_Female	1.000000	-1.000000	...	
Gender_Male	-1.000000	1.000000	...	
family_history_with_overweight_no	0.102512	-0.102512	...	
family_history_with_overweight_yes	-0.102512	0.102512	...	
FAVC_no	0.064934	-0.064934	...	
FAVC_yes	-0.064934	0.064934	...	
CAEC_Always	-0.019296	0.019296	...	
CAEC_Frequently	0.123224	-0.123224	...	
CAEC_Sometimes	-0.071790	0.071790	...	
CAEC_no	-0.062930	0.062930	...	
SMOKE_no	0.044698	-0.044698	...	
SMOKE_yes	-0.044698	0.044698	...	
SCC_no	-0.102633	0.102633	...	
SCC_yes	0.102633	-0.102633	...	
CALC_Always	-0.021514	0.021514	...	
CALC_Frequently	-0.034848	0.034848	...	
CALC_Sometimes	0.037694	-0.037694	...	
CALC_no	-0.024163	0.024163	...	
MTRANS_Automobile	-0.137560	0.137560	...	
MTRANS_Bike	-0.057001	0.057001	...	
MTRANS_Motorbike	-0.045203	0.045203	...	
MTRANS_Public_Transportation	0.160184	-0.160184	...	
MTRANS_Walking	-0.039317	0.039317	...	
NObeyesdad_Insufficient_Weight	0.109192	-0.109192	...	
NObeyesdad_Normal_Weight	-0.002213	0.002213	...	
NObeyesdad_Obesity_Type_I	-0.044334	0.044334	...	
NObeyesdad_Obesity_Type_II	-0.394418	0.394418	...	
NObeyesdad_Obesity_Type_III	0.428249	-0.428249	...	
NObeyesdad_Overweight_Level_I	0.004726	-0.004726	...	
NObeyesdad_Overweight_Level_II	-0.110873	0.110873	...	

	MTRANS_Motorbike
Age	0.020286
Height	-0.010471
Weight	-0.037300
FCVC	-0.007512
NCP	0.003875
CH2O	-0.000946
FAF	-0.016351
TUE	-0.067402
Gender_Female	-0.045203
Gender_Male	0.045203
family_history_with_overweight_no	0.051011
family_history_with_overweight_yes	-0.051011
FAVC_no	0.035401
FAVC_yes	-0.035401
CAEC_Always	0.030441
CAEC_Frequently	0.077219
CAEC_Sometimes	-0.074596
CAEC_no	-0.011388
SMOKE_no	-0.035496
SMOKE_yes	0.035496
SCC_no	-0.047362
SCC_yes	0.047362
CALC_Always	-0.001576
CALC_Frequently	-0.013403
CALC_Sometimes	-0.018109
CALC_no	0.023921
MTRANS_Automobile	-0.038043
MTRANS_Bike	-0.004175
MTRANS_Motorbike	1.000000
MTRANS_Public_Transportation	-0.124844
MTRANS_Walking	-0.011947
NObeyesdad_Insufficient_Weight	-0.027834
NObeyesdad_Normal_Weight	0.086472
NObeyesdad_Obesity_Type_I	0.020693
NObeyesdad_Obesity_Type_II	-0.029285
NObeyesdad_Obesity_Type_III	-0.030817
NObeyesdad_Overweight_Level_I	-0.009769
NObeyesdad_Overweight_Level_II	-0.009769

	MTRANS_Public_Transportation
Age	-0.554448
Height	-0.089684
Weight	0.059332
FCVC	0.057966
NCP	-0.056849
CH2O	0.037504
FAF	-0.048751
TUE	0.143597
Gender_Female	0.160184
Gender_Male	-0.160184
family_history_with_overweight_no	0.058927
family_history_with_overweight_yes	-0.058927
FAVC_no	-0.028539
FAVC_yes	0.028539
CAEC_Always	-0.046534
CAEC_Frequently	0.068101
CAEC_Sometimes	-0.062029
CAEC_no	0.055664
SMOKE_no	0.022409

SMOKE_yes	-0.022409
SCC_no	-0.011255
SCC_yes	0.011255
CALC_Always	-0.037553
CALC_Frequently	-0.087755
CALC_Sometimes	0.097994
CALC_no	-0.064794
MTRANS_Automobile	-0.906717
MTRANS_Bike	-0.099496
MTRANS_Motorbike	-0.124844
MTRANS_Public_Transportation	1.000000
MTRANS_Walking	-0.284754
NObeyesdad_Insufficient_Weight	0.053503
NObeyesdad_Normal_Weight	-0.047169
NObeyesdad_Obesity_Type_I	-0.078320
NObeyesdad_Obesity_Type_II	-0.069997
NObeyesdad_Obesity_Type_III	0.243819
NObeyesdad_Overweight_Level_I	-0.016027
NObeyesdad_Overweight_Level_II	-0.088970

	MTRANS_Walking \
Age	-0.063064
Height	0.036951
Weight	-0.100805
FCVC	0.013991
NCP	0.006502
CH2O	0.008307
FAF	0.108927
TUE	0.073404
Gender_Female	-0.039317
Gender_Male	0.039317
family_history_with_overweight_no	0.067076
family_history_with_overweight_yes	-0.067076
FAVC_no	0.179473
FAVC_yes	-0.179473
CAEC_Always	0.086561
CAEC_Frequently	0.042381
CAEC_Sometimes	-0.078208
CAEC_no	0.012423
SMOKE_no	-0.017184
SMOKE_yes	0.017184
SCC_no	-0.048860
SCC_yes	0.048860
CALC_Always	0.131877
CALC_Frequently	0.018819
CALC_Sometimes	-0.050946
CALC_no	0.038811
MTRANS_Automobile	-0.086772
MTRANS_Bike	-0.009522
MTRANS_Motorbike	-0.011947
MTRANS_Public_Transportation	-0.284754
MTRANS_Walking	1.000000
NObeyesdad_Insufficient_Weight	-0.010695
NObeyesdad_Normal_Weight	0.209742
NObeyesdad_Obesity_Type_I	-0.057885
NObeyesdad_Obesity_Type_II	-0.058318
NObeyesdad_Obesity_Type_III	-0.070291
NObeyesdad_Overweight_Level_I	0.011192
NObeyesdad_Overweight_Level_II	-0.014498

	NObeyesdad_Insufficient_Weight
Age	-0.274559
Height	-0.043538
Weight	-0.538726
FCVC	0.044486
NCP	0.113111
CH2O	-0.085809
FAF	0.108464
TUE	0.114718
Gender_Female	0.109192
Gender_Male	-0.109192
family_history_with_overweight_no	0.352946
family_history_with_overweight_yes	-0.352946
FAVC_no	0.085781
FAVC_yes	-0.085781
CAEC_Always	-0.043642
CAEC_Frequently	0.398627
CAEC_Sometimes	-0.310974
CAEC_no	-0.032887
SMOKE_no	0.046214
SMOKE_yes	-0.046214
SCC_no	-0.065357
SCC_yes	0.065357
CALC_Always	-0.008372
CALC_Frequently	-0.063326
CALC_Sometimes	-0.079359
CALC_no	0.106686
MTRANS_Automobile	-0.044232
MTRANS_Bike	-0.022183
MTRANS_Motorbike	-0.027834
MTRANS_Public_Transportation	0.053503
MTRANS_Walking	-0.010695
NObeyesdad_Insufficient_Weight	1.000000
NObeyesdad_Normal_Weight	-0.152553
NObeyesdad_Obesity_Type_I	-0.171748
NObeyesdad_Obesity_Type_II	-0.155616
NObeyesdad_Obesity_Type_III	-0.163758
NObeyesdad_Overweight_Level_I	-0.153475
NObeyesdad_Overweight_Level_II	-0.153475

	NObeyesdad_Normal_Weight
Age	-0.160927
Height	-0.106700
Weight	-0.370099
FCVC	-0.062828
NCP	0.027052
CH2O	-0.102168
FAF	0.110591
TUE	0.011789
Gender_Female	-0.002213
Gender_Male	0.002213
family_history_with_overweight_no	0.285109
family_history_with_overweight_yes	-0.285109
FAVC_no	0.197164
FAVC_yes	-0.197164
CAEC_Always	0.245545
CAEC_Frequently	0.217346
CAEC_Sometimes	-0.302269
CAEC_no	0.027602
SMOKE_no	-0.067897

SMOKE_yes	0.067897
SCC_no	-0.112432
SCC_yes	0.112432
CALC_Always	0.054882
CALC_Frequently	0.065482
CALC_Sometimes	-0.086219
CALC_no	0.060544
MTRANS_Automobile	-0.057491
MTRANS_Bike	0.073286
MTRANS_Motorbike	0.086472
MTRANS_Public_Transportation	-0.047169
MTRANS_Walking	0.209742
NObeyesdad_Insufficient_Weight	-0.152553
NObeyesdad_Normal_Weight	1.000000
NObeyesdad_Obesity_Type_I	-0.177144
NObeyesdad_Obesity_Type_II	-0.160505
NObeyesdad_Obesity_Type_III	-0.168904
NObeyesdad_Overweight_Level_I	-0.158297
NObeyesdad_Overweight_Level_II	-0.158297

	NObeyesdad_Obesity_Type_I \
Age	0.110675
Height	-0.037693
Weight	0.107174
FCVC	-0.194922
NCP	-0.145691
CH2O	0.075940
FAF	-0.012367
TUE	0.013847
Gender_Female	-0.044334
Gender_Male	0.044334
family_history_with_overweight_no	-0.187851
family_history_with_overweight_yes	0.187851
FAVC_no	-0.118122
FAVC_yes	0.118122
CAEC_Always	-0.022872
CAEC_Frequently	-0.136733
CAEC_Sometimes	0.153046
CAEC_no	-0.061980
SMOKE_no	0.011720
SMOKE_yes	-0.011720
SCC_no	0.085262
SCC_yes	-0.085262
CALC_Always	-0.009722
CALC_Frequently	0.016776
CALC_Sometimes	-0.164128
CALC_no	0.162704
MTRANS_Automobile	0.105077
MTRANS_Bike	-0.025759
MTRANS_Motorbike	0.020693
MTRANS_Public_Transportation	-0.078320
MTRANS_Walking	-0.057885
NObeyesdad_Insufficient_Weight	-0.171748
NObeyesdad_Normal_Weight	-0.177144
NObeyesdad_Obesity_Type_I	1.000000
NObeyesdad_Obesity_Type_II	-0.180699
NObeyesdad_Obesity_Type_III	-0.190155
NObeyesdad_Overweight_Level_I	-0.178214
NObeyesdad_Overweight_Level_II	-0.178214

	NObeyesdad_Obesity_Type_II
Age	0.250082
Height	0.304150
Weight	0.443793
FCVC	-0.021042
NCP	0.030653
CH2O	-0.086071
FAF	-0.018291
TUE	-0.094833
Gender_Female	-0.394418
Gender_Male	0.394418
family_history_with_overweight_no	-0.187576
family_history_with_overweight_yes	0.187576
FAVC_no	-0.116843
FAVC_yes	0.116843
CAEC_Always	-0.047518
CAEC_Frequently	-0.141324
CAEC_Sometimes	0.164432
CAEC_no	-0.054793
SMOKE_no	-0.084014
SMOKE_yes	0.084014
SCC_no	0.081781
SCC_yes	-0.081781
CALC_Always	-0.008809
CALC_Frequently	-0.059718
CALC_Sometimes	0.077544
CALC_no	-0.056052
MTRANS_Automobile	0.101569
MTRANS_Bike	0.000359
MTRANS_Motorbike	-0.029285
MTRANS_Public_Transportation	-0.069997
MTRANS_Walking	-0.058318
NObeyesdad_Insufficient_Weight	-0.155616
NObeyesdad_Normal_Weight	-0.160505
NObeyesdad_Obesity_Type_I	-0.180699
NObeyesdad_Obesity_Type_II	1.000000
NObeyesdad_Obesity_Type_III	-0.172294
NObeyesdad_Overweight_Level_I	-0.161474
NObeyesdad_Overweight_Level_II	-0.161474

	NObeyesdad_Obesity_Type_III
Age	-0.054835
Height	-0.064443
Weight	0.558662
FCVC	0.463421
NCP	0.172090
CH2O	0.139303
FAF	-0.172988
TUE	-0.037240
Gender_Female	0.428249
Gender_Male	-0.428249
family_history_with_overweight_no	-0.201104
family_history_with_overweight_yes	0.201104
FAVC_no	-0.150187
FAVC_yes	0.150187
CAEC_Always	-0.068332
CAEC_Frequently	-0.149094
CAEC_Sometimes	0.184978
CAEC_no	-0.066998
SMOKE_no	0.052926

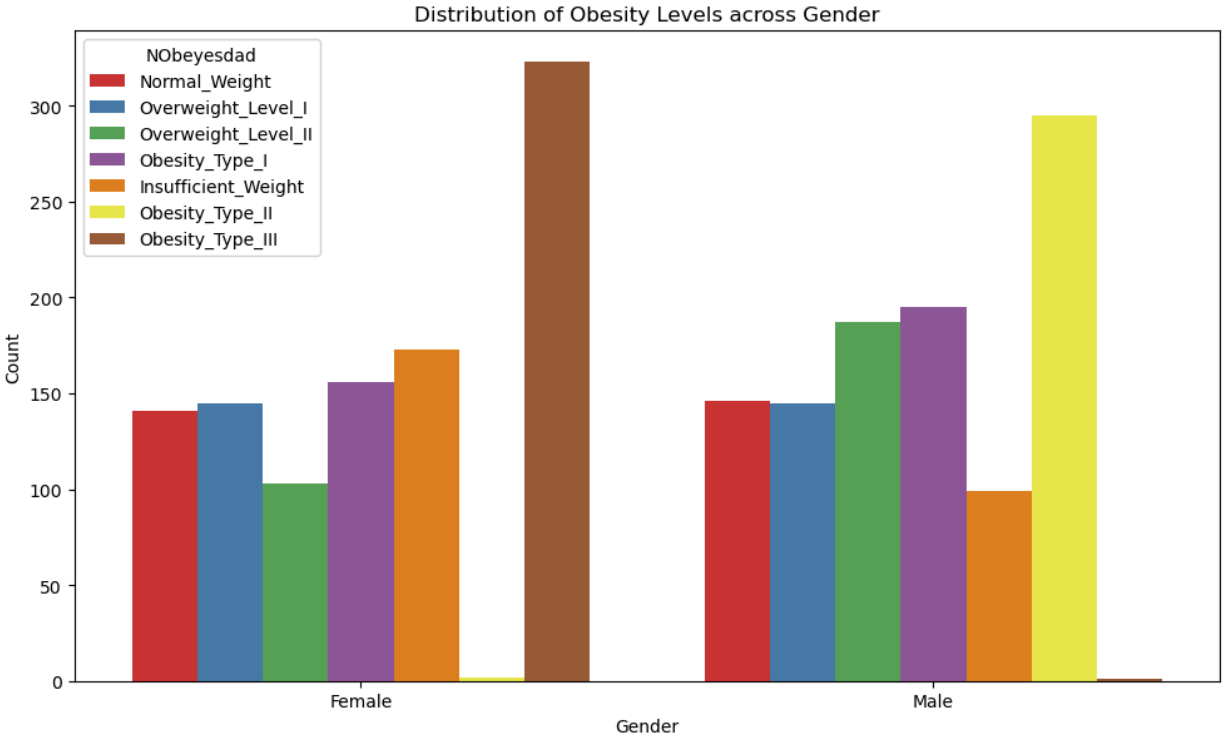
SMOKE_yes	-0.052926
SCC_no	0.092941
SCC_yes	-0.092941
CALC_Always	-0.009270
CALC_Frequently	-0.078857
CALC_Sometimes	0.300342
CALC_no	-0.277687
MTRANS_Automobile	-0.220630
MTRANS_Bike	-0.024560
MTRANS_Motorbike	-0.030817
MTRANS_Public_Transportation	0.243819
MTRANS_Walking	-0.070291
NObeyesdad_Insufficient_Weight	-0.163758
NObeyesdad_Normal_Weight	-0.168904
NObeyesdad_Obesity_Type_I	-0.190155
NObeyesdad_Obesity_Type_II	-0.172294
NObeyesdad_Obesity_Type_III	1.000000
NObeyesdad_Overweight_Level_I	-0.169924
NObeyesdad_Overweight_Level_II	-0.169924

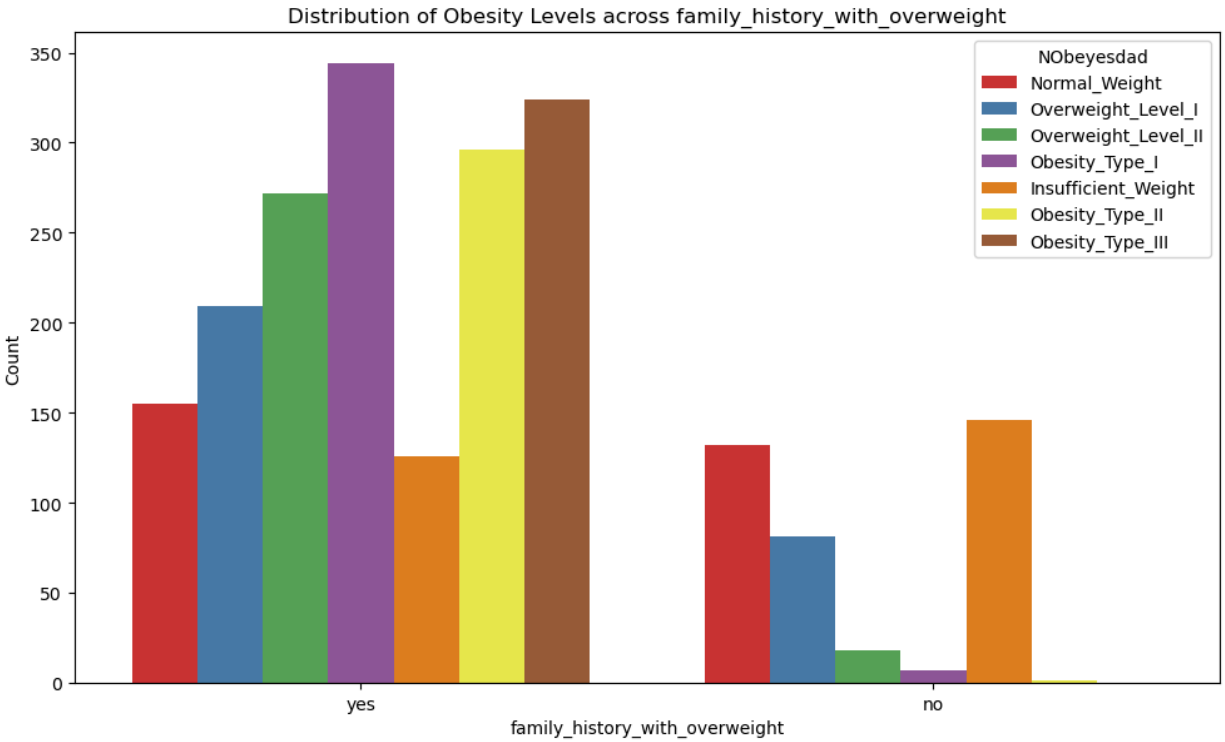
	NObeyesdad_Overweight_Level_I \
Age	-0.056291
Height	-0.059212
Weight	-0.187748
FCVC	-0.115437
NCP	-0.093069
CH2O	0.033025
FAF	0.021820
TUE	-0.029415
Gender_Female	0.004726
Gender_Male	-0.004726
family_history_with_overweight_no	0.100173
family_history_with_overweight_yes	-0.100173
FAVC_no	-0.050082
FAVC_yes	0.050082
CAEC_Always	-0.020062
CAEC_Frequently	-0.083126
CAEC_Sometimes	-0.024043
CAEC_no	0.250887
SMOKE_no	0.029326
SMOKE_yes	-0.029326
SCC_no	-0.157274
SCC_yes	0.157274
CALC_Always	-0.008688
CALC_Frequently	0.049061
CALC_Sometimes	0.091855
CALC_no	-0.113169
MTRANS_Automobile	0.010757
MTRANS_Bike	0.024855
MTRANS_Motorbike	-0.009769
MTRANS_Public_Transportation	-0.016027
MTRANS_Walking	0.011192
NObeyesdad_Insufficient_Weight	-0.153475
NObeyesdad_Normal_Weight	-0.158297
NObeyesdad_Obesity_Type_I	-0.178214
NObeyesdad_Obesity_Type_II	-0.161474
NObeyesdad_Obesity_Type_III	-0.169924
NObeyesdad_Overweight_Level_I	1.000000
NObeyesdad_Overweight_Level_II	-0.159253

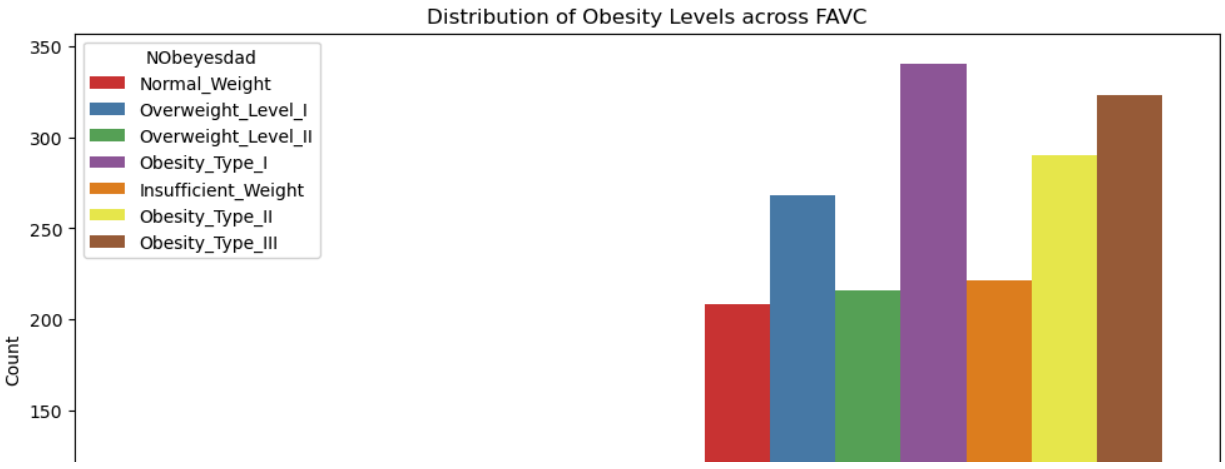
	NObeyesdad_Overweight_Level_II
Age	0.168847
Height	0.008857
Weight	-0.068593
FCVC	-0.118467
NCP	-0.097527
CH2O	0.011150
FAF	-0.024508
TUE	0.025833
Gender_Female	-0.110873
Gender_Male	0.110873
family_history_with_overweight_no	-0.124331
family_history_with_overweight_yes	0.124331
FAVC_no	0.173326
FAVC_yes	-0.173326
CAEC_Always	-0.037654
CAEC_Frequently	-0.074487
CAEC_Sometimes	0.102344
CAEC_no	-0.053829
SMOKE_no	0.010061
SMOKE_yes	-0.010061
SCC_no	0.060686
SCC_yes	-0.060686
CALC_Always	-0.008688
CALC_Frequently	0.072117
CALC_Sometimes	-0.144068
CALC_no	0.120459
MTRANS_Automobile	0.104311
MTRANS_Bike	-0.023018
MTRANS_Motorbike	-0.009769
MTRANS_Public_Transportation	-0.088970
MTRANS_Walking	-0.014498
NObeyesdad_Insufficient_Weight	-0.153475
NObeyesdad_Normal_Weight	-0.158297
NObeyesdad_Obesity_Type_I	-0.178214
NObeyesdad_Obesity_Type_II	-0.161474
NObeyesdad_Obesity_Type_III	-0.169924
NObeyesdad_Overweight_Level_I	-0.159253
NObeyesdad_Overweight_Level_II	1.000000

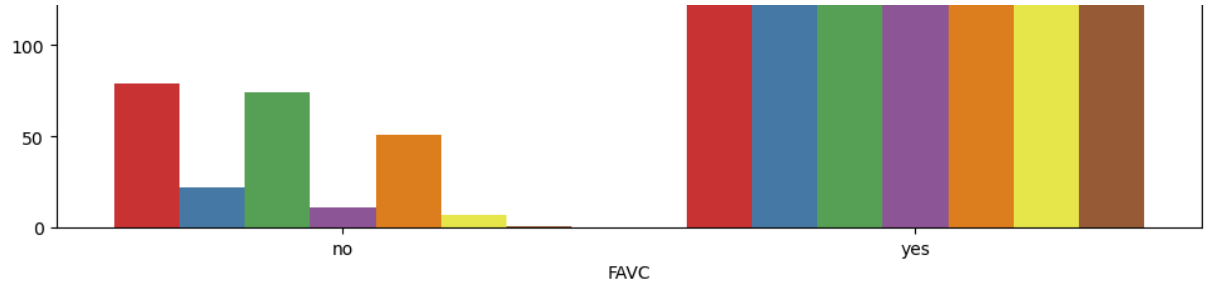
[38 rows x 38 columns]

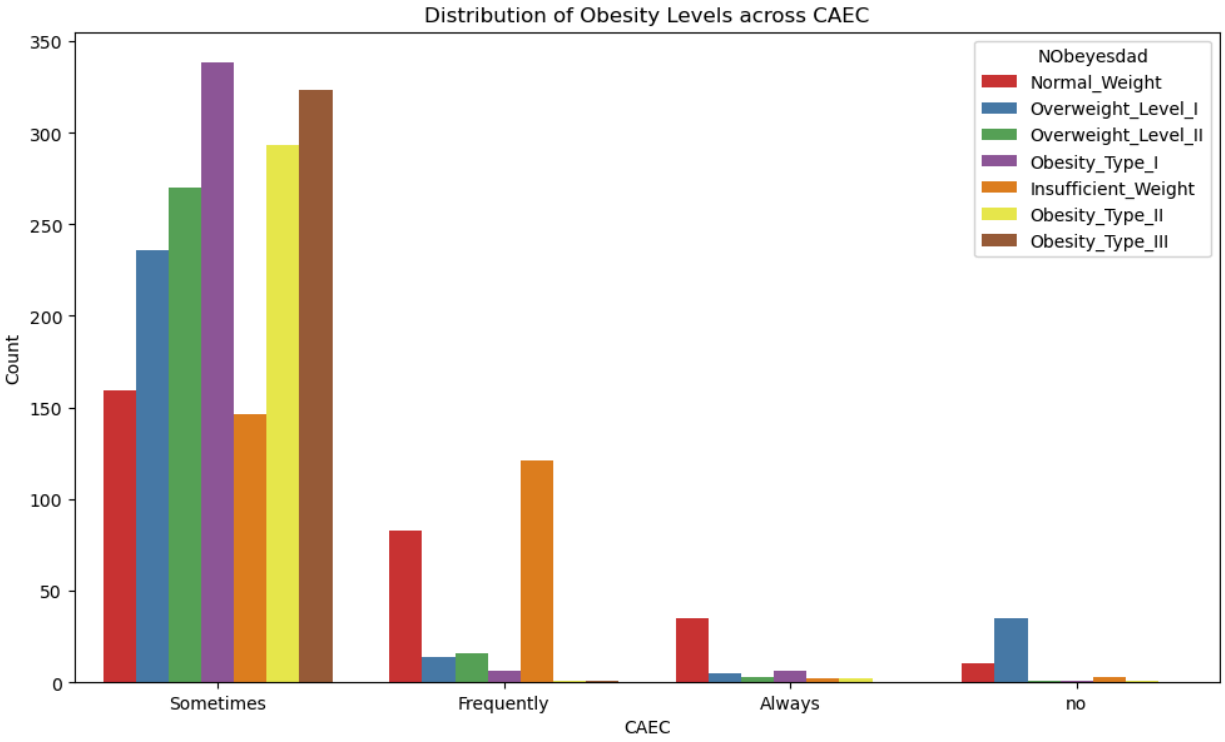
In [21]:
```
categorical_variables = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', '
fig, axes = plt.subplots(nrows=len(categorical_variables), ncols=1, figsize=(10, 6*ler
for i, cat_var in enumerate(categorical_variables):
    sns.countplot(x=cat_var, hue='NObeyesdad', data=df, ax=axes[i], palette='Set1')
    axes[i].set_title(f'Distribution of Obesity Levels across {cat_var}')
    axes[i].set_xlabel(cat_var)
    axes[i].set_ylabel('Count')

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

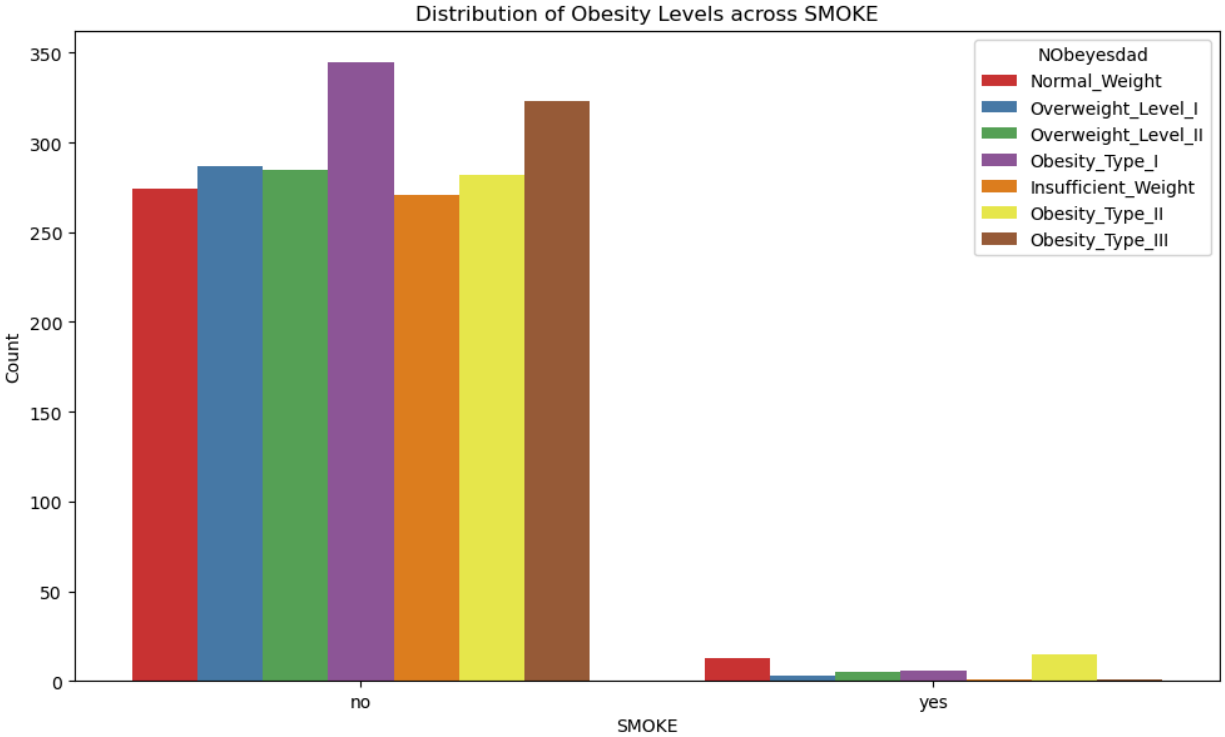

Distribution of Obesity Levels across Gender
NObeyesdad
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Insufficient_Weight
Obesity_Type_II
Obesity_Type_III
Count
0
50
100
150
200
250
300
Female
Male
Gender


Distribution of Obesity Levels across family_history_with_overweight
NObeyesdad
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Insufficient_Weight
Obesity_Type_II
Obesity_Type_III
Count
0
50
100
150
200
250
300
350
yes
no
family_history_with_overweight


Distribution of Obesity Levels across FAVC
NObeyesdad
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Insufficient_Weight
Obesity_Type_II
Obesity_Type_III
Count
150
200
250
300
350


100
50
0
no
yes
FAVC


Distribution of Obesity Levels across CAEC
NObeyesdad
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Insufficient_Weight
Obesity_Type_II
Obesity_Type_III
Count
350
300
250
200
150
100
50
0
Sometimes
Frequently
Always
no
CAEC


Distribution of Obesity Levels across SMOKE
NObeyesdad
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Insufficient_Weight
Obesity_Type_II
Obesity_Type_III
Count
350
300
250
200
150
100
50
0
no
yes
SMOKE


Distribution of Obesity Levels across SCC
NObeyesdad
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
350
300

Encoding Categorical Variables

In [22]:
```
categorical_vars = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE
obesity_data_encoded = pd.get_dummies(df, columns=categorical_vars, drop_first=True)
```

In [23]:
```
#Label Encoding
label_encoder = LabelEncoder()
obesity_data_encoded['NObeyesdad'] = label_encoder.fit_transform(obesity_data_encoded[
```

Normalizing Numerical Features

In [24]:
```
numerical_features = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']
```

In [25]:
```
scaler = MinMaxScaler()
obesity_data_encoded[numerical_features] = scaler.fit_transform(obesity_data_encoded[r
```

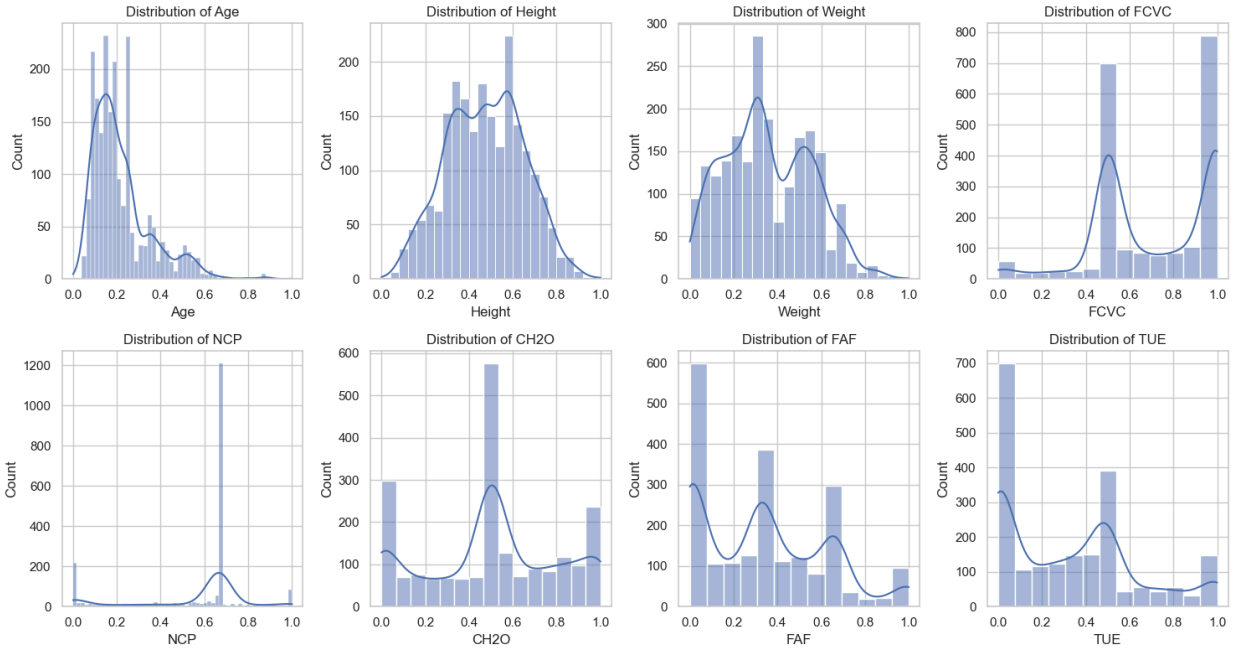
In [26]:
```
obesity_data_encoded.head()
```

Out[26]:

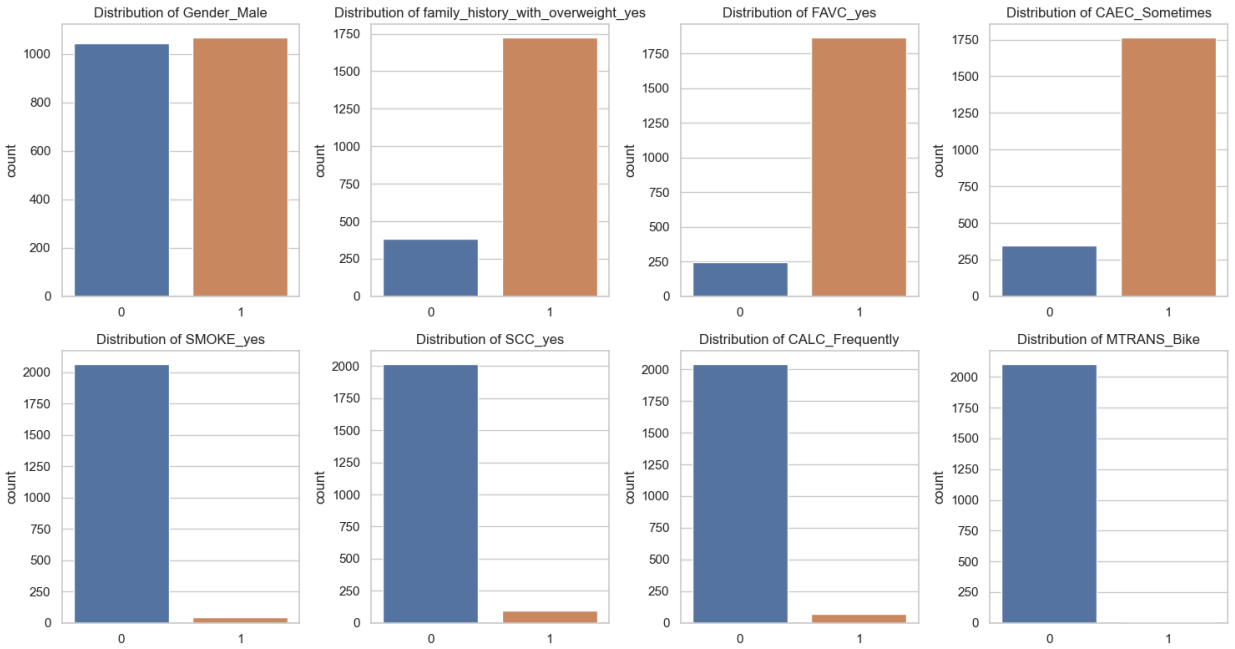
	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE	NObeyesdad	Gender_Male	...
0	0.148936	0.320755	0.186567	0.5	0.666667	0.5	0.000000	0.5	1	0	...
1	0.148936	0.132075	0.126866	1.0	0.666667	1.0	1.000000	0.0	1	0	...
2	0.191489	0.660377	0.283582	0.5	0.666667	0.5	0.666667	0.5	1	1	...
3	0.276596	0.660377	0.358209	1.0	0.666667	0.5	0.666667	0.0	5	1	...
4	0.170213	0.622642	0.379104	0.5	0.000000	0.5	0.000000	0.0	6	1	...

5 rows × 24 columns

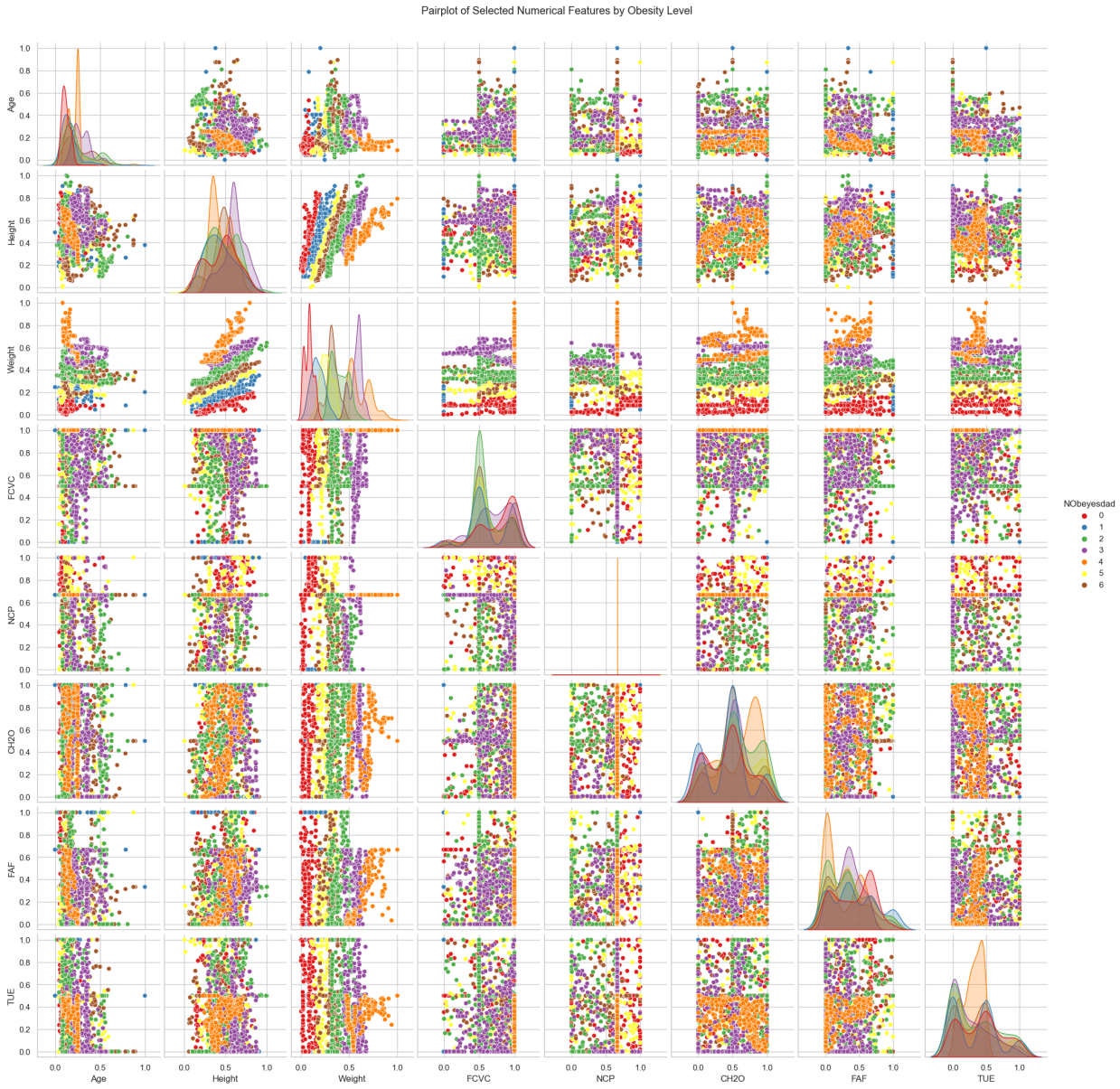
In [27]:
```
numerical_features = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']
sns.set(style="whitegrid")
plt.figure(figsize=(15, 8))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(2, 4, i)
    sns.histplot(data=obesity_data_encoded, x=feature, kde=True)
    plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()
```



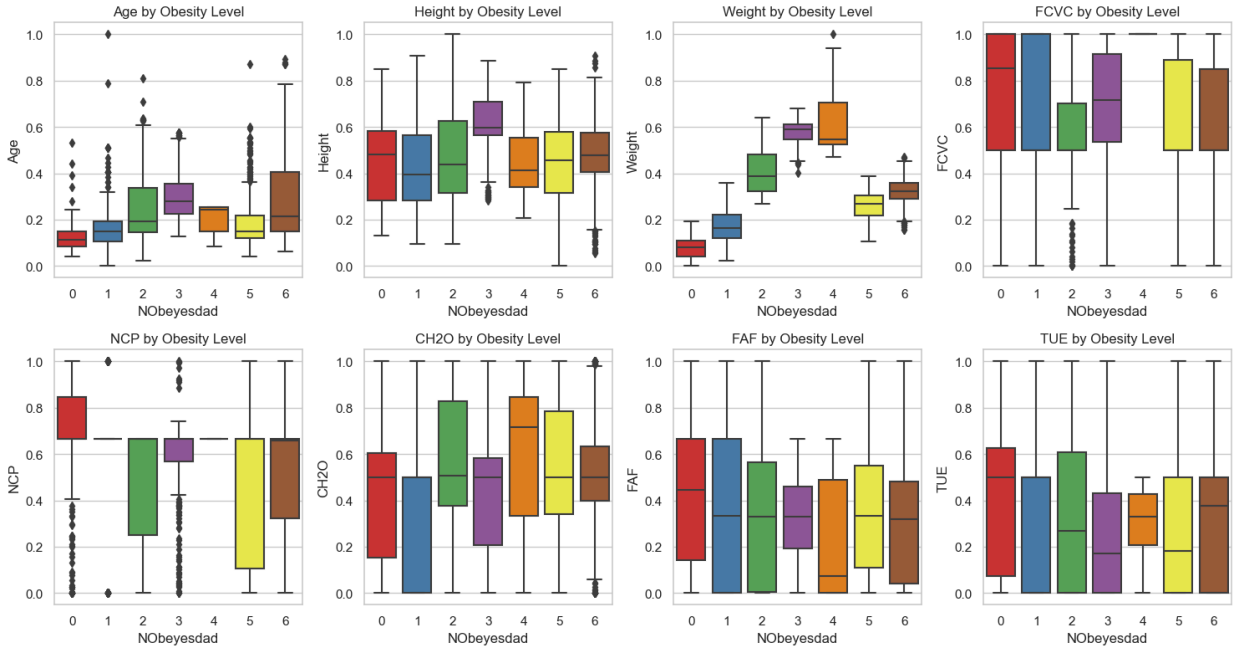
In [28]:
```
categorical_features = ['Gender_Male', 'family_history_with_overweight_yes', 'FAVC_yes
plt.figure(figsize=(15, 8))
for i, feature in enumerate(categorical_features, 1):
    plt.subplot(2, 4, i)
    sns.countplot(data=obesity_data_encoded, x=feature)
    plt.title(f'Distribution of {feature}')
    plt.xlabel('')
plt.tight_layout()
plt.show()
```



In [29]:
```
sns.pairplot(data=obesity_data_encoded, vars=numerical_features, hue='NObeyesdad', pal
plt.suptitle('Pairplot of Selected Numerical Features by Obesity Level', y=1.02)
plt.show()
```

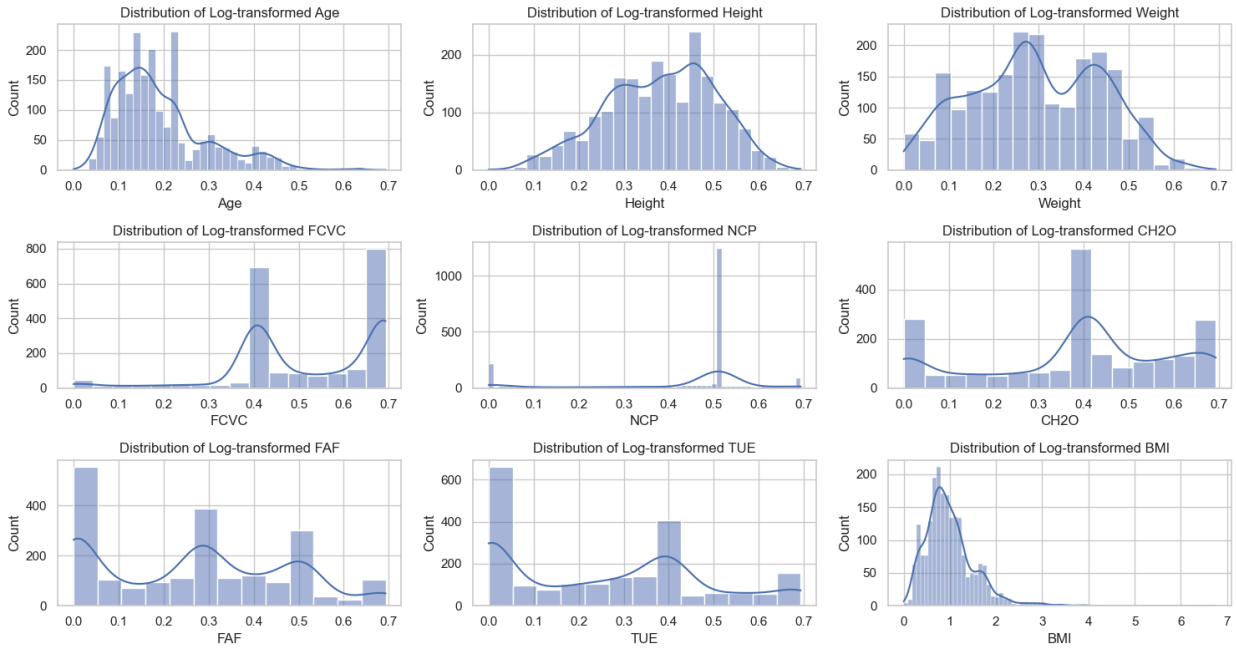


In [30]:
```
plt.figure(figsize=(15, 8))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(2, 4, i)
    sns.boxplot(x='NObeyesdad', y=feature, data=obesity_data_encoded, palette='Set1')
    plt.title(f'{feature} by Obesity Level')
plt.tight_layout()
plt.show()
```



Feature Engineering

In [31]:
```
# Feature Creation: Calculate BMI

obesity_data_encoded['BMI'] = obesity_data_encoded['Weight'] / (obesity_data_encoded['
numerical_features = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE',
obesity_data_encoded[numerical_features] = np.log1p(obesity_data_encoded[numerical_fea
plt.figure(figsize=(15, 8))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(3, 3, i)
    sns.histplot(data=obesity_data_encoded, x=feature, kde=True)
    plt.title(f'Distribution of Log-transformed {feature}')
plt.tight_layout()
plt.show()
```



In [32]:
```
X = obesity_data_encoded.drop('NObeyesdad', axis=1)
y = obesity_data_encoded['NObeyesdad']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)
X_train.dropna(inplace=True)
X_test.dropna(inplace=True)


y_train = y_train.loc[X_train.index]
y_test = y_test.loc[X_test.index]


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Basic Model Testing

Logistic Regression

In [33]:
```
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train_scaled, y_train)
```

Out[33]:

LogisticRegression

```
LogisticRegression(max_iter=1000)
```

In [34]:
```
logistic_pred = logistic_model.predict(X_test_scaled)

print("\nLogistic Regression:")
print("Accuracy:", accuracy_score(y_test, logistic_pred))
print("\nClassification Report:")
print(classification_report(y_test, logistic_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, logistic_pred))
```

```
Logistic Regression:
Accuracy: 0.8676122931442081

Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94        56
           1       0.91      0.68      0.78        62
           2       0.93      0.85      0.89        78
           3       0.85      0.97      0.90        58
           4       0.98      1.00      0.99        63
           5       0.74      0.75      0.74        56
           6       0.75      0.84      0.79        50

    accuracy                           0.87       423
   macro avg       0.86      0.87      0.86       423
weighted avg       0.87      0.87      0.87       423


Confusion Matrix:
[[56  0  0  0  0  0  0]
 [ 7 42  0  0  0 10  3]
 [ 0  0 66 10  0  0  2]
 [ 0  0  2 56  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0  4  0  0  1 42  9]
 [ 0  0  3  0  0  5 42]]
```

K Nearest Neighbours (KNN)

In [35]:

```
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_scaled, y_train)
knn_pred = knn_model.predict(X_test_scaled)

print("K-Nearest Neighbors:")
print("Accuracy:", accuracy_score(y_test, knn_pred))
print("\nClassification Report:")
print(classification_report(y_test, knn_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, knn_pred))
```

```
K-Nearest Neighbors:
Accuracy: 0.8085106382978723

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.86      0.81        56
           1       0.57      0.52      0.54        62
           2       0.84      0.87      0.86        78
           3       0.90      0.97      0.93        58
           4       0.98      1.00      0.99        63
           5       0.76      0.68      0.72        56
           6       0.77      0.74      0.76        50

    accuracy                           0.81       423
   macro avg       0.80      0.80      0.80       423
weighted avg       0.80      0.81      0.80       423


Confusion Matrix:
[[48  6  0  0  0  2  0]
 [13 32  6  0  0  6  5]
 [ 0  2 68  3  0  1  4]
 [ 0  1  0 56  0  0  1]
 [ 0  0  0  0 63  0  0]
 [ 1 11  4  1  0 38  1]
 [ 0  4  3  2  1  3 37]]
```

Support Vector Machines

In [36]:
```
svm_model = SVC()
svm_model.fit(X_train_scaled, y_train)
```

Out[36]:

SVC

SVC()

In [37]:
```
svm_pred = svm_model.predict(X_test_scaled)

print("\nSupport Vector Machine:")
print("Accuracy:", accuracy_score(y_test, svm_pred))
print("\nClassification Report:")
print(classification_report(y_test, svm_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, svm_pred))
```

```
Support Vector Machine:
Accuracy: 0.8865248226950354

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        56
           1       0.74      0.87      0.80        62
           2       0.94      0.86      0.90        78
           3       0.92      0.98      0.95        58
           4       1.00      1.00      1.00        63
           5       0.76      0.70      0.73        56
           6       0.87      0.80      0.83        50

    accuracy                           0.89       423
   macro avg       0.89      0.88      0.88       423
weighted avg       0.89      0.89      0.89       423


Confusion Matrix:
[[55  1  0  0  0  0  0]
 [ 2 54  1  0  0  4  1]
 [ 0  3 67  5  0  1  2]
 [ 0  0  1 57  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0 14  0  0  0 39  3]
 [ 0  1  2  0  0  7 40]]
```

Gaussian Naive Bayes Classifier

In [38]:
```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

Out[38]:
```
GaussianNB()
```

In [39]:
```
nb_pred = nb_model.predict(X_test)

print("\nNaive Bayes:")
print("Accuracy:", accuracy_score(y_test, nb_pred))
print("\nClassification Report:")
print(classification_report(y_test, nb_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, nb_pred))
```

```
Naive Bayes:
Accuracy: 0.5721040189125296

Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.95      0.70        56
           1       0.50      0.08      0.14        62
           2       0.38      0.49      0.43        78
           3       0.59      0.95      0.73        58
           4       1.00      1.00      1.00        63
           5       0.55      0.20      0.29        56
           6       0.40      0.34      0.37        50

    accuracy                           0.57       423
   macro avg       0.57      0.57      0.52       423
weighted avg       0.57      0.57      0.52       423


Confusion Matrix:
[[53  0  2  0  0  1  0]
 [37  5  6  0  0  6  8]
 [ 0  1 38 32  0  2  5]
 [ 0  0  2 55  0  0  1]
 [ 0  0  0  0 63  0  0]
 [ 5  3 25  1  0 11 11]
 [ 0  1 27  5  0  0 17]]
```

Logistic Regression: Accuracy: 0.868 Precision: Highest for class 4 (0.98), lowest for class 5 (0.74) Recall: Highest for class 0 (1.00), lowest for class 5 (0.75) F1-score: Ranges from 0.74 (class 5) to 0.99 (class 4) Insight: Logistic Regression performs reasonably well across all metrics, with high accuracy and good precision-recall balance for most classes.

K-Nearest Neighbors (KNN): Accuracy: 0.809 Precision: Highest for class 4 (0.98), lowest for class 1 (0.57) Recall: Highest for class 4 (1.00), lowest for class 5 (0.68) F1-score: Ranges from 0.54 (class 1) to 0.99 (class 4) Insight: KNN shows lower performance compared to Logistic Regression, especially in precision for some classes, but maintains good recall in general.

Support Vector Machine (SVM): Accuracy: 0.887 Precision: Highest for class 4 (1.00), lowest for class 5 (0.76) Recall: Highest for class 4 (1.00), lowest for class 5 (0.70) F1-score: Ranges from 0.73 (class 5) to 1.00 (class 4) Insight: SVM performs competitively with Logistic Regression, with high accuracy and excellent precision-recall for most classes.

Gaussian Naive Bayes (NB): Accuracy: 0.572 Precision: Highest for class 4 (1.00), lowest for class 1 (0.50) Recall: Highest for class 4 (1.00), lowest for class 5 (0.20) F1-score: Ranges from 0.14 (class 1) to 1.00 (class 4) Insight: NB shows significantly lower accuracy and F1-scores compared to other models, indicating poorer performance in classification.

Comparison and Conclusion: Logistic Regression and SVM consistently show the highest accuracy and F1-scores across most classes, indicating robust performance. KNN performs decently but lags behind in precision for some classes compared to Logistic Regression and SVM. Gaussian NB performs the poorest among the models tested, with significantly lower accuracy and F1-scores across all classes.

Based on these results, Logistic Regression or SVM can be preferred for this classification task due to their higher accuracy and balanced precision-recall metrics across classes. Further tuning of models on the train and test csv with tree based and boosting classifiers can be helpful for this multiclass classification task.

In [40]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from imblearn.over_sampling import SMOTE

# Function to get variable types
def get_variable_types(dataframe):
    continuous_vars = []
    categorical_vars = []
    for column in dataframe.columns:
        if dataframe[column].dtype == 'object':
            categorical_vars.append(column)
        else:
            continuous_vars.append(column)
    return continuous_vars, categorical_vars

# Copy data and get variable types
df2 = df1.copy()
continuous_vars, categorical_vars = get_variable_types(df_train)
categorical_vars.remove('NObeyesdad')

# Prepare train and test datasets
train = pd.concat([df_train, df]).drop(['id'], axis=1).drop_duplicates()
test = df_test.drop(['id'], axis=1)

# One-hot encode categorical variables
train = pd.get_dummies(train, columns=categorical_vars, drop_first=True)
test = pd.get_dummies(test, columns=categorical_vars, drop_first=True)

# Ensure the test set has the same dummy variables as the train set
test = test.reindex(columns=train.columns, fill_value=0)
```

Check for calss imbalance

In [41]:
```
# Separate features and target variable
X = train.drop(['NObeyesdad'], axis=1)
y = train['NObeyesdad']

# Splitting data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42

# Check for class imbalance
class_distribution = y_train.value_counts()
print("Class Distribution:")
print(class_distribution)

# Check the imbalance ratio (ratio of the majority class to the minority class)
imbalance_ratio = class_distribution.max() / class_distribution.min()
print("Imbalance Ratio:", imbalance_ratio)
```

```
# If the imbalance ratio is greater than a threshold (e.g., 2), apply SMOTE
threshold = 2
if imbalance_ratio > threshold:
    smote = SMOTE(sampling_strategy='auto', random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
    resampled_class_distribution = y_train_resampled.value_counts()
    print("\nClass Distribution After SMOTE:")
    print(resampled_class_distribution)
    X_train = X_train_resampled
    y_train = y_train_resampled
    print("\nSMOTE Applied. Training data resampled.")
else:
    print("\nNo significant class imbalance. SMOTE not applied.")
```

```
Class Distribution:
Obesity_Type_III       3492
Obesity_Type_II        2848
Normal_Weight          2687
Obesity_Type_I         2579
Overweight_Level_II    2276
Insufficient_Weight    2216
Overweight_Level_I     2178
Name: NObeyesdad, dtype: int64
Imbalance Ratio: 1.603305785123967

No significant class imbalance. SMOTE not applied.
```

Remove outliers from training dataset using IQR

In [42]:
```
# Remove Outliers
selected_columns = ['Age']
Q1 = X_train[selected_columns].quantile(0.25)
Q3 = X_train[selected_columns].quantile(0.75)
IQR = Q3 - Q1
threshold = 1.5
outlier_mask = (
    (X_train[selected_columns] < (Q1 - threshold * IQR)) |
    (X_train[selected_columns] > (Q3 + threshold * IQR))
).any(axis=1)
X_train_clean = X_train[~outlier_mask]
y_train_clean = y_train[~outlier_mask]
num_rows_removed = len(X_train) - len(X_train_clean)
print(f"Number of rows removed due to outliers: {num_rows_removed}")
X_train = X_train_clean
y_train = y_train_clean
```

```
Number of rows removed due to outliers: 1005
```

In [43]:
```
# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_val = sc.transform(X_val)
X_test = sc.transform(test.drop(['NObeyesdad'], axis=1))  # Assuming test has the targ

# Check shapes
print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"X_val shape: {X_val.shape}")
```

```
print(f"y_val shape: {y_val.shape}")
print(f"X_test shape: {X_test.shape}")
```

```
X_train shape: (17271, 23)
y_train shape: (17271,)
X_val shape: (4569, 23)
y_val shape: (4569,)
X_test shape: (13840, 23)
```

Model fitting and Prediction

1. Decision Tree

In [44]:

```
# Best parameters for Decision Tree
best_params_dt = {
    'criterion': 'entropy',
    'max_depth': 14,
    'min_samples_split': 14,
    'min_samples_leaf': 16
}

# Create and train the model
dt_classifier = DecisionTreeClassifier(**best_params_dt, random_state=42)
dt_classifier.fit(X_train, y_train)

# Predict and evaluate
y_pred_dt = dt_classifier.predict(X_val)  # Use validation set for evaluation
accuracy_dt = accuracy_score(y_val, y_pred_dt)
print(f"Decision Tree Accuracy: {accuracy_dt}")
print(classification_report(y_val, y_pred_dt))
```

```
Decision Tree Accuracy: 0.8872838695557015
                     precision    recall  f1-score   support

Insufficient_Weight       0.92      0.92      0.92       574
      Normal_Weight       0.86      0.85      0.85       677
     Obesity_Type_I       0.90      0.83      0.86       682
    Obesity_Type_II       0.95      0.96      0.95       697
   Obesity_Type_III       0.99      1.00      0.99       878
 Overweight_Level_I       0.75      0.79      0.77       525
Overweight_Level_II       0.77      0.80      0.79       536

           accuracy                           0.89      4569
          macro avg       0.88      0.88      0.88      4569
       weighted avg       0.89      0.89      0.89      4569
```

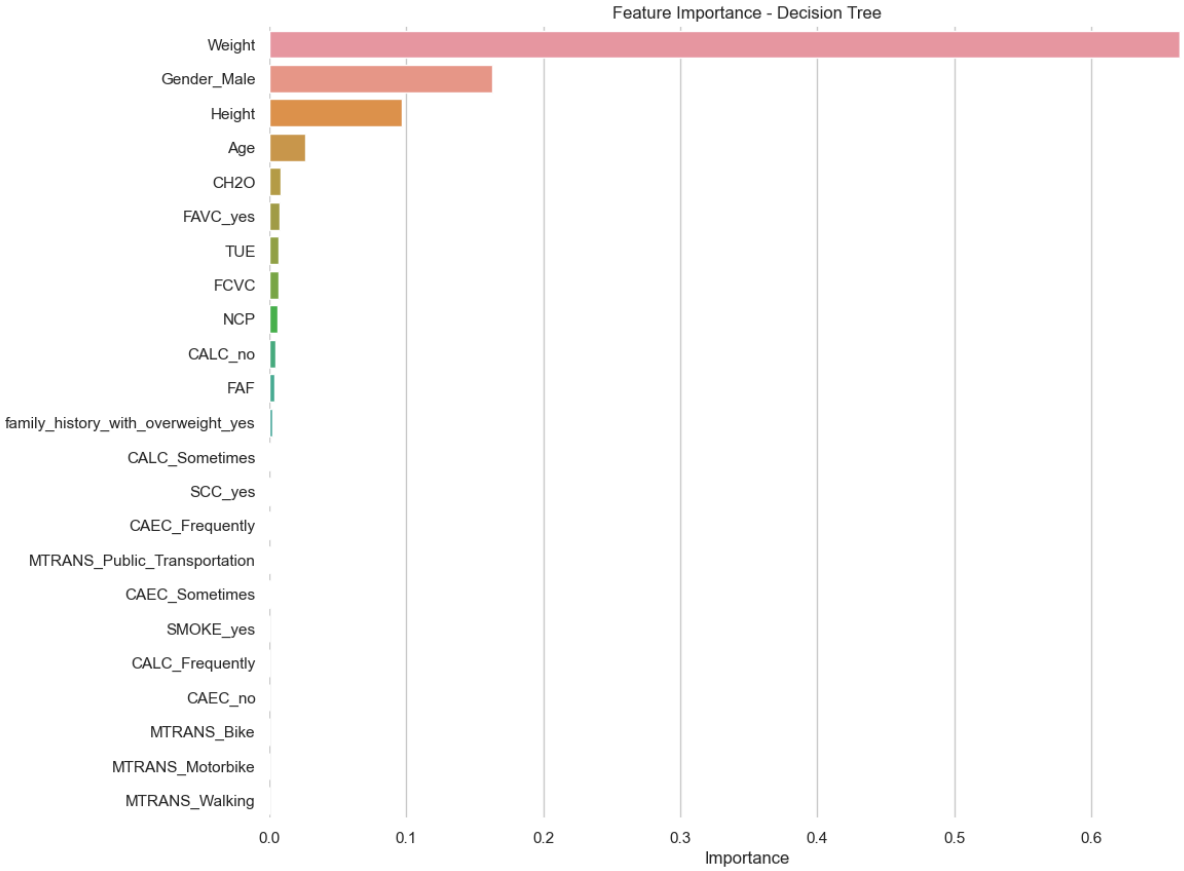
In [45]:

```
# Perform cross-validation with appropriate metrics
cv_scores_accuracy = cross_val_score(dt_classifier, X, y, cv=5, scoring='accuracy')
cv_scores_log_loss = cross_val_score(dt_classifier, X, y, cv=5, scoring='neg_log_loss'
cv_scores_f1_macro = cross_val_score(dt_classifier, X, y, cv=5, scoring='f1_macro')

print(f"Cross-Validation Accuracy Scores: {cv_scores_accuracy}")
print(f"Mean CV Accuracy Score: {np.mean(cv_scores_accuracy)}")

print(f"Cross-Validation Log Loss Scores: {cv_scores_log_loss}")
print(f"Mean CV Log Loss Score: {np.mean(cv_scores_log_loss)}")
```

```
print(f"Cross-Validation F1 Macro Scores: {cv_scores_f1_macro}")
print(f"Mean CV F1 Macro Score: {np.mean(cv_scores_f1_macro)}")
```

```
Cross-Validation Accuracy Scores: [0.8715255  0.8708689  0.88006128 0.88487634 0.8997
5925]
Mean CV Accuracy Score: 0.8814182534471439
Cross-Validation Log Loss Scores: [-1.23948374 -1.41289197 -1.42701925 -1.34260973 -
0.8585291 ]
Mean CV Log Loss Score: -1.2561067587364187
Cross-Validation F1 Macro Scores: [0.85901122 0.85859204 0.86885226 0.87352259 0.8898
5542]
Mean CV F1 Macro Score: 0.8699667072141335
```

In [46]:
```
# Feature importances
feature_importance = dt_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_impc
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=F
plt.figure(figsize=(12, 10))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance - Decision Tree')
plt.xlabel('Importance')
plt.ylabel('')
sns.despine(left=True, bottom=True)
plt.show()
```



2. Random Forest

In [47]:
```
from sklearn.ensemble import RandomForestClassifier

# Best parameters for Random Forest
best_params_rf = {
```

```
    'n_estimators': 744,
    'max_depth': 26,
    'min_samples_split': 12,
    'min_samples_leaf': 1,
    'max_features': 'sqrt'
}

# Create and train the model
rf_classifier = RandomForestClassifier(**best_params_rf, random_state=42)
rf_classifier.fit(X_train, y_train)

# Predict and evaluate
y_pred_rf = rf_classifier.predict(X_val)
accuracy_rf = accuracy_score(y_val, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf}")
print(classification_report(y_val, y_pred_rf))
```

```
Random Forest Accuracy: 0.8923177938279712
                     precision    recall  f1-score   support

Insufficient_Weight       0.93      0.92      0.93       574
      Normal_Weight       0.83      0.90      0.86       677
     Obesity_Type_I       0.95      0.81      0.87       682
    Obesity_Type_II       0.97      0.97      0.97       697
   Obesity_Type_III       1.00      1.00      1.00       878
 Overweight_Level_I       0.77      0.75      0.76       525
Overweight_Level_II       0.74      0.82      0.78       536

           accuracy                           0.89      4569
          macro avg       0.88      0.88      0.88      4569
       weighted avg       0.90      0.89      0.89      4569
```

In [48]:

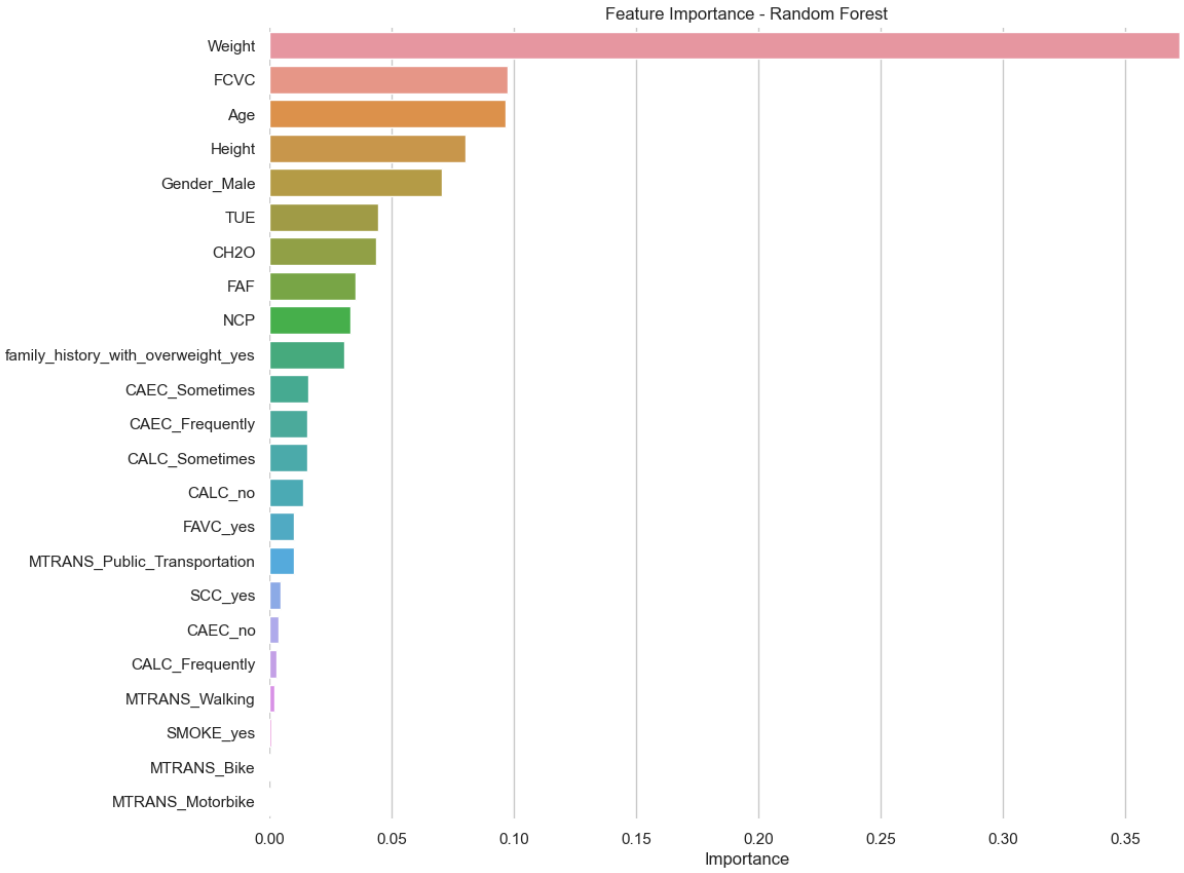
```
# Perform cross-validation with appropriate metrics
cv_scores_accuracy = cross_val_score(rf_classifier, X, y, cv=5, scoring='accuracy')
cv_scores_log_loss = cross_val_score(rf_classifier, X, y, cv=5, scoring='neg_log_loss'
cv_scores_f1_macro = cross_val_score(rf_classifier, X, y, cv=5, scoring='f1_macro')

print(f"Cross-Validation Accuracy Scores: {cv_scores_accuracy}")
print(f"Mean CV Accuracy Score: {np.mean(cv_scores_accuracy)}")

print(f"Cross-Validation Log Loss Scores: {cv_scores_log_loss}")
print(f"Mean CV Log Loss Score: {np.mean(cv_scores_log_loss)}")

print(f"Cross-Validation F1 Macro Scores: {cv_scores_f1_macro}")
print(f"Mean CV F1 Macro Score: {np.mean(cv_scores_f1_macro)}")
```

```
Cross-Validation Accuracy Scores: [0.88991026 0.8896914  0.89800832 0.90260451 0.9194
5721]
Mean CV Accuracy Score: 0.8999343401181878
Cross-Validation Log Loss Scores: [-0.38031872 -0.38358506 -0.37354463 -0.36987124 -
0.34120488]
Mean CV Log Loss Score: -0.3697049037468917
Cross-Validation F1 Macro Scores: [0.87829895 0.87805748 0.88829274 0.89224186 0.9119
472 ]
Mean CV F1 Macro Score: 0.8897676497167903
```

In [49]:

```
# Feature importances
feature_importance = rf_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_impc
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=F
```

```
plt.figure(figsize=(12, 10))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance - Random Forest')
plt.xlabel('Importance')
plt.ylabel('')
sns.despine(left=True, bottom=True)
plt.show()
```



4. Gradient Boosting Classifier

In [50]:

```
from sklearn.ensemble import GradientBoostingClassifier

# Best parameters for Gradient Boosting
best_params_gb = {
    'n_estimators': 618,
    'learning_rate': 0.1851561046543922,
    'max_depth': 3,
    'min_samples_split': 4,
    'min_samples_leaf': 1,
    'subsample': 0.7301321323053057,
    'max_features': 'log2'
}

# Create and train the model
gb_classifier = GradientBoostingClassifier(**best_params_gb, random_state=42)
gb_classifier.fit(X_train, y_train)

# Predict and evaluate
y_pred_gb = gb_classifier.predict(X_val)
accuracy_gb = accuracy_score(y_val, y_pred_gb)
```

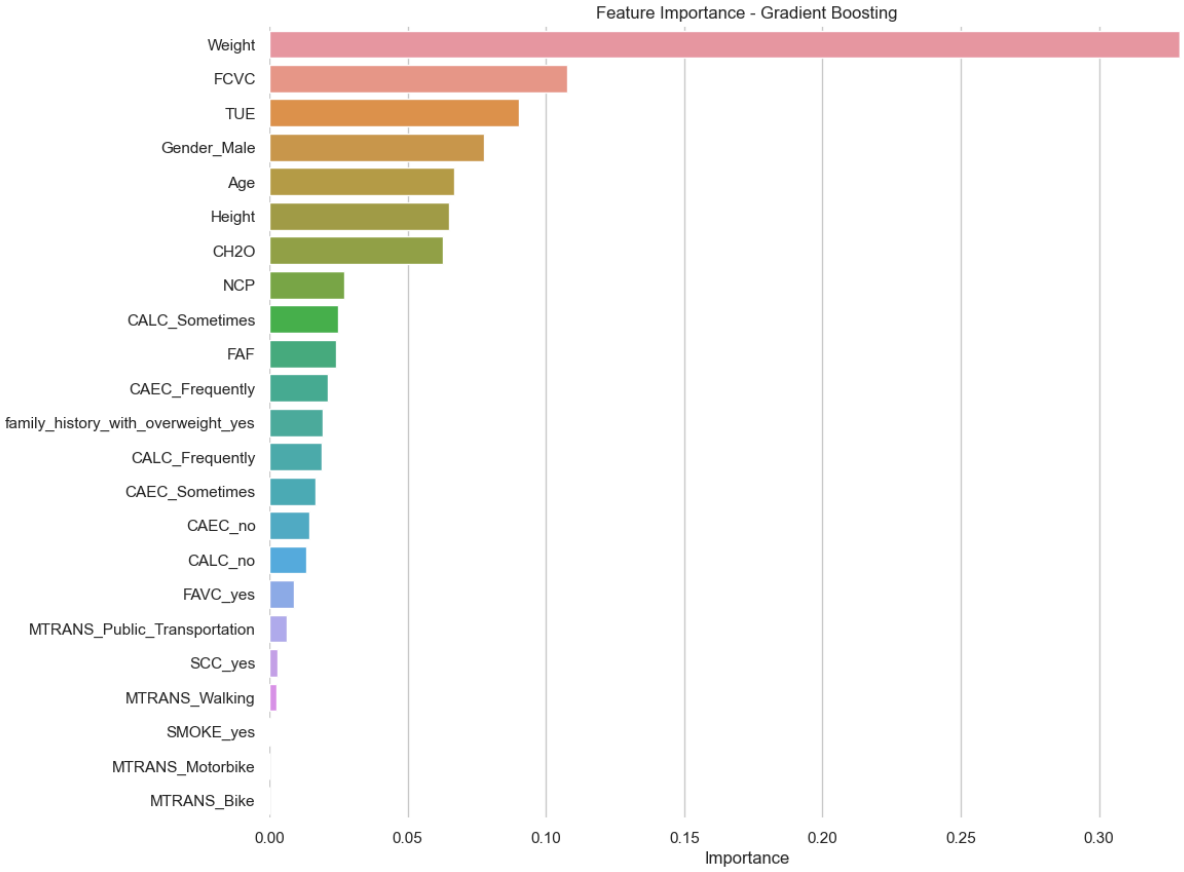
```
print(f"Gradient Boosting Accuracy: {accuracy_gb}")
print(classification_report(y_val, y_pred_gb))
```

```
Gradient Boosting Accuracy: 0.273145108338805
                     precision    recall  f1-score   support

Insufficient_Weight       0.33      0.19      0.24       574
      Normal_Weight       0.59      0.69      0.64       677
     Obesity_Type_I       0.39      0.07      0.12       682
    Obesity_Type_II       0.62      0.30      0.41       697
   Obesity_Type_III       0.25      0.00      0.00       878
 Overweight_Level_I       0.31      0.14      0.19       525
Overweight_Level_II       0.12      0.63      0.21       536

           accuracy                           0.27      4569
          macro avg       0.38      0.29      0.26      4569
       weighted avg       0.38      0.27      0.25      4569
```

In [51]:
```
# Perform cross-validation with appropriate metrics
cv_scores_accuracy = cross_val_score(gb_classifier, X, y, cv=5, scoring='accuracy')
cv_scores_log_loss = cross_val_score(gb_classifier, X, y, cv=5, scoring='neg_log_loss'
cv_scores_f1_macro = cross_val_score(gb_classifier, X, y, cv=5, scoring='f1_macro')

print(f"Cross-Validation Accuracy Scores: {cv_scores_accuracy}")
print(f"Mean CV Accuracy Score: {np.mean(cv_scores_accuracy)}")

print(f"Cross-Validation Log Loss Scores: {cv_scores_log_loss}")
print(f"Mean CV Log Loss Score: {np.mean(cv_scores_log_loss)}")

print(f"Cross-Validation F1 Macro Scores: {cv_scores_f1_macro}")
print(f"Mean CV F1 Macro Score: {np.mean(cv_scores_f1_macro)}")
```

```
Cross-Validation Accuracy Scores: [0.90019698 0.89757058 0.39746115 0.256949   0.1626
1764]
Mean CV Accuracy Score: 0.5229590720070036
Cross-Validation Log Loss Scores: [ -0.29996553  -0.3839511  -21.71770142 -26.7822725
4 -30.18231952]
Mean CV Log Loss Score: -15.873242022775958
Cross-Validation F1 Macro Scores: [0.89017734 0.88801346 0.38666468 0.24675418 0.1544
7919]
Mean CV F1 Macro Score: 0.5132177693324314
```

In [52]:
```
# Feature importances
feature_importance = gb_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_impo
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=F
plt.figure(figsize=(12, 10))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance - Gradient Boosting')
plt.xlabel('Importance')
plt.ylabel('')
sns.despine(left=True, bottom=True)
plt.show()
```



4. XGBoost Classifier

In [53]:

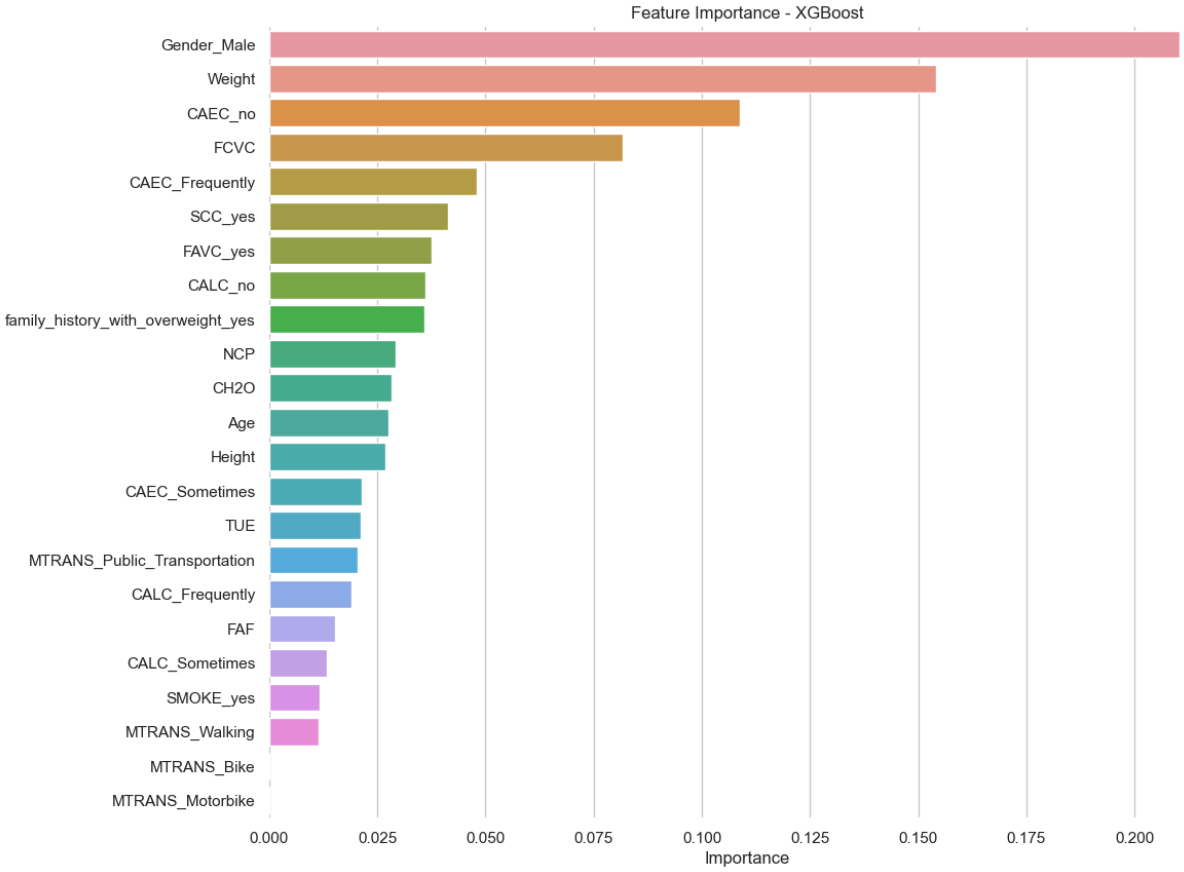
```
from xgboost import XGBClassifier

# Best parameters for XGBoost
best_params_xgb = {
    'n_estimators': 484,
    'learning_rate': 0.057802283277477794,
    'max_depth': 10,
    'min_child_weight': 6,
    'subsample': 0.6873001371240143,
    'colsample_bytree': 0.708723729237405
}

# Label Encoding for target variable
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)
y_encoded = label_encoder.transform(y)

# Create and train the model
xgb_classifier = XGBClassifier(**best_params_xgb, random_state=42, use_label_encoder=F
xgb_classifier.fit(X_train, y_train_encoded)

# Predict and evaluate
y_pred_xgb = xgb_classifier.predict(X_val)
accuracy_xgb = accuracy_score(y_val_encoded, y_pred_xgb)
print(f"XGBoost Accuracy: {accuracy_xgb}")
print(classification_report(y_val_encoded, y_pred_xgb, target_names=label_encoder.clas
```

```
XGBoost Accuracy: 0.9100459619172685
                     precision    recall  f1-score   support

Insufficient_Weight       0.93      0.95      0.94       574
      Normal_Weight       0.89      0.90      0.89       677
     Obesity_Type_I       0.93      0.85      0.89       682
    Obesity_Type_II       0.97      0.97      0.97       697
   Obesity_Type_III       1.00      1.00      1.00       878
 Overweight_Level_I       0.79      0.82      0.80       525
Overweight_Level_II       0.80      0.83      0.81       536

           accuracy                           0.91      4569
          macro avg       0.90      0.90      0.90      4569
       weighted avg       0.91      0.91      0.91      4569
```

In [54]:
```
# Perform cross-validation with appropriate metrics
cv_scores_accuracy = cross_val_score(xgb_classifier, X, y_encoded, cv=5, scoring='accu
cv_scores_log_loss = cross_val_score(xgb_classifier, X, y_encoded, cv=5, scoring='neg_
cv_scores_f1_macro = cross_val_score(xgb_classifier, X, y_encoded, cv=5, scoring='f1_m

print(f"Cross-Validation Accuracy Scores: {cv_scores_accuracy}")
print(f"Mean CV Accuracy Score: {np.mean(cv_scores_accuracy)}")

print(f"Cross-Validation Log Loss Scores: {cv_scores_log_loss}")
print(f"Mean CV Log Loss Score: {np.mean(cv_scores_log_loss)}")

print(f"Cross-Validation F1 Macro Scores: {cv_scores_f1_macro}")
print(f"Mean CV F1 Macro Score: {np.mean(cv_scores_f1_macro)}")
```

```
Cross-Validation Accuracy Scores: [0.90435544 0.9058875  0.91179689 0.90829503 0.9255
8547]
Mean CV Accuracy Score: 0.9111840665353469
Cross-Validation Log Loss Scores: [-0.2768559  -0.29954309 -0.27540553 -0.28060139 -
0.21180251]
Mean CV Log Loss Score: -0.2688416850875971
Cross-Validation F1 Macro Scores: [0.89488449 0.89657546 0.90319054 0.89909372 0.9183
4194]
Mean CV F1 Macro Score: 0.9024172300112733
```

In [55]:
```
# Feature importances
feature_importance = xgb_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_impo
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=F
plt.figure(figsize=(12, 10))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance - XGBoost')
plt.xlabel('Importance')
plt.ylabel('')
sns.despine(left=True, bottom=True)
plt.show()
```



5. CatBoost Classifier

In [56]:

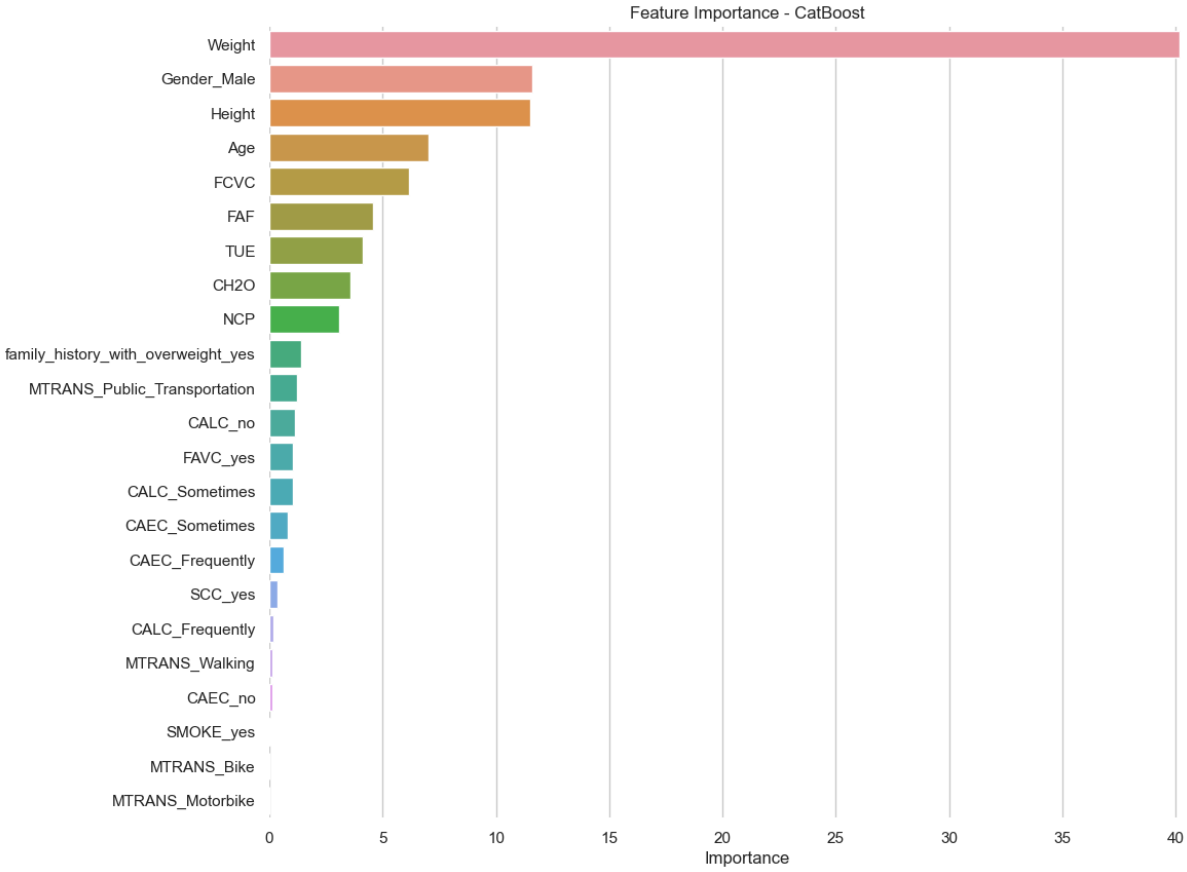
```
from catboost import CatBoostClassifier

# Best parameters for CatBoost
best_params_cb = {
    'iterations': 994,
    'learning_rate': 0.1520048955879897,
    'depth': 5,
    'l2_leaf_reg': 3.374126414651382,
    'border_count': 186
}

# Label Encoding for target variable
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)
y_encoded = label_encoder.transform(y)  # Encode the entire target variable for cross-

# Create and train the model
cb_classifier = CatBoostClassifier(**best_params_cb, random_state=42, verbose=0)
cb_classifier.fit(X_train, y_train_encoded)

# Predict and evaluate
y_pred_cb = cb_classifier.predict(X_val)
accuracy_cb = accuracy_score(y_val_encoded, y_pred_cb)
print(f"CatBoost Accuracy: {accuracy_cb}")
print(classification_report(y_val_encoded, y_pred_cb, target_names=label_encoder.class
```

```
CatBoost Accuracy: 0.9058875027358284
                     precision    recall  f1-score   support

Insufficient_Weight       0.93      0.94      0.93       574
      Normal_Weight       0.89      0.88      0.89       677
     Obesity_Type_I       0.94      0.84      0.89       682
    Obesity_Type_II       0.96      0.98      0.97       697
   Obesity_Type_III       1.00      1.00      1.00       878
 Overweight_Level_I       0.79      0.81      0.80       525
Overweight_Level_II       0.78      0.84      0.81       536

           accuracy                           0.91      4569
          macro avg       0.90      0.90      0.90      4569
       weighted avg       0.91      0.91      0.91      4569
```

In [57]:
```
# Perform cross-validation with appropriate metrics
cv_scores_accuracy = cross_val_score(cb_classifier, X, y_encoded, cv=5, scoring='accur
cv_scores_log_loss = cross_val_score(cb_classifier, X, y_encoded, cv=5, scoring='neg_l
cv_scores_f1_macro = cross_val_score(cb_classifier, X, y_encoded, cv=5, scoring='f1_ma

print(f"Cross-Validation Accuracy Scores: {cv_scores_accuracy}")
print(f"Mean CV Accuracy Score: {np.mean(cv_scores_accuracy)}")

print(f"Cross-Validation Log Loss Scores: {cv_scores_log_loss}")
print(f"Mean CV Log Loss Score: {np.mean(cv_scores_log_loss)}")

print(f"Cross-Validation F1 Macro Scores: {cv_scores_f1_macro}")
print(f"Mean CV F1 Macro Score: {np.mean(cv_scores_f1_macro)}")
```

```
Cross-Validation Accuracy Scores: [0.90151018 0.90479317 0.91157803 0.9052309  0.9317
1372]
Mean CV Accuracy Score: 0.9109652002626396
Cross-Validation Log Loss Scores: [-0.2680714  -0.28859266 -0.27025589 -0.27601638 -
0.21073788]
Mean CV Log Loss Score: -0.262734843596997
Cross-Validation F1 Macro Scores: [0.89171754 0.89539703 0.90308309 0.89539194 0.9253
0024]
Mean CV F1 Macro Score: 0.9021779673320045
```

In [58]:
```
# Feature importances
feature_importance = cb_classifier.get_feature_importance()
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_impo
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=F
plt.figure(figsize=(12, 10))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance - CatBoost')
plt.xlabel('Importance')
plt.ylabel('')
sns.despine(left=True, bottom=True)
plt.show()
```



6. Light Gradient Boosting

In [61]:

```
from lightgbm import LGBMClassifier

# Best parameters for LightGBM
best_params_lgbm = {
    "objective": "multiclass",
    "metric": "multi_logloss",
    "verbosity": -1,
    "boosting_type": "gbdt",
    "random_state": 42,
    "num_class": 7,
    'learning_rate':  0.011978527382971352,
    'n_estimators': 509,
    'lambda_l1': 0.009715116714365276,
    'lambda_l2':  0.03853395161282091,
    'max_depth': 11,
    'colsample_bytree':  0.7364306508830605,
    'subsample':  0.9529973839959326,
    'min_child_samples': 17
}

# Create and train the model
lgbm_classifier = LGBMClassifier(**best_params_lgbm)
lgbm_classifier.fit(X_train, y_train)

# Predict and evaluate
y_pred_lgbm = lgbm_classifier.predict(X_val)
accuracy_lgbm = accuracy_score(y_val, y_pred_lgbm)
```

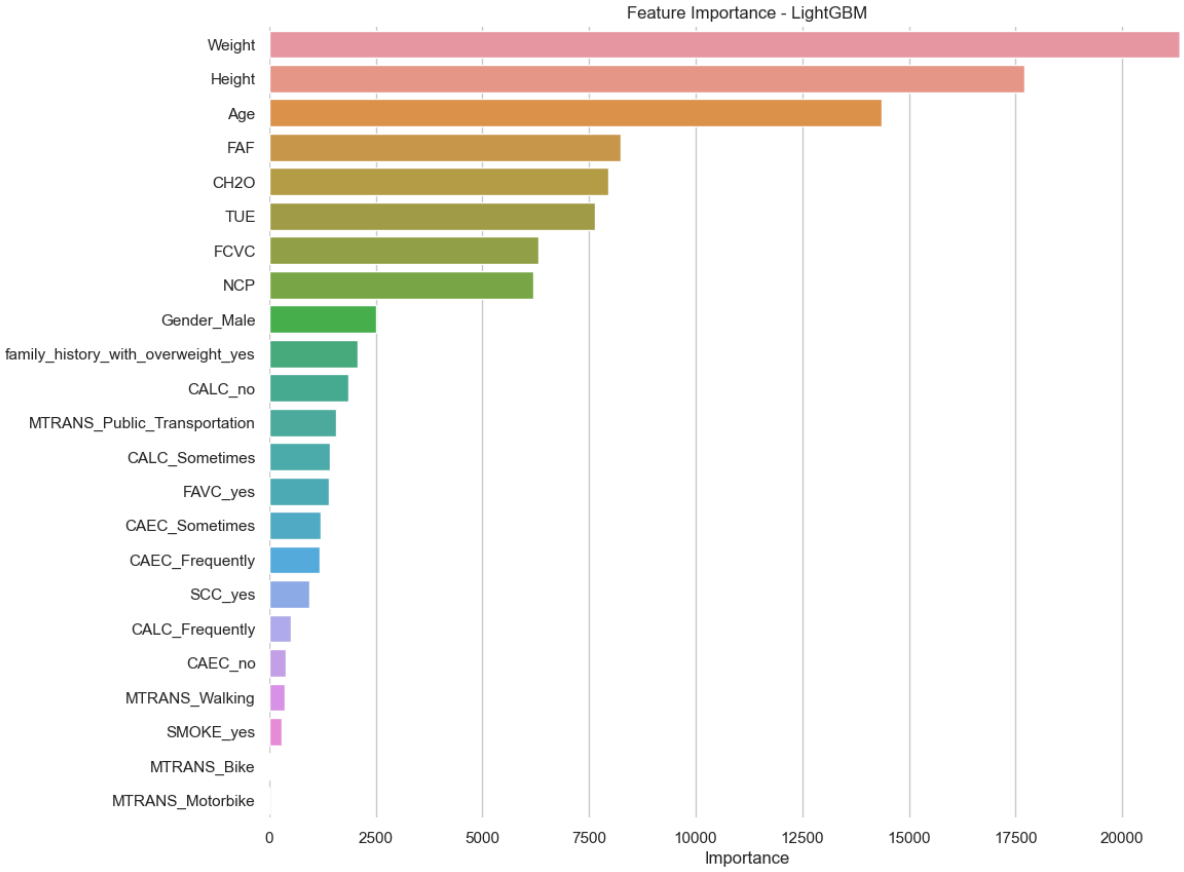
```
print(f"LightGBM Accuracy: {accuracy_lgbm}")
print(classification_report(y_val, y_pred_lgbm))
```

```
LightGBM Accuracy: 0.9133289560078792
                     precision    recall  f1-score   support

Insufficient_Weight       0.93      0.94      0.94       574
      Normal_Weight       0.89      0.90      0.90       677
     Obesity_Type_I       0.93      0.87      0.90       682
    Obesity_Type_II       0.97      0.97      0.97       697
   Obesity_Type_III       1.00      1.00      1.00       878
 Overweight_Level_I       0.81      0.81      0.81       525
Overweight_Level_II       0.80      0.84      0.82       536

           accuracy                           0.91      4569
          macro avg       0.90      0.90      0.90      4569
       weighted avg       0.91      0.91      0.91      4569
```

In [63]:
```
# Perform cross-validation with appropriate metrics
cv_scores_accuracy = cross_val_score(lgbm_classifier, X, y, cv=5, scoring='accuracy')
cv_scores_log_loss = cross_val_score(lgbm_classifier, X, y, cv=5, scoring='neg_log_los
cv_scores_f1_macro = cross_val_score(lgbm_classifier, X, y, cv=5, scoring='f1_macro')

print(f"Cross-Validation Accuracy Scores: {cv_scores_accuracy}")
print(f"Mean CV Accuracy Score: {np.mean(cv_scores_accuracy)}")

print(f"Cross-Validation Log Loss Scores: {cv_scores_log_loss}")
print(f"Mean CV Log Loss Score: {np.mean(cv_scores_log_loss)}")

print(f"Cross-Validation F1 Macro Scores: {cv_scores_f1_macro}")
print(f"Mean CV F1 Macro Score: {np.mean(cv_scores_f1_macro)}")
```

```
Cross-Validation Accuracy Scores: [0.90369884 0.90676297 0.91092143 0.91070256 0.9317
1372]
Mean CV Accuracy Score: 0.91275990369884
Cross-Validation Log Loss Scores: [-0.26829541 -0.28034211 -0.26390528 -0.26168369 -
0.20833737]
Mean CV Log Loss Score: -0.2565127721390516
Cross-Validation F1 Macro Scores: [0.89413371 0.89740541 0.9023965  0.90123    0.9251
1204]
Mean CV F1 Macro Score: 0.9040555330423852
```

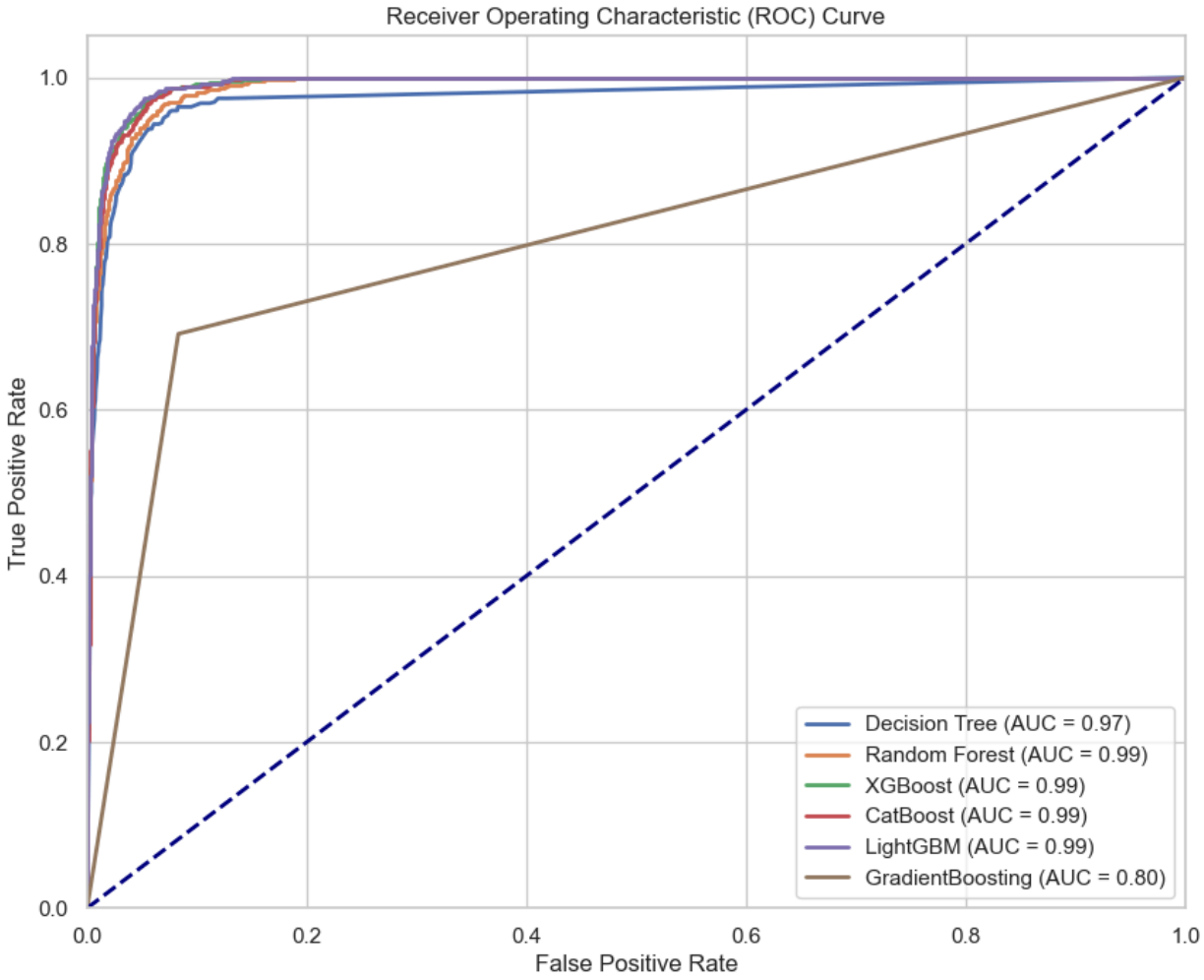
In [64]:
```
# Feature importances
feature_importance = lgbm_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_impo
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=F
plt.figure(figsize=(12, 10))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance - LightGBM')
plt.xlabel('Importance')
plt.ylabel('')
sns.despine(left=True, bottom=True)
plt.show()
```



In [70]:

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Define classifiers and their predictions
classifiers = {
    'Decision Tree': dt_classifier,
    'Random Forest': rf_classifier,
    'XGBoost': xgb_classifier,
    'CatBoost': cb_classifier,
    'LightGBM': lgbm_classifier,
    'GradientBoosting':gb_classifier
}

# Define a function to plot ROC curve
def plot_roc_curve(classifiers, X_val, y_val):
    plt.figure(figsize=(10, 8))
    for name, clf in classifiers.items():
        if 'CatBoost' in name:  # CatBoost needs special handling due to its get_roc_c
            y_pred_proba = clf.predict_proba(X_val)
            fpr, tpr, _ = roc_curve(y_val, y_pred_proba[:, 1], pos_label=1)
        else:
            y_pred_proba = clf.predict_proba(X_val)[:, 1]
            fpr, tpr, _ = roc_curve(y_val, y_pred_proba, pos_label=1)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
```

```
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()

# Plot ROC curves for all classifiers
plot_roc_curve(classifiers, X_val, y_val_encoded)
```



Based on the analysis of classification reports, cross-validation scores, feature importances, and AUC values provided, let's compare the performance of the models:

1. **Decision Tree:**
    - Accuracy: 0.887
    - Mean CV Accuracy: 0.881
    - AUC: 0.97
2. **Random Forest:**
    - Accuracy: 0.892
    - Mean CV Accuracy: 0.900
    - AUC: 0.99
3. **XGBoost:**
    - Accuracy: 0.910
    - Mean CV Accuracy: 0.911

- AUC: 0.99

4. **LightGBM:**
   - Accuracy: 0.913
   - Mean CV Accuracy: 0.913
   - AUC: 0.99
5. **CatBoost:**
   - Accuracy: 0.906
   - Mean CV Accuracy: 0.911
   - AUC: 0.99
6. **Gradient Boosting (GB):**
   - Accuracy: 0.273
   - Mean CV Accuracy: 0.523
   - AUC: 0.80

Evaluation:

- **Accuracy:** LightGBM (0.913) and XGBoost (0.910) have the highest accuracy scores on the validation set.
- **Mean CV Accuracy:** LightGBM (0.913) has the highest mean cross-validation accuracy score, closely followed by XGBoost (0.911).
- **AUC:** Random Forest, XGBoost, LightGBM, and CatBoost have the highest AUC scores (0.99), indicating excellent overall performance in terms of ranking predictions.

Conclusion:

Based on the above comparisons across various metrics (accuracy, cross-validation accuracy, and AUC), **LightGBM** and **XGBoost** emerge as the top-performing models for this classification task. LightGBM slightly edges out XGBoost with marginally higher accuracy and mean cross-validation scores. Therefore, **LightGBM** is the best performing model for this particular dataset and task.

Prediction on Test set

In [75]:
```
# Drop the extra feature from test
test = test.drop(columns='NObeyesdad')
```

In [76]:
```
# Evaluate the best model on the test set
predictions = lgbm_classifier.predict(test)
```

In [77]:
```
print(predictions)
```

```
['Obesity_Type_II' 'Obesity_Type_III' 'Obesity_Type_III' ...
 'Obesity_Type_III' 'Obesity_Type_I' 'Obesity_Type_II']
```

In [80]:
```
submission = pd.read_csv('sample_submission.csv')
submission["NObeyesdad"] = predictions
submission.to_csv("submission1.csv", index=False)
submission.head()
```

Out[80]:

	id	NObeyesdad
0	20758	Obesity_Type_II
1	20759	Obesity_Type_III
2	20760	Obesity_Type_III
3	20761	Obesity_Type_II
4	20762	Overweight_Level_II