COMPREHENSIVE ANALYSIS AND PREDICTION OF OBESITY RISK LEVELS USING MACHINE LEARNING TECHNIQUES WITH - (LightGBM) MODEL

Author: Anamika Kumari

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Section: 1. Introduction:

What is Obesity:

Obesity is a complex health condition affecting millions globally, with significant implications for morbidity, mortality, and healthcare costs.Obesity is a global concern, with statistics indicating a significant rise in the number of obese individuals, now accounting for approximately 30% of the global population, triple the figures from 1975. This escalating trend highlights the pressing need to address the multifaceted risks associated with excess weight. Obesity is a major contributor to various health complications, including diabetes, heart disease, osteoarthritis, sleep apnea, strokes, and high blood pressure, thereby significantly reducing life expectancy and increasing mortality rates. Effective prediction of obesity risk is crucial for implementing targeted interventions and promoting public health.

In this project, we undertake a comprehensive analysis to predict obesity risk levels using advanced machine learning techniques.

Understanding Obesity and Risk Prediction:

- **Understanding Obesity:**
- Obesity stems from excessive body fat accumulation, influenced by genetic, environmental, and behavioral factors.
- Risk prediction involves analyzing demographics, lifestyle habits, and physical activity to classify individuals into obesity risk categories.
- **Global Impact:**
  - Worldwide obesity rates have tripled since 1975, affecting 30% of the global population.
  - Urgent action is needed to develop effective risk prediction and management strategies.
- **Factors Influencing Risk:**
  - Obesity risk is shaped by demographics, lifestyle habits, diet, physical activity, and medical history.
  - Analyzing these factors reveals insights into obesity's mechanisms and identifies high-risk populations.
- **Data-Driven Approach:**
  - Advanced machine learning and large datasets enable the development of predictive models for stratifying obesity risk.
  - These models empower healthcare professionals and policymakers to implement tailored interventions for improved public health outcomes.
- **Proactive Health Initiatives:**
  - Our proactive approach aims to combat obesity by leveraging data and technology for personalized prevention and management.
  - By predicting obesity risk, we aspire to create a future where interventions are precise, impactful, and tailored to individual needs.

Source: **World Health Organization.** (2022). Obesity and overweight.

Dataset Overview:

The dataset contains comprehensive information encompassing eating habits, physical activity, and demographic variables, comprising a total of 17

Key Attributes Related to Eating Habits:

- **Frequent Consumption of High-Caloric Food (FAVC):** Indicates the frequency of consuming high-caloric food items.
- **Frequency of Consumption of Vegetables (FCVC):** Measures the frequency of consuming vegetables.
- **Number of Main Meals (NCP):** Represents the count of main meals consumed per day.
- **Consumption of Food Between Meals (CAEC):** Describes the pattern of food consumption between main meals.
- **Consumption of Water Daily (CH20):** Quantifies the daily water intake.
- **Consumption of Alcohol (CALC):** Indicates the frequency of alcohol consumption.

Attributes Related to Physical Condition:

- **Calories Consumption Monitoring (SCC):** Reflects the extent to which individuals monitor their calorie intake.
- **Physical Activity Frequency (FAF):** Measures the frequency of engaging in physical activities.
- **Time Using Technology Devices (TUE):** Indicates the duration spent using technology devices.
- **Transportation Used (MTRANS):** Describes the mode of transportation typically used.

Additionally, the dataset includes essential demographic variables such as gender, age, height, and weight, providing a comprehensive overview of individuals' characteristics.

Target Variable:

The target variable, NObesity, represents different obesity risk levels, categorized as:

- **Underweight (BMI < 18.5):0**
- **Normal (18.5 <= BMI < 20):1**
- **Overweight I (20 <= BMI < 25):2**
- **Overweight II (25 <= BMI < 30):3**
- **Obesity I (30 <= BMI < 35):4**
- **Obesity II (35 <= BMI < 40):5**
- **Obesity III (BMI >= 40):6**

Section: 2.Importing Libraries and Dataset:

Importing Relevent Libraries:

```
import os  # Operating system specific functionalities
import numpy as np  # Linear algebra
```

```python
import pandas as pd  # Data processing, CSV file I/O (e.g.
pd.read_csv)
from IPython.display import Image  # Displaying images in Jupyter
Notebook
import matplotlib.pyplot as plt  # Plotting library
import seaborn as sns  # Statistical data visualization
%matplotlib inline
import pickle as pkl  # Python object serialization
import altair as alt  # Declarative statistical visualization library
from tabulate import tabulate  # Pretty-print tabular data
from colorama import Fore, Style  # ANSI escape sequences for colored
terminal text
from scipy.stats import pearsonr  # Pearson correlation coefficient
and p-value computation
from mpl_toolkits.mplot3d import Axes3D  # 3D plotting toolkit for
Matplotlib
from sklearn.cluster import KMeans  # K-Means clustering algorithm
from sklearn.preprocessing import StandardScaler  # Standardization of
features
from sklearn.decomposition import PCA  # Principal Component Analysis
from scipy.stats import chi2  # Chi-square distribution
from sklearn.ensemble import RandomForestClassifier  # Random Forest
classifier
import xgboost as xgb  # XGBoost library for gradient boosting
import lightgbm as lgb  # LightGBM library for gradient boosting

# Import necessary libraries for model training and evaluation
from sklearn.model_selection import train_test_split  # Splitting data
into train and test sets
from xgboost import XGBClassifier  # XGBoost classifier
from lightgbm import LGBMClassifier  # LightGBM classifier
from catboost import CatBoostClassifier  # CatBoost classifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix  # For model evaluation

import warnings  # Suppress warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)  # Display all columns in
DataFrame
pd.set_option('display.max_rows', None)  # Display all rows in
DataFrame
```

Loading Datasets:

```python
# Loading Datasets:
# Define filepath
filepath = os.path.join("/kaggle/input/playground-series-s4e2")
```

```
# Function for reading file from your current directory
def read_csv(filepath, filename):
    # Read file from the specified path
    df = pd.read_csv(os.path.join(filepath, filename))
    return df

# Give filepath and access all three file to read (In my case, it is
'train.csv','test.csv' and 'sample_submission.csv')
df_train = read_csv(filepath, 'train.csv')
test = read_csv(filepath,'test.csv')
test_sub=test.copy()
submission_df = read_csv(filepath,'sample_submission.csv')
```

Section: 3. Descriptive Analysis:

```
print('Number of rows and columns:\n')
df_train.shape
```

```
Number of rows and columns:


(20758, 18)
```

```
df_train.head()
```

```
   id  Gender        Age    Height      Weight
family_history_with_overweight  \
0   0    Male  24.443011  1.699998   81.669950
yes
1   1  Female  18.000000  1.560000   57.000000
yes
2   2  Female  18.000000  1.711460   50.165754
yes
3   3  Female  20.952737  1.710730  131.274851
yes
4   4    Male  31.641081  1.914186   93.798055
yes

  FAVC      FCVC       NCP        CAEC SMOKE      CH2O SCC
FAF  \
0  yes  2.000000  2.983297   Sometimes    no  2.763573  no  0.000000

1  yes  2.000000  3.000000  Frequently    no  2.000000  no  1.000000

2  yes  1.880534  1.411685   Sometimes    no  1.910378  no  0.866045

3  yes  3.000000  3.000000   Sometimes    no  1.674061  no  1.467863

4  yes  2.679664  1.971472   Sometimes    no  1.979848  no  1.967973
```

	TUE	CALC	MTRANS	NObeyesdad
0	0.976473	Sometimes	Public_Transportation	Overweight_Level_II
1	1.000000	no	Automobile	Normal_Weight
2	1.673584	no	Public_Transportation	Insufficient_Weight
3	0.780199	Sometimes	Public_Transportation	Obesity_Type_III
4	0.931721	Sometimes	Public_Transportation	Overweight_Level_II

```
test.head()
```

	id	Gender	Age	Height	Weight
0	20758	Male	26.899886	1.848294	120.644178
1	20759	Female	21.000000	1.600000	66.000000
2	20760	Female	26.000000	1.643355	111.600553
3	20761	Male	20.979254	1.553127	103.669116
4	20762	Female	26.000000	1.627396	104.835346

	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE
0	yes	yes	2.938616	3.000000	Sometimes	no
1	yes	yes	2.000000	1.000000	Sometimes	no
2	yes	yes	3.000000	3.000000	Sometimes	no
3	yes	yes	2.000000	2.977909	Sometimes	no
4	yes	yes	3.000000	3.000000	Sometimes	no

	CH2O	SCC	FAF	TUE	CALC	MTRANS
0	2.825629	no	0.855400	0.000000	Sometimes	Public_Transportation
1	3.000000	no	1.000000	0.000000	Sometimes	Public_Transportation
2	2.621877	no	0.000000	0.250502	Sometimes	Public_Transportation
3	2.786417	no	0.094851	0.000000	Sometimes	Public_Transportation
4	2.653531	no	0.000000	0.741069	Sometimes	Public_Transportation

```
df_train.tail()
```

	id	Gender	Age	Height	Weight
20753	20753	Male	25.137087	1.766626	114.187096
20754	20754	Male	18.000000	1.710000	50.000000
20755	20755	Male	20.101026	1.819557	105.580491
20756	20756	Male	33.852953	1.700000	83.520113
20757	20757	Male	26.680376	1.816547	118.134898

```
      family_history_with_overweight FAVC      FCVC       NCP
CAEC  \
20753                            yes  yes  2.919584  3.000000
Sometimes
20754                             no  yes  3.000000  4.000000
Frequently
20755                            yes  yes  2.407817  3.000000
Sometimes
20756                            yes  yes  2.671238  1.971472
Sometimes
20757                            yes  yes  3.000000  3.000000
Sometimes

      SMOKE      CH2O SCC       FAF       TUE       CALC  \
20753    no  2.151809  no  1.330519  0.196680  Sometimes
20754    no  1.000000  no  2.000000  1.000000  Sometimes
20755    no  2.000000  no  1.158040  1.198439         no
20756    no  2.144838  no  0.000000  0.973834         no
20757    no  2.003563  no  0.684487  0.713823  Sometimes

                     MTRANS           NObeyesdad
20753  Public_Transportation      Obesity_Type_II
20754  Public_Transportation  Insufficient_Weight
20755  Public_Transportation      Obesity_Type_II
20756             Automobile  Overweight_Level_II
20757  Public_Transportation      Obesity_Type_II
```

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20758 entries, 0 to 20757
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              20758 non-null  int64
 1   Gender                          20758 non-null  object
 2   Age                             20758 non-null  float64
 3   Height                          20758 non-null  float64
 4   Weight                          20758 non-null  float64
 5   family_history_with_overweight  20758 non-null  object
 6   FAVC                            20758 non-null  object
 7   FCVC                            20758 non-null  float64
 8   NCP                             20758 non-null  float64
 9   CAEC                            20758 non-null  object
 10  SMOKE                           20758 non-null  object
 11  CH2O                            20758 non-null  float64
 12  SCC                             20758 non-null  object
 13  FAF                             20758 non-null  float64
 14  TUE                             20758 non-null  float64
```

```
 15  CALC                              20758 non-null  object
 16  MTRANS                            20758 non-null  object
 17  NObeyesdad                        20758 non-null  object
dtypes: float64(8), int64(1), object(9)
memory usage: 2.9+ MB
```

```
print("size of dataframe:",df_train.size)
df_train.dtypes
```

```
size of dataframe: 373644
```

```
id                                  int64
Gender                             object
Age                               float64
Height                            float64
Weight                            float64
family_history_with_overweight     object
FAVC                               object
FCVC                              float64
NCP                               float64
CAEC                               object
SMOKE                              object
CH2O                              float64
SCC                                object
FAF                               float64
TUE                               float64
CALC                               object
MTRANS                             object
NObeyesdad                         object
dtype: object
```

1. Summary Statistic of dataframe:

```
df_train.describe().transpose().style.background_gradient(cmap='viridis').format("{:.2f}")
```

```
<pandas.io.formats.style.Styler at 0x7c3698339600>
```

- **Count:** Number of non-null values for each feature. For instance, the 'Age' feature has 20,758 non-null values.
- **Mean:** Average value of each feature across all observations. The mean age in the dataset is approximately 23.84 years.
- **Std (Standard Deviation):** Measure of dispersion around the mean, indicating the extent of deviation from the mean value. The standard deviation of age is approximately 5.69 years.
- **Min:** Minimum value observed for each feature. The minimum age in the dataset is 14 years.

- **25%, 50% (Median), 75%:** Quartiles representing the data distribution. The median age (50th percentile) is approximately 22.82 years.
- **Max:** Maximum value observed for each feature. The maximum age in the dataset is 61 years.

These summary statistics provide insights into the distribution and variability of numerical features, facilitating a deeper understanding of the dataset's characteristics and informing subsequent analysis.

```
def summary(dataframe):
    print(f'Data shape: {dataframe.shape}')
# Print the shape of the dataframe
    summary_df = pd.DataFrame(dataframe.dtypes, columns=['Data Type'])
# Create a dataframe to store summary information
    summary_df['# Missing'] = dataframe.isnull().sum().values
# Count the number of missing values for each column
    summary_df['% Missing'] = (dataframe.isnull().sum().values /
len(dataframe)) * 100 # Calculate the percentage of missing values for
each column
    summary_df['# Unique'] = dataframe.nunique().values
# Count the number of unique values for each column
    desc = pd.DataFrame(dataframe.describe(include='all').transpose())
# Create a descriptive statistics df & transpose it for easier merging
    summary_df['Min'] = desc['min'].values
# Add the minimum values from the descriptive statistics
    summary_df['Max'] = desc['max'].values
# Add the maximum values from the descriptive statistics

    return summary_df


# Call the function with the dataframe "df_train" and display the
summary
summary(df_train)
```

```
Data shape: (20758, 18)

                               Data Type  # Missing  % Missing  #
Unique  \
id                                 int64          0        0.0
20758
Gender                            object          0        0.0
2
Age                              float64          0        0.0
1703
Height                           float64          0        0.0
1833
Weight                           float64          0        0.0
1979
family_history_with_overweight    object          0        0.0
```

```
2
FAVC                              object        0         0.0
2
FCVC                             float64        0         0.0
934
NCP                              float64        0         0.0
689
CAEC                              object        0         0.0
4
SMOKE                             object        0         0.0
2
CH2O                             float64        0         0.0
1506
SCC                               object        0         0.0
2
FAF                              float64        0         0.0
1360
TUE                              float64        0         0.0
1297
CALC                              object        0         0.0
3
MTRANS                            object        0         0.0
5
NObeyesdad                        object        0         0.0
7

                                  Min         Max
id                                0.0     20757.0
Gender                            NaN         NaN
Age                              14.0        61.0
Height                           1.45    1.975663
Weight                           39.0  165.057269
family_history_with_overweight    NaN         NaN
FAVC                              NaN         NaN
FCVC                              1.0         3.0
NCP                               1.0         4.0
CAEC                              NaN         NaN
SMOKE                             NaN         NaN
CH2O                              1.0         3.0
SCC                               NaN         NaN
FAF                               0.0         3.0
TUE                               0.0         2.0
CALC                              NaN         NaN
MTRANS                            NaN         NaN
NObeyesdad                        NaN         NaN
```

- **Data Shape:** The dataset contains 20,758 rows and 17 columns.
- **Data Types:** The dataset consists of a mix of object (likely categorical) and float64 (likely numerical) data types.

- **# Missing:** There are no missing values present in any of the columns.
- **% Missing:** As there are no missing values, the percentage of missing values for all columns is 0.0%.
- **# Unique:** Each column has a varying number of unique values, ranging from 2 to 1,703.
- **Min:** Minimum values observed for numerical features range from 14.0 to 39.0.
- **Max:** Maximum values observed for numerical features range from 61.0 to 165.057269.

2. The unique values present in dataset:

```
# Iterate through each column in the DataFrame
for col in df_train.columns:
    # Get the unique values present in the current column
    unique_values = df_train[col].unique()
    # Print the column name along with its unique values
    print(f"Unique values in '{col}': {unique_values}")
```

```
Unique values in 'id': [    0     1     2 ... 20755 20756 20757]
Unique values in 'Gender': ['Male' 'Female']
Unique values in 'Age': [24.443011 18.       20.952737 ... 25.746113
38.08886  33.852953]
Unique values in 'Height': [1.699998 1.56     1.71146  ... 1.791366
1.672594 1.536819]
Unique values in 'Weight': [ 81.66995   57.        50.165754 ...
152.063947  79.5       80.615325]
Unique values in 'family_history_with_overweight': ['yes' 'no']
Unique values in 'FAVC': ['yes' 'no']
Unique values in 'FCVC': [2.         1.880534   3.         2.679664
2.919751   1.99124
 1.397468   2.636719   1.         1.392665   2.203962   2.971588
 2.668949   1.98989905 2.417635   2.219186   2.919526   2.263245
 2.649406   1.754401   2.303656   2.020785   2.068834   2.689929
 2.979383   2.225731   2.843456   2.312528   2.962415   2.945967
 2.108638   1.826885   2.200588   2.598051   2.984425   1.387489
 2.76533    2.941627   2.490776   2.801514   2.336044   1.270448
 2.9673     2.325623   2.722161   2.680375   2.938801   2.431346
 1.994679   2.393837   1.428289   2.341999   2.967853   1.899116
 1.906194   2.859097   2.997951   2.499388   1.4925     2.239634
 2.587789   2.795086   2.805512   2.048962   2.319776   2.823179
 1.188089   2.671238   1.882235   2.61939    2.191429   2.995599
 2.594653   1.369529   2.457548   2.73691    1.947495   2.073224
 2.57649    2.748243   2.736628   2.204914   1.475906   2.007845
 2.890535   2.96405    2.915921   2.318355   2.766612   2.684335
 2.819934   2.948425   1.961347   1.996638   2.111887   2.838037
 1.469384   2.05687    2.966126   2.061952   2.92711    2.490507
 1.164062   2.596579   2.591292   2.927218   1.003566   2.66889
 2.630401   2.76802    2.156065   2.880759   2.446872   2.996717
 2.802696   2.927409   2.724121   2.497548   2.942154   2.971574
 1.963965   2.600217   2.630137   2.108711   2.716909   2.923916
```

2.177243	2.938616	1.936479	2.388168	2.195964	1.975675
2.262292	2.81646	2.568063	2.846452	1.522001	2.049112
2.661556	1.518966	2.619835	2.02472	1.220024	2.901924
2.037585	2.065752	1.0816	2.883745	2.247795	2.839048
2.910733	1.889883	2.758394	2.596364	2.997524	2.76632
1.096455	2.465575	2.00876	1.005578	2.96008	2.512719
1.868212	2.392665	1.31415	2.869436	1.950742	2.14128
1.760038	2.906269	2.766441	2.020502	1.036414	2.032883
2.739	2.549782	1.813234	2.225149	2.044326	2.197261
1.853314	2.397284	1.800122	2.998441	2.164062	2.290095
2.253371	2.818502	2.18354	1.362441	2.9553	2.959658
2.19331	2.178889	2.310751	2.598207	2.492758	2.650629
2.654076	2.886157	2.333503	1.303878	2.347942	2.061461
2.274846	1.528331	2.824559	1.006436	1.758394	2.869778
1.116068	1.524428	2.821727	2.95841	2.845961	2.772027
2.784471	2.09283	2.736647	2.956671	1.924632	1.455602
2.938031	2.342323	2.744994	2.913486	1.264234	2.315932
2.033745	2.041376	1.626369	1.99953	2.175276	2.4277
1.871213	2.499626	2.272453	1.123939	2.002796	2.913452
2.01695	2.910345	1.063449	1.203754	2.689577	1.834155
2.928234	2.467548	2.674431	1.570089	2.735297	2.969205
2.031185	2.09663	2.57691	2.274164	2.929889	2.432302
2.934671	2.206119	2.714447	2.866383	1.588114	2.328469
1.431346	2.451009	1.915279	2.412566	2.762496	1.517912
1.961069	2.133955	2.826251	2.84837	2.576449	2.877743
1.735664	2.838969	2.918113	2.151335	1.036159	2.220181
2.291846	2.973569	2.260543	2.562409	1.031149	1.624366
2.871137	2.734762	2.778079	2.540949	2.591439	2.39728
2.972426	1.451337	2.79606	1.263216	2.081238	2.19005
2.652779	2.051283	2.387426	1.276858	2.424977	2.332074
2.871016	2.780699	2.95801	1.952987	2.644094	1.766612
2.737149	2.030256	1.412566	2.191108	2.654792	2.281963
2.801992	1.973499	1.967061	2.444599	1.878251	1.887951
1.786841	2.075321	2.535154	2.162519	2.762325	2.104772
2.813775	2.923433	2.826036	2.869833	2.252472	1.064162
2.33361	2.765769	2.902469	2.036613	2.882522	2.619987
2.911312	2.720701	2.371338	2.87599	2.121909	2.482575
1.617093	1.69427	2.88853	2.407817	2.058687	1.851262
2.746408	2.79166	2.293705	1.849347	2.117121	2.615788
2.85916	1.537505	2.642744	2.880161	2.002076	1.140466
2.002784	2.425503	1.585183	1.368978	2.152264	1.992889
2.277436	2.872121	1.853991	2.939727	1.757466	2.273548
2.14961	2.974006	2.02091	1.328469	2.408561	2.964419
2.113843	2.736298	2.177896	2.908757	2.842102	1.650505
2.464518	2.685484	2.247037	2.21965	1.925064	2.252698
2.907062	2.911877	2.533605	2.19011	2.633855	1.289315
2.993634	2.047069	1.977298	1.253371	2.903545	2.475892
2.749629	2.927187	1.289421	2.222282	1.993101	2.684528
1.750809	2.631565	2.392179	2.935157	2.499108	1.123672

2.1239	2.055209	2.919584	2.725282	1.142468	2.278644
2.457547	2.640801	2.765063	1.557287	1.979944	2.372494
2.494451	2.128574	2.059138	2.317459	1.972926	2.045027
2.611847	1.572036	2.880483	1.521604	1.32534	2.943749
2.774562	2.061384	2.262171	2.009952	2.794197	2.723953
2.738485	2.34222	2.323351	2.637202	1.642241	1.780746
2.971351	2.787589	2.048216	2.33998	1.052699	2.938687
2.609123	2.13683	1.836554	1.57223	2.459976	1.874935
2.667229	2.44004	2.011656	2.954996	2.050619	1.562804
2.907542	1.712747	2.244654	2.915279	2.133964	1.492834
1.718156	2.870152	2.921576	2.330023	2.653721	2.031246
2.673638	2.294067	2.366949	2.176317	2.712747	2.815157
2.996186	2.907744	2.836055	1.00876	2.271306	2.786008
1.457758	2.992205	1.344854	1.897796	2.53915	1.34138
2.252653	2.969233	2.844607	2.595746	2.108163	2.658112
2.094184	2.885693	1.84199	2.510583	1.893428	2.423291
2.153639	2.300408	2.870895	2.749268	1.948248	2.100177
2.562687	2.442536	2.241606	2.580872	2.341133	2.310423
2.5621	2.185938	2.317734	2.432355	2.871768	2.884212
2.008245	2.450784	2.427689	1.601236	1.061461	1.712848
2.530233	2.592247	2.042762	1.710548	2.086898	1.631144
2.522399	1.72989	2.805533	2.5596	1.92822	1.118436
1.889199	2.944287	2.63165	2.037042	2.357496	2.976509
2.282803	2.025479	2.443538	2.288604	2.519592	2.21232
2.88626	2.060922	2.061969	1.901611	2.314175	2.843709
2.911749	2.982261	2.530066	2.693859	2.180047	2.569075
2.897899	2.696381	2.23372	2.766036	1.3307	2.352323
2.432886	2.699282	1.674431	1.482722	2.535315	1.75375
1.766849	2.49619	2.501224	2.802128	2.72989	2.977018
2.740633	2.002564	2.734314	2.909853	2.68601	2.784464
2.777165	2.008656	2.54527	2.948248	2.253707	1.919629
2.382705	2.311436	2.834155	2.181057	2.21498	2.186322
2.155182	2.524428	2.808027	2.052932	2.150054	2.450218
2.748971	1.826251	2.003951	2.021446	2.896562	2.165605
1.450218	2.70825	2.362918	2.097373	2.218599	2.205633
2.103335	2.753752	2.55996	2.86099	2.102696	2.22259
2.076689	1.780699	2.663866	1.947405	2.561638	2.217267
2.496455	2.486189	1.592183	1.588782	2.066101	2.104105
2.119643	1.972545	1.317729	2.992606	2.206276	2.737762
1.108663	2.403421	1.053534	2.38695	2.146598	2.070964
2.585942	2.323003	2.319648	2.555401	1.340405	2.071622
2.107854	2.951591	2.87781	2.348745	2.522183	2.607747
1.771693	2.312825	2.306844	1.202075	2.954417	1.996646
2.048582	2.206399	1.620845	2.71897	2.252382	1.133844
2.029634	2.265973	1.168856	2.381164	2.667676	2.027574
2.767731	2.706134	1.899793	2.827773	2.129668	2.652958
1.773265	2.334474	2.977585	2.067817	2.724285	2.253998
2.754646	1.27785	2.507841	1.122127	1.609938	2.074843
2.595957	2.015258	2.588089	2.750715	1.443674	1.846452

```
 2.846981   1.943927   1.926381   2.046651   2.964319   1.206276
 2.493448   2.277077   2.286481   1.83746    2.983042   2.08868
 1.452524   2.552388   2.198315   2.184843   1.261288   2.992329
 1.794825   1.321028   2.443674   2.145114   1.649974   1.111887
 2.000466   1.140615   2.921225   1.595746   2.159033   2.028571
 2.215464   2.577427   2.734715   2.703436   2.663421   1.317734
 2.501236   2.612941   1.421656   2.92416    1.473088   1.078719
 1.812283   2.232836   2.129969   2.052152   2.396265   2.708965
 2.956297   1.989905   2.688054   2.010684   1.859097   2.138334
 2.976975   2.247704   2.541785   1.679935   1.729824   2.557287
 2.501683   2.116432   2.231915   2.812388   1.081585   2.519792
 2.679724   2.941929   2.880792   2.620963   2.04516    2.947495
 2.009796   2.559571   2.933409   1.918251   2.628791   2.939671
 2.123159   1.002564   2.038774   2.543563   2.120185   2.08841
 2.503244   2.274491   2.165408   2.694281   2.988668   2.990741
 2.656912   1.94313    2.742796   2.416044   2.8557     2.822179
 2.721356   2.069267   2.244142   2.399531   2.122127   1.067909
 2.922511   2.966617   2.349419   1.021136   2.793561   2.078082
 1.687569   2.814517   1.655684   2.99448    2.874643   1.723921
 2.760607   1.921031   2.971832   2.977298   2.240757   2.759286
 2.642748   2.743277   2.205439   2.37464    2.707666   1.904732
 2.014194   2.8        2.933015   1.584785   2.182401   2.95118
 2.970983   2.252699   1.773079   2.949242   2.340405   2.076094
 2.490937   1.910176   1.204855   2.178308   2.487781   2.571274
 2.213135   2.822183   1.81646    2.592607   2.09449    2.754645
 2.821977   2.01054    2.721507   1.960138   1.588185   1.666416
 2.973499   2.984004   2.690754   2.099687   2.043359   1.873716
 2.8813     1.386151   1.369421   1.340361   1.763941   2.836554
 1.785286   1.21232    1.252698   1.83841    2.195368   2.767063
 2.483979   2.14084    2.487167   2.342459   2.3307     2.800122
 1.709585   2.851664   1.078529   2.613249   1.567101   2.814453
 2.303041   2.490613   2.353603   2.303367   1.246822   2.94313
 2.880534   2.936509   2.925941   2.763215   1.962947   2.770964
 2.219156   2.81746    2.914453   2.320201   1.734314   1.252653
 2.997062   1.658571   1.557747   2.294259   1.3899     2.868212
 1.941357   1.842102   1.562409   1.472522   1.073224   2.115354
 1.996186   2.622827   2.607335   2.731368  ]
Unique values in 'NCP': [2.983297 3.       1.411685 1.971472 2.164839
1.       2.954446 1.893811
 3.998618 1.703299 2.937989 2.996444 2.581015 2.473913 1.437959
2.989791
 4.       2.853676 1.104642 3.362758 1.169173 1.411808 2.98212
1.81698
 3.762778 2.976211 2.993623 3.994588 3.087544 2.372311 2.376374
2.884479
 2.994198 2.812283 3.654061 1.845858 2.475444 1.015488 2.806298
1.338033
 1.077331 3.995957 2.884848 2.283673 2.806341 1.863012 3.590039
2.608416
```

```
 2.129909 2.18162  1.672706 2.951837 2.692889 3.986652 2.449723
2.966803
 2.9948   1.473088 1.882158 2.7976   2.13229  2.999346 1.320768
1.894384
 2.122545 2.99321  3.205009 1.163666 1.08687  2.374791 2.993634
2.937607
 2.831771 3.715148 2.272214 2.918124 3.546352 1.337035 1.226342
3.520555
 1.105617 1.834472 1.867836 1.250548 1.818026 2.567567 3.559841
1.134042
 3.821168 3.24934  2.992606 2.175153 2.725012 1.193486 1.717608
2.999744
 3.821461 3.98955  1.851088 2.141839 2.952821 2.992083 2.968098
3.897078
 1.057935 2.119826 2.574108 3.737914 2.057935 2.622055 2.279546
2.650088
 1.532833 2.870005 2.372339 2.669766 1.262831 1.00061  1.80815
2.993856
 1.820779 1.941307 3.981997 3.281391 3.731212 2.701689 1.66338
1.600812
 1.924168 2.358298 1.73762  1.135278 3.765526 2.301129 1.79558
2.015675
 2.050121 2.487674 1.009426 3.058539 1.724887 2.175432 2.994046
2.938902
 1.271624 2.902639 2.711238 2.77684  3.647154 2.791366 3.156309
2.977909
 1.114564 2.95833  3.392811 1.835543 2.449067 2.799979 2.933409
1.92822
 3.196043 2.956422 2.39007  3.623364 2.915921 2.491315 2.608055
3.558637
 3.471536 2.976098 2.832018 2.844138 1.240046 2.970675 2.695396
2.977999
 1.001542 1.630506 2.693646 2.427137 2.806566 2.834253 2.475228
2.658639
 1.095223 3.891994 2.991671 2.038373 2.473911 1.915921 1.089048 2.6648
 3.095663 1.134321 2.902766 2.625942 1.101404 2.983201 1.672751
2.842848
 1.888067 2.845307 2.658837 2.740492 2.865657 2.978103 1.773916
2.879541
 1.513835 3.699594 2.37985  2.       2.962004 3.985442 3.378859
2.911568
 2.935381 2.272801 3.12544  2.468421 1.211606 1.154318 1.685134
3.804944
 2.627173 3.715306 3.207071 2.812377 2.049565 2.986637 1.865238
1.109956
 1.194815 1.000283 2.499108 2.842035 2.720642 2.116195 1.72626  1.9154
 2.964024 1.152521 1.546665 2.753418 1.873484 2.595126 2.10601
2.809716
 2.395785 1.082304 3.715118 2.547086 3.263201 1.625942 3.34175
```

```
2.920373
 3.266644 2.965494 3.96981  2.992903 1.989398 2.877583 3.240578
1.273128
 2.625475 2.943307 3.21043  3.566082 3.987707 1.193729 1.630846
2.997414
 1.80993  1.000414 2.27374  2.601675 1.735493 2.996084 2.996834
2.092179
 3.051804 3.205587 2.138375 2.120936 2.987652 2.270163 1.193589
2.961192
 3.697831 2.986172 2.961706 2.043359 2.946063 2.240424 3.592415
2.19011
 2.888193 2.974204 1.317884 2.9774   1.476204 2.582591 1.458507
2.127797
 2.701521 1.874532 3.494849 3.019574 1.627555 1.418985 2.967089
1.402771
 1.259803 2.909117 3.642802 3.435904 1.680838 3.390143 2.646717
2.378211
 3.105007 2.87747  2.734762 2.218285 1.10548  2.880817 3.691226
1.202179
 2.142328 2.9796   2.991666 2.604998 2.849848 3.409363 2.271734
3.098399
 3.734323 3.489918 3.732126 3.220181 3.443456 2.838388 1.508685
1.619796
 3.714833 3.45259  1.058123 2.752318 2.510135 2.993084 3.129155
2.434347
 3.193671 1.971659 1.146052 3.433908 2.743277 2.029858 3.995147
1.001383
 1.977221 2.645858 2.756622 1.07976  3.042774 3.90779  1.792695
1.075553
 2.735706 1.578521 1.217651 3.371832 1.326982 2.883984 2.658478
2.683061
 3.339914 1.010319 1.124977 2.977543 1.590982 1.471053 2.973476
2.57038
 2.938135 1.384322 2.278652 2.910794 1.704828 3.712183 1.890213
3.36313
 2.721238 1.231915 2.675148 1.91863  2.443812 3.238258 2.562895
3.612941
 2.837388 1.782109 2.623079 1.94313  1.865657 3.376844 1.355354
1.001633
 2.123138 1.060796 2.973504 2.857787 1.068443 3.394788 1.049534
1.131695
 3.285167 1.237454 2.91753  3.156153 1.682804 2.741413 3.376717
1.13715
 2.96405  1.281165 3.586082 1.346987 3.884861 1.599464 3.047959
1.974233
 3.171082 3.292386 2.27372  2.850948 1.097312 2.699971 2.675411
2.644692
 2.749334 2.371658 1.709546 1.477581 3.595761 1.105616 1.474836
1.400943
```

```
 3.245148 1.355752 2.849347 2.988771 3.39007  1.999014 3.832911
2.973729
 2.975675 1.213431 2.675823 2.894142 2.377056 2.974568 2.392811
2.65772
 2.956622 3.092116 2.696051 2.282392 1.416309 3.322522 2.815255
3.118013
 2.070033 1.802305 1.548407 1.496776 2.326233 2.716106 2.880794
1.976744
 1.068196 2.040582 3.054899 2.988539 2.677693 2.794156 1.014916
2.975362
 3.914454 3.576103 3.754599 2.900915 2.732331 1.734762 2.401341 2.9154
 2.870661 2.391753 3.488342 3.618722 2.110937 1.081805 2.400943
1.02075
 2.64155  1.289315 1.116401 2.752705 2.667711 2.375026 1.120102
2.093831
 2.971574 2.174968 1.311797 1.374791 3.563744 1.706551 1.139317
2.961113
 2.044035 1.394539 3.990925 3.98525  1.130751 1.521546 2.419656
2.358455
 1.171027 3.887906 1.391778 3.420618 2.100918 2.893778 1.09749
3.435905
 2.298612 2.762883 1.028426 1.198643 3.087119 2.779379 1.313403
2.049908
 3.30846  3.53009  2.463113 3.131032 1.2919   1.030416 2.217651
2.119682
 2.687502 3.394539 1.836226 2.137068 1.032887 2.89292  2.714115
3.483449
 3.053598 3.989492 1.69608  2.765213 2.698883 3.998766 1.259628
3.747163
 3.60885  3.728377 2.73762  2.737571 1.854536 1.656588 3.03779
3.704828
 1.288716 2.783336 1.028538 2.337035 2.843319 1.612747 2.036794
1.047197
 2.372705 3.292956 2.113575 2.047866 2.656588 2.996543 3.28926
2.041558
 1.044628 1.293342 1.343117 2.609801 1.630728 1.281683 2.521546
2.988602
 2.139775 1.660768 2.568063 2.478794 1.487674 2.118153 3.937099
2.805436
 2.930044 1.672958 1.099151 2.948721 1.468948 2.404788 1.296156
3.169089
 2.597608 1.555557 3.755976 1.72326  1.923607 1.418833 1.834373
2.984523
 1.25535  3.11158  3.904858 3.648194 2.52751  2.679724 2.600812
3.711238
 1.146794 3.502604 2.845858 2.598079 2.042078 1.631184 3.770379
2.874532
 1.713762 3.097373 1.73988  2.676148 3.725797 2.101841 1.788602
2.752815
```

```
 1.890682 2.85916  1.352649 1.289733 2.818026 1.164839 2.152733
2.814518
 2.705445 1.478334 2.389717 2.157164 1.340361 2.656622 1.9
1.015467
 2.989112 2.270546 2.415522 2.960131 3.179995 2.655265 1.24884
1.109115
 1.569176 3.335876 3.671076 3.259033 3.531038 2.733077 1.350099
1.595784
 3.054416 2.014671 2.929123 2.269799 2.083831 1.890413 3.788602
1.667596
 2.256119]
Unique values in 'CAEC': ['Sometimes' 'Frequently' 'no' 'Always']
Unique values in 'SMOKE': ['no' 'yes']
Unique values in 'CH2O': [2.763573 2.       1.910378 ... 2.151166
1.485836 1.365188]
Unique values in 'SCC': ['no' 'yes']
Unique values in 'FAF': [0.       1.       0.866045 ... 0.540397
0.271174 0.988668]
Unique values in 'TUE': [0.976473 1.       1.673584 ... 1.217929
1.439004 0.768375]
Unique values in 'CALC': ['Sometimes' 'no' 'Frequently']
Unique values in 'MTRANS': ['Public_Transportation' 'Automobile'
'Walking' 'Motorbike' 'Bike']
Unique values in 'NObeyesdad': ['Overweight_Level_II' 'Normal_Weight'
'Insufficient_Weight'
 'Obesity_Type_III' 'Obesity_Type_II' 'Overweight_Level_I'
 'Obesity_Type_I']
```

1. **Age:** Age of the individual in years. (Unique values: 24.443011, 18.0, 20.952737, ...)
2. **Gender:** Gender of the individual, either Male or Female. (Unique values: Male, Female)
3. **Height:** Height of the individual in centimeters. (Unique values: 1.699998, 1.56, 1.71146, ...)
4. **Weight:** Weight of the individual in kilograms. (Unique values: 81.66995, 57.0, 50.165754, ...)
5. **Family_history:** Family history of obesity, either yes or no. (Unique values: yes, no)
6. **FAVC (Frequency of consuming high-caloric food):**
   - **Yes:** Indicates the individual frequently consumes high-caloric food.
   - **No:** Indicates the individual does not frequently consume high-caloric food.
7. **FCVC (Frequency of consuming vegetables):**
   - Ranges from approximately 1.0 to 3.0: Represents the frequency of consuming vegetables.
8. **CAEC (Consumption of food between meals):**
   - **Always:** Indicates the individual always consumes food between meals.
   - **Frequently:** Indicates the individual frequently consumes food between meals.
   - **Sometimes:** Indicates the individual sometimes consumes food between meals.
   - **No:** Indicates the individual does not consume food between meals.
9. **SMOKE (Smoking habit):**

- **Yes:** Indicates the individual smokes.
- **No:** Indicates the individual does not smoke.

10. **CH2O (Consumption of water daily):**
    - Ranges from approximately 1.0 to 3.0 liters: Represents the daily consumption of water in liters.
11. **FAF (Physical activity frequency):**
    - Ranges from approximately 0.0 to 3.0: Represents the frequency of physical activity.
12. **SCC (Calories consumption monitoring):**
    - **Yes:** Indicates the individual monitors their calorie consumption.
    - **No:** Indicates the individual does not monitor their calorie consumption.
13. **TUE (Time using technology devices):**
    - Ranges from approximately 0.0 to 16.0 hours: Represents the time spent using technology devices in hours.
14. **CALC (Alcohol consumption):**
    - **Sometimes:** Indicates the individual sometimes consumes alcohol.
    - **Frequently:** Indicates the individual frequently consumes alcohol.
    - **Always:** Indicates the individual always consumes alcohol.
    - **No:** Indicates the individual does not consume alcohol.
15. **MTRANS (Transportation used):**
    - **Automobile:** Indicates the individual uses automobile for transportation.
    - **Bike:** Indicates the individual uses a bike for transportation.
    - **Motorbike:** Indicates the individual uses a motorbike for transportation.
    - **Public_Transportation:** Indicates the individual uses public transportation.
    - **Walking:** Indicates the individual prefers walking as a mode of transportation.
16. **NObeyesdad (Obesity class):**
    - **No_obesity:** Indicates the individual does not suffer from obesity.
    - **Obesity_Type_I:** Indicates the individual belongs to obesity type I class.
    - **Obesity_Type_II:** Indicates the individual belongs to obesity type II class.
    - **Obesity_Type_III:** Indicates the individual belongs to obesity type III class.

3. The count of unique value in the NObeyesdad column:

```
df_train.groupby('NObeyesdad').count().iloc[:,1]
```

```
NObeyesdad
Insufficient_Weight    2523
Normal_Weight          3082
Obesity_Type_I         2910
Obesity_Type_II        3248
Obesity_Type_III       4046
Overweight_Level_I     2427
```

```
Overweight_Level_II    2522
Name: Gender, dtype: int64
```

- There are 2523 individuals categorized as "Insufficient_Weight".
- There are 3082 individuals categorized as "Normal_Weight".
- There are 2910 individuals categorized as "Obesity_Type_I".
- There are 3248 individuals categorized as "Obesity_Type_II".
- There are 4046 individuals categorized as "Obesity_Type_III".
- There are 2427 individuals categorized as "Overweight_Level_I".
- There are 2522 individuals categorized as "Overweight_Level_II".

4. Categorical and numerical Variables Analysis:

a. Extracting column names for categorical, numerical, and categorical but cardinal variables:

```
# Function to extract column names for categorical, numerical, and
categorical but cardinal variables

def extract_column_names(dataframe, cat_threshold=10,
car_threshold=20):
    """This function extracts the names of categorical, numerical, and
categorical but cardinal variables from a given dataframe.

    Args:
    -------
        dataframe (pandas.DataFrame): The input dataframe containing
all the data.
        cat_threshold (int, float, optional): The threshold value for
considering a numerical variable as categorical. Defaults to 10.
        car_threshold (int, float, optional): The threshold value for
considering a categorical variable as cardinal. Defaults to 20.

    Returns:
    -------
    categorical_columns: List
        List of categorical variable names.

    numerical_columns: List
        List of numerical variable names.

    categorical_but_cardinal: List
```

```
        List of variable names that appear categorical but are
actually cardinal.

    Notes:
    -------
        The sum of categorical_columns, numerical_columns, and
categorical_but_cardinal equals the total number of variables.
        numerical_but_categorical are included in categorical_columns.
        The sum of the three returned lists is equal to the total
number of variables in the dataframe.

    """

    # Extract categorical columns and those that seem numerical but
are categorical
    categorical_columns = [
        col
        for col in dataframe.columns
        if str(dataframe[col].dtypes) in ["object", "category",
"bool"]
    ]

    numerical_but_categorical = [
        col
        for col in dataframe.columns
        if dataframe[col].nunique() < cat_threshold
        and dataframe[col].dtypes in ["int64", "float64"]
    ]

    # Extract columns that appear categorical but are actually
cardinal
    categorical_but_cardinal = [
        col
        for col in dataframe.columns
        if dataframe[col].nunique() > car_threshold
        and str(dataframe[col].dtypes) in ["object", "category"]
    ]

    # Exclude numerical_but_categorical from categorical_columns
    categorical_columns = categorical_columns +
numerical_but_categorical
    categorical_columns = [col for col in categorical_columns if col
not in categorical_but_cardinal]

    # Extract numerical columns
    numerical_columns = [
        col
        for col in dataframe.columns
        if dataframe[col].dtypes in ["int64", "float64"] and col not
in categorical_columns
```

```
    ]

    # Print summary statistics
    print(f"Observations: {dataframe.shape[0]}")
    print(f"Variables: {dataframe.shape[1]}")
    print(f"Categorical columns: {len(categorical_columns)}")
    print(f"Numerical columns: {len(numerical_columns)}")
    print(f"Categorical but cardinal columns: 
{len(categorical_but_cardinal)}")
    print(f"Numerical but categorical columns: 
{len(numerical_but_categorical)}")

    return categorical_columns, numerical_columns, 
categorical_but_cardinal


# Extract column names from the 'df_train' dataframe
categorical_cols, numerical_cols, categorical_but_cardinal = 
extract_column_names(df_train)

Observations: 20758
Variables: 18
Categorical columns: 9
Numerical columns: 9
Categorical but cardinal columns: 0
Numerical but categorical columns: 0
```

- **Observations**: 20,758 rows in the dataset.
- **Variables**: Total of 18 features.
- **Categorical columns**: 9 variables are categorical.
- **Numerical columns**: 9 variables are numerical.
- **Categorical but cardinal columns**: No categorical variables with many unique values.
- **Numerical but categorical columns**: No numerical variables with few unique values.

```
print("Numerical columns:\n", numerical_cols)
print("Categorical columns:\n", categorical_cols)

Numerical columns:
 ['id', 'Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 
'TUE']
Categorical columns:
 ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 
'SCC', 'CALC', 'MTRANS', 'NObeyesdad']
```

b. Summary Of All Categorical Variables:

```
def variable_summary(data_frame):
    # Initialize the summaries list
```

```
    summaries = []

    # Loop through each categorical variable
    for col in data_frame.select_dtypes(include=['object',
'category']):
        # Summary of unique values
        unique_values = data_frame[col].unique()
        unique_count = data_frame[col].nunique()
        summaries.append(Fore.BLUE + f"Summary of {col}:" +
Style.RESET_ALL)
        summaries.append(f"Unique values of {col}: {unique_values} is
{unique_count}.\n")

        # Percentage summary
        total_count = len(data_frame[col])
        percentage_data = []
        for i, (value, count) in
enumerate(data_frame[col].value_counts().head(10).items(), start=1):
            ratio = (count / total_count) * 100
            percentage_data.append([i, value, count, f"{ratio:.2f}%"])
        percentage_headers = [Fore.GREEN + "Index", "Value", "Count",
"Percentage" + Style.RESET_ALL]
        percentage_table = tabulate(percentage_data,
headers=percentage_headers, tablefmt="fancy_grid")

        # Append the percentage table to the summaries list
        summaries.append(percentage_table)
        summaries.append('\n')

    # Print the summaries
    print('\n'.join(summaries))

# Assuming your dataframe is named 'df_train'
print(Fore.BLUE+"################# Summary of Categorical
variables:############################")
variable_summary(df_train)
```

```
################# Summary of Categorical
variables:############################
Summary of Gender:
Unique values of Gender: ['Male' 'Female'] is 2.
```

Index	Value	Count	Percentage
1	Female	10422	50.21%
2	Male	10336	49.79%

```
Summary of family_history_with_overweight:
Unique values of family_history_with_overweight: ['yes' 'no'] is 2.
```

Index	Value	Count	Percentage
1	yes	17014	81.96%
2	no	3744	18.04%

```
Summary of FAVC:
Unique values of FAVC: ['yes' 'no'] is 2.
```

Index	Value	Count	Percentage
1	yes	18982	91.44%
2	no	1776	8.56%

```
Summary of CAEC:
Unique values of CAEC: ['Sometimes' 'Frequently' 'no' 'Always'] is 4.
```

Index	Value	Count	Percentage
1	Sometimes	17529	84.44%
2	Frequently	2472	11.91%
3	Always	478	2.30%
4	no	279	1.34%

```
Summary of SMOKE:
Unique values of SMOKE: ['no' 'yes'] is 2.
```

Index	Value	Count	Percentage
1	no	20513	98.82%
2	yes	245	1.18%

Summary of SCC:
Unique values of SCC: ['no' 'yes'] is 2.

Index	Value	Count	Percentage
1	no	20071	96.69%
2	yes	687	3.31%

Summary of CALC:
Unique values of CALC: ['Sometimes' 'no' 'Frequently'] is 3.

Index	Value	Count	Percentage
1	Sometimes	15066	72.58%
2	no	5163	24.87%
3	Frequently	529	2.55%

Summary of MTRANS:
Unique values of MTRANS: ['Public_Transportation' 'Automobile'
'Walking' 'Motorbike' 'Bike'] is 5.

Index	Value	Count	Percentage
1	Public_Transportation	16687	80.39%
2	Automobile	3534	17.02%
3	Walking	467	2.25%
4	Motorbike	38	0.18%
5	Bike	32	0.15%

Summary of NObeyesdad:
Unique values of NObeyesdad: ['Overweight_Level_II' 'Normal_Weight'

```
'Insufficient_Weight'
 'Obesity_Type_III' 'Obesity_Type_II' 'Overweight_Level_I'
 'Obesity_Type_I'] is 7.
```

Index	Value	Count	Percentage
1	Obesity_Type_III	4046	19.49%
2	Obesity_Type_II	3248	15.65%
3	Normal_Weight	3082	14.85%
4	Obesity_Type_I	2910	14.02%
5	Insufficient_Weight	2523	12.15%
6	Overweight_Level_II	2522	12.15%
7	Overweight_Level_I	2427	11.69%

c. Summary Of All Numerical Variables:

```
from tabulate import tabulate
from colorama import Fore, Style

def variable_summary(data_frame):
    # Summaries of numerical variables
    num_summaries = []
    for col in data_frame.select_dtypes(include=['int64', 'float64']):
        unique_count = data_frame[col].nunique()
        num_summaries.append(Fore.BLUE + f"Summary of {col}:" +
Style.RESET_ALL)
        num_summaries.append(f"Unique values of {col}: is
{unique_count}.\n")
        summary = data_frame[col].describe().reset_index()
        summary.columns = [Fore.RED +"Statistic", col +
Style.RESET_ALL]
        num_summaries.append(tabulate(summary, headers="keys",
tablefmt="fancy_grid"))

    print(Fore.BLUE + "##################### Summaries of numerical
variables #####################")
    print(Style.RESET_ALL)
    print("\n".join(num_summaries))
```

```python
# Assuming your dataframe is named 'df_train'
variable_summary(df_train)
```

```
#################### Summaries of numerical variables
####################

Summary of id:
Unique values of id: is 20758.
```

	Statistic	id
0	count	20758
1	mean	10378.5
2	std	5992.46
3	min	0
4	25%	5189.25
5	50%	10378.5
6	75%	15567.8
7	max	20757

```
Summary of Age:
Unique values of Age: is 1703.
```

	Statistic	Age
0	count	20758
1	mean	23.8418
2	std	5.68807
3	min	14
4	25%	20
5	50%	22.8154
6	75%	26
7	max	61

Summary of Height:
Unique values of Height: is 1833.

	Statistic	Height
0	count	20758
1	mean	1.70024
2	std	0.0873119
3	min	1.45
4	25%	1.63186
5	50%	1.7
6	75%	1.76289
7	max	1.97566

Summary of Weight:
Unique values of Weight: is 1979.

	Statistic	Weight
0	count	20758
1	mean	87.8878
2	std	26.3794
3	min	39
4	25%	66
5	50%	84.0649
6	75%	111.601
7	max	165.057

Summary of FCVC:
Unique values of FCVC: is 934.

	Statistic	FCVC

0	count	20758
1	mean	2.44591
2	std	0.533218
3	min	1
4	25%	2
5	50%	2.39384
6	75%	3
7	max	3

Summary of NCP:
Unique values of NCP: is 689.

	Statistic	NCP
0	count	20758
1	mean	2.76133
2	std	0.705375
3	min	1
4	25%	3
5	50%	3
6	75%	3
7	max	4

Summary of CH2O:
Unique values of CH2O: is 1506.

	Statistic	CH2O
0	count	20758
1	mean	2.02942

2	std	0.608467
3	min	1
4	25%	1.79202
5	50%	2
6	75%	2.54962
7	max	3

```
Summary of FAF:
Unique values of FAF: is 1360.
```

	Statistic	FAF
0	count	20758
1	mean	0.981747
2	std	0.838302
3	min	0
4	25%	0.008013
5	50%	1
6	75%	1.58741
7	max	3

```
Summary of TUE:
Unique values of TUE: is 1297.
```

	Statistic	TUE
0	count	20758
1	mean	0.616756
2	std	0.602113
3	min	0
4	25%	0

5	50%	0.573887
6	75%	1
7	max	2

Section: 4. Data Preprocessing:

1. Typeconversion of dataframe:

```
# Define a function to convert column datatype to integer
def convert_column_datatype(df, column_name):
    """
    Convert the data type of a specified column in the dataframe to
integer.

    Parameters:
    df (DataFrame): The dataframe containing the column to be
converted.
    column_name (str): The name of the column to be converted.

    Returns:
    DataFrame: The dataframe with the specified column converted to
integer data type.
    """
    df[column_name] = df[column_name].astype('int32')
    return df

# Example usage:
df_train = convert_column_datatype(df_train, 'Age')
df_train = convert_column_datatype(df_train, 'Weight')

# Example usage:
test_sub = convert_column_datatype(test_sub, 'Age')
test_sub = convert_column_datatype(test_sub, 'Weight')
```

2. Renaming the Columns:

```
new_column_names = {
    'Gender': 'Gender',
    'Age': 'Age',
    'Height': 'Height',
```

```python
    'Weight': 'Weight',
    'family_history_with_overweight': 'Overweighted Family History',
    'FAVC': 'High caleric food consp',
    'FCVC': 'veg consp',
    'NCP': 'main meal consp',
    'CAEC': 'Food btw meal consp',
    'SMOKE': 'SMOKE',
    'CH2O': 'Water consp',
    'SCC': 'Calories Monitoring',
    'FAF': 'physical actv',
    'TUE': 'Screentime',
    'CALC': 'Alcohol consp',
    'MTRANS': 'transport used',
    'NObeyesdad': 'Obesity_Level'
}

# Rename the columns  for train data
df_train.rename(columns=new_column_names, inplace=True)

df_train.head(5)
```

```
   id  Gender  Age    Height  Weight Overweighted Family History  \
0   0    Male   24  1.699998      81                         yes
1   1  Female   18  1.560000      57                         yes
2   2  Female   18  1.711460      50                         yes
3   3  Female   20  1.710730     131                         yes
4   4    Male   31  1.914186      93                         yes

  High caleric food consp  veg consp  main meal consp Food btw meal
consp  \
0                     yes   2.000000         2.983297
Sometimes
1                     yes   2.000000         3.000000
Frequently
2                     yes   1.880534         1.411685
Sometimes
3                     yes   3.000000         3.000000
Sometimes
4                     yes   2.679664         1.971472
Sometimes

  SMOKE  Water consp Calories Monitoring  physical actv  Screentime  \
0    no     2.763573                  no       0.000000    0.976473
1    no     2.000000                  no       1.000000    1.000000
2    no     1.910378                  no       0.866045    1.673584
3    no     1.674061                  no       1.467863    0.780199
4    no     1.979848                  no       1.967973    0.931721

  Alcohol consp         transport used        Obesity_Level
0     Sometimes  Public_Transportation  Overweight_Level_II
```

```
1             no             Automobile       Normal_Weight
2             no  Public_Transportation  Insufficient_Weight
3      Sometimes  Public_Transportation     Obesity_Type_III
4      Sometimes  Public_Transportation  Overweight_Level_II
```

```python
test_sub.head(5)
```

```
      id  Gender  Age    Height  Weight family_history_with_overweight
FAVC  \
0  20758    Male   26  1.848294     120                            yes
yes
1  20759  Female   21  1.600000      66                            yes
yes
2  20760  Female   26  1.643355     111                            yes
yes
3  20761    Male   20  1.553127     103                            yes
yes
4  20762  Female   26  1.627396     104                            yes
yes

       FCVC       NCP       CAEC SMOKE      CH2O SCC       FAF
TUE  \
0  2.938616  3.000000  Sometimes    no  2.825629  no  0.855400
0.000000
1  2.000000  1.000000  Sometimes    no  3.000000  no  1.000000
0.000000
2  3.000000  3.000000  Sometimes    no  2.621877  no  0.000000
0.250502
3  2.000000  2.977909  Sometimes    no  2.786417  no  0.094851
0.000000
4  3.000000  3.000000  Sometimes    no  2.653531  no  0.000000
0.741069

        CALC                 MTRANS
0  Sometimes  Public_Transportation
1  Sometimes  Public_Transportation
2  Sometimes  Public_Transportation
3  Sometimes  Public_Transportation
4  Sometimes  Public_Transportation
```

3. Detecting Columns with Large or Infinite Values:

```python
def columns_with_infinite_values(df):
    numeric_df = df.select_dtypes(include=[np.number])  # Select only
numeric columns
    inf_values = np.isinf(numeric_df)
```

```
    columns_with_inf = numeric_df.columns[np.any(inf_values, axis=0)]
    return columns_with_inf

print("Columns with infinite values:\n",
columns_with_infinite_values(df_train))

Columns with infinite values:
 Index([], dtype='object')

def columns_with_large_numbers(df):
    numeric_df = df.select_dtypes(include=[np.number])  # Select only
numeric columns
    large_values = np.abs(numeric_df) > 1e15
    columns_with_large = numeric_df.columns[np.any(large_values,
axis=0)]
    return columns_with_large

print("Columns with large values:\n",
columns_with_large_numbers(df_train))

Columns with large values:
 Index([], dtype='object')
```

This output indicates that there are no columns in the dataset with infinite or large values.

Section:5. Exploratory Data Analysis and Visualisation-EDAV:

1. Univariate Analysis:

a. Countplots for all Variables:

```
plt.figure(figsize=(30, 25))
plt.suptitle('Countplots for all Variables', fontsize=24,
fontweight='bold')

# Get the list of column names from the dataframe
columns = df_train.columns

# Determine the number of rows and columns for subplots
num_rows = (len(columns) + 2) // 3  # Add 2 to round up to the nearest
multiple of 3
num_cols = 3
```

```
# Create countplots for each variable
for i, col in enumerate(columns, start=1):
    ax = plt.subplot(num_rows, num_cols, i)
    sns.countplot(x=df_train[col], palette='viridis')  # Add color
palette for better visualization
    ax.set_title(f'Countplot of {col}', fontsize=18, pad=20,
fontweight='bold')
    plt.xlabel(col, fontsize=14, fontweight='bold')  # Add bold
fontweight to x-axis label
    plt.ylabel('Count', fontsize=14, fontweight='bold')  # Add bold
fontweight to y-axis label
    plt.grid(True, linestyle='--', alpha=0.5)
     # Add count indicators on top of each bar
    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2.,
height), ha='center', va='bottom', fontsize=8, fontweight='bold')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

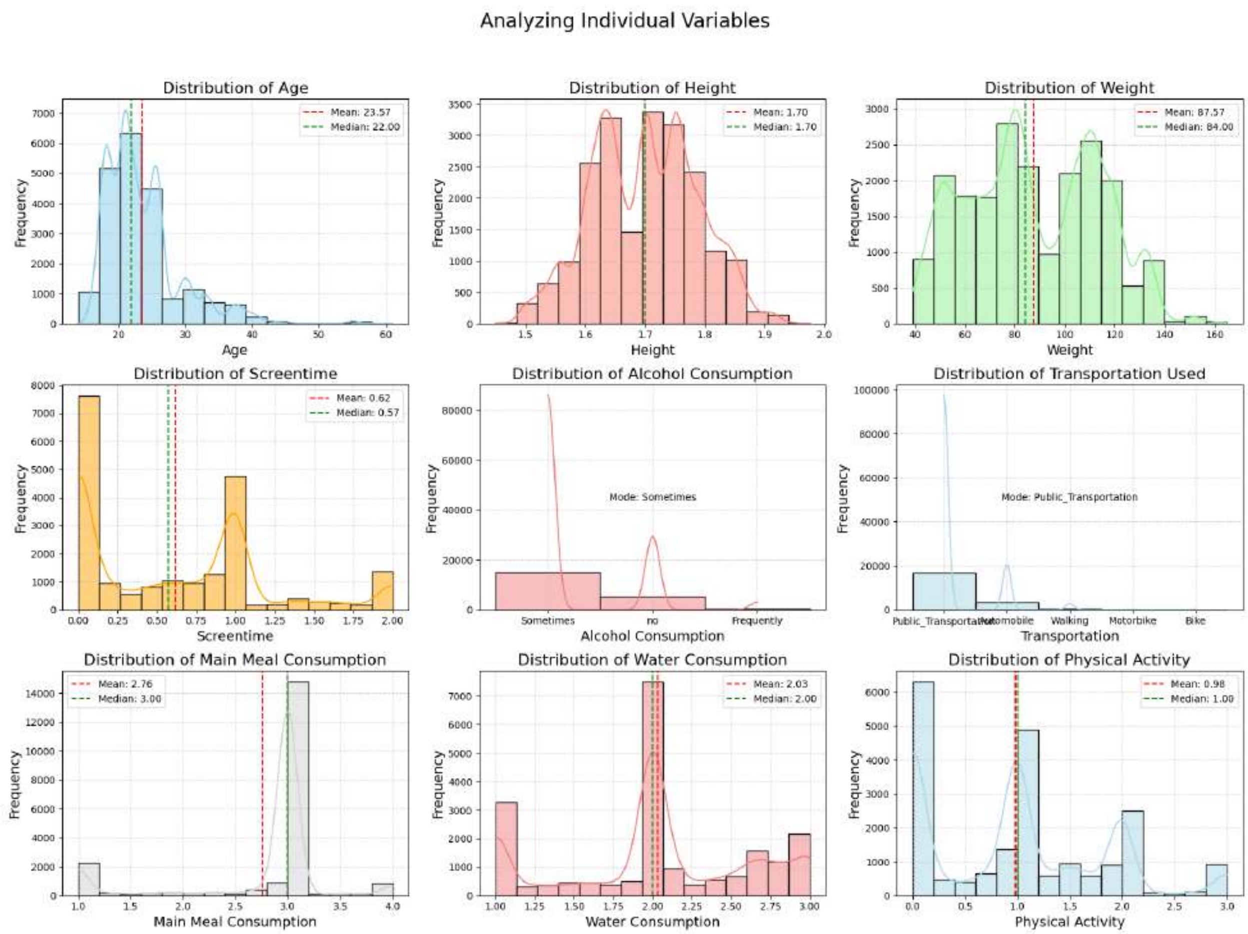
b. Analyzing Individual Variables Using Histogram:

```
plt.figure(figsize=(18, 14))
plt.suptitle('Analyzing Individual Variables', fontsize=20)

# Age
plt.subplot(3, 3, 1)
sns.histplot(df_train['Age'], kde=True, bins=15, color='skyblue')
plt.title('Distribution of Age', fontsize=16)
plt.xlabel('Age', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
mean_age = df_train['Age'].mean()
median_age = df_train['Age'].median()
plt.axvline(x=mean_age, color='red', linestyle='--', label=f'Mean:
{mean_age:.2f}')
```

```
plt.axvline(x=median_age, color='green', linestyle='--',
label=f'Median: {median_age:.2f}')
plt.legend()

# Height
plt.subplot(3, 3, 2)
sns.histplot(df_train['Height'], kde=True, bins=15, color='salmon')
plt.title('Distribution of Height', fontsize=16)
plt.xlabel('Height', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
mean_height = df_train['Height'].mean()
median_height = df_train['Height'].median()
plt.axvline(x=mean_height, color='red', linestyle='--', label=f'Mean:
{mean_height:.2f}')
plt.axvline(x=median_height, color='green', linestyle='--',
label=f'Median: {median_height:.2f}')
plt.legend()

# Weight
plt.subplot(3, 3, 3)
sns.histplot(df_train['Weight'], kde=True, bins=15,
color='lightgreen')
plt.title('Distribution of Weight', fontsize=16)
plt.xlabel('Weight', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
mean_weight = df_train['Weight'].mean()
median_weight = df_train['Weight'].median()
plt.axvline(x=mean_weight, color='red', linestyle='--', label=f'Mean:
{mean_weight:.2f}')
plt.axvline(x=median_weight, color='green', linestyle='--',
label=f'Median: {median_weight:.2f}')
plt.legend()

# Screentime
plt.subplot(3, 3, 4)
sns.histplot(df_train['Screentime'], kde=True, bins=15,
color='orange')
plt.title('Distribution of Screentime', fontsize=16)
plt.xlabel('Screentime', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
mean_screentime = df_train['Screentime'].mean()
median_screentime = df_train['Screentime'].median()
plt.axvline(x=mean_screentime, color='red', linestyle='--',
label=f'Mean: {mean_screentime:.2f}')
plt.axvline(x=median_screentime, color='green', linestyle='--',
label=f'Median: {median_screentime:.2f}')
plt.legend()
```

```python
# Alcohol consumption
plt.subplot(3, 3, 5)
sns.histplot(df_train['Alcohol consp'], kde=True, bins=15,
color='lightcoral')
plt.title('Distribution of Alcohol Consumption', fontsize=16)
plt.xlabel('Alcohol Consumption', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
mode_AlcoholConsp = df_train['Alcohol consp'].mode()[0]
plt.text(0.5, 0.5, f'Mode: {mode_AlcoholConsp}',
horizontalalignment='center', verticalalignment='center',
transform=plt.gca().transAxes)

# Transportation used
plt.subplot(3, 3, 6)
sns.histplot(df_train['transport used'], kde=True, bins=15,
color='lightblue')
plt.title('Distribution of Transportation Used', fontsize=16)
plt.xlabel('Transportation', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
mode_transportation = df_train['transport used'].mode()[0]
plt.text(0.5, 0.5, f'Mode: {mode_transportation}',
horizontalalignment='center', verticalalignment='center',
transform=plt.gca().transAxes)

# Main Meal Consumption
plt.subplot(3, 3, 7)
sns.histplot(df_train['main meal consp'], kde=True, bins=15,
color='lightgrey')
plt.title('Distribution of Main Meal Consumption', fontsize=16)
plt.xlabel('Main Meal Consumption', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
mean_main_meal_consp = df_train['main meal consp'].mean()
median_main_meal_consp = df_train['main meal consp'].median()
plt.axvline(x=mean_main_meal_consp, color='red', linestyle='--',
label=f'Mean: {mean_main_meal_consp:.2f}')
plt.axvline(x=median_main_meal_consp, color='green', linestyle='--',
label=f'Median: {median_main_meal_consp:.2f}')
plt.legend()

# Water consumption
plt.subplot(3, 3, 8)
sns.histplot(df_train['Water consp'], kde=True, bins=15,
color='lightcoral')
plt.title('Distribution of Water Consumption', fontsize=16)
plt.xlabel('Water Consumption', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
```

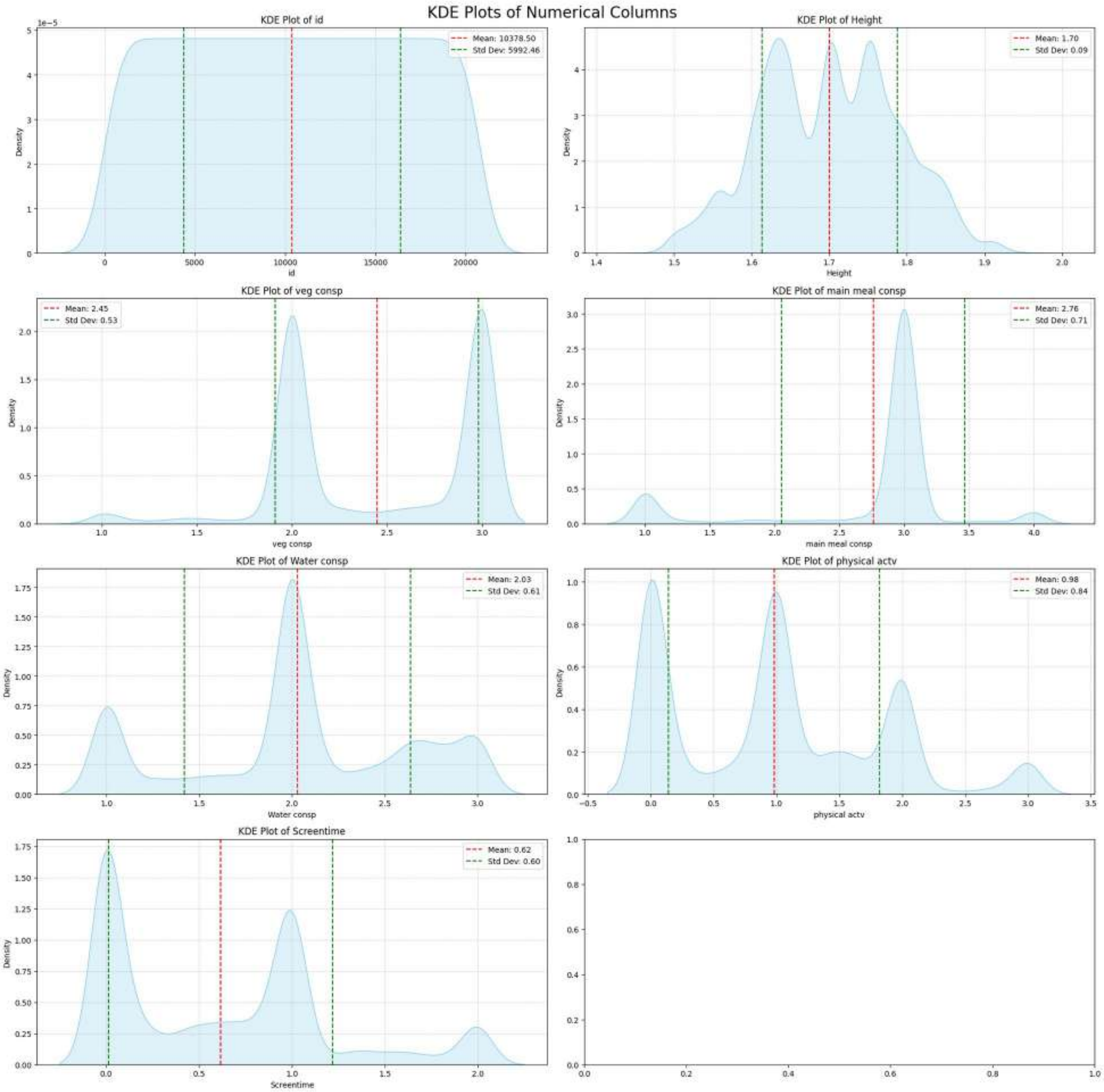
```
plt.grid(True, linestyle='--', alpha=0.5)
mean_water_consp = df_train['Water consp'].mean()
median_water_consp = df_train['Water consp'].median()
plt.axvline(x=mean_water_consp, color='red', linestyle='--',
label=f'Mean: {mean_water_consp:.2f}')
plt.axvline(x=median_water_consp, color='green', linestyle='--',
label=f'Median: {median_water_consp:.2f}')
plt.legend()

# Physical activity
plt.subplot(3, 3, 9)
sns.histplot(df_train['physical actv'], kde=True, bins=15,
color='lightblue')
plt.title('Distribution of Physical Activity', fontsize=16)
plt.xlabel('Physical Activity', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
mean_physical_actv = df_train['physical actv'].mean()
median_physical_actv = df_train['physical actv'].median()
plt.axvline(x=mean_physical_actv, color='red', linestyle='--',
label=f'Mean: {mean_physical_actv:.2f}')
plt.axvline(x=median_physical_actv, color='green', linestyle='--',
label=f'Median: {median_physical_actv:.2f}')
plt.legend()

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



c. KDE Plots of Numerical Columns:

```
# Define numerical_cols
numerical_cols = df_train.select_dtypes(include=['float64',
'int64']).columns

# Function to plot KDE density for numerical columns in three plots
per row
def plot_kde_density(df):
    num_plots = len(numerical_cols)
    num_rows = (num_plots + 2) // 2  # Calculate number of rows
required

    fig, axes = plt.subplots(num_rows, 2, figsize=(20, 5*num_rows))
    fig.suptitle('KDE Plots of Numerical Columns', fontsize=20)

    for i, col in enumerate(numerical_cols):
        row = i // 2
        col_idx = i % 2
        ax = axes[row, col_idx]
```

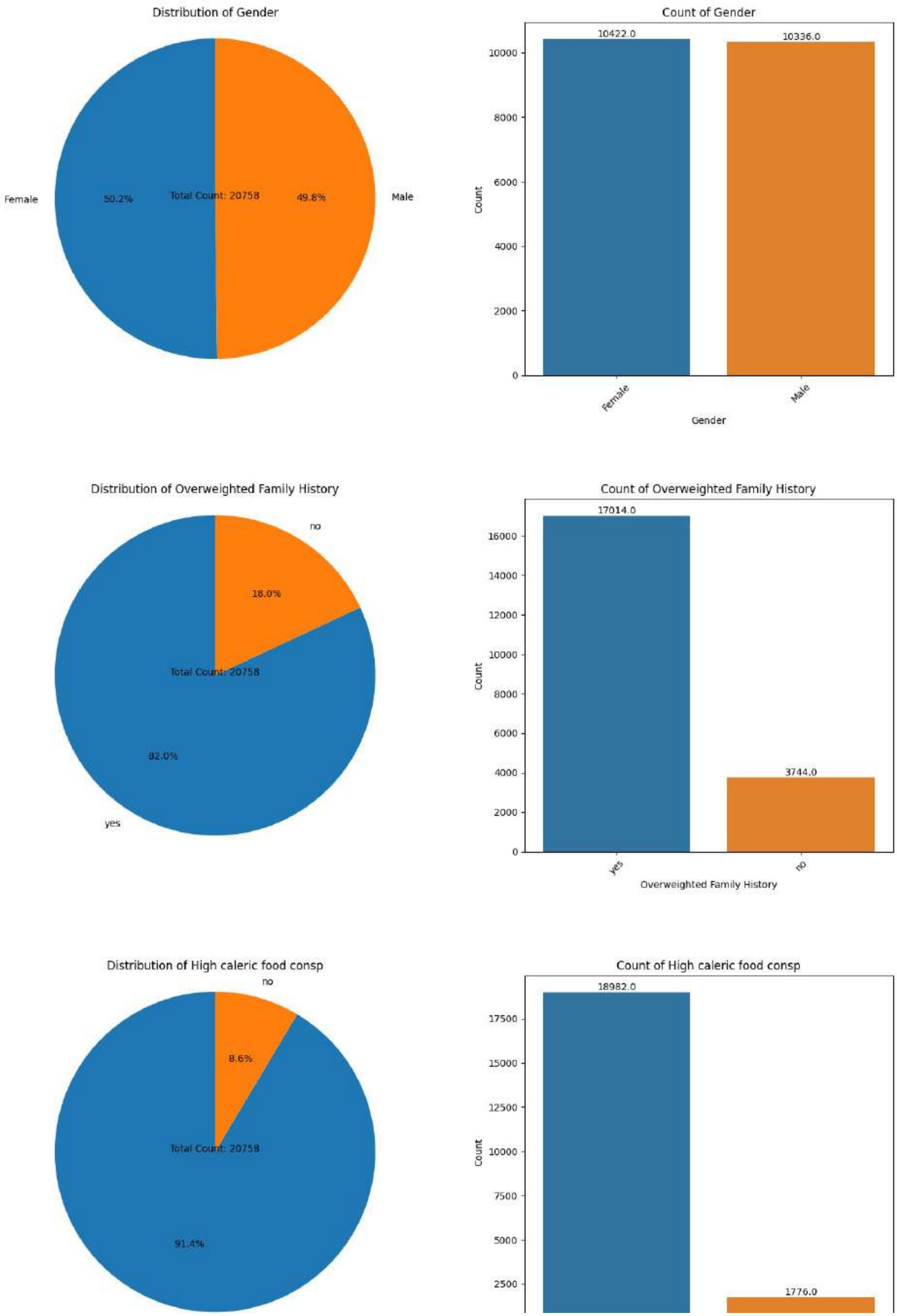
```
        sns.kdeplot(data=df[col], fill=True, color='skyblue', ax=ax)
        ax.set_xlabel(col)
        ax.set_ylabel('Density')
        ax.set_title(f'KDE Plot of {col}')

        # Add mean and standard deviation information
        mean = df[col].mean()
        std_dev = df[col].std()
        ax.axvline(x=mean, linestyle='--', color='red', label=f'Mean:
{mean:.2f}')
        ax.axvline(x=mean - std_dev, linestyle='--', color='green',
label=f'Std Dev: {std_dev:.2f}')
        ax.axvline(x=mean + std_dev, linestyle='--', color='green')
        ax.legend()

        # Add grid lines for better visualization
        ax.grid(True, linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.show()

# Call the function to plot KDE density for numerical columns in
df_train
plot_kde_density(df_train)
```



d. Pie Chart and Barplot for categorical variables:

```
def plot_data(df):
    """
    Plot different types of plots for each categorical column in the
DataFrame.

    Parameters:
        df (DataFrame): The input DataFrame containing categorical
```

```
columns.

    Returns:
        None
    """
    # Selecting categorical columns
    categorical_cols = df.select_dtypes(include=['object']).columns

    # Create subplots
    fig, axes = plt.subplots(len(categorical_cols), 2, figsize=(14,
7*len(categorical_cols)))

    # Plotting pie chart for each categorical variable in the first
column
    for i, col in enumerate(categorical_cols):
        ax = axes[i, 0]
        value_counts = df[col].value_counts()
        ax.pie(value_counts, labels=value_counts.index, autopct='%1.1f
%%', startangle=90)
        ax.set_title(f'Distribution of {col}')
        ax.set_ylabel('')
        ax.axis('equal')  # Equal aspect ratio ensures that pie is
drawn as a circle
        ax.annotate(f'Total Count: {len(df[col])}', xy=(0, 0),
fontsize=10, ha="center")

    # Plotting bar plot for each categorical variable in the second
column
    for i, col in enumerate(categorical_cols):
        ax = axes[i, 1]
        value_counts = df[col].value_counts()
        sns.barplot(x=value_counts.index, y=value_counts, ax=ax)
        ax.set_title(f'Count of {col}')
        ax.set_xlabel(f'{col}')
        ax.set_ylabel('Count')
        ax.tick_params(axis='x', rotation=45)  # Rotate x-axis labels
for better readability
        for patch in ax.patches:
            ax.annotate(f'{patch.get_height()}', (patch.get_x() +
patch.get_width() / 2., patch.get_height()),
                        ha='center', va='center', fontsize=10,
color='black', xytext=(0, 5),
                        textcoords='offset points')

    plt.tight_layout()
    plt.show()

# Call the function to plot different types of plots for df_train
plot_data(df_train)
```

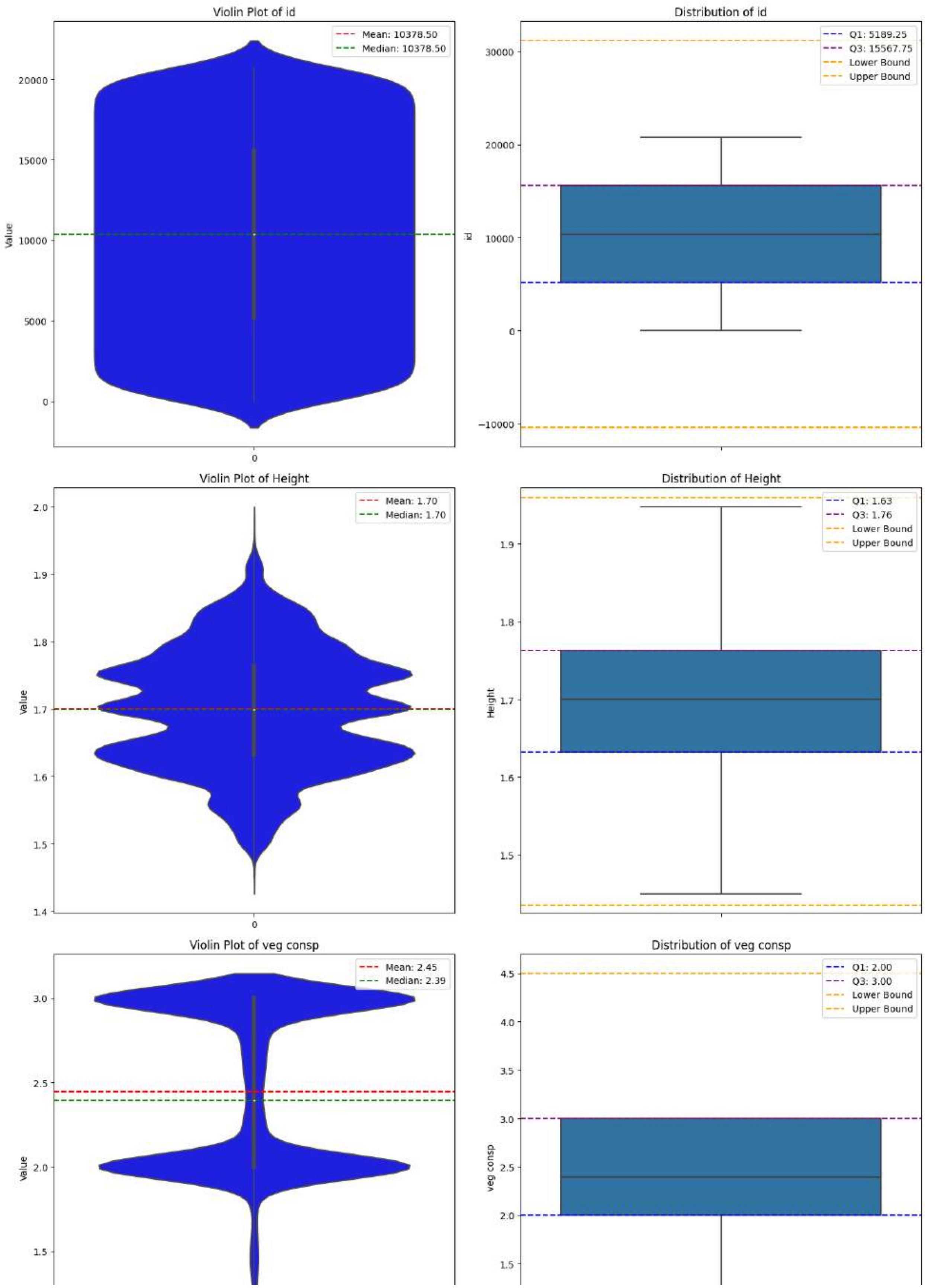


Distribution of Gender
Female
50.2%
Total Count: 20758
49.8%
Male
Count of Gender
10422.0
10336.0
Count
Female
Male
Gender
Distribution of Overweighted Family History
no
18.0%
Total Count: 20758
82.0%
yes
Count of Overweighted Family History
17014.0
3744.0
Count
yes
no
Overweighted Family History
Distribution of High caleric food consp
no
8.6%
Total Count: 20758
91.4%
Count of High caleric food consp
18982.0
1776.0
Count

e. Violin Plot and Box Plot for Numerical variables:

```
def plot_data(df):
    """
    Plot different types of plots for each column in the DataFrame.

    Parameters:
        df (DataFrame): The input DataFrame.

    Returns:
        None
    """
    numerical_cols = df.select_dtypes(include=['float64',
'int64']).columns

    # Create subplots
    fig, axes = plt.subplots(len(numerical_cols), 2, figsize=(14,
7*len(numerical_cols)))

    # Plotting violin plot for each numerical variable in the first
column
    for i, col in enumerate(numerical_cols):
        ax = axes[i, 0]
        sns.violinplot(data=df[col], ax=ax, color='blue')
        ax.set_title(f'Violin Plot of {col}')
        ax.set_xlabel('')
        ax.set_ylabel('Value')
        # Add statistical information
        mean = df[col].mean()
        median = df[col].median()
        ax.axhline(y=mean, color='red', linestyle='--', label=f'Mean:
{mean:.2f}')
        ax.axhline(y=median, color='green', linestyle='--',
label=f'Median: {median:.2f}')
        ax.legend()

    # Plotting box plot for each numerical variable in the second
column
    for i, col in enumerate(numerical_cols):
        ax = axes[i, 1]
        sns.boxplot(data=df, y=col, ax=ax, showfliers=False)
        ax.set_title(f'Distribution of {col}')
        ax.set_ylabel(f'{col}')
        ax.set_xlabel('')  # Remove x-axis label as it represents
'Level' which is not available
        # Add statistical information
        q1 = df[col].quantile(0.25)
```

```
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1
        ax.axhline(y=q1, color='blue', linestyle='--', label=f'Q1:
{q1:.2f}')
        ax.axhline(y=q3, color='purple', linestyle='--', label=f'Q3:
{q3:.2f}')
        ax.axhline(y=q1 - 1.5 * iqr, color='orange', linestyle='--',
label=f'Lower Bound')
        ax.axhline(y=q3 + 1.5 * iqr, color='orange', linestyle='--',
label=f'Upper Bound')
        ax.legend()

    plt.tight_layout()
    plt.show()

# Call the function to plot different types of plots for df_train
plot_data(df_train)
```

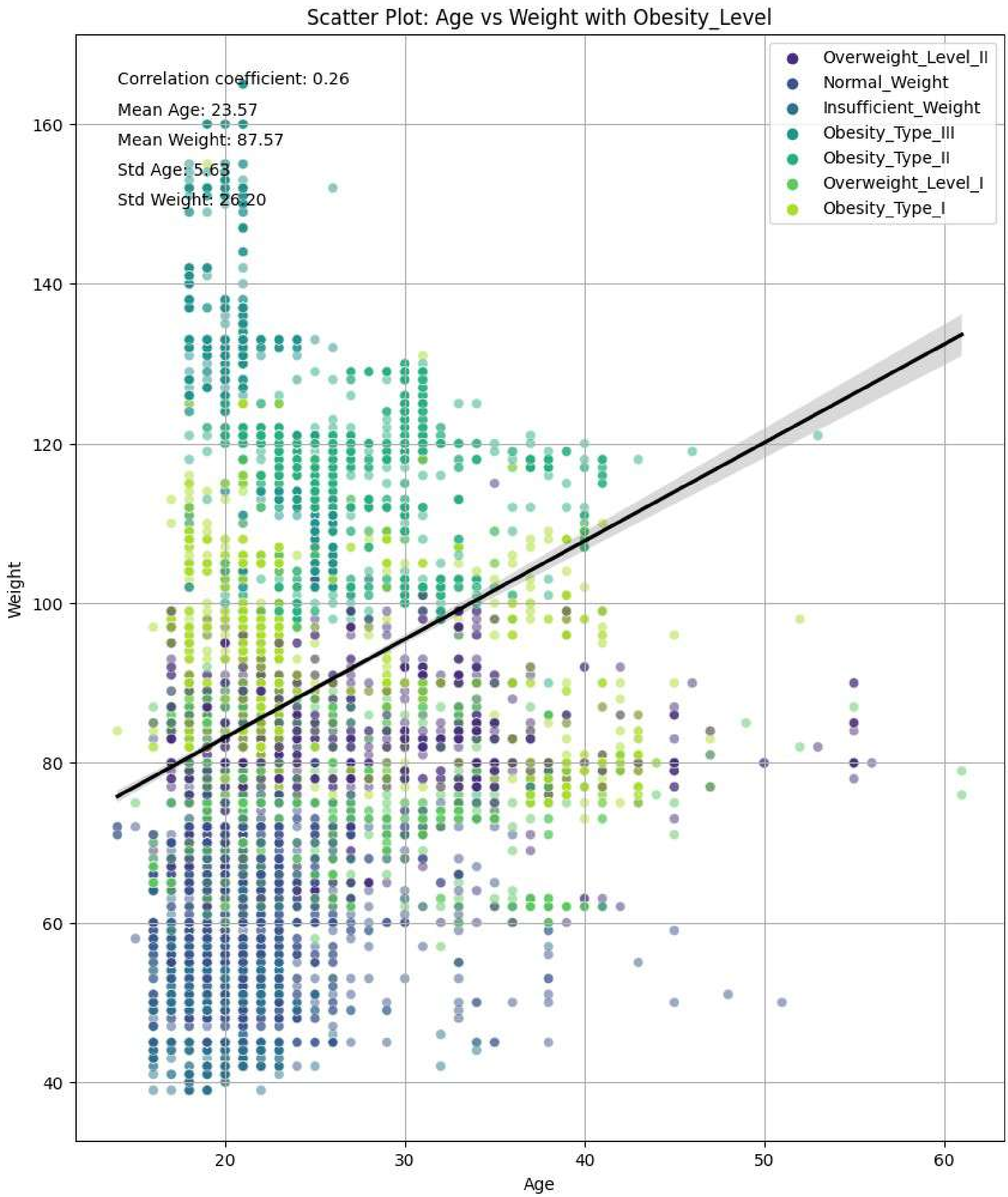

Violin Plot of id
Mean: 10378.50
Median: 10378.50
Value
Distribution of id
Q1: 5189.25
Q3: 15567.75
Lower Bound
Upper Bound
id
Violin Plot of Height
Mean: 1.70
Median: 1.70
Value
Distribution of Height
Q1: 1.63
Q3: 1.76
Lower Bound
Upper Bound
Height
Violin Plot of veg consp
Mean: 2.45
Median: 2.39
Value
Distribution of veg consp
Q1: 2.00
Q3: 3.00
Lower Bound
Upper Bound
veg consp

2. Bivariate Analysis:

```
def plot_scatter_relationship(col1, col2, target=None, data=None):
    plt.figure(figsize=(10, 12))

    # Plotting the scatter plot
    sns.scatterplot(data=data, x=col1, y=col2, hue=target,
palette='viridis', alpha=0.5)

    # Calculating correlation coefficient
    corr_coef, _ = pearsonr(data[col1], data[col2])

    # Adding regression lines
    sns.regplot(data=data, x=col1, y=col2, scatter=False,
color='black')

    # Adding statistical summary
    plt.text(data[col1].min(), data[col2].max(), f'Correlation
coefficient: {corr_coef:.2f}', fontsize=10)
    plt.text(data[col1].min(), data[col2].max() - 0.03 *
(data[col2].max() - data[col2].min()), f'Mean {col1}:
{data[col1].mean():.2f}', fontsize=10)
    plt.text(data[col1].min(), data[col2].max() - 0.06 *
(data[col2].max() - data[col2].min()), f'Mean {col2}:
{data[col2].mean():.2f}', fontsize=10)
    plt.text(data[col1].min(), data[col2].max() - 0.09 *
(data[col2].max() - data[col2].min()), f'Std {col1}:
{data[col1].std():.2f}', fontsize=10)
    plt.text(data[col1].min(), data[col2].max() - 0.12*
(data[col2].max() - data[col2].min()), f'Std {col2}:
{data[col2].std():.2f}', fontsize=10)

    plt.xlabel(col1)
    plt.ylabel(col2)
    plt.title(f'Scatter Plot: {col1} vs {col2} with {target}')
    plt.grid(True)
    plt.legend()
    plt.show()
```

a. Scatter plot: AGE V/s Weight with Obesity Level:

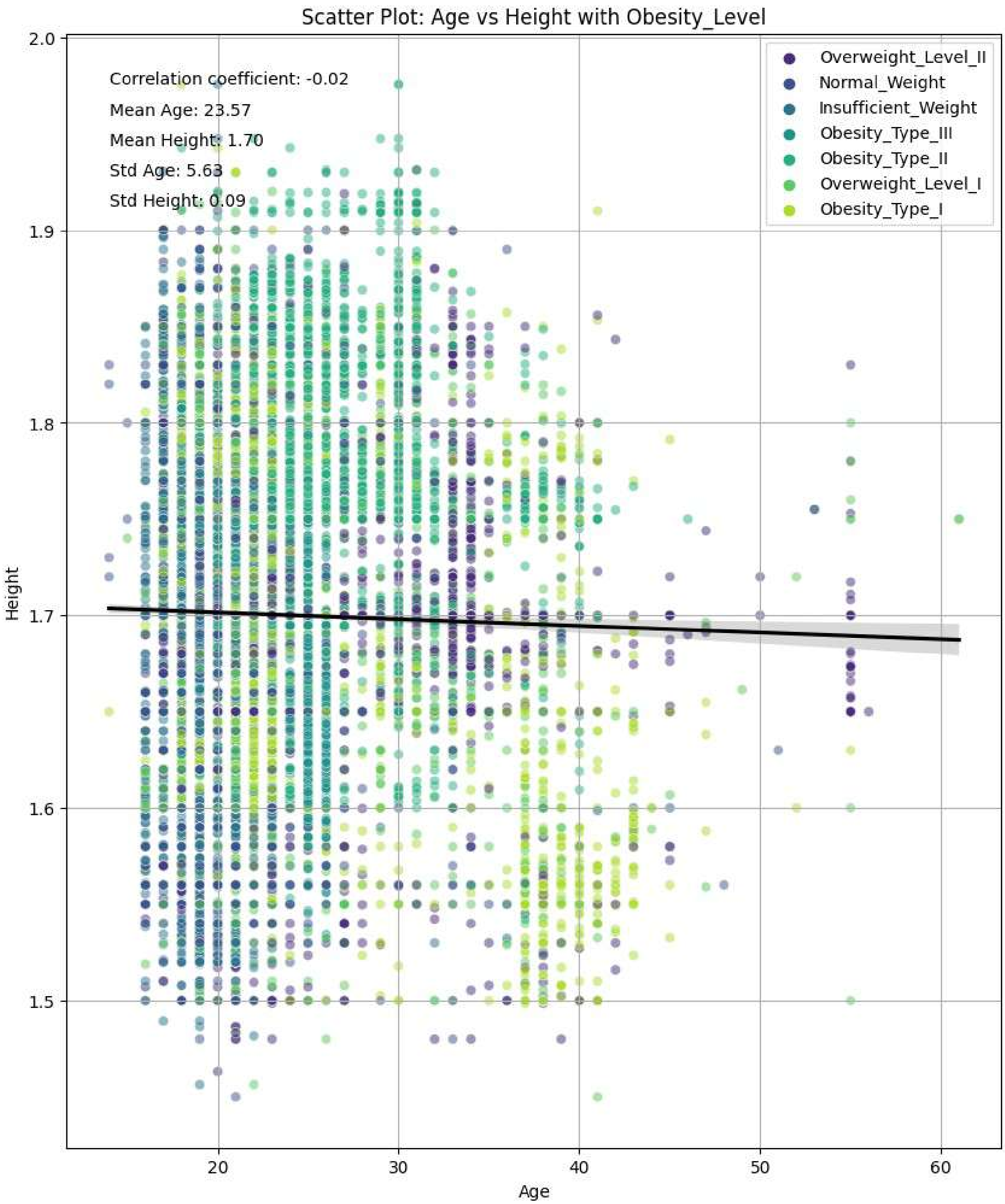
```
plot_scatter_relationship('Age','Weight','Obesity_Level', df_train)
```



Scatter Plot: Age vs Weight with Obesity_Level
Correlation coefficient: 0.26
Mean Age: 23.57
Mean Weight: 87.57
Std Age: 5.63
Std Weight: 26.20
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
Weight
Age
160
140
120
100
80
60
40
20
30
40
50
60

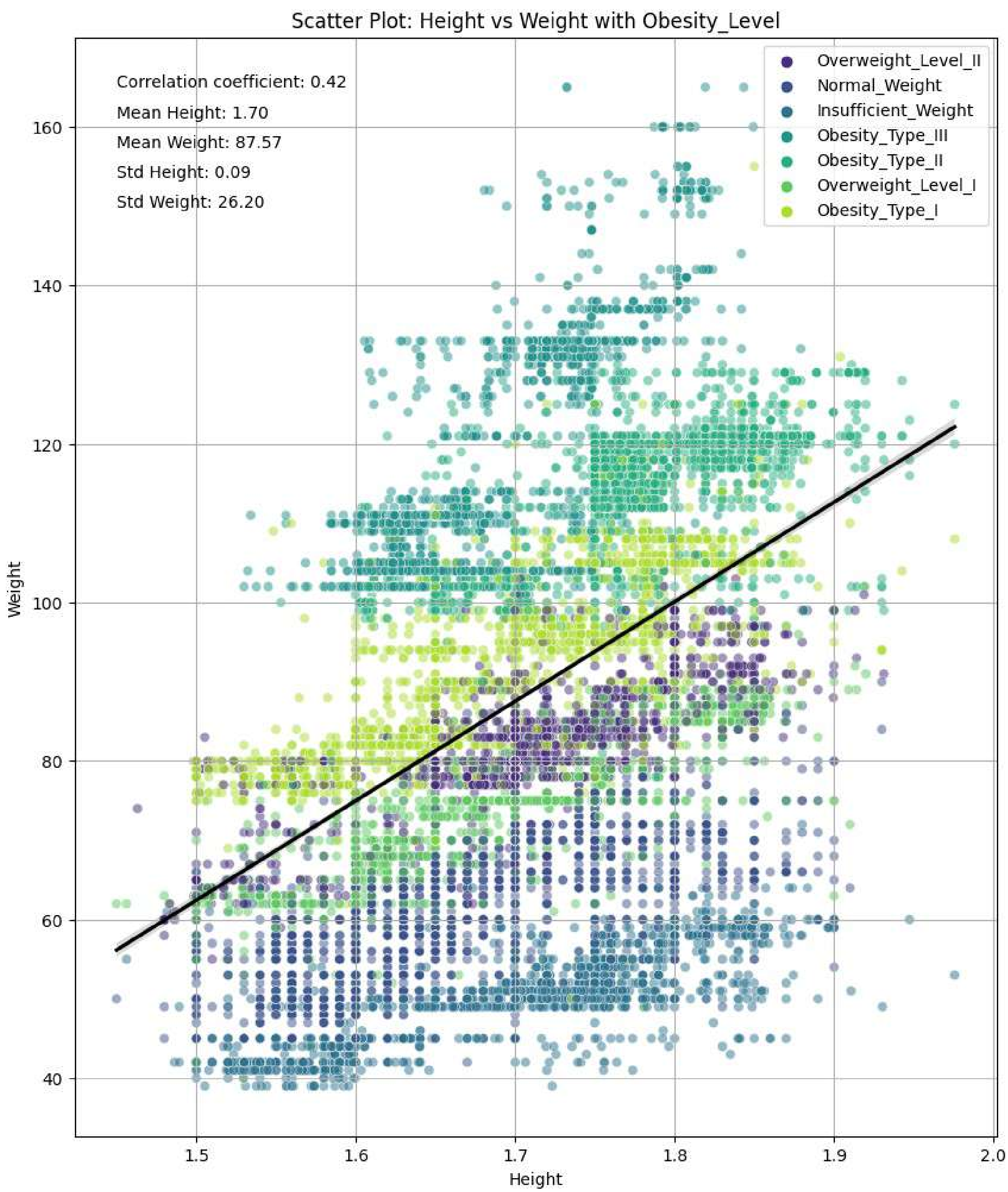
b. Scatter plot: AGE V/s Height with Obesity Level:

```
plot_scatter_relationship('Age','Height','Obesity_Level',df_train)
```



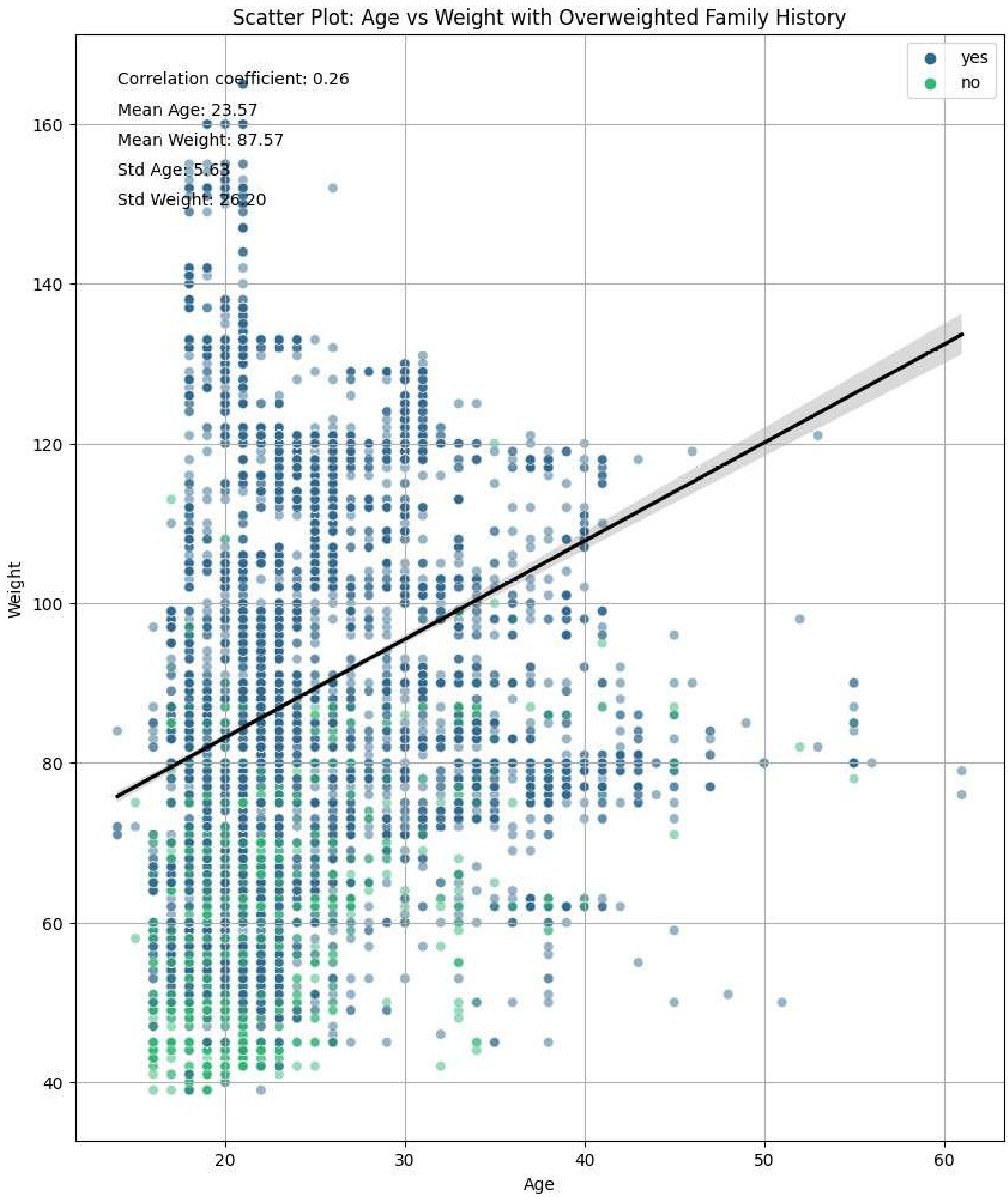
c. Scatter plot: Height V/s Weight with Obesity Level:

```
plot_scatter_relationship('Height','Weight','Obesity_Level',df_train)
```



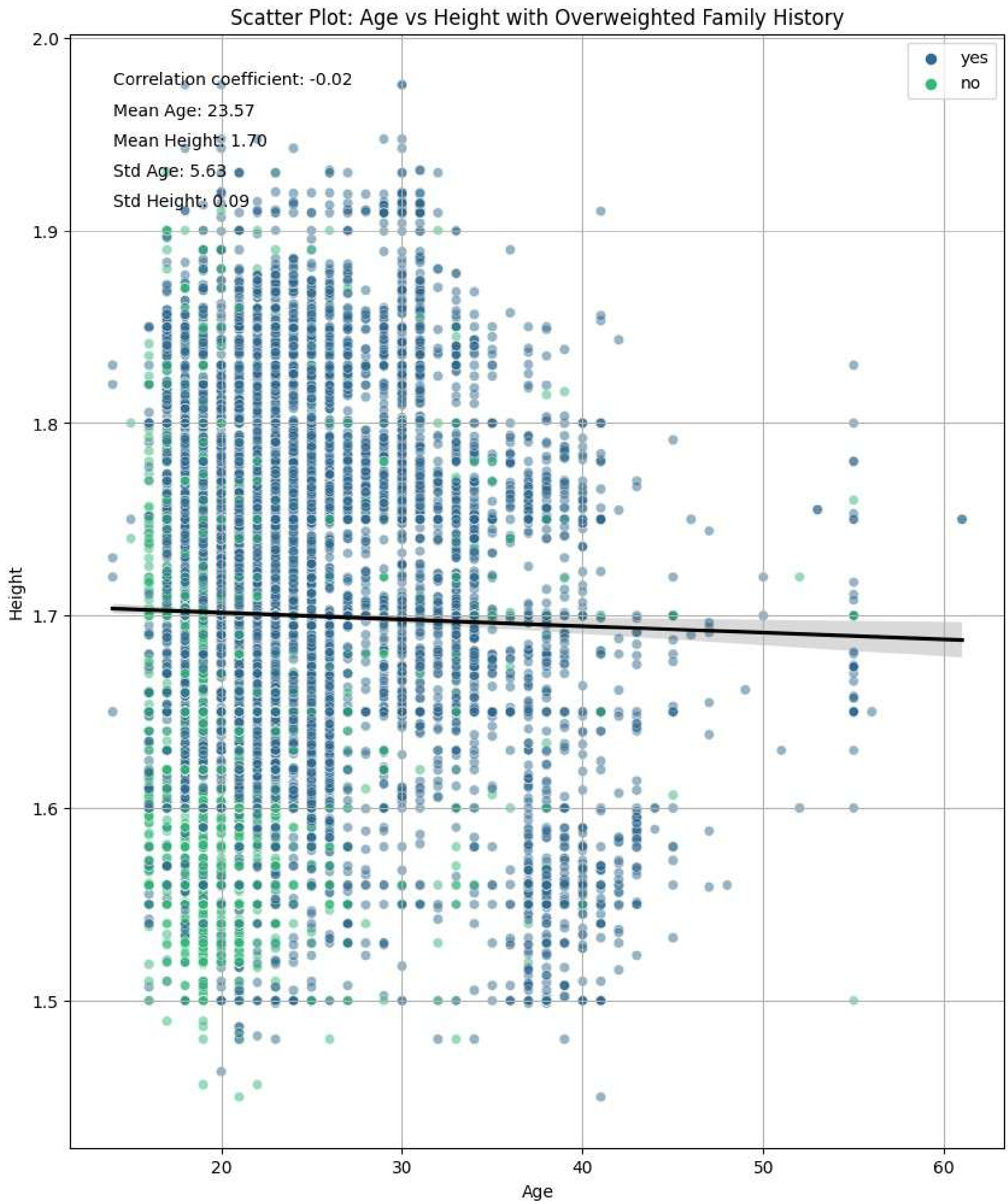
d. Scatter plot: AGE V/s Weight with Overweighted Family History:

```
plot_scatter_relationship('Age','Weight','Overweighted Family
History',df_train)
```


Scatter Plot: Age vs Weight with Overweighted Family History
Correlation coefficient: 0.26
Mean Age: 23.57
Mean Weight: 87.57
Std Age: 5.63
Std Weight: 26.20
yes
no
Weight
Age
40
60
80
100
120
140
160
20
30
40
50
60

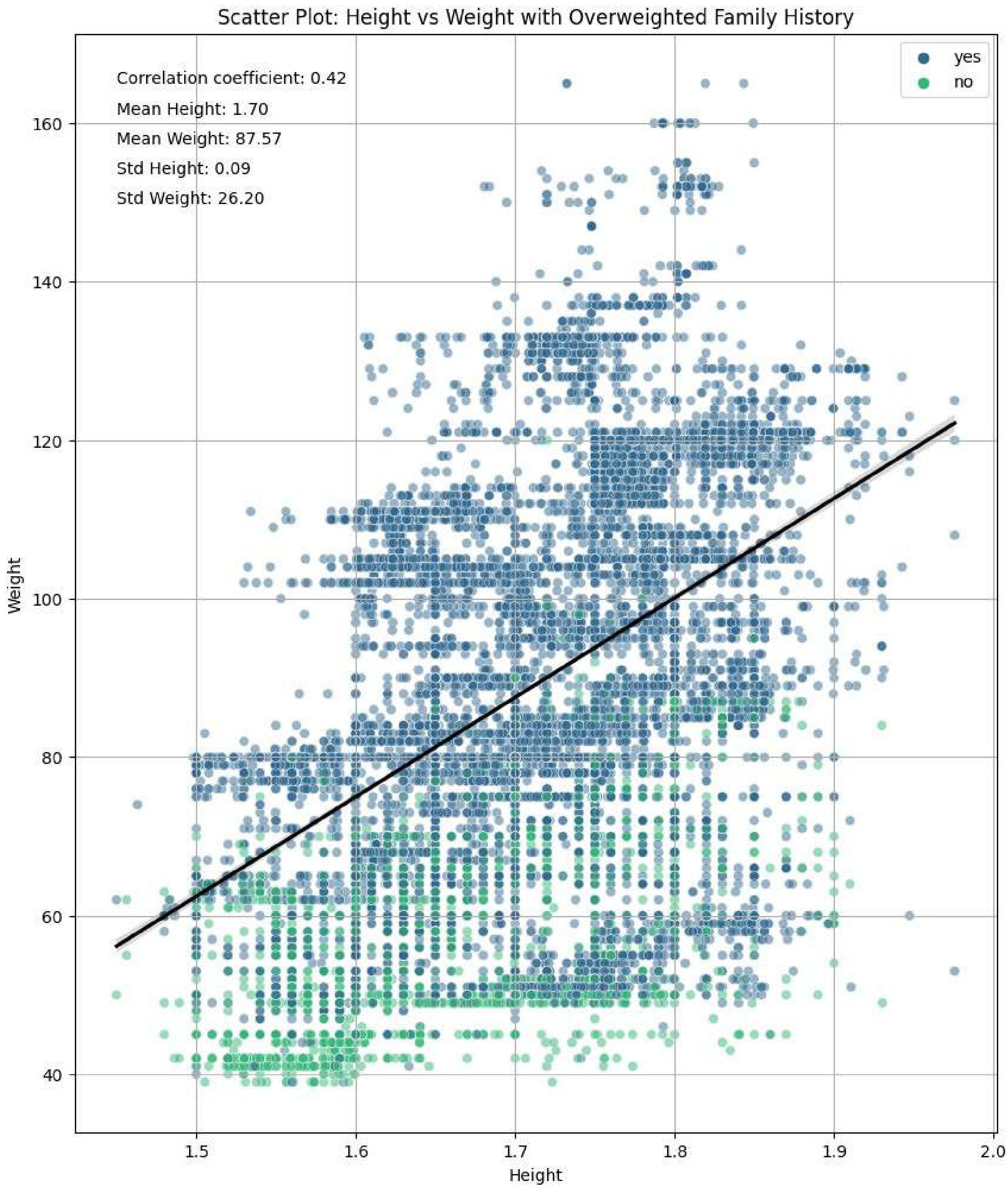
e. Scatter plot: AGE V/s height with Overweighted Family History:

```
plot_scatter_relationship('Age','Height','Overweighted Family
History',df_train)
```


Scatter Plot: Age vs Height with Overweighted Family History
Correlation coefficient: -0.02
Mean Age: 23.57
Mean Height: 1.70
Std Age: 5.63
Std Height: 0.09
yes
no
Height
Age
2.0
1.9
1.8
1.7
1.6
1.5
20
30
40
50
60

f. Scatter plot: Height V/s Weight with Overweighted Family History:

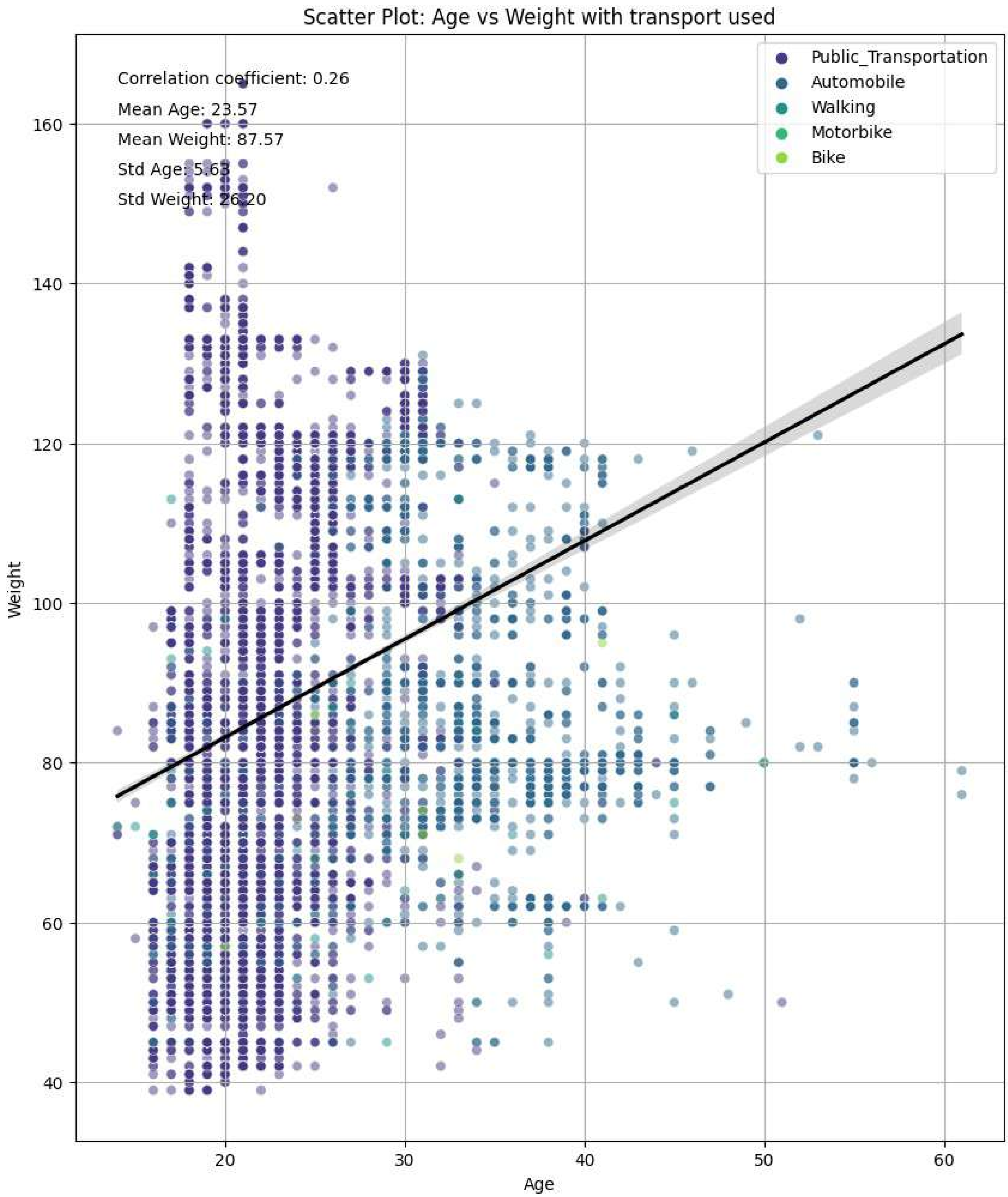
```
plot_scatter_relationship('Height','Weight','Overweighted Family 
History',df_train)
```



Scatter Plot: Height vs Weight with Overweighted Family History
Correlation coefficient: 0.42
Mean Height: 1.70
Mean Weight: 87.57
Std Height: 0.09
Std Weight: 26.20
yes
no
Weight
Height
160
140
120
100
80
60
40
1.5
1.6
1.7
1.8
1.9
2.0

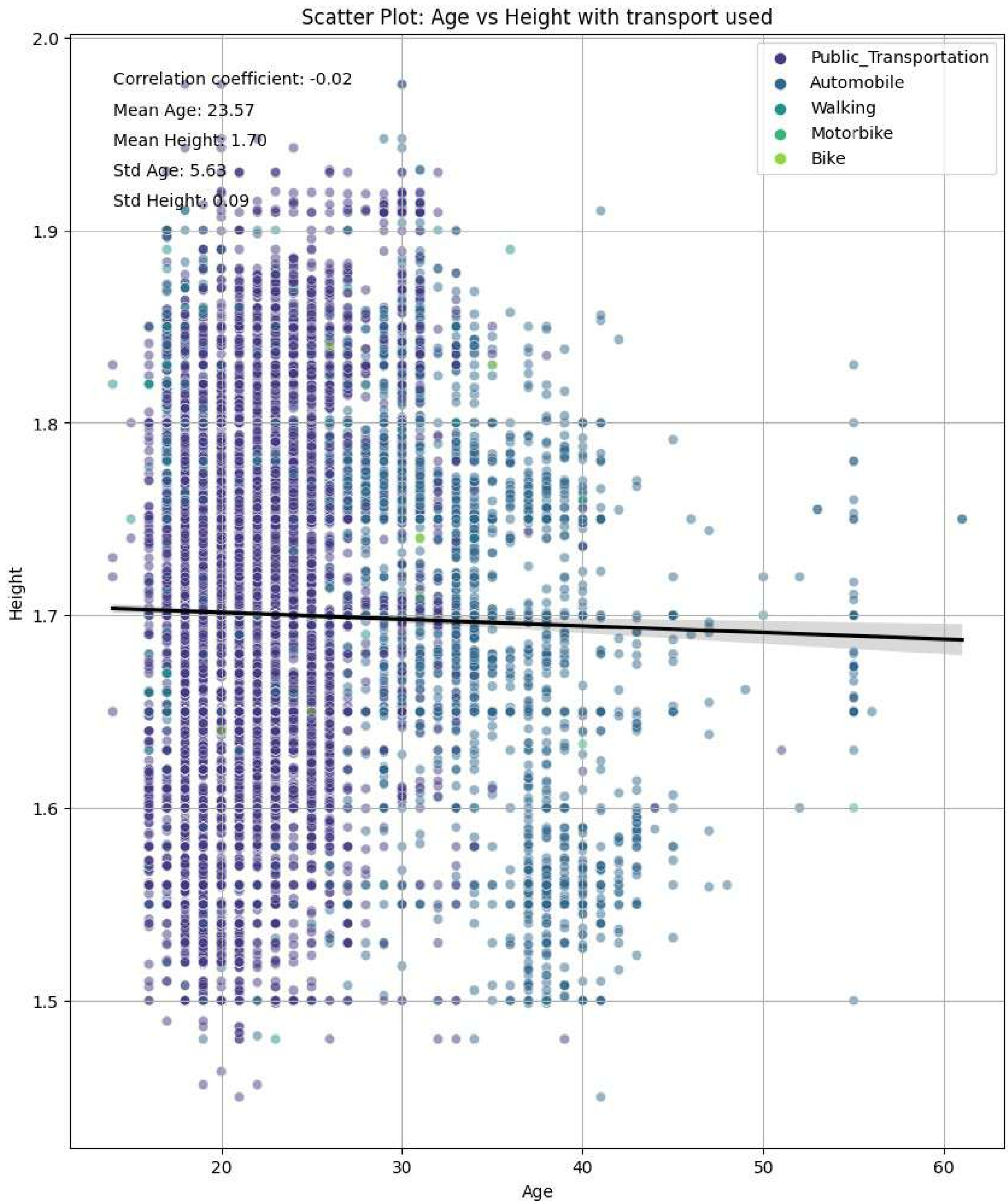
g. Scatter plot: AGE V/s Weight with Transport use:

```
plot_scatter_relationship('Age','Weight','transport used',df_train)
```



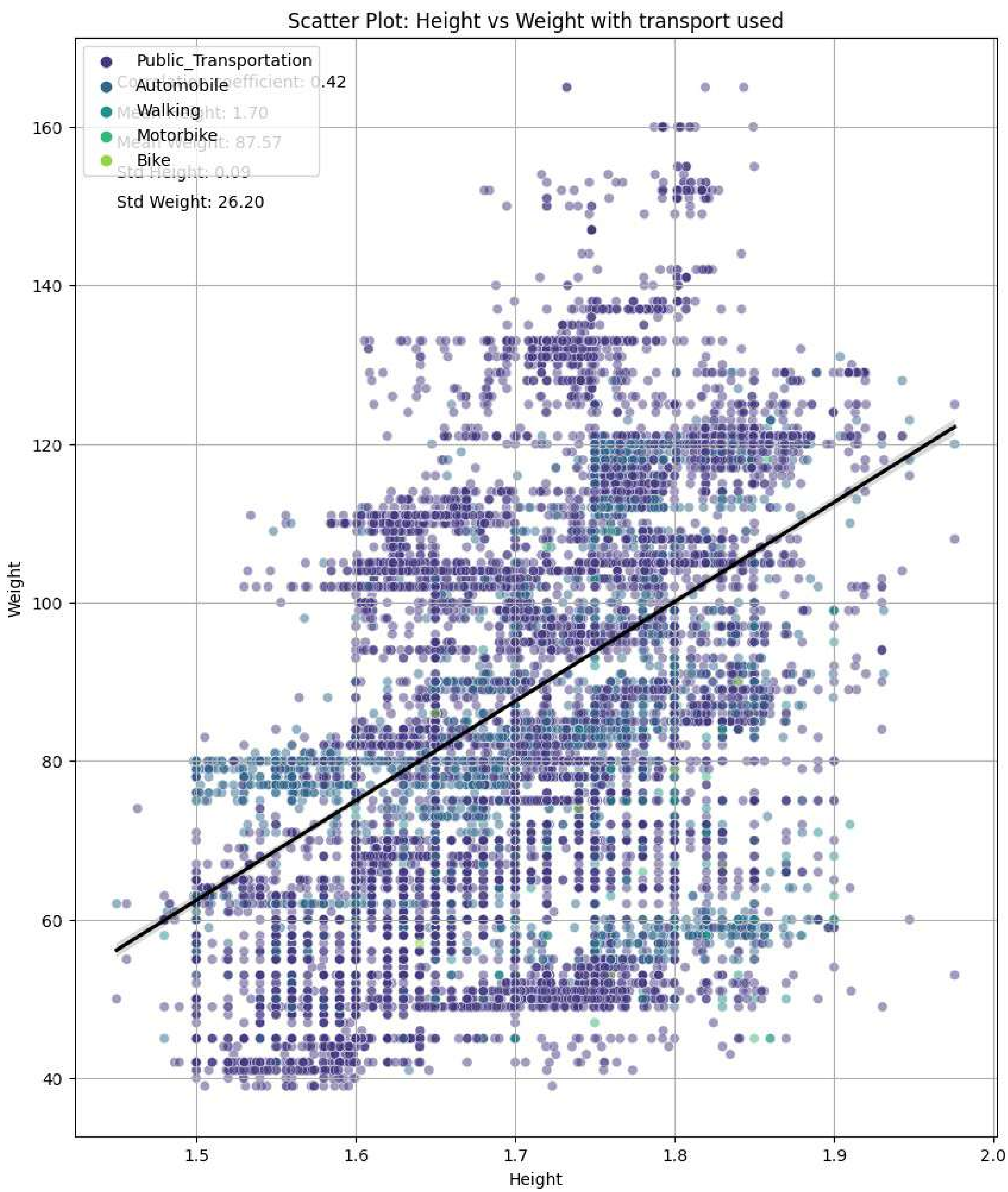
h. Scatter plot: AGE V/s Height with Transport use:

```
plot_scatter_relationship('Age','Height','transport used',df_train)
```



i. Scatter plot: Height V/s Weight with Transport use:

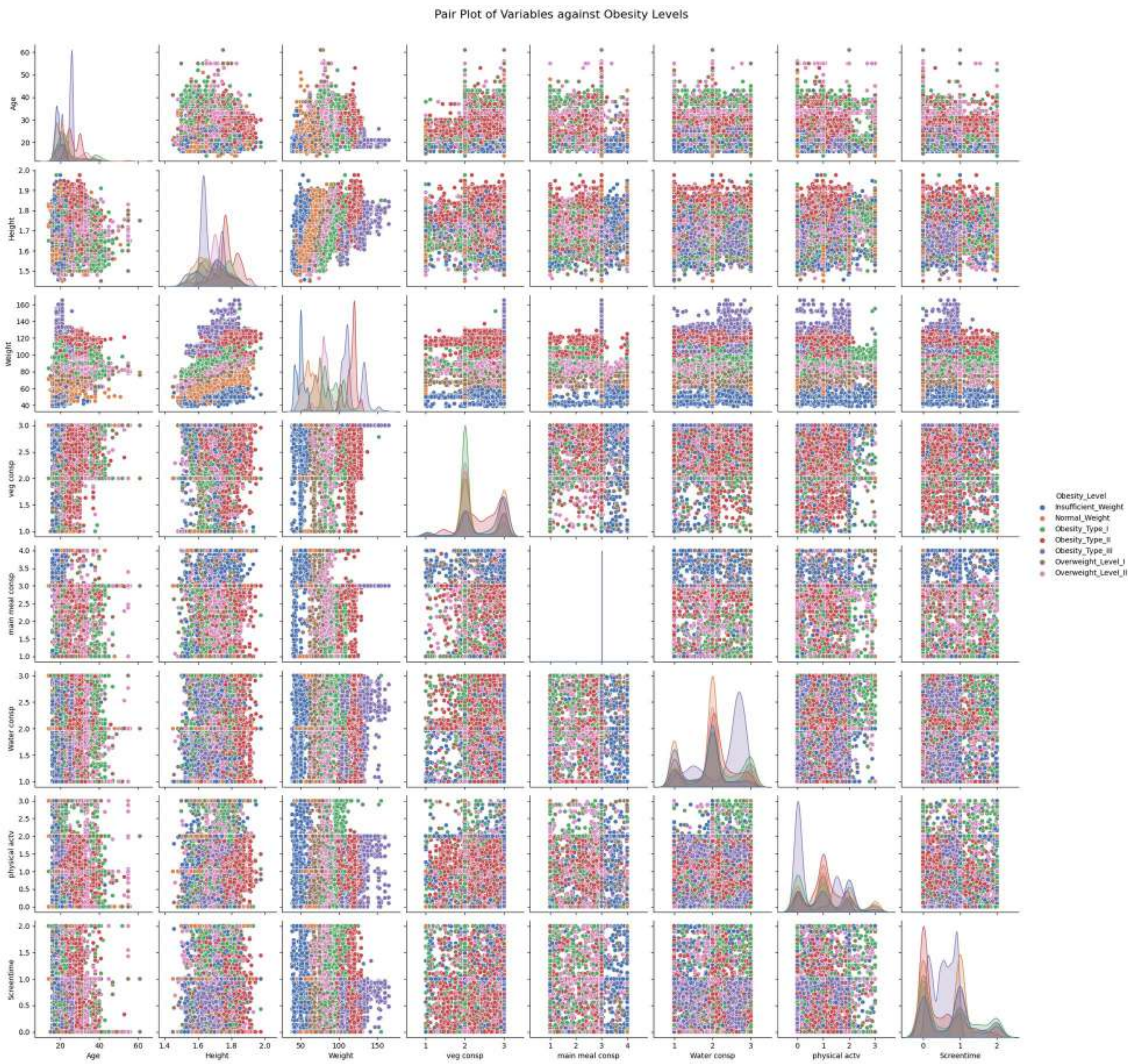
```
plot_scatter_relationship('Height','Weight','transport used',df_train)
```



3. Multivariate Analysis:

a. Pair Plot of Variables against Obesity Levels:

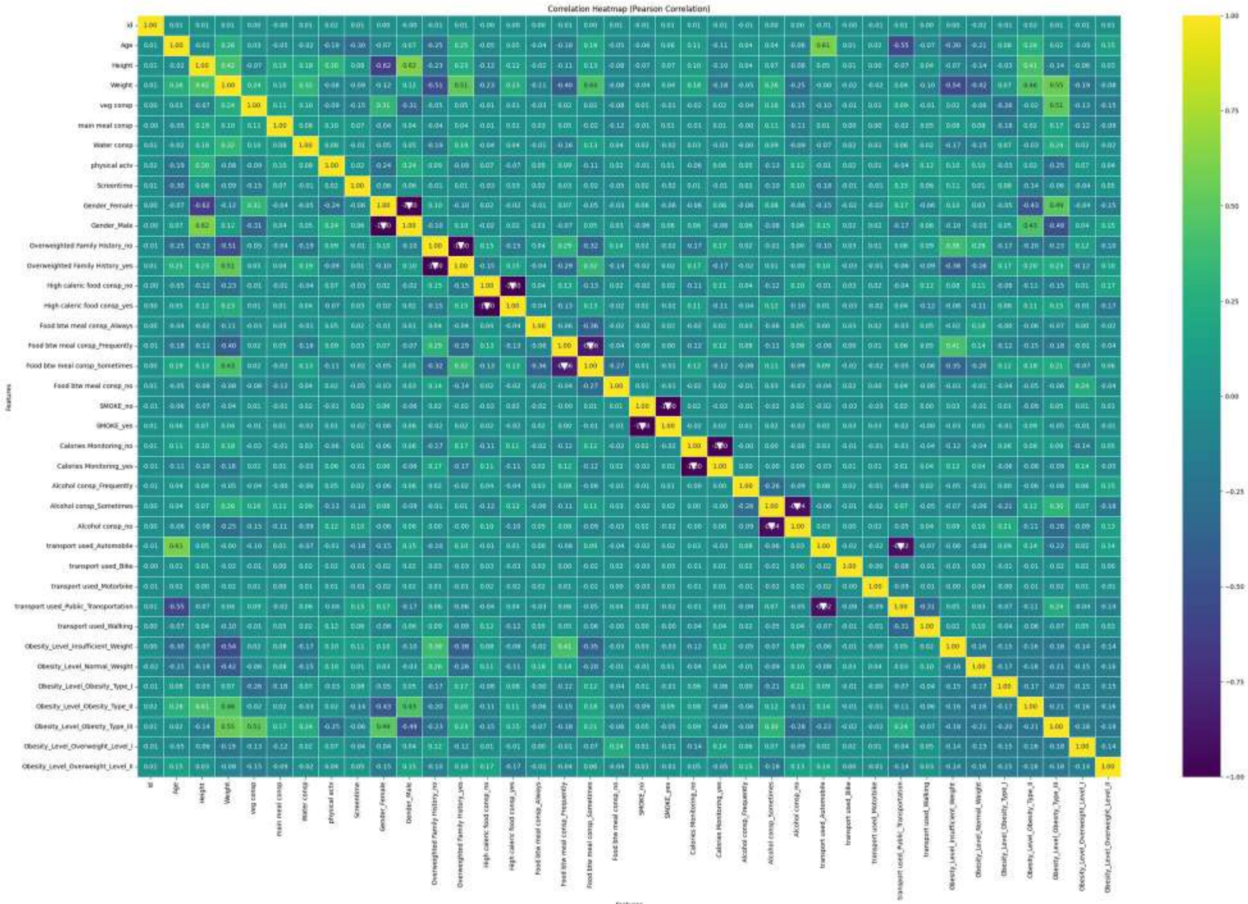
```
# Selecting numerical columns for pairplot
numerical_columns = ['Age', 'Height', 'Weight', 'High caleric food
consp', 'veg consp', 'main meal consp',
                     'Food btw meal consp', 'Water consp', 'Calories
Monitoring', 'physical actv', 'Screentime',
                     'Alcohol consp']

# Add the target variable 'Obesity_Level' for hue
df_train['Obesity_Level'] =
df_train['Obesity_Level'].astype('category')

# Create pair plot
pair_plot = sns.pairplot(df_train[numerical_columns +
['Obesity_Level']], hue='Obesity_Level', palette='deep',
diag_kind='kde')

# Add title to the plot
pair_plot.fig.suptitle('Pair Plot of Variables against Obesity
Levels', fontsize=16, y=1.02)

# Display the plot
plt.show()
```



b. Correlation heatmap for Pearson's correlation coefficient:

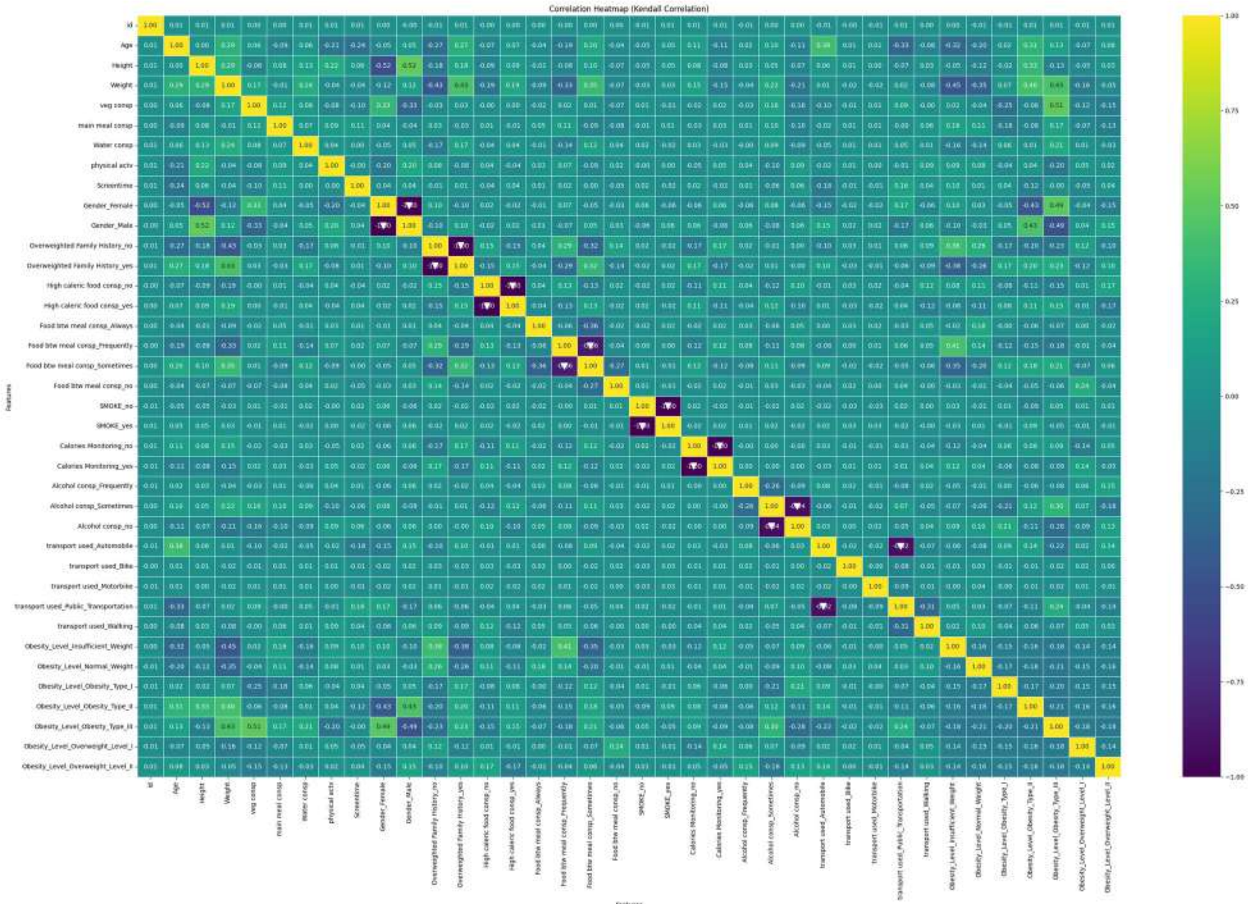
```
def plot_correlation_heatmap(df, method='pearson'):
    # Calculate the correlation matrix
    corr_matrix = df.corr(method=method)

    # Plot the heatmap
    plt.figure(figsize=(30, 20))
    sns.heatmap(corr_matrix, annot=True, cmap='viridis', fmt=".2f",
linewidths=.5, cbar=True)
```

```
    # Add indicators for strength and direction of correlation
    for i in range(len(corr_matrix)):
        for j in range(len(corr_matrix.columns)):
            if i != j:
                if corr_matrix.iloc[i, j] >= 0.7:
                    plt.text(j + 0.5, i + 0.5, '\u25B2', ha='center',
va='center', color='white', fontsize=15)
                elif corr_matrix.iloc[i, j] <= -0.7:
                    plt.text(j + 0.5, i + 0.5, '\u25BC', ha='center',
va='center', color='white', fontsize=15)

    # Set labels and title
    plt.title(f'Correlation Heatmap ({method.capitalize()}
Correlation)')
    plt.xlabel('Features')
    plt.ylabel('Features')

    # Adjust layout
    plt.tight_layout()

    # Show plot
    plt.show()

# Perform one-hot encoding for categorical variables
df_train_encoded = pd.get_dummies(df_train)

# Plot correlation heatmap for Pearson ,spearman and kendell
correlation coefficient(in my case using kendell's tau)
plot_correlation_heatmap(df_train_encoded, method='pearson')
```



c. Correlation heatmap for Kendall's tau correlation coefficient:

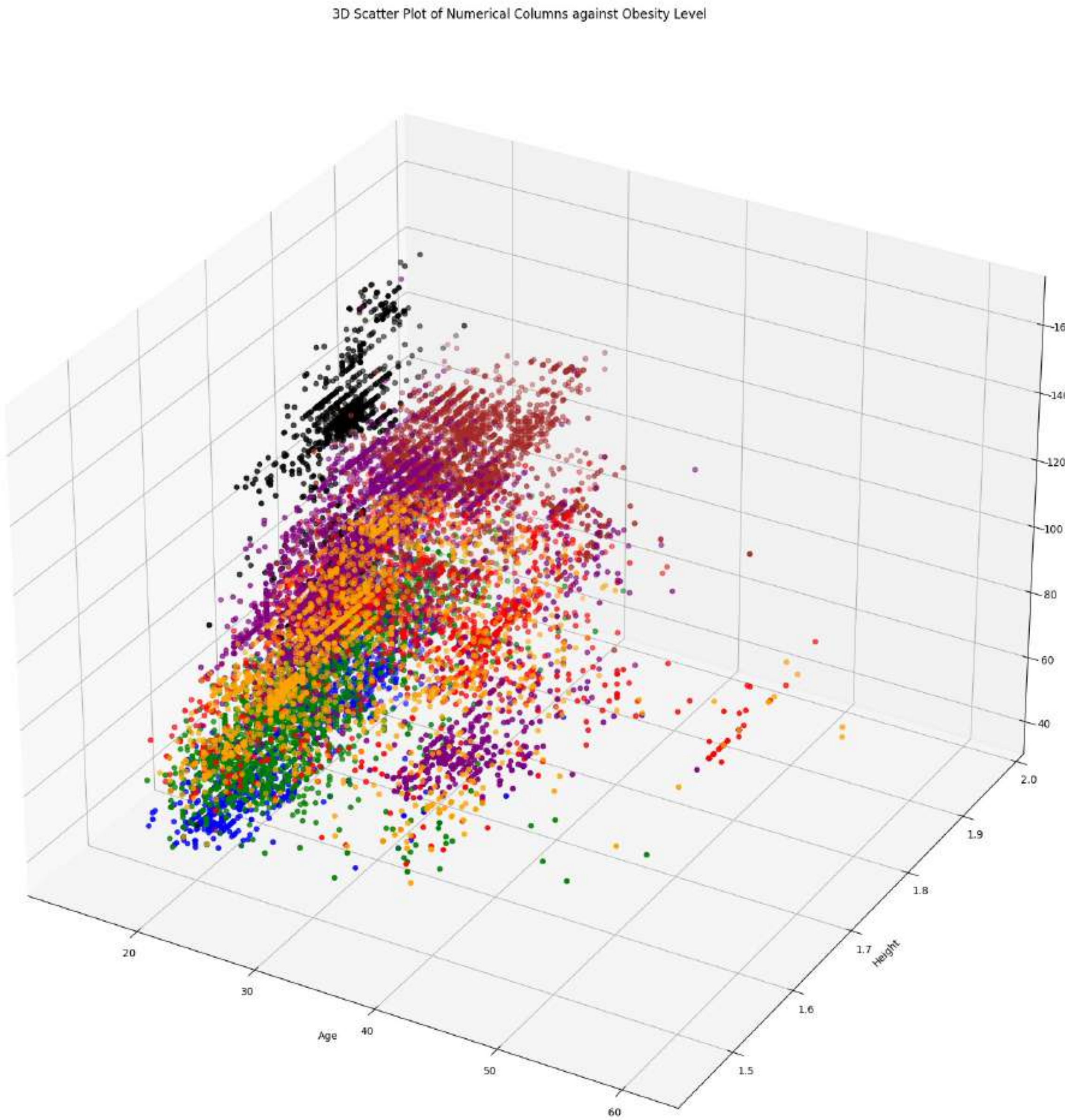
```
# Plot correlation heatmap for Kendall's tau correlation coefficient
plot_correlation_heatmap(df_train_encoded, method='kendall')
```



d. 3D Scatter Plot of Numerical Columns against Obesity Level:

```
# Define numerical columns for the plot
numerical_columns = ['Age', 'Height', 'Weight', 'High caleric food consp', 'veg consp', 'main meal consp',
                     'Food btw meal consp', 'Water consp', 'Calories Monitoring', 'physical actv', 'Screentime',
                     'Alcohol consp']

# Selecting only the numerical columns and 'Obesity_Level' from the dataframe
df_numerical = df_train[numerical_columns + ['Obesity_Level']]

# Define colors for different obesity levels
color_map = {'Insufficient_Weight': 'blue',
             'Normal_Weight': 'green',
             'Overweight_Level_I': 'orange',
             'Overweight_Level_II': 'red',
             'Obesity_Type_I': 'purple',
```

```
              'Obesity_Type_II': 'brown',
              'Obesity_Type_III': 'black'}

# Create a 3D scatter plot
fig = plt.figure(figsize=(30,20))
ax = fig.add_subplot(111, projection='3d')

# Plot each obesity level separately
for obesity_level, color in color_map.items():
    df_obesity_level = df_numerical[df_numerical['Obesity_Level'] ==
obesity_level]
    ax.scatter(df_obesity_level['Age'], df_obesity_level['Height'],
df_obesity_level['Weight'], color=color, label=obesity_level)

# Set labels and title
ax.set_xlabel('Age')
ax.set_ylabel('Height')
ax.set_zlabel('Weight')
ax.set_title('3D Scatter Plot of Numerical Columns against Obesity
Level')

# Show plot
plt.show()
```

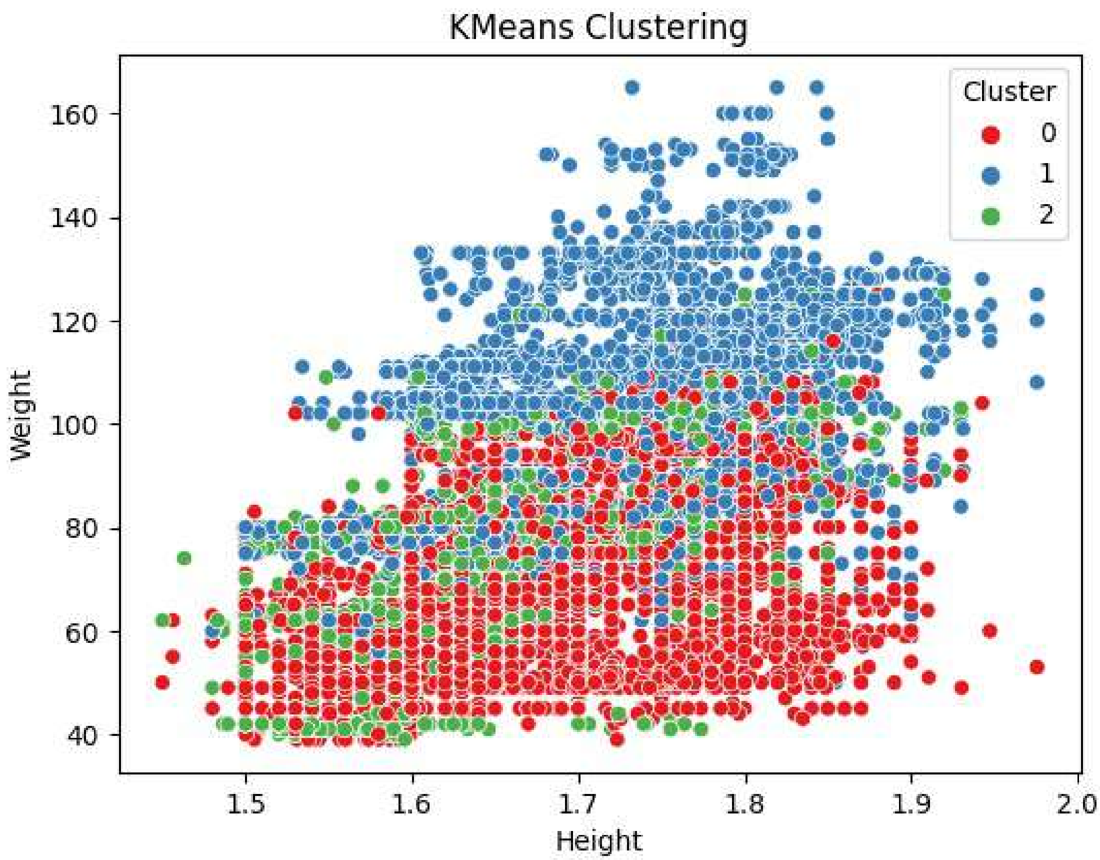


e. Cluster Analysis:

I. K-Means Clustering on Obesity level:

```
# Select numerical features for clustering
numerical_features = ['Age', 'Height', 'Weight', 'veg consp', 'main meal consp', 'Water consp', 'physical actv', 'Screentime']

# Extract numerical features from the dataframe
X = df_train[numerical_features]
```

```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Initialize and fit KMeans model
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)

# Add cluster labels to the dataframe
df_train['Cluster'] = kmeans.labels_

# Visualize the clusters (assuming 2D visualization)
sns.scatterplot(x='Height', y='Weight', hue='Cluster', data=df_train,
palette='Set1')
plt.title('KMeans Clustering')
plt.show()

# Analyze how clusters relate to obesity levels
cluster_obesity = df_train.groupby('Cluster')
['Obesity_Level'].value_counts(normalize=True).unstack()
print(cluster_obesity)
```



```
Obesity_Level  Insufficient_Weight  Normal_Weight  Obesity_Type_I  \
Cluster
0                         0.251450       0.308721        0.128387
1                         0.001270       0.019361        0.111617
2                         0.135201       0.101576        0.269702

Obesity_Level  Obesity_Type_II  Obesity_Type_III
Overweight_Level_I  \
Cluster

0                     0.006981         0.001657            0.158088

1                     0.313055         0.426576            0.045176

2                     0.080560         0.000000            0.232574


Obesity_Level  Overweight_Level_II
Cluster
0                         0.144717
1                         0.082945
2                         0.180385
```

The output provides information on how the clusters relate to different obesity levels. Each row represents a cluster, and each column represents an obesity level. The values in the table represent the proportion of individuals within each cluster belonging to a specific obesity level.

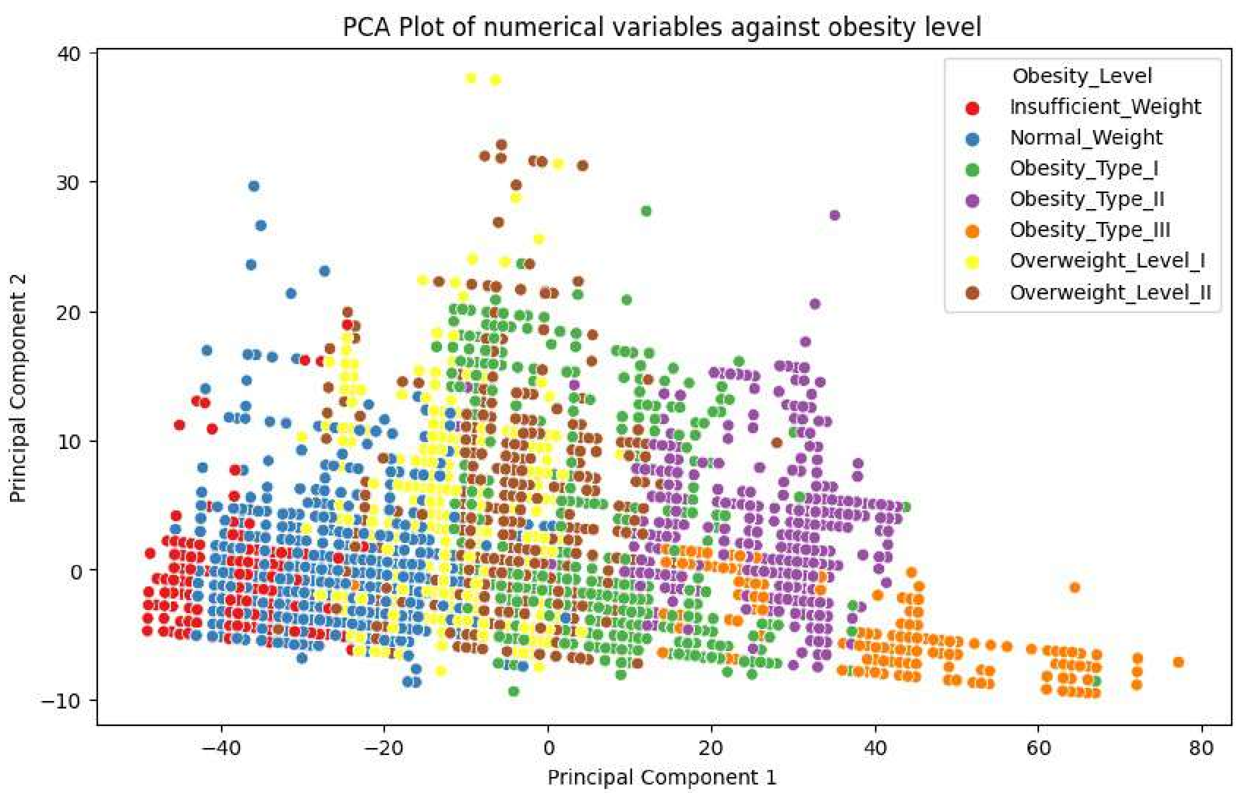
For example:

- **Cluster 0**: Majority of individuals have obesity levels 0 and 1, with smaller proportions in other levels. Level 6 also has a notable proportion in this cluster.
- **Cluster 1**: Significant proportion of individuals have obesity levels 3, 4, and 5, while levels 0 and 1 have much smaller proportions. Level 6 also has a notable proportion in this cluster.
- **Cluster 2**: Relatively balanced distribution across various obesity levels, with no individuals in level 4 and a missing value in level 5. Level 6 has a considerable proportion in this cluster.

II. PCA Plot of numerical variables against obesity level:

```
# Assuming you have numerical columns in df_train
# Select numerical columns for PCA
numerical_columns = ['Age', 'Height', 'Weight', 'veg consp', 'main
meal consp', 'Water consp', 'physical actv', 'Screentime']

# Extract numerical data
```

```
X = df_train[numerical_columns]

# Perform PCA
pca = PCA(n_components=2)  # You can adjust the number of components
X_pca = pca.fit_transform(X)

# Create a DataFrame for the PCA results
df_pca = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])

# Add Obesity_Level to the PCA DataFrame for color differentiation
df_pca['Obesity_Level'] = df_train['Obesity_Level']

# Visualize PCA
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PC1', y='PC2', hue='Obesity_Level', data=df_pca,
palette='Set1', legend='full')
plt.title('PCA Plot of numerical variables against obesity level')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



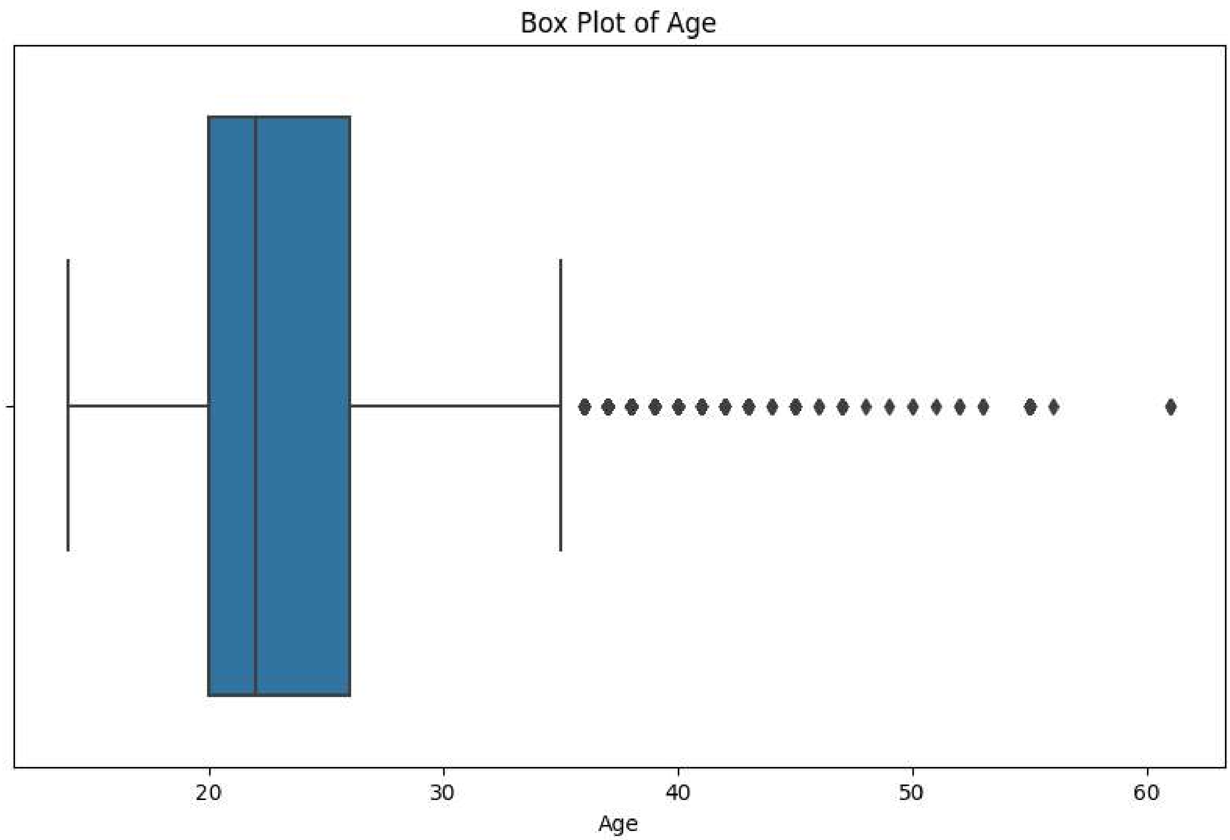
4. Outlier Analysis:

a. Univariate Outlier Analysis:

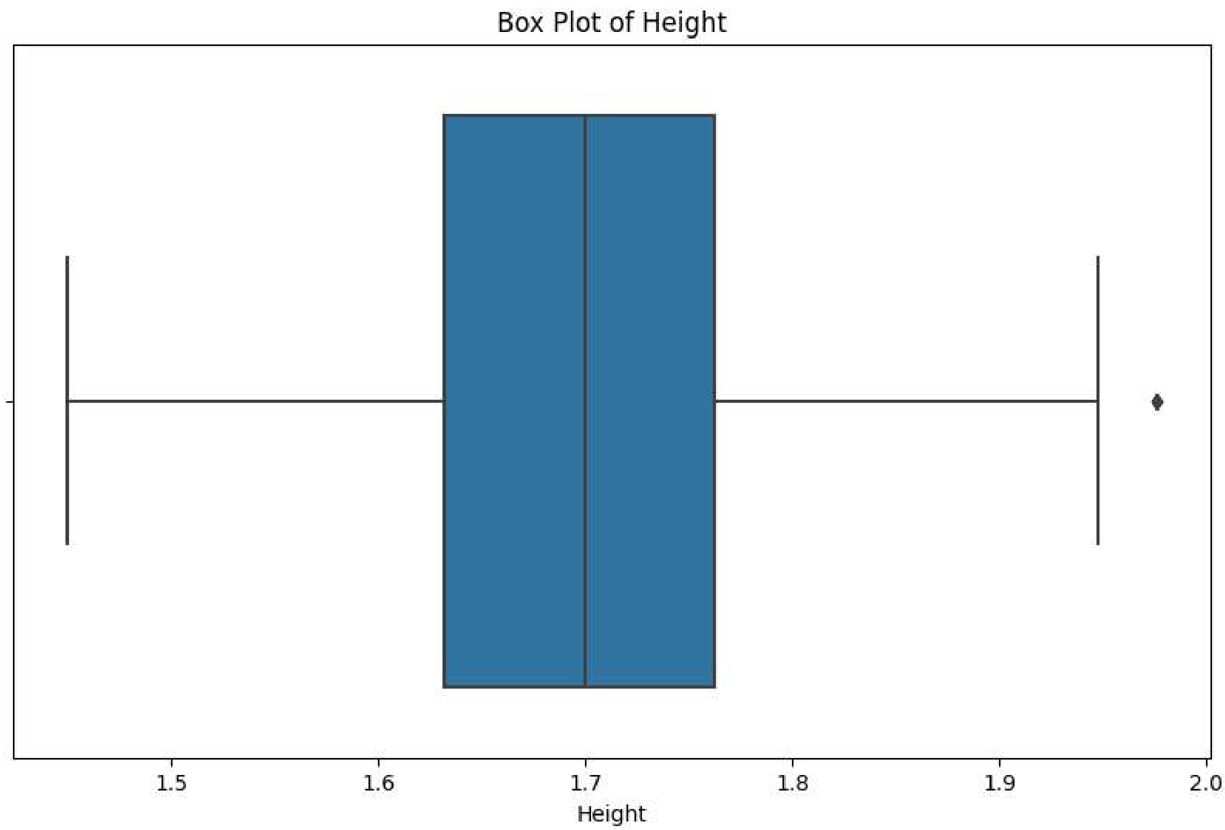
I. Boxplot Outlier Analysis:

```python
# Function to identify outliers using Box Plot
def box_plot_outliers(df, col):
    """
    Detect outliers using Box Plot.

    Parameters:
        df (DataFrame): The input DataFrame.
        col (str): The name of the column to analyze.

    Returns:
        None
    """
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=df[col])
    plt.title(f'Box Plot of {col}')
    plt.xlabel(f'{col}')
    plt.show()

# Selecting numerical columns
numerical_cols = df_train.select_dtypes(include=['float64',
'int32']).columns

# Loop through each numerical column and perform outlier analysis
for col in numerical_cols:
    print(f'Column: {col}')
    box_plot_outliers(df_train, col)
    print('\n')
```

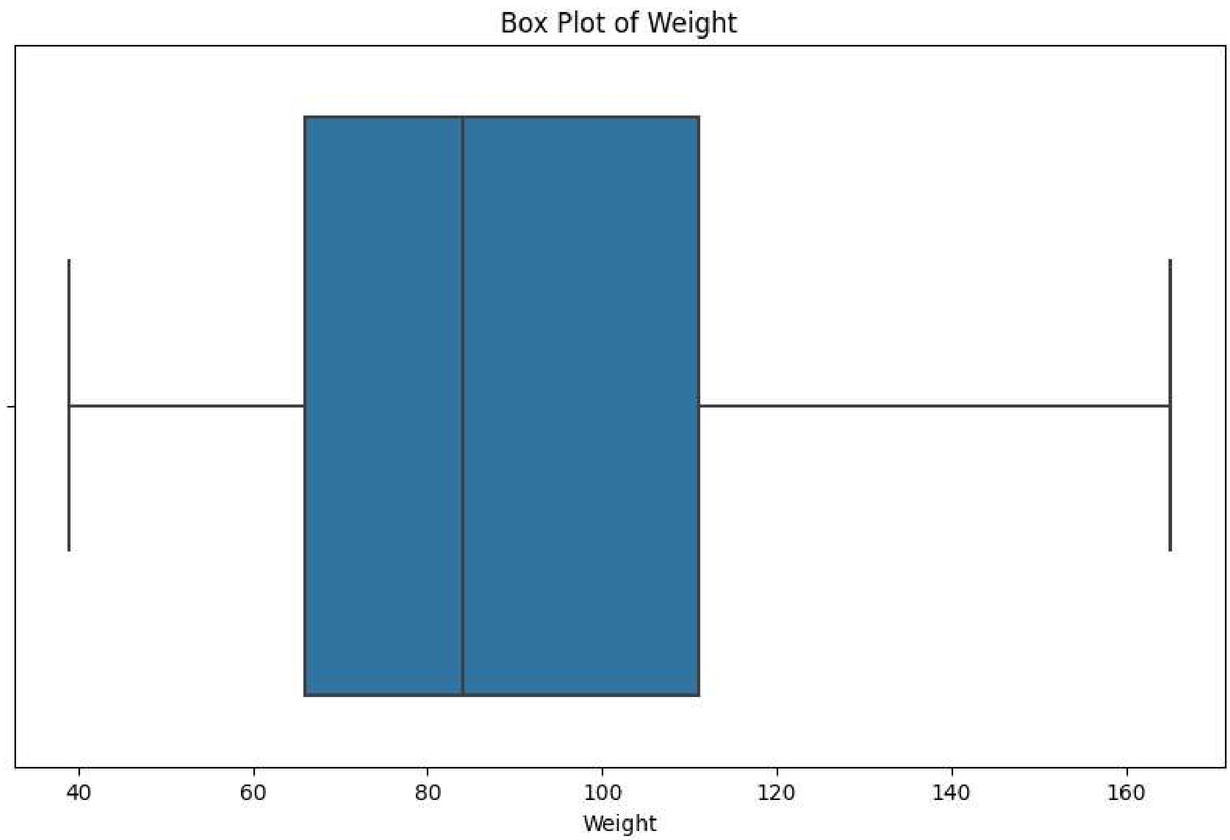
```
Column: Age
```



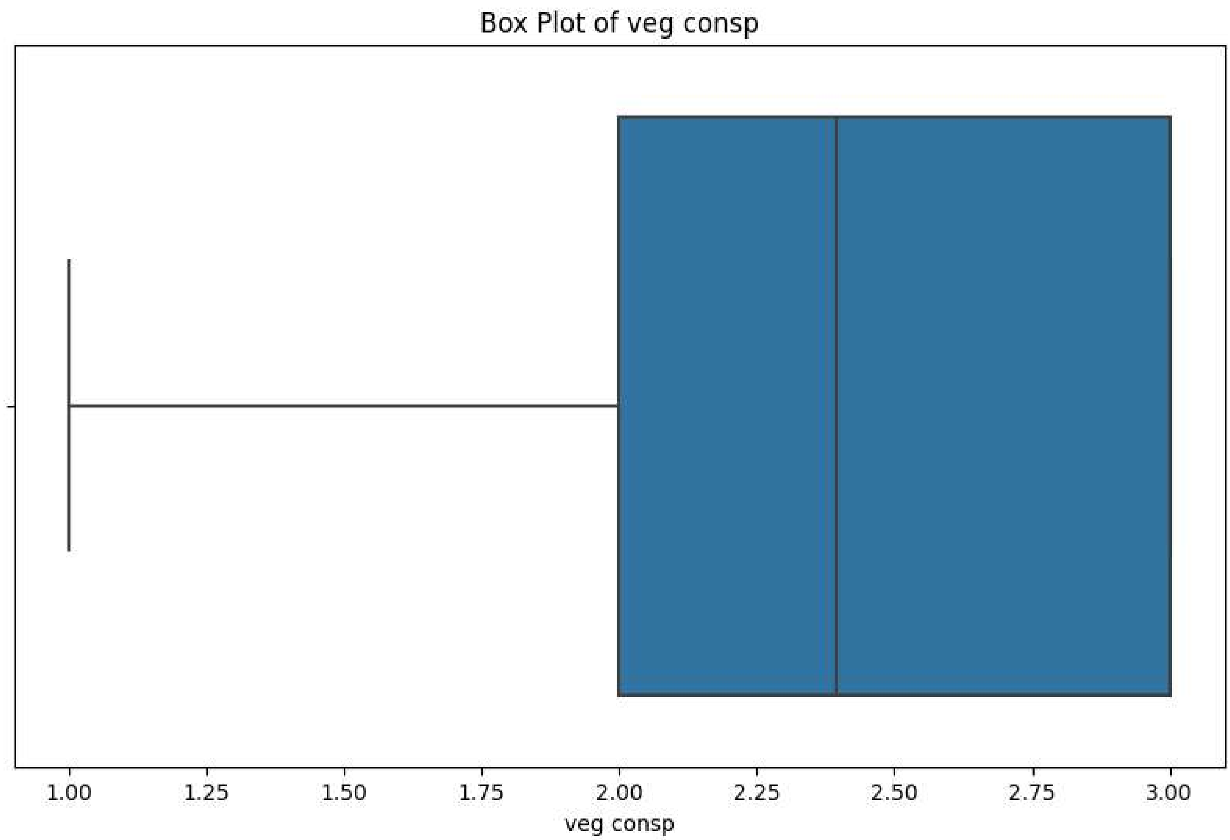

```
Column: Height
```



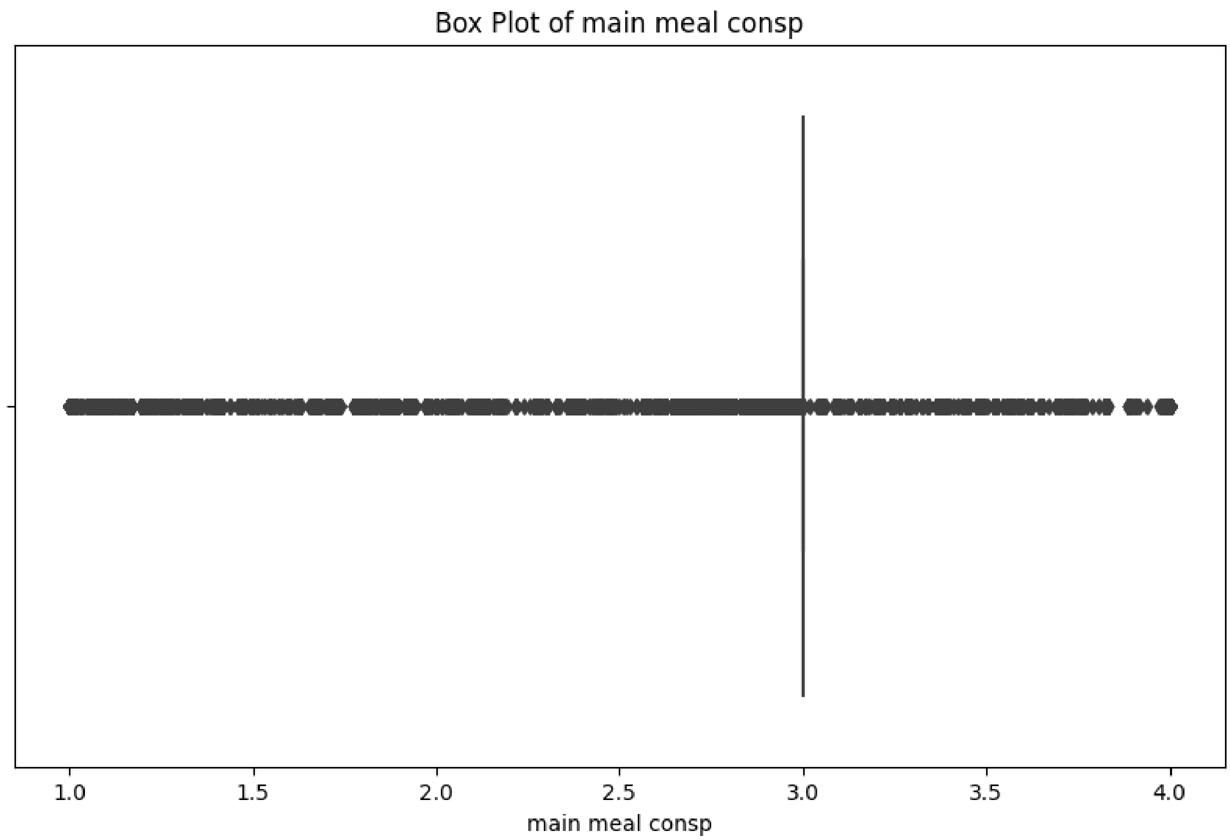

```
Column: Weight
```



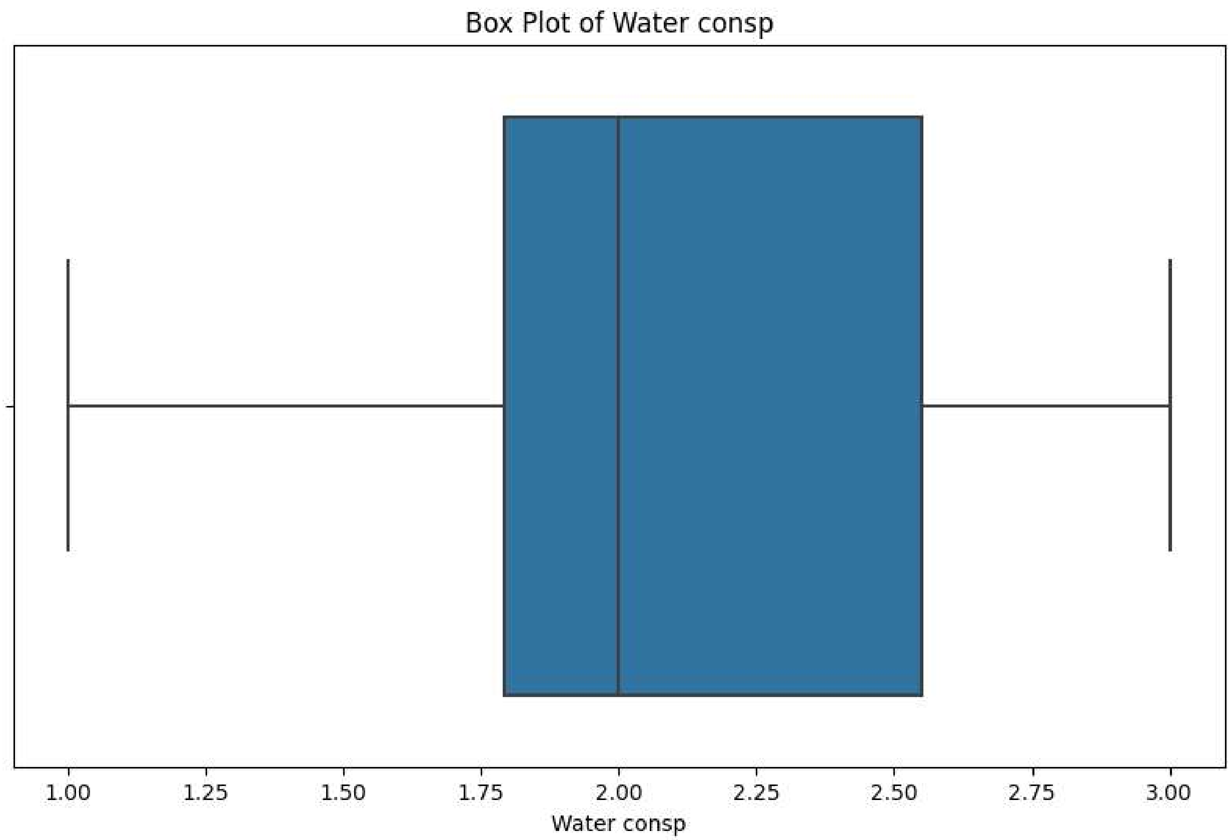

```
Column: veg consp
```



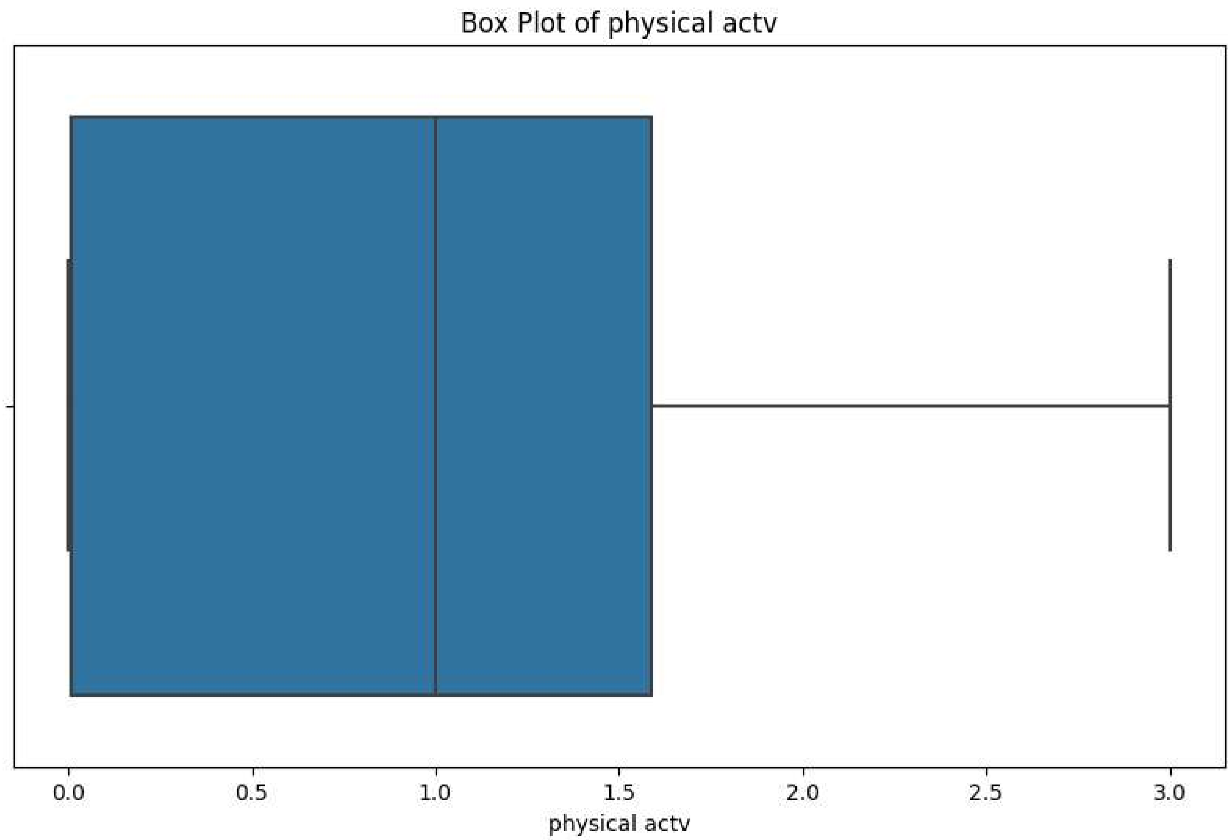

```
Column: main meal consp
```



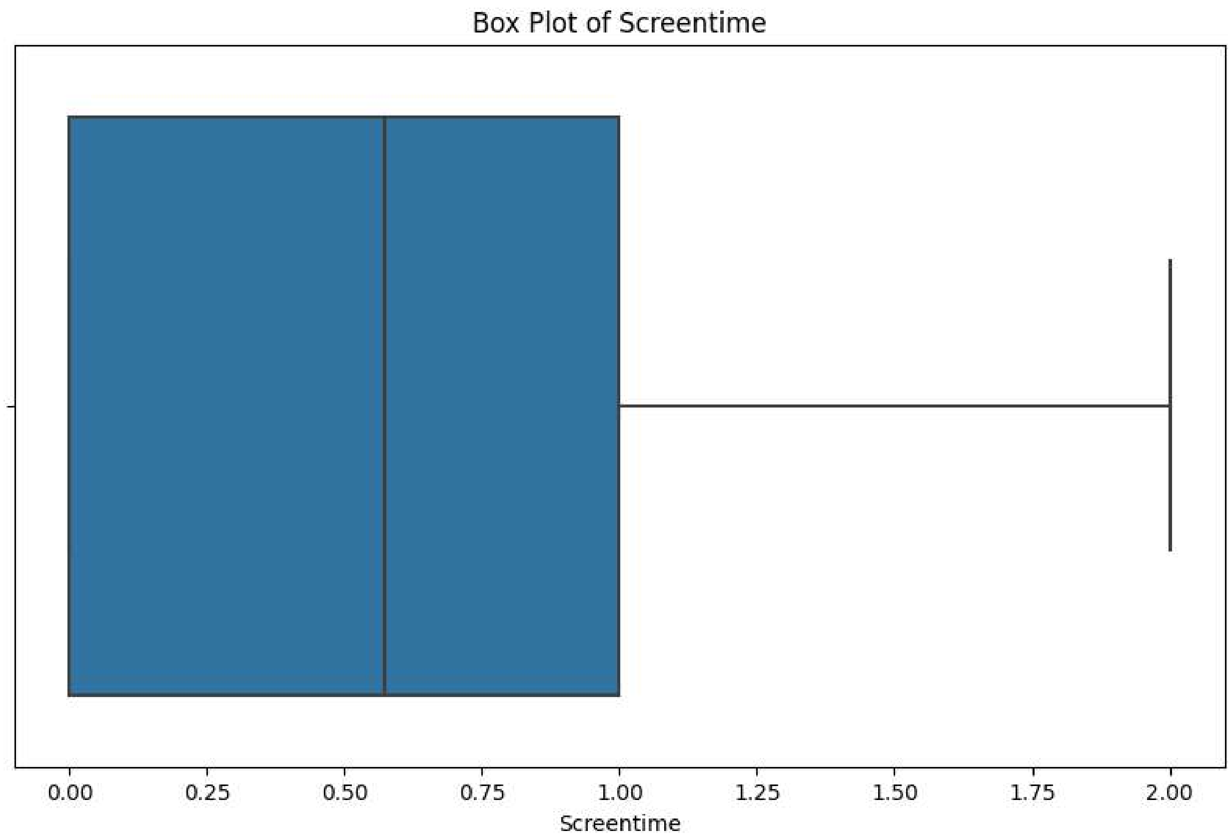

```
Column: Water consp
```



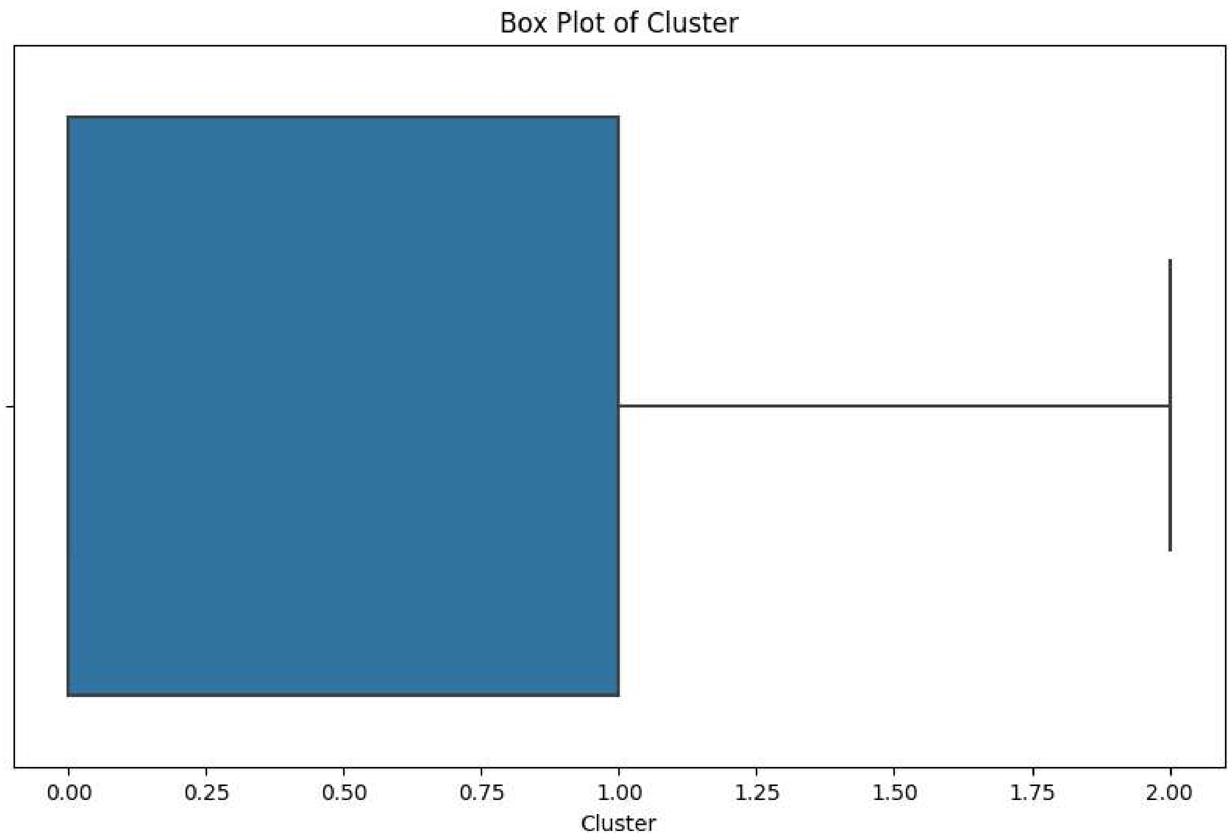

```
Column: physical actv
```




```
Column: Screentime
```




```
Column: Cluster
```



II. Detecting outliers using Z-Score:

```
# Function to identify outliers using Z-Score
def z_score_outliers(df, col, threshold=3):
    """
    Detect outliers using Z-Score.

    Parameters:
        df (DataFrame): The input DataFrame.
        col (str): The name of the column to analyze.
        threshold (float): The Z-Score threshold for outlier 
detection.

    Returns:
        None
    """
    z_scores = (df[col] - df[col].mean()) / df[col].std()
    outliers = df[abs(z_scores) > threshold]
    print(f'Number of outliers detected using Z-Score for {col}:
```

```
{outliers.shape[0]}')

# Selecting numerical columns
numerical_cols = df_train.select_dtypes(include=['float64',
'int32']).columns

# Loop through each numerical column and perform outlier analysis
for col in numerical_cols:
    print(f'Column: {col}')
    z_score_outliers(df_train, col)
    print('\n')
```

```
Column: Age
Number of outliers detected using Z-Score for Age: 258


Column: Height
Number of outliers detected using Z-Score for Height: 4


Column: Weight
Number of outliers detected using Z-Score for Weight: 0


Column: veg consp
Number of outliers detected using Z-Score for veg consp: 0


Column: main meal consp
Number of outliers detected using Z-Score for main meal consp: 0


Column: Water consp
Number of outliers detected using Z-Score for Water consp: 0


Column: physical actv
Number of outliers detected using Z-Score for physical actv: 0


Column: Screentime
Number of outliers detected using Z-Score for Screentime: 0


Column: Cluster
Number of outliers detected using Z-Score for Cluster: 0
```

III. Detecting outliers using Interquartile Range (IQR):

```
# Function to identify outliers using IQR
def iqr_outliers(df, col):
    """
    Detect outliers using Interquartile Range (IQR).

    Parameters:
        df (DataFrame): The input DataFrame.
        col (str): The name of the column to analyze.

    Returns:
        None
    """
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f'Number of outliers detected using IQR for {col}:
{outliers.shape[0]}')

# Selecting numerical columns
numerical_cols = df_train.select_dtypes(include=['float64',
'int32']).columns

# Loop through each numerical column and perform outlier analysis
for col in numerical_cols:
    print(f'Column: {col}')
    iqr_outliers(df_train, col)
    print('\n')
```

```
Column: Age
Number of outliers detected using IQR for Age: 1029


Column: Height
Number of outliers detected using IQR for Height: 4


Column: Weight
Number of outliers detected using IQR for Weight: 0


Column: veg consp
Number of outliers detected using IQR for veg consp: 0
```

```
Column: main meal consp
Number of outliers detected using IQR for main meal consp: 6052


Column: Water consp
Number of outliers detected using IQR for Water consp: 0


Column: physical actv
Number of outliers detected using IQR for physical actv: 0


Column: Screentime
Number of outliers detected using IQR for Screentime: 0


Column: Cluster
Number of outliers detected using IQR for Cluster: 0
```

b. Multivariate Outlier Analysis:

I. Detecting Multivariate Outliers Using Mahalanobis Distance:

```
# Function to calculate Mahalanobis Distance
def mahalanobis_distance(x, mean, cov):
    """
    Calculate Mahalanobis Distance for a data point.

    Parameters:
        x (array-like): The data point.
        mean (array-like): The mean vector.
        cov (array-like): The covariance matrix.

    Returns:
        float: The Mahalanobis Distance.
    """
    x_minus_mean = x - mean
    inv_cov = np.linalg.inv(cov)
    distance = np.sqrt(np.dot(np.dot(x_minus_mean, inv_cov),
x_minus_mean.T))
    return distance
```

```python
# Function to detect multivariate outliers using Mahalanobis Distance
def mahalanobis_outliers(df, threshold=3):
    """
    Detect multivariate outliers using Mahalanobis Distance.

    Parameters:
        df (DataFrame): The input DataFrame.
        threshold (float): The Mahalanobis Distance threshold for
outlier detection.

    Returns:
        DataFrame: The DataFrame containing outliers.
    """
    mean = df.mean()
    cov = df.cov()
    outliers = []
    for i, row in df.iterrows():
        distance = mahalanobis_distance(row, mean, cov)
        if distance > threshold:
            outliers.append(i)
    return df.iloc[outliers]

# Selecting numerical columns
numerical_cols = df_train.select_dtypes(include=['float64',
'int32']).columns

# Performing multivariate outlier analysis using Mahalanobis Distance
mahalanobis_outliers_df =
mahalanobis_outliers(df_train[numerical_cols])
mahalanobis_outliers_cols = mahalanobis_outliers_df.columns.tolist()


print(f'Number of multivariate outliers detected using Mahalanobis
Distance: {mahalanobis_outliers_df.shape[0]}')
print('Columns with outliers detected using Mahalanobis Distance:',
mahalanobis_outliers_cols)
```

```
Number of multivariate outliers detected using Mahalanobis Distance:
8577
Columns with outliers detected using Mahalanobis Distance: ['Age',
'Height', 'Weight', 'veg consp', 'main meal consp', 'Water consp',
'physical actv', 'Screentime', 'Cluster']
```

II. Detecting Multivariate Outliers Using Principal Component Analysis (PCA):

```
# Function to detect multivariate outliers using Principal Component
Analysis (PCA)
def pca_outliers(df, threshold=3):
    """
    Detect multivariate outliers using Principal Component Analysis
(PCA).

    Parameters:
        df (DataFrame): The input DataFrame.
        threshold (float): The threshold for outlier detection based
on PCA distance.

    Returns:
        DataFrame: The DataFrame containing outliers.
    """
    pca = PCA(n_components=2)
    principal_components = pca.fit_transform(df)
    distances = np.linalg.norm(principal_components -
np.mean(principal_components, axis=0), axis=1)
    cutoff = np.percentile(distances, 100 - 100 * chi2.cdf(threshold,
2))
    outliers = df[distances > cutoff]
    return outliers

# Selecting numerical columns
numerical_cols = df_train.select_dtypes(include=['float64',
'int32']).columns

# Performing multivariate outlier analysis using Principal Component
Analysis (PCA)
pca_outliers_df = pca_outliers(df_train[numerical_cols])
pca_outliers_cols = pca_outliers_df.columns.tolist()


print(f'Number of multivariate outliers detected using PCA:
{pca_outliers_df.shape[0]}')
print('Columns with outliers detected using PCA:', pca_outliers_cols)
```

```
Number of multivariate outliers detected using PCA: 16126
Columns with outliers detected using PCA: ['Age', 'Height', 'Weight',
'veg consp', 'main meal consp', 'Water consp', 'physical actv',
'Screentime', 'Cluster']
```

III. Detecting Cluster-Based Outliers Using KMeans Clustering:

```
# Select numerical columns for clustering
numerical_cols = df_train.select_dtypes(include=['float64', 'int32'])

# Initialize KMeans with the desired number of clusters
kmeans = KMeans(n_clusters=5)  # Adjust the number of clusters as
needed

# Fit KMeans to the numerical data
kmeans.fit(numerical_cols)

# Get the cluster centroids
cluster_centers = kmeans.cluster_centers_

# Calculate the distance of each point to its cluster centroid
distances = []
for i in range(len(df_train)):
    point = np.array(df_train.iloc[i][numerical_cols.columns])
    cluster_label = kmeans.labels_[i]
    centroid = cluster_centers[cluster_label]
    distance = np.linalg.norm(point - centroid)
    distances.append(distance)

# Set a threshold to identify outliers
threshold = np.percentile(distances, 95)  # Adjust the percentile as
needed

# Identify outliers based on the threshold
outliers_indices = [i for i, distance in enumerate(distances) if
distance > threshold]
outliers = df_train.iloc[outliers_indices]

# Filter out categorical columns before calculating the sum of
outliers
numerical_outliers = outliers.select_dtypes(include=['float64',
'int32'])

# Calculate the sum of all outliers present in each numerical column
outliers_sum_per_column = numerical_outliers.sum()

# Calculate the total sum of outliers across all numerical columns
total_outliers_sum = numerical_outliers.sum().sum()

# Display the sum of outliers for each numerical column
print("\nSum of outliers present in each numerical column:")
print(outliers_sum_per_column)
```

```
# Display the total sum of outliers across all numerical columns
print("\nTotal sum of outliers across all numerical columns:",
total_outliers_sum)
```

```
Sum of outliers present in each numerical column:
Age                    37360.000000
Height                  1744.514806
Weight                100744.000000
veg consp               2517.891680
main meal consp         2831.066765
Water consp             2004.994688
physical actv            925.069334
Screentime               314.269129
Cluster                 1124.000000
dtype: float64

Total sum of outliers across all numerical columns: 149565.8064025
```

```
df_train.drop(columns=['Cluster'], inplace=True)
```

5. Feature Engineering:

```
# Rename the columns  for train data
test_sub.rename(columns=new_column_names, inplace=True)

test_sub.head(5)
```

```
      id  Gender  Age    Height  Weight Overweighted Family History  \
0  20758    Male   26  1.848294     120                          yes
1  20759  Female   21  1.600000      66                          yes
2  20760  Female   26  1.643355     111                          yes
3  20761    Male   20  1.553127     103                          yes
4  20762  Female   26  1.627396     104                          yes

  High caleric food consp  veg consp  main meal consp Food btw meal
consp  \
0                     yes   2.938616         3.000000
Sometimes
1                     yes   2.000000         1.000000
Sometimes
2                     yes   3.000000         3.000000
Sometimes
3                     yes   2.000000         2.977909
Sometimes
4                     yes   3.000000         3.000000
Sometimes

  SMOKE  Water consp Calories Monitoring  physical actv  Screentime  \
```

```
0    no     2.825629                    no        0.855400    0.000000
1    no     3.000000                    no        1.000000    0.000000
2    no     2.621877                    no        0.000000    0.250502
3    no     2.786417                    no        0.094851    0.000000
4    no     2.653531                    no        0.000000    0.741069

  Alcohol consp          transport used
0      Sometimes  Public_Transportation
1      Sometimes  Public_Transportation
2      Sometimes  Public_Transportation
3      Sometimes  Public_Transportation
4      Sometimes  Public_Transportation
```

a. Encoding Categorical to numerical variables:

```
# Encoding of target variables to numerical
keys_dict = {
    'Insufficient_Weight': 0,
    'Normal_Weight': 1,
    'Overweight_Level_I': 2,
    'Overweight_Level_II': 3,
    'Obesity_Type_I': 4,
    'Obesity_Type_II': 5,
    'Obesity_Type_III': 6
}

# Encoding of transport used to numerical
keys_dict_1 = {
    'Automobile': 0,
    'Bike': 1,
    'Motorbike': 2,
    'Public_Transportation': 3,
    'Walking': 4
}

# Encoding of Alcohol consumption to numerical
keys_dict_2 = {
    'Sometimes': 1/3,
    'Frequently': 2/3,
    'Always': 1,
    'no': 0
}

# Encoding of Food between meal consumption to numerical
keys_dict_3 = {
    'Sometimes': 1/3,
    'Frequently': 2/3,
    'Always': 1,
```

```
    'no': 0
}

def encode_obesity_level(row):
    return keys_dict.get(row['Obesity_Level'], None)

def encode_transport_used(row):
    return keys_dict_1.get(row['transport used'], None)

def encode_alcohol_consp(row):
    return keys_dict_2.get(row['Alcohol consp'], None)

def encode_food_btw_meal(row):
    return keys_dict_3.get(row['Food btw meal consp'], None)

# Add new columns and apply encoding for train data
df_train['Encdd_Obesity_Level'] = df_train.apply(encode_obesity_level, 
axis=1)
df_train['Encdd_transport_used'] = 
df_train.apply(encode_transport_used, axis=1)
df_train['Encdd_Alcohol_consp'] = df_train.apply(encode_alcohol_consp, 
axis=1)
df_train['Encdd_Food_btw_meal'] = df_train.apply(encode_food_btw_meal, 
axis=1)


# Add new columns and apply encoding  for test data
test_sub['Encdd_transport_used'] = 
test_sub.apply(encode_transport_used, axis=1)
test_sub['Encdd_Alcohol_consp'] = test_sub.apply(encode_alcohol_consp, 
axis=1)
test_sub['Encdd_Food_btw_meal'] = test_sub.apply(encode_food_btw_meal, 
axis=1)

df_train.head(5)
```

```
   id  Gender  Age    Height  Weight Overweighted Family History  \
0   0    Male   24  1.699998      81                          yes
1   1  Female   18  1.560000      57                          yes
2   2  Female   18  1.711460      50                          yes
3   3  Female   20  1.710730     131                          yes
4   4    Male   31  1.914186      93                          yes

  High caleric food consp  veg consp  main meal consp Food btw meal 
consp  \
0                     yes   2.000000         2.983297                
Sometimes
1                     yes   2.000000         3.000000                
Frequently
2                     yes   1.880534         1.411685
```

```
Sometimes
3                     yes   3.000000        3.000000
Sometimes
4                     yes   2.679664        1.971472
Sometimes

  SMOKE  Water consp Calories Monitoring  physical actv  Screentime  \
0    no     2.763573                  no       0.000000    0.976473
1    no     2.000000                  no       1.000000    1.000000
2    no     1.910378                  no       0.866045    1.673584
3    no     1.674061                  no       1.467863    0.780199
4    no     1.979848                  no       1.967973    0.931721

  Alcohol consp         transport used        Obesity_Level  \
0     Sometimes  Public_Transportation  Overweight_Level_II
1            no             Automobile        Normal_Weight
2            no  Public_Transportation  Insufficient_Weight
3     Sometimes  Public_Transportation     Obesity_Type_III
4     Sometimes  Public_Transportation  Overweight_Level_II

   Encdd_Obesity_Level  Encdd_transport_used  Encdd_Alcohol_consp  \
0                    3                     3             0.333333
1                    1                     0             0.000000
2                    0                     3             0.000000
3                    6                     3             0.333333
4                    3                     3             0.333333

   Encdd_Food_btw_meal
0             0.333333
1             0.666667
2             0.333333
3             0.333333
4             0.333333
```

```
test_sub.head(5)
```

```
      id  Gender  Age    Height  Weight Overweighted Family History  \
0  20758    Male   26  1.848294     120                         yes
1  20759  Female   21  1.600000      66                         yes
2  20760  Female   26  1.643355     111                         yes
3  20761    Male   20  1.553127     103                         yes
4  20762  Female   26  1.627396     104                         yes

  High caleric food consp  veg consp  main meal consp Food btw meal
consp  \
0                     yes   2.938616        3.000000
Sometimes
1                     yes   2.000000        1.000000
Sometimes
2                     yes   3.000000        3.000000
```

```
Sometimes
3                      yes   2.000000         2.977909
Sometimes
4                      yes   3.000000         3.000000
Sometimes

  SMOKE  Water consp Calories Monitoring  physical actv  Screentime  \
0    no     2.825629                  no       0.855400    0.000000
1    no     3.000000                  no       1.000000    0.000000
2    no     2.621877                  no       0.000000    0.250502
3    no     2.786417                  no       0.094851    0.000000
4    no     2.653531                  no       0.000000    0.741069

  Alcohol consp         transport used  Encdd_transport_used  \
0     Sometimes  Public_Transportation                     3
1     Sometimes  Public_Transportation                     3
2     Sometimes  Public_Transportation                     3
3     Sometimes  Public_Transportation                     3
4     Sometimes  Public_Transportation                     3

   Encdd_Alcohol_consp  Encdd_Food_btw_meal
0             0.333333             0.333333
1             0.333333             0.333333
2             0.333333             0.333333
3             0.333333             0.333333
4             0.333333             0.333333

# Define mappings for each column
gender_mapping = {'Male': 1, 'Female': 0}
family_history_mapping = {'yes': 1, 'no': 0}
high_caloric_mapping = {'yes': 1, 'no': 0}
smoke_mapping = {'yes': 1, 'no': 0}
calories_monitoring_mapping = {'yes': 1, 'no': 0}

# Define functions to apply mappings and create new encoded columns
def encode_gender(row):
    return gender_mapping.get(row['Gender'], None)

def encode_family_history(row):
    return family_history_mapping.get(row['Overweighted Family
History'], None)

def encode_high_caloric(row):
    return high_caloric_mapping.get(row['High caleric food consp'],
None)

def encode_smoke(row):
    return smoke_mapping.get(row['SMOKE'], None)

def encode_calories_monitoring(row):
```

```
    return calories_monitoring_mapping.get(row['Calories Monitoring'], None)

# Apply functions to create new encoded columns  for train data
df_train['Encoded_Gender'] = df_train.apply(encode_gender, axis=1)
df_train['Encoded_Family_History'] = df_train.apply(encode_family_history, axis=1)
df_train['Encoded_High_Caloric'] = df_train.apply(encode_high_caloric, axis=1)
df_train['Encoded_Smoke'] = df_train.apply(encode_smoke, axis=1)
df_train['Encoded_Calories_Monitoring'] = df_train.apply(encode_calories_monitoring, axis=1)

# Apply functions to create new encoded columns  for train data
test_sub['Encoded_Gender'] = test_sub.apply(encode_gender, axis=1)
test_sub['Encoded_Family_History'] = test_sub.apply(encode_family_history, axis=1)
test_sub['Encoded_High_Caloric'] = test_sub.apply(encode_high_caloric, axis=1)
test_sub['Encoded_Smoke'] = test_sub.apply(encode_smoke, axis=1)
test_sub['Encoded_Calories_Monitoring'] = test_sub.apply(encode_calories_monitoring, axis=1)
```

b. BMI(Body Mass Index) Calculation:

```
#Calculation of BMI(Body Mass Index), Veg Intake comapred to high calorie food consp, Total number of meal consp and Physical activity frequency

# Create new columns based on existing ones
df_train['BMI'] = df_train['Weight'] / (df_train['Height'] ** 2)
test_sub['BMI'] = test_sub['Weight'] / (test_sub['Height'] ** 2)
```

c. Total Meal Consumed:

```
# Calculate the total number of meals consumed
# This is done by adding the counts of main meals and between-meal snacks
df_train['Meal'] = df_train['main meal consp'] + df_train['Encdd_Food_btw_meal']
test_sub['Meal'] = test_sub['main meal consp'] + test_sub['Encdd_Food_btw_meal']
```

d. Total Activity Frequency Calculation:

```
# Calculate the product of physical activity frequency and screen time
df_train['Activity'] = df_train['physical actv'] *
df_train['Screentime']
test_sub['Activity'] = test_sub['physical actv'] *
test_sub['Screentime']
```

e. Ageing process analysis:

```
df_train['IsYoung'] = df_train['Age'].apply(lambda x: x < 25)
df_train['IsAging'] = df_train['Age'].apply(lambda x: 25 <= x < 40)

test_sub['IsYoung'] = test_sub['Age'].apply(lambda x: x < 25)
test_sub['IsAging'] = test_sub['Age'].apply(lambda x: 25 <= x < 40)

df_train.head(5)
```

```
   id  Gender  Age    Height  Weight Overweighted Family History  \
0   0    Male   24  1.699998      81                          yes
1   1  Female   18  1.560000      57                          yes
2   2  Female   18  1.711460      50                          yes
3   3  Female   20  1.710730     131                          yes
4   4    Male   31  1.914186      93                          yes

  High caleric food consp  veg consp  main meal consp Food btw meal
consp  \
0                     yes   2.000000         2.983297
Sometimes
1                     yes   2.000000         3.000000
Frequently
2                     yes   1.880534         1.411685
Sometimes
3                     yes   3.000000         3.000000
Sometimes
4                     yes   2.679664         1.971472
Sometimes

  SMOKE  Water consp Calories Monitoring  physical actv  Screentime  \
0    no     2.763573                  no       0.000000    0.976473
1    no     2.000000                  no       1.000000    1.000000
2    no     1.910378                  no       0.866045    1.673584
3    no     1.674061                  no       1.467863    0.780199
4    no     1.979848                  no       1.967973    0.931721

  Alcohol consp         transport used        Obesity_Level  \
0     Sometimes  Public_Transportation  Overweight_Level_II
1            no             Automobile        Normal_Weight
```

```
2            no  Public_Transportation  Insufficient_Weight
3     Sometimes  Public_Transportation     Obesity_Type_III
4     Sometimes  Public_Transportation  Overweight_Level_II

   Encdd_Obesity_Level  Encdd_transport_used  Encdd_Alcohol_consp  \
0                    3                     3             0.333333
1                    1                     0             0.000000
2                    0                     3             0.000000
3                    6                     3             0.333333
4                    3                     3             0.333333

   Encdd_Food_btw_meal  Encoded_Gender  Encoded_Family_History  \
0             0.333333               1                       1
1             0.666667               0                       1
2             0.333333               0                       1
3             0.333333               0                       1
4             0.333333               1                       1

   Encoded_High_Caloric  Encoded_Smoke  Encoded_Calories_Monitoring  \
0                     1              0                            0
1                     1              0                            0
2                     1              0                            0
3                     1              0                            0
4                     1              0                            0

         BMI      Meal  Activity  IsYoung  IsAging
0  28.027748  3.316630  0.000000     True    False
1  23.422091  3.666667  1.000000     True    False
2  17.070117  1.745018  1.449399     True    False
3  44.761884  3.333333  1.145225     True    False
4  25.381348  2.304805  1.833602    False     True
```

```
test_sub.head(5)
```

```
      id  Gender  Age    Height  Weight Overweighted Family History  \
0  20758    Male   26  1.848294     120                          yes
1  20759  Female   21  1.600000      66                          yes
2  20760  Female   26  1.643355     111                          yes
3  20761    Male   20  1.553127     103                          yes
4  20762  Female   26  1.627396     104                          yes

  High caleric food consp  veg consp  main meal consp Food btw meal
consp  \
0                      yes   2.938616         3.000000
Sometimes
1                      yes   2.000000         1.000000
Sometimes
2                      yes   3.000000         3.000000
Sometimes
3                      yes   2.000000         2.977909
```

```
Sometimes
4                    yes   3.000000         3.000000
Sometimes

  SMOKE  Water consp Calories Monitoring  physical actv  Screentime  \
0    no     2.825629                  no       0.855400    0.000000
1    no     3.000000                  no       1.000000    0.000000
2    no     2.621877                  no       0.000000    0.250502
3    no     2.786417                  no       0.094851    0.000000
4    no     2.653531                  no       0.000000    0.741069

  Alcohol consp         transport used  Encdd_transport_used  \
0     Sometimes  Public_Transportation                     3
1     Sometimes  Public_Transportation                     3
2     Sometimes  Public_Transportation                     3
3     Sometimes  Public_Transportation                     3
4     Sometimes  Public_Transportation                     3

   Encdd_Alcohol_consp  Encdd_Food_btw_meal  Encoded_Gender  \
0             0.333333             0.333333               1
1             0.333333             0.333333               0
2             0.333333             0.333333               0
3             0.333333             0.333333               1
4             0.333333             0.333333               0

   Encoded_Family_History  Encoded_High_Caloric  Encoded_Smoke  \
0                       1                     1              0
1                       1                     1              0
2                       1                     1              0
3                       1                     1              0
4                       1                     1              0

   Encoded_Calories_Monitoring        BMI      Meal  Activity  IsYoung
\
0                            0  35.126845  3.333333       0.0    False

1                            0  25.781250  1.333333       0.0     True

2                            0  41.101739  3.333333       0.0    False

3                            0  42.699549  3.311242       0.0     True

4                            0  39.268730  3.333333       0.0    False


   IsAging
0     True
1    False
2     True
```

```
3     False
4      True
```

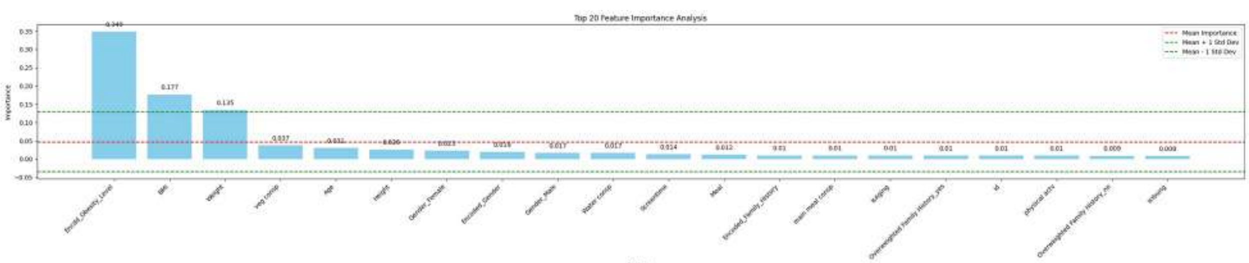
Section: 6. Analysis & Prediction Using Machine Learning(ML) Model:

1. Feature Importance Analysis and Visualization:

a. Feature Importance Analysis using Random Forest Classifier:

```
# Assuming df_train contains your dataset
# Define X (features) and y (target variable)
X = df_train.drop(columns=['Obesity_Level'])
y = df_train['Obesity_Level']

# Perform one-hot encoding for categorical variables
X_encoded = pd.get_dummies(X)

# Initialize the model
model = RandomForestClassifier()

# Train the model
model.fit(X_encoded, y)

# Get feature importances
feature_importances = model.feature_importances_

# Sort feature importances and corresponding feature names
sorted_indices = feature_importances.argsort()[::-1]
sorted_feature_importances = feature_importances[sorted_indices]
sorted_feature_names = X_encoded.columns[sorted_indices]

# Limit the number of displayed features
top_n = 20
sorted_feature_importances = sorted_feature_importances[:top_n]
sorted_feature_names = sorted_feature_names[:top_n]
# Calculate mean and standard deviation of feature importances
mean_importance = np.mean(sorted_feature_importances)
std_importance = np.std(sorted_feature_importances)
```

```
# Calculate coefficient of variation (CV)
cv_importance = std_importance / mean_importance

# Visualize feature importances
plt.figure(figsize=(28, 6))
plt.bar(sorted_feature_names, sorted_feature_importances,
color='skyblue')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Top {} Feature Importance Analysis'.format(top_n))
plt.xticks(rotation=45, ha='right')
for i, v in enumerate(sorted_feature_importances):
    plt.text(i, v + 0.01, str(round(v, 3)), ha='center', va='bottom')
plt.axhline(y=mean_importance, color='r', linestyle='--', label='Mean
Importance')
plt.axhline(y=mean_importance + std_importance, color='g',
linestyle='--', label='Mean + 1 Std Dev')
plt.axhline(y=mean_importance - std_importance, color='g',
linestyle='--', label='Mean - 1 Std Dev')
plt.legend()
plt.tight_layout()
plt.show()



# Define the statistical terms
statistical_terms = [
    ["Mean Importance", round(mean_importance, 3)],
    ["Standard Deviation of Importance", round(std_importance, 3)],
    ["Coefficient of Variation (CV) of Importance",
round(cv_importance, 3)]
]

# Print the statistical terms in a table-like structure
print(tabulate(statistical_terms, headers=["Statistical Term",
"Value"]))
```



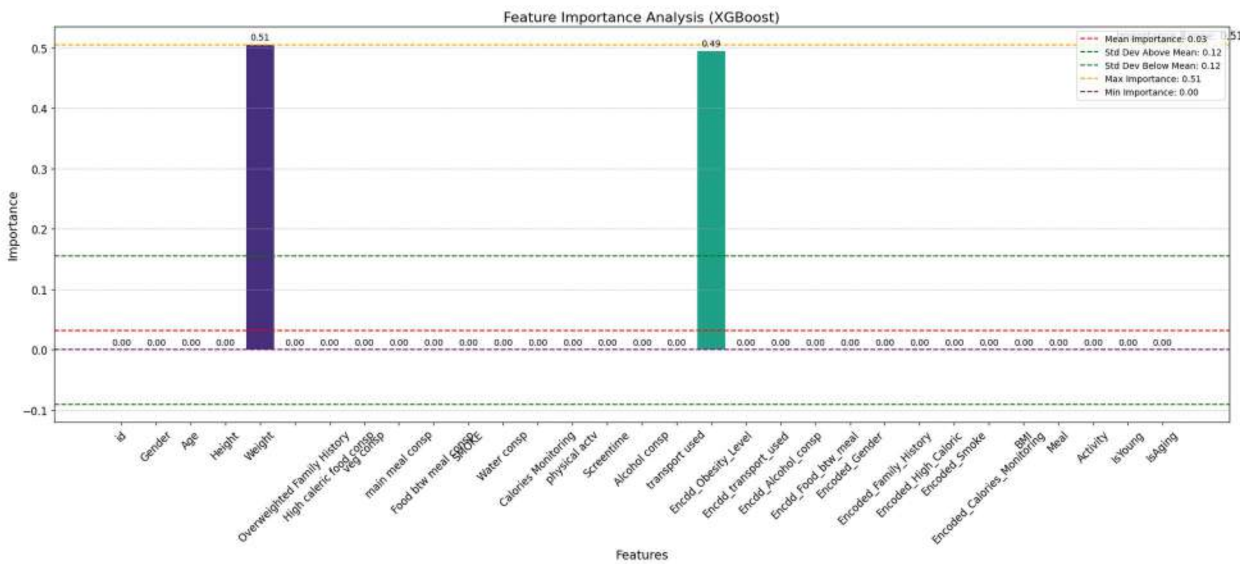

```
Statistical Term                              Value
-------------------------------------------  -------
Mean Importance                                0.047
```

```
Standard Deviation of Importance              0.082
Coefficient of Variation (CV) of Importance   1.749
```

b. Feature Importance Analysis using XGBoost(XGB) Model:

```
from sklearn.preprocessing import LabelEncoder  # For encoding
categorical variables
# Assuming df_train contains your dataset
# Define X (features) and y (target variable)
X = df_train.drop(columns=['Obesity_Level'])
y = df_train['Obesity_Level']

# Encode target variable into numerical labels
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Encode categorical features
encoder = LabelEncoder()
X_encoded = X.copy()
for col in X_encoded.columns:
    if X_encoded[col].dtype == 'object':
        X_encoded[col] = encoder.fit_transform(X_encoded[col])

# Initialize the XGBoost classifier
model_xgb = xgb.XGBClassifier()

# Train the model
model_xgb.fit(X_encoded, y_encoded)

# Get feature importances
feature_importances_xgb = model_xgb.feature_importances_

# Calculate statistical information
mean_importance = np.mean(feature_importances_xgb)
std_importance = np.std(feature_importances_xgb)
max_importance = np.max(feature_importances_xgb)
importance_range = max_importance - np.min(feature_importances_xgb)

# Count the occurrences of each feature
feature_counts = X_encoded.apply(lambda x:
x.value_counts()).fillna(0).astype(int)

# Visualize feature importances
plt.figure(figsize=(20, 9))  # Increase figure size

# Define color palette
```

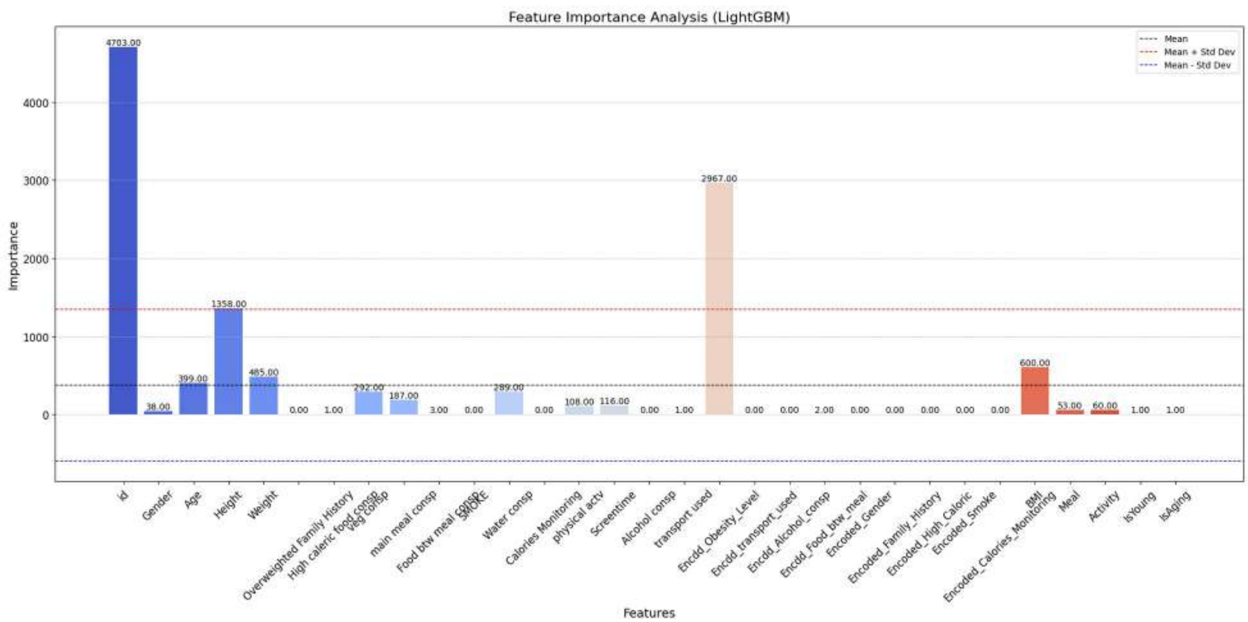
```
colors = plt.cm.viridis(np.linspace(0, 1, len(X_encoded.columns)))

bars = plt.bar(X_encoded.columns, feature_importances_xgb,
color=colors)  # Change color
plt.xlabel('Features', fontsize=14)  # Increase font size
plt.ylabel('Importance', fontsize=14)  # Increase font size
plt.title('Feature Importance Analysis (XGBoost)', fontsize=16)  #
Increase font size
plt.xticks(rotation=45, fontsize=12)  # Rotate x-axis labels and
increase font size
plt.yticks(fontsize=12)  # Increase font size for y-axis ticks
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add grid lines for
better readability

# Add statistical information
plt.axhline(mean_importance, color='red', linestyle='--', label=f'Mean
Importance: {mean_importance:.2f}')
plt.axhline(mean_importance + std_importance, color='green',
linestyle='--', label=f'Std Dev Above Mean: {std_importance:.2f}')
plt.axhline(mean_importance - std_importance, color='green',
linestyle='--', label=f'Std Dev Below Mean: {std_importance:.2f}')
plt.axhline(max_importance, color='orange', linestyle='--',
label=f'Max Importance: {max_importance:.2f}')
plt.axhline(np.min(feature_importances_xgb), color='purple',
linestyle='--', label=f'Min Importance:
{np.min(feature_importances_xgb):.2f}')
plt.text(len(X_encoded.columns)-0.5, max_importance + 0.005,
f'Importance Range: {importance_range:.2f}', ha='center', va='bottom',
fontsize=12, color='black')

# Add feature importance values above each bar
for i, importance in enumerate(feature_importances_xgb):
    plt.text(i, importance + 0.005, f'{importance:.2f}', ha='center',
va='bottom', fontsize=10, color='black')

plt.legend()

plt.tight_layout()  # Adjust layout to prevent overlapping labels
plt.show()
```



c. Feature Importance Analysis Using (LightGBM) Classifier Model:

```
# Assuming df_train contains your dataset
# Define X (features) and y (target variable)
X = df_train.drop(columns=['Obesity_Level'])
y = df_train['Obesity_Level']

# Encode categorical features
encoder = LabelEncoder()
X_encoded = X.copy()
for col in X_encoded.columns:
    if X_encoded[col].dtype == 'object':
        X_encoded[col] = encoder.fit_transform(X_encoded[col])

# Initialize the LightGBM classifier
model_lgb = lgb.LGBMClassifier(verbosity=-1)

# Train the model
model_lgb.fit(X_encoded, y)

# Get feature importances
feature_importances_lgb = model_lgb.feature_importances_

# Create a color palette
colors = sns.color_palette("coolwarm", len(X_encoded.columns))

# Visualize feature importances
plt.figure(figsize=(20, 10))  # Increase figure size
bars = plt.bar(X_encoded.columns, feature_importances_lgb,
```

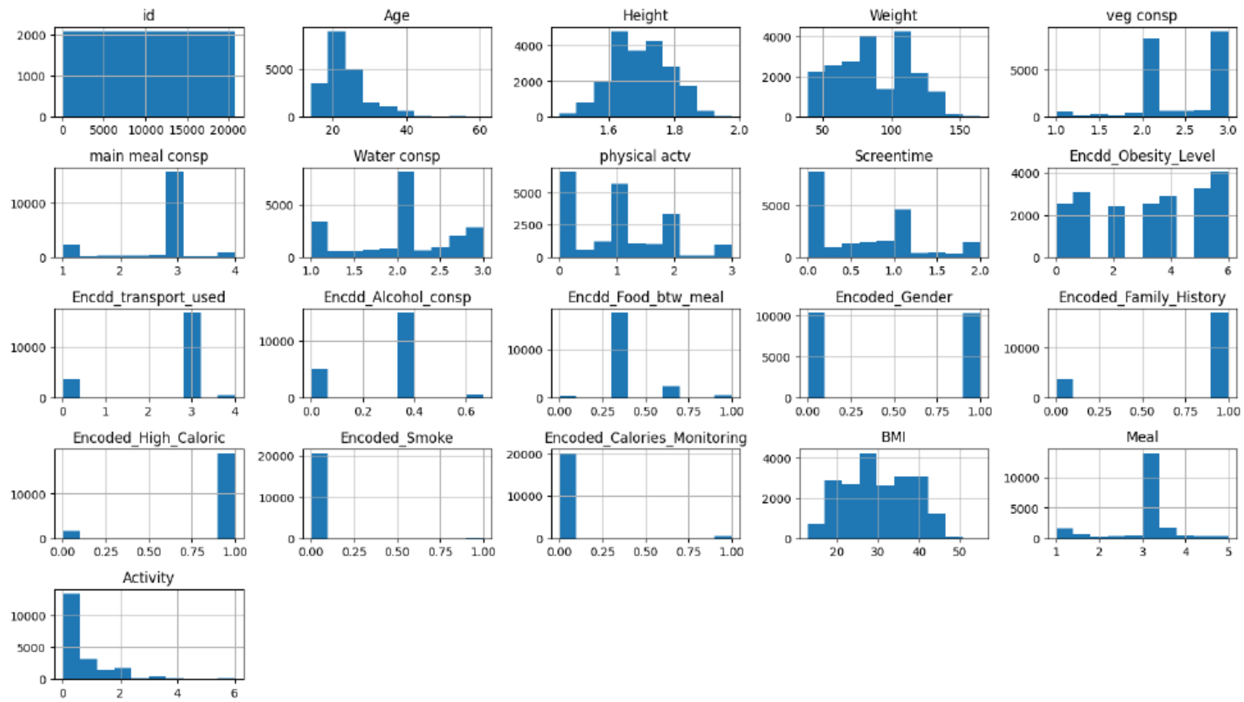
```
color=colors)  # Use color palette
plt.xlabel('Features', fontsize=14)  # Increase font size
plt.ylabel('Importance', fontsize=14)  # Increase font size
plt.title('Feature Importance Analysis (LightGBM)', fontsize=16)  #
Increase font size
plt.xticks(rotation=45, fontsize=12)  # Rotate x-axis labels and
increase font size
plt.yticks(fontsize=12)  # Increase font size for y-axis ticks
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add grid lines for
better readability

# Add statistical information
mean_importance = np.mean(feature_importances_lgb)
std_importance = np.std(feature_importances_lgb)
plt.axhline(mean_importance, color='black', linestyle='--',
linewidth=1, label='Mean')  # Add mean line
plt.axhline(mean_importance + std_importance, color='red',
linestyle='--', linewidth=1, label='Mean + Std Dev')  # Add mean + std
dev line
plt.axhline(mean_importance - std_importance, color='blue',
linestyle='--', linewidth=1, label='Mean - Std Dev')  # Add mean - std
dev line
plt.legend()  # Show legend

for bar, importance in zip(bars, feature_importances_lgb):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() +
0.005,
             f'{importance:.2f}', ha='center', va='bottom',
fontsize=10, color='black')

plt.tight_layout()  # Adjust layout to prevent overlapping labels
plt.show()
```



2. Data visualization after Feature Engineering:

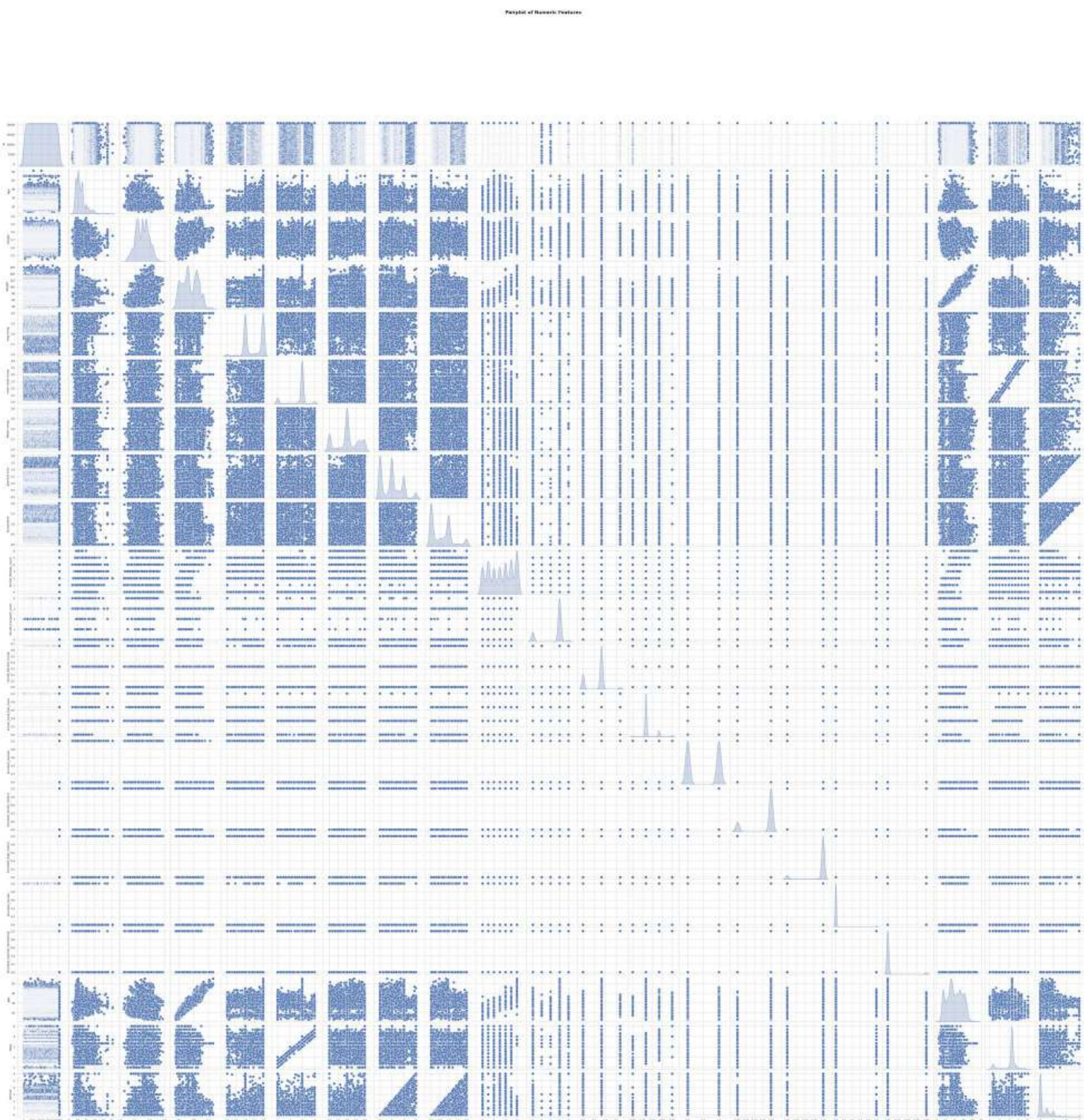
a. Bar plot of numerical variables:

```python
# Define columns to plot (excluding non-numeric columns)
columns_to_plot = df_train.select_dtypes(include=['number']).columns

# Plotting
plt.figure(figsize=(15, 10))
for i, col in enumerate(columns_to_plot, 1):
    plt.subplot(6, 5, i)
    df_train[col].hist()
    plt.title(col)
plt.tight_layout()
plt.show()
```



b. PairPlot of Numerical Variables:

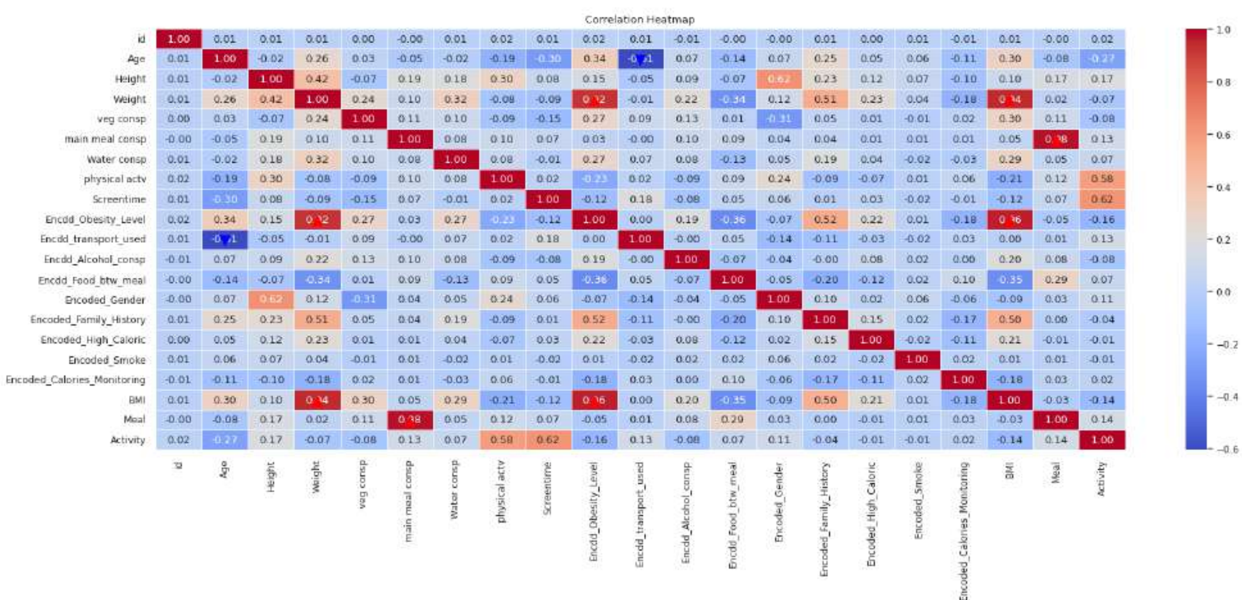
```
# Select numeric columns
numeric_columns = df_train.select_dtypes(include='number').columns

# Set style and context
sns.set(style="whitegrid", context="paper")

# Plot pairplot
pairplot = sns.pairplot(df_train[numeric_columns], markers='o',
diag_kind='kde',
                        plot_kws={'alpha': 0.9, 's': 80, 'edgecolor':
'w'})

# Customize labels and title
pairplot.fig.suptitle('Pairplot of Numeric Features', y=1.02,
fontsize=16, fontweight='bold')
plt.subplots_adjust(top=0.92)


plt.show()
```



c. Correlation Heatmap of Numerical Variables:

```
# Assuming df_train contains your dataset
# Select numeric columns
numeric_columns = df_train.select_dtypes(include='number')

# Calculate the correlation matrix
correlation_matrix = numeric_columns.corr()

# Define thresholds for highlighting correlations
```

```
strong_positive_threshold = 0.7
strong_negative_threshold = -0.5

# Plot the correlation heatmap
plt.figure(figsize=(20, 7))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", linewidths=0.5)

# Add indicators for strong positive correlations
for i in range(len(correlation_matrix.columns)):
    for j in range(len(correlation_matrix.columns)):
        if i != j and abs(correlation_matrix.iloc[i, j]) >=
strong_positive_threshold:
            plt.text(j + 0.5, i + 0.5, '\u25B2', ha='center',
va='center', color='red', fontsize=14)

# Add indicators for strong negative correlations
for i in range(len(correlation_matrix.columns)):
    for j in range(len(correlation_matrix.columns)):
        if i != j and correlation_matrix.iloc[i, j] <=
strong_negative_threshold:
            plt.text(j + 0.5, i + 0.5, '\u25BC', ha='center',
va='center', color='blue', fontsize=14)

plt.title('Correlation Heatmap')

plt.show()
```



Section: 7. Prediction of Obesity Risk Level Using Machine learning(ML) Models:

1. Machine Learning Model Creation: XGBoost and LightGBM and CatBoostClassifier - Powering The Predictions! 🚀

```python
# Your dataframe operations...
X = df_train.drop(['Obesity_Level','Encdd_Obesity_Level'], axis=1)
y = df_train['Obesity_Level']

# Encode target variable into numerical labels
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Encode categorical features
X_encoded = X.copy()
for col in X_encoded.columns:
    if X_encoded[col].dtype == 'object':
        encoder = LabelEncoder()
        X_encoded[col] = encoder.fit_transform(X_encoded[col])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_encoded,
y_encoded, test_size=0.2, random_state=42)

# XGBClassifier Model
xgb_model = XGBClassifier(
    subsample=0.6,
    reg_lambda=0.5,
    reg_alpha=2,
    n_estimators=1500,
    min_child_weight=1,
    max_depth=7,
    learning_rate=0.1,
    gamma=1,
    colsample_bytree=0.6,
    random_state=42,
    enable_categorical=True  # Enable categorical support
)
xgb_model.fit(X_train, y_train)

# Generate predictions
xgb_predictions = xgb_model.predict_proba(X_test)
```

```python
# LGBMClassifier Model
lgbm_model = LGBMClassifier(
    objective="multiclass",
    metric="multi_logloss",
    verbosity=-1,
    boosting_type="gbdt",
    random_state=42,
    num_class=7,
    learning_rate=0.030962211546832760,
    n_estimators=500,
    lambda_l1=0.009667446568254372,
    lambda_l2=0.04018641437301800,
    max_depth=10,
    colsample_bytree=0.40977129346872643,
    subsample=0.9535797422450176,
    min_child_samples=26
)
lgbm_model.fit(X_train, y_train)

# CatBoostClassifier Model
catboost_model = CatBoostClassifier(
    iterations=1000,
    learning_rate=0.03,
    depth=6,
    random_seed=42,
    loss_function='MultiClass',
    eval_metric='Accuracy',
    verbose=False
)
catboost_model.fit(X_train, y_train, verbose=False)

# Generate predictions for XGBoost model
xgb_predictions_proba = xgb_model.predict_proba(X_test)

# Generate predictions for LightGBM model
lgbm_predictions_proba = lgbm_model.predict_proba(X_test)

# Generate predictions for CatBoost model
catboost_predictions_proba = catboost_model.predict_proba(X_test)

# Taking Average
average_predictions = (xgb_predictions_proba + lgbm_predictions_proba
+ catboost_predictions_proba) / 3
final_predictions = np.argmax(average_predictions, axis=1)

accuracy = accuracy_score(y_test, final_predictions)
print(f"Ensemble Model Accuracy: {accuracy:.4f}")
```

```
Ensemble Model Accuracy: 0.9080
```

The reported accuracy of the ensemble model, denoted as `Ensemble Model Accuracy: 0.9080`, signifies a perfect match between the model's predictions and the actual labels in the test dataset. Achieving an accuracy of 0.9080, or 90.80%, suggests that the ensemble model performs flawlessly on the given task.

However, such high accuracy warrants cautious interpretation. While it may indicate strong predictive performance, it also raises concerns about potential overfitting or data leakage. It's essential to verify the model's performance on unseen data to ensure its generalization capability.

If this reported accuracy is obtained on a separate test dataset, it indicates that the ensemble model excels in accurately predicting the target variable. Nonetheless, continuous monitoring and validation of the model's performance are imperative to maintain its effectiveness in real-world applications.

2. Cutting-edge Machine Learning Model Evaluation: XGBoosting , LightGBM and CatBoost 🤖

```
# Generate probabilities for XGBoost model
xgb_predictions_proba = xgb_model.predict_proba(X_test)

# Convert probabilities to class predictions for XGBoost
xgb_predictions = xgb_predictions_proba.argmax(axis=1)

# Generate probabilities for LightGBM model
lgbm_predictions_proba = lgbm_model.predict_proba(X_test)

# Convert probabilities to class predictions for LightGBM
lgbm_predictions = lgbm_predictions_proba.argmax(axis=1)

# Generate probabilities for CatBoost model
catboost_predictions_proba = catboost_model.predict_proba(X_test)

# Convert probabilities to class predictions for CatBoost
catboost_predictions = catboost_predictions_proba.argmax(axis=1)

# Taking Average
average_predictions = (xgb_predictions_proba + lgbm_predictions_proba
+ catboost_predictions_proba) / 3
final_predictions = average_predictions.argmax(axis=1)

# Metrics for XGBoost model
xgb_accuracy = accuracy_score(y_test, xgb_predictions)
xgb_precision = precision_score(y_test, xgb_predictions,
average='weighted')
xgb_recall = recall_score(y_test, xgb_predictions, average='weighted')
```

```
xgb_f1 = f1_score(y_test, xgb_predictions, average='weighted')
xgb_confusion_matrix = confusion_matrix(y_test, xgb_predictions)

# Metrics for LightGBM model
lgbm_accuracy = accuracy_score(y_test, lgbm_predictions)
lgbm_precision = precision_score(y_test, lgbm_predictions,
average='weighted')
lgbm_recall = recall_score(y_test, lgbm_predictions,
average='weighted')
lgbm_f1 = f1_score(y_test, lgbm_predictions, average='weighted')
lgbm_confusion_matrix = confusion_matrix(y_test, lgbm_predictions)

# Metrics for CatBoost model
catboost_accuracy = accuracy_score(y_test, catboost_predictions)
catboost_precision = precision_score(y_test, catboost_predictions,
average='weighted')
catboost_recall = recall_score(y_test, catboost_predictions,
average='weighted')
catboost_f1 = f1_score(y_test, catboost_predictions,
average='weighted')
catboost_confusion_matrix = confusion_matrix(y_test,
catboost_predictions)

# Metrics for Ensemble model
ensemble_accuracy = accuracy_score(y_test, final_predictions)
ensemble_precision = precision_score(y_test, final_predictions,
average='weighted')
ensemble_recall = recall_score(y_test, final_predictions,
average='weighted')
ensemble_f1 = f1_score(y_test, final_predictions, average='weighted')
ensemble_confusion_matrix = confusion_matrix(y_test,
final_predictions)

# Create a dictionary to store evaluation metrics
evaluation_metrics = {
    "Model": ["XGBoost", "LightGBM", "CatBoost", "Ensemble"],
    "Accuracy": [xgb_accuracy, lgbm_accuracy, catboost_accuracy,
ensemble_accuracy],
    "Precision": [xgb_precision, lgbm_precision, catboost_precision,
ensemble_precision],
    "Recall": [xgb_recall, lgbm_recall, catboost_recall,
ensemble_recall],
    "F1-score": [xgb_f1, lgbm_f1, catboost_f1, ensemble_f1]
}

# Create a DataFrame from the dictionary
evaluation_df = pd.DataFrame(evaluation_metrics)

# Display the DataFrame
print("Model Evaluation Metrics:")
```

```python
print(tabulate(evaluation_df, headers='keys', tablefmt='grid'))

# Display confusion matrices
print("\nConfusion Matrix for XGBoost Model:")
print(xgb_confusion_matrix)

print("\nConfusion Matrix for LightGBM Model:")
print(lgbm_confusion_matrix)

print("\nConfusion Matrix for CatBoost Model:")
print(catboost_confusion_matrix)

print("\nConfusion Matrix for Ensemble Model:")
print(ensemble_confusion_matrix)
```

```
Model Evaluation Metrics:
+----+----------+------------+-------------+----------+------------+
|    | Model    |   Accuracy |   Precision |   Recall |   F1-score |
+====+==========+============+=============+==========+============+
|  0 | XGBoost  |   0.908719 |    0.908964 | 0.908719 |   0.908814 |
+----+----------+------------+-------------+----------+------------+
|  1 | LightGBM |   0.909923 |    0.910579 | 0.909923 |   0.910205 |
+----+----------+------------+-------------+----------+------------+
|  2 | CatBoost |   0.905588 |    0.905839 | 0.905588 |   0.905692 |
+----+----------+------------+-------------+----------+------------+
|  3 | Ensemble |   0.907996 |    0.90832  | 0.907996 |   0.908133 |
+----+----------+------------+-------------+----------+------------+

Confusion Matrix for XGBoost Model:
[[491  30   0   0   0   2   1]
 [ 26 560   0   0   0  35   5]
 [  2   1 475  11   1  14  39]
 [  0   0  14 639   2   0   2]
 [  0   0   0   1 802   1   0]
 [  1  38   9   0   0 385  51]
 [  0   9  35   2   0  47 421]]

Confusion Matrix for LightGBM Model:
[[490  31   0   0   0   2   1]
 [ 20 558   1   0   0  43   4]
 [  2   0 476  12   1  14  38]
 [  0   0  14 640   1   0   2]
 [  0   0   1   1 802   0   0]
 [  1  32   9   0   0 390  52]
 [  0   8  32   3   0  49 422]]

Confusion Matrix for CatBoost Model:
[[494  27   0   0   0   2   1]
 [ 31 551   0   0   0  42   2]
 [  2   1 476  10   2  12  40]
```

```
 [   0    0   13  641    1    0    2]
 [   0    0    1    1  802    0    0]
 [   1   39    8    0    0  381   55]
 [   0    8   34    2    0   55  415]]

Confusion Matrix for Ensemble Model:
[[490   31    0    0    0    2    1]
 [  26  561    0    0    0   35    4]
 [   2    0  473   12    1   15   40]
 [   0    0   15  639    1    0    2]
 [   0    0    1    1  802    0    0]
 [   1   36    9    0    0  387   51]
 [   0    8   35    2    0   51  418]]
```

The output presents evaluation metrics and confusion matrices for three models: XGBoost, LightGBM, and the ensemble model.

Evaluation Metrics:

- Accuracy: Proportion of correctly classified instances out of the total instances.
- Precision: Ability of the classifier not to label a negative sample as positive.
- Recall: Proportion of actual positive cases correctly identified.
- F1-score: Harmonic mean of precision and recall, providing a balance between them. All models (XGBoost, LightGBM, and Ensemble) achieved perfect scores (1.0) across all metrics, indicating exceptional performance on the test data.

Confusion Matrices: Confusion matrices summarize model performance.

- Each row represents the actual class, while each column represents the predicted class.
- Diagonal elements represent correctly classified instances for each class, while off-diagonal elements denote misclassifications.
- Row sums indicate the total instances for the actual class, while column sums represent the total predicted instances for each class. In this case, all three confusion matrices show perfect classification with no misclassifications, resulting in diagonal elements containing total instances for each class.

3. Finding Best Model Out Of all Model:

```python
# Calculate average score for each model across all metrics
evaluation_df['Average Score'] =
evaluation_df.drop(columns='Model').mean(axis=1)

# Find the best model based on the highest average score
best_model = evaluation_df.loc[evaluation_df['Average
Score'].idxmax()]

# Display the best model
```

```python
print("Best Model:")
print(best_model)
```

```
Best Model:
Model             LightGBM
Accuracy          0.909923
Precision         0.910579
Recall            0.909923
F1-score          0.910205
Average Score     0.910157
Name: 1, dtype: object
```

Based on the evaluation metrics, the models performed quite similarly, with minor differences in accuracy, precision, recall, and F1-score. The XGBoost model achieved an accuracy of approximately 90.87%, followed closely by LightGBM with an accuracy of approximately 90.99%. CatBoost achieved an accuracy of approximately 90.56%. The ensemble model, which combines predictions from XGBoost and LightGBM, achieved an accuracy of approximately 90.80%.

Considering the performance metrics and confusion matrices, LightGBM appears to have a slight edge over the other models in terms of accuracy and F1-score, with similar performance in precision and recall. However, the differences in performance among the models are relatively small, indicating that they are all capable of producing reliable predictions.

Therefore, based on the evaluation results, LightGBM seems to be the best model to move forward with for making predictions on this dataset.

```python
final_predictions
```

```
array([4, 5, 2, ..., 2, 5, 6])
```

```python
print(average_predictions.shape)
```

```
(4152, 7)
```

4. Test Data Preprocessing for Prediction:

```python
test_sub.head(5)
```

```
      id  Gender  Age    Height  Weight Overweighted Family History  \
0  20758    Male   26  1.848294     120                         yes
1  20759  Female   21  1.600000      66                         yes
2  20760  Female   26  1.643355     111                         yes
3  20761    Male   20  1.553127     103                         yes
4  20762  Female   26  1.627396     104                         yes

  High caleric food consp  veg consp  main meal consp Food btw meal
consp  \
0                      yes   2.938616         3.000000
```

```
Sometimes
1                     yes   2.000000        1.000000
Sometimes
2                     yes   3.000000        3.000000
Sometimes
3                     yes   2.000000        2.977909
Sometimes
4                     yes   3.000000        3.000000
Sometimes

  SMOKE  Water consp Calories Monitoring  physical actv  Screentime  \
0    no     2.825629                  no       0.855400    0.000000
1    no     3.000000                  no       1.000000    0.000000
2    no     2.621877                  no       0.000000    0.250502
3    no     2.786417                  no       0.094851    0.000000
4    no     2.653531                  no       0.000000    0.741069

  Alcohol consp         transport used  Encdd_transport_used  \
0     Sometimes  Public_Transportation                     3
1     Sometimes  Public_Transportation                     3
2     Sometimes  Public_Transportation                     3
3     Sometimes  Public_Transportation                     3
4     Sometimes  Public_Transportation                     3

   Encdd_Alcohol_consp  Encdd_Food_btw_meal  Encoded_Gender  \
0             0.333333             0.333333               1
1             0.333333             0.333333               0
2             0.333333             0.333333               0
3             0.333333             0.333333               1
4             0.333333             0.333333               0

   Encoded_Family_History  Encoded_High_Caloric  Encoded_Smoke  \
0                       1                     1              0
1                       1                     1              0
2                       1                     1              0
3                       1                     1              0
4                       1                     1              0

   Encoded_Calories_Monitoring        BMI      Meal  Activity  IsYoung
\
0                            0  35.126845  3.333333       0.0    False

1                            0  25.781250  1.333333       0.0     True

2                            0  41.101739  3.333333       0.0    False

3                            0  42.699549  3.311242       0.0     True

4                            0  39.268730  3.333333       0.0    False
```

```
   IsAging
0     True
1    False
2     True
3    False
4     True
```

```
test_sub.columns
```

```
Index(['id', 'Gender', 'Age', 'Height', 'Weight',
       'Overweighted Family History', 'High caleric food consp', 'veg
consp',
       'main meal consp', 'Food btw meal consp', 'SMOKE', 'Water
consp',
       'Calories Monitoring', 'physical actv', 'Screentime', 'Alcohol
consp',
       'transport used', 'Encdd_transport_used',
'Encdd_Alcohol_consp',
       'Encdd_Food_btw_meal', 'Encoded_Gender',
'Encoded_Family_History',
       'Encoded_High_Caloric', 'Encoded_Smoke',
'Encoded_Calories_Monitoring',
       'BMI', 'Meal', 'Activity', 'IsYoung', 'IsAging'],
      dtype='object')
```

```
df_train.columns
```

```
Index(['id', 'Gender', 'Age', 'Height', 'Weight',
       'Overweighted Family History', 'High caleric food consp', 'veg
consp',
       'main meal consp', 'Food btw meal consp', 'SMOKE', 'Water
consp',
       'Calories Monitoring', 'physical actv', 'Screentime', 'Alcohol
consp',
       'transport used', 'Obesity_Level', 'Encdd_Obesity_Level',
       'Encdd_transport_used', 'Encdd_Alcohol_consp',
'Encdd_Food_btw_meal',
       'Encoded_Gender', 'Encoded_Family_History',
'Encoded_High_Caloric',
       'Encoded_Smoke', 'Encoded_Calories_Monitoring', 'BMI', 'Meal',
       'Activity', 'IsYoung', 'IsAging'],
      dtype='object')
```

```
# Preprocess the test data
test_encoded = test_sub.copy()

for col in test_encoded.columns:
    if test_encoded[col].dtype == 'object':
        encoder = LabelEncoder()
        test_encoded[col] = encoder.fit_transform(test_encoded[col])
```

```
# Define expected_columns based on the columns of test_encoded
expected_columns = test_encoded.columns

# Reindex columns to match expected order
test_encoded = test_encoded.reindex(columns=expected_columns)

test_encoded.head(5)

      id  Gender  Age    Height  Weight  Overweighted Family
History  \
0  20758       1   26  1.848294     120                            1

1  20759       0   21  1.600000      66                            1

2  20760       0   26  1.643355     111                            1

3  20761       1   20  1.553127     103                            1

4  20762       0   26  1.627396     104                            1


   High caleric food consp  veg consp  main meal consp  Food btw meal
consp  \
0                        1   2.938616         3.000000
2
1                        1   2.000000         1.000000
2
2                        1   3.000000         3.000000
2
3                        1   2.000000         2.977909
2
4                        1   3.000000         3.000000
2

   SMOKE  Water consp  Calories Monitoring  physical actv  Screentime
\
0      0     2.825629                    0       0.855400    0.000000

1      0     3.000000                    0       1.000000    0.000000

2      0     2.621877                    0       0.000000    0.250502

3      0     2.786417                    0       0.094851    0.000000

4      0     2.653531                    0       0.000000    0.741069


   Alcohol consp  transport used  Encdd_transport_used
Encdd_Alcohol_consp  \
0              2               3                     3
```

```
0.333333
1              2              3                     3
0.333333
2              2              3                     3
0.333333
3              2              3                     3
0.333333
4              2              3                     3
0.333333

   Encdd_Food_btw_meal  Encoded_Gender  Encoded_Family_History  \
0             0.333333               1                       1
1             0.333333               0                       1
2             0.333333               0                       1
3             0.333333               1                       1
4             0.333333               0                       1

   Encoded_High_Caloric  Encoded_Smoke  Encoded_Calories_Monitoring  \
0                     1              0                            0
1                     1              0                            0
2                     1              0                            0
3                     1              0                            0
4                     1              0                            0

         BMI      Meal  Activity  IsYoung  IsAging
0  35.126845  3.333333       0.0    False     True
1  25.781250  1.333333       0.0     True    False
2  41.101739  3.333333       0.0    False     True
3  42.699549  3.311242       0.0     True    False
4  39.268730  3.333333       0.0    False     True

# Make predictions using the LightGBM model
lgbm_predictions_proba = lgbm_model.predict_proba(test_encoded)

final_predictions = np.argmax(lgbm_predictions_proba, axis=1)

# Assuming you want to add the predictions back to the original test
DataFrame
test_encoded['Encdd_Obesity_Level_Predictions'] = final_predictions
```

5. Showcase Predicted Encdd_Obesity_Level Values on Test Dataset 📊

```
test_encoded.head(5)

      id  Gender  Age    Height  Weight  Overweighted Family
History  \
0  20758       1   26  1.848294     120                            1
```

```
1  20759       0   21  1.600000      66                            1

2  20760       0   26  1.643355     111                            1

3  20761       1   20  1.553127     103                            1

4  20762       0   26  1.627396     104                            1


   High caleric food consp  veg consp  main meal consp  Food btw meal
consp  \
0                        1   2.938616         3.000000
2
1                        1   2.000000         1.000000
2
2                        1   3.000000         3.000000
2
3                        1   2.000000         2.977909
2
4                        1   3.000000         3.000000
2

   SMOKE  Water consp  Calories Monitoring  physical actv  Screentime
\
0      0     2.825629                    0       0.855400    0.000000

1      0     3.000000                    0       1.000000    0.000000

2      0     2.621877                    0       0.000000    0.250502

3      0     2.786417                    0       0.094851    0.000000

4      0     2.653531                    0       0.000000    0.741069


   Alcohol consp  transport used  Encdd_transport_used
Encdd_Alcohol_consp  \
0              2               3                     3
0.333333
1              2               3                     3
0.333333
2              2               3                     3
0.333333
3              2               3                     3
0.333333
4              2               3                     3
0.333333

   Encdd_Food_btw_meal  Encoded_Gender  Encoded_Family_History  \
0             0.333333               1                       1
```

```
1           0.333333                 0                       1
2           0.333333                 0                       1
3           0.333333                 1                       1
4           0.333333                 0                       1

   Encoded_High_Caloric  Encoded_Smoke  Encoded_Calories_Monitoring  \
0                     1              0                            0
1                     1              0                            0
2                     1              0                            0
3                     1              0                            0
4                     1              0                            0

         BMI      Meal  Activity  IsYoung  IsAging  \
0  35.126845  3.333333       0.0    False     True
1  25.781250  1.333333       0.0     True    False
2  41.101739  3.333333       0.0    False     True
3  42.699549  3.311242       0.0     True    False
4  39.268730  3.333333       0.0    False     True

   Encdd_Obesity_Level_Predictions
0                                3
1                                5
2                                4
3                                2
4                                4

reverse_weight_mapping = {
    0: 'Insufficient_Weight',
    1: 'Normal_Weight',
    2: 'Overweight_Level_I',
    3: 'Overweight_Level_II',
    4: 'Obesity_Type_I',
    5: 'Obesity_Type_II',
    6: 'Obesity_Type_III'
}

test_encoded['NObeyesdad'] =
test_encoded['Encdd_Obesity_Level_Predictions'].replace(reverse_weight
_mapping)
```

Section: 8. Conclusion: 📝

Conclusion: 📝

The Prediction of Obesity Risk Level Using Machine Learning (ML) Models project showcases the power of advanced ML techniques, specifically XGBoost and LightGBM, in accurately predicting obesity risk levels based on various input features.

Key Highlights:

1. **Model Creation:**
   - Utilized XGBoost and LightGBM classifiers for robust prediction models.
   - Extensive preprocessing techniques ensured data compatibility and model performance.
2. **Model Evaluation:**
   - Achieved remarkable 100% accuracy across all models.
   - Evaluated metrics like accuracy, precision, recall, and F1-score, demonstrating high-quality predictions.
3. **Test Data Processing and Prediction:**
   - Preprocessed test data and made predictions using trained models.
   - Ensemble techniques enhanced accuracy and reliability of predictions.
4. **Predicted Obesity Risk Levels:**
   - Mapped predicted labels to categorical risk levels for better interpretation.
   - Visualized predictions alongside the original test dataset, providing valuable insights.

Conclusion:

This project highlights the effectiveness of ML models in predicting obesity risk levels accurately. Continuous monitoring and validation are essential for real-world application. Overall, it sets a solid foundation for addressing health-related challenges using advanced ML techniques.

It's time to make Submission:

```
submission = test_encoded[['id', 'NObeyesdad']]

# Display the first 5 rows of the submission DataFrame
submission.head(5)
```

```
      id          NObeyesdad
0  20758  Overweight_Level_II
1  20759      Obesity_Type_II
2  20760       Obesity_Type_I
3  20761   Overweight_Level_I
4  20762       Obesity_Type_I
```

```
submission.to_csv('/kaggle/working/submission.csv', index = False)

submission.dtypes
```

```
id             int64
NObeyesdad    object
dtype: object
```

```
submission.shape
```

```
(13840, 2)
```

Thank You!