```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'playground-series-s4e2:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-competitions-data%2Fkaggle-v2%2F68479%2F7609535%2Fbundle%2Farchi

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
# %% [markdown]
# ## **<span style="color:red">COMPREHENSIVE ANALYSIS AND PREDICTION OF OBESITY RISK LEVELS USING MACHINE LEARNING TECHNIQUES WITH - (LightGBM) MODEL</span>
# **Author**: **Anamika Kumari**

# %% [markdown]
# ## Table of Contents
#
```

```
# | Table of Contents                       |
# |---------------------------------------|
# | **[Section: 1.Introduction](#Section:-1.-Introduction:)**  |
# |[]() |
# | [1. What is Obesity?](#What-is-Obesity:) |
# | [2. Understanding Obesity and Risk Prediction](#Understanding-Obesity-and-Risk-Prediction:) |
# | [3. Dataset Overview](#Dataset-Overview:) |
# |[]() |
# |**[Section: 2.Importing Libraries and Dataset](#Section:-2.Importing-Libraries-and-Dataset:)**  |
# |[]() |
# | [1. Importing Relevent Libraries](#Importing-Relevent-Libraries:)   |
# | [2. Loading Datasets](#Loading-Datasets:)        |
# |[]() |
# |**[Section: 3. Descriptive Analysis](#Section:-3.-Descriptive-Analysis:)** |
# |[]() |
# | [1. Summary Statistic of dataframe](#1.-Summary-Statistic-of-dataframe:)  |
# | [2. The unique values present in dataset](#2.-The-unique-values-present-in-dataset:) |
# | [3. The count of unique value in the NObeyesdad column](#3.-The-count-of-unique-value-in-the-NObeyesdad-column:)  |
# | [4. Categorical and numerical Variables Analysis](#4.-Categorical-and-numerical-Variables-Analysis:)   |
# |[]() |
# |[]() |
# | [a. Extracting column names for categorical, numerical, and categorical but cardinal variables](#a.-Extracting-column-names-for-categorical,-numerical,-
# | [b. Summary Of All Categorical Variables](#b.-Summary-Of-All-Categorical-Variables:)  |
# | [c. Summary Of All Numerical Variables](#c.-Summary-Of-All-Numerical-Variables:)    |
# |[]() |
# |[]() |
# |**[Section: 4. Data Preprocessing](#Section:-4.-Data-Preprocessing:)**   |
# |[]() |
# | [1. Typeconversion of dataframe](#1.-Typeconversion-of-dataframe:)    |
# | [2. Renaming the Columns](#2.-Renaming-the-Columns:)             |
# | [3. Detecting Columns with Large or Infinite Values](#3.-Detecting-Columns-with-Large-or-Infinite-Values:)          |
# |[]() |
# |**[Section:5. Exploratory Data Analysis and Visualisation-EDAV](#Section:5.-Exploratory-Data-Analysis-and-Visualisation-EDAV:)**    |
# |[]() |
# |**[1. Univariate Analysis](#1.-Univariate-Analysis)**            |
# |[]() |
# |[]() |
# | [a. Countplots for all Variables](#a.-Countplots-for-all-Variables:)         |
# | [b. Analyzing Individual Variables Using Histogram](#b.-Analyzing-Individual-Variables-Using-Histogram:)  |
# | [c. KDE Plots of Numerical Columns](#c.-KDE-Plots-of-Numerical-Columns:)    |
# | [d. Pie Chart and Barplot for categorical variables](#d.-Pie-Chart-and-Barplot-for-categorical-variables:)     |
# | [e. Violin Plot and Box Plot for Numerical variables](#e.-Violin-Plot-and-Box-Plot-for-Numerical-variables:)   |
# |[]() |
# |[]() |
# |**[2. Bivariate Analysis](#2.-Bivariate-Analysis:)**              |
# |[]() |
# |[]() |
# | [a. Scatter plot: AGE V/s Weight with Obesity Level](#a.-Scatter-plot:-AGE-V/s-Weight-with-Obesity-Level:)    |
# | [b. Scatter plot: AGE V/s Height with Obesity Level](#b.-Scatter-plot:-AGE-V/s-Height-with-Obesity-Level:)    |
# | [c. Scatter plot: Height V/s Weight with Obesity Level](#c.-Scatter-plot:-Height-V/s-Weight-with-Obesity-Level:)   |
# | [d. Scatter plot: AGE V/s Weight with Overweighted Family History](#d.-Scatter-plot:-AGE-V/s-Weight-with-Overweighted-Family-History:)    |
# | [e. Scatter plot: AGE V/s height with Overweighted Family History](#e.-Scatter-plot:-AGE-V/s-height-with-Overweighted-Family-History:)    |
# | [f. Scatter plot: Height V/s Weight with Overweighted Family History](#f.-Scatter-plot:-Height-V/s-Weight-with-Overweighted-Family-History:)   |
# | [g. Scatter plot: AGE V/s Weight with Transport use](#g.-Scatter-plot:-AGE-V/s-Weight-with-Transport-use:)    |
# | [h. Scatter plot: AGE V/s Height with Transport use](#h.-Scatter-plot:-AGE-V/s-Height-with-Transport-use:)    |
# | [i. Scatter plot: Height V/s Weight with Transport use](#i.-Scatter-plot:-Height-V/s-Weight-with-Transport-use:)   |
# |[]() |
# |[]() |
# |**[3. Multivariate Analysis](#3.-Multivariate-Analysis:)**        |
# |[]() |
# |[]() |
# | [a. Pair Plot of Variables against Obesity Levels](#a.-Pair-Plot-of-Variables-against-Obesity-Levels:)    |
# | [b. Correlation heatmap for Pearson's correlation coefficient](#b.-Correlation-heatmap-for-Pearson's-correlation-coefficient:)   |
# | [c. Correlation heatmap for Kendall's tau correlation coefficient](#c.-Correlation-heatmap-for-Kendall's-tau-correlation-coefficient:)  |
# | [d. 3D Scatter Plot of Numerical Columns against Obesity Level](#d.-3D-Scatter-Plot-of-Numerical-Columns-against-Obesity-Level:)    |
# |[]() |
# |**[e. Cluster Analysis](#e.-Cluster-Analysis:)**                 |
# |[]() |
# |[]() |
# | [I. K-Means Clustering on Obesity level](#I.-K-Means-Clustering-on-Obesity-level:)   |
# | [II. PCA Plot of numerical variables against obesity level](#II.-PCA-Plot-of-numerical-variables-against-obesity-level:)   |
# |[]() |
# |[]() |
# |**[4. Outlier Analysis](#4.-Outlier-Analysis:)**                 |
# |[]() |
# | [a. Univariate Outlier Analysis](#a.-Univariate-Outlier-Analysis:)     |
# |[]() |
# |[]() |
# | [I. Boxplot Outlier Analysis](#I.-Boxplot-Outlier-Analysis:)      |
# | [II. Detecting outliers using Z-Score](#II.-Detecting-outliers-using-Z-Score:)  |
# | [III. Detecting outliers using Interquartile Range (IQR)](#III.-Detecting-outliers-using-Interquartile-Range-(IQR): )  |
# |[]() |
# |[]() |
# | [b. Multivariate Outlier Analysis](#b.-Multivariate-Outlier-Analysis:)     |
```

```
# | [b. Multivariate Outlier Analysis](#b.-Multivariate-Outlier-Analysis:)     |
# |[]() |
# | [I. Detecting Multivariate Outliers Using Mahalanobis Distance](#I.-Detecting-Multivariate-Outliers-Using-Mahalanobis-Distance:)    |
# | [II. Detecting Multivariate Outliers Using Principal Component Analysis (PCA)](#II.-Detecting-Multivariate-Outliers-Using-Principal-Component-Analysis-(
# | [III. Detecting Cluster-Based Outliers Using KMeans Clustering](#III.-Detecting-Cluster-Based-Outliers-Using-KMeans-Clustering:)    |
# |[]() |
# |[]() |
# |**[5. Feature Engineering:](#5.-Feature-Engineering:)**            |
# |[]() |
# | [a. Encoding Categorical to numerical variables](#a.-Encoding-Categorical-to-numerical-variables:)   |
# | [b. BMI(Body Mass Index) Calculation](#b.-BMI(Body-Mass-Index)-Calculation:)    |
# | [c. Total Meal Consumed:](#c.-Total-Meal-Consumed:)        |
# | [d. Total Activity Frequency Calculation](#d.-Total-Activity-Frequency-Calculation:)          |
# | [e. Ageing process analysis](#e.-Ageing-process-analysis:)             |
# |[]() |
# |**[Section: 6. Analysis & Prediction Using Machine Learning(ML) Model](#Section:-6.-Analysis-&-Prediction-Using-Machine-Learning(ML)-Model:)**   |
# |[]() |
# | [1. Feature Importance Analysis and Visualization](#1.-Feature-Importance-Analysis-and-Visualization:)  |
# |[]() |
# | [a. Feature Importance Analysis  using Random Forest Classifier](#a.-Feature-Importance-Analysis--using-Random-Forest-Classifier:)  |
# | [b. Feature Importance Analysis using XGBoost(XGB) Model](#b.-Feature-Importance-Analysis-using-XGBoost(XGB)-Model:)   |
# | [c. Feature Importance Analysis Using (LightGBM) Classifier Model](#c.-Feature-Importance-Analysis-Using-(LightGBM)-Classifier-Model:) |
# |[]() |
# | [2. Data visualization after Feature Engineering](#2.-Data-visualization-after-Feature-Engineering:)                    |
# |[]() |
# | [a. Bar plot of numerical variables](#a.-Bar-plot-of-numerical-variables:)   |
# | [b. PairPlot of Numerical Variables](#b.-PairPlot-of-Numerical-Variables:)  |
# | [c. Correlation Heatmap of Numerical Variables](#c.-Correlation-Heatmap-of-Numerical-Variables:)   |
# |[]() |
# |**[Section: 7. Prediction of Obesity Risk Level Using Machine learning(ML) Models](#Section:-7.-Prediction-of-Obesity-Risk-Level-Using-Machine-learning(M
# |[]() |
# |[1. Machine Learning Model Creation: XGBoost and LightGBM and CatBoostClassifier - Powering The Predictions! 🚀](#1.-Machine-Learning-Model-Creation:-XGB
# | [2. Cutting-edge Machine Learning Model Evaluation: XGBoosting and LightGBM 🤖](#2.-Cutting-edge-Machine-Learning-Model-Evaluation:-XGBoosting-and-Light
# | [3. Finding Best Model Out Of all Model](#3.-Finding-Best-Model-Out-Of-all-Model:)   |
# | [4. Test Data Preprocessing for Prediction](#4.-Test-Data-Preprocessing-for-Prediction:)   |
# | [5. Showcase Predicted Encdd_Obesity_Level Values on Test Dataset 📊](#5.-Showcase-Predicted-Encdd_Obesity_Level-Values-on-Test-Dataset-📊) |
# |[]() |
# |**[Section: 8. Conclusion: 📝](#Section:-8.-Conclusion:-📝)**  |
# |[]() |
# | [Conclusion: 📝](#Conclusion:-📝) |
# | [It's time to make Submission:](#It's-time-to-make-Submission:) |
# |[]() |
# |[]() |

# %% [markdown]
# # Section: 1. Introduction:

# %% [markdown]
# # <span style="color:blue">**What is Obesity:**</span>
#
#
# **Obesity** is a complex health condition affecting millions globally, with significant implications for morbidity, mortality, and healthcare costs.Obesit
#
# In this project, we undertake a comprehensive analysis to predict obesity risk levels using advanced machine learning techniques.

# %% [markdown]
# <img src="https://www.limarp.com/wp-content/uploads/2023/02/obesity-risk-factors.png" alt="Obesity-Risk-Factors" width="1500">

# %% [markdown]
# # <span style="color:blue">**Understanding Obesity and Risk Prediction:**</span>
#
#
#  - **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](https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight).
```

```
# %% [markdown]
# # <span style="color:blue">**Dataset Overview:**</span>
#
# 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 individu
#
# ### **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**

# %% [markdown]
# # Section: 2.Importing Libraries and Dataset:

# %% [markdown]
# # <span style="color:blue">Importing Relevent Libraries:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:36:55.512087Z","iopub.execute_input":"2024-02-28T14:36:55.512503Z","iopub.status.idle":"2024-02
import os  # Operating system specific functionalities
import numpy as np  # Linear algebra
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

# %% [markdown]
# # <span style="color:blue">Loading Datasets:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:00.597993Z","iopub.execute_input":"2024-02-28T14:37:00.598779Z","iopub.status.idle":"2024-02
# 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))
```

```
    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')

# %% [markdown]
# # Section: 3. Descriptive Analysis:

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:00.833856Z","iopub.execute_input":"2024-02-28T14:37:00.834306Z","iopub.status.idle":"2024-02
print('Number of rows and columns:\n')
df_train.shape

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:00.848241Z","iopub.execute_input":"2024-02-28T14:37:00.848919Z","iopub.status.idle":"2024-02
df_train.head()

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:00.884255Z","iopub.execute_input":"2024-02-28T14:37:00.884632Z","iopub.status.idle":"2024-02
test.head()

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:00.910400Z","iopub.execute_input":"2024-02-28T14:37:00.910787Z","iopub.status.idle":"2024-02
df_train.tail()

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:00.937493Z","iopub.execute_input":"2024-02-28T14:37:00.937911Z","iopub.status.idle":"2024-02
df_train.info()

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:00.992767Z","iopub.execute_input":"2024-02-28T14:37:00.993265Z","iopub.status.idle":"2024-02
print("size of dataframe:",df_train.size)
df_train.dtypes

# %% [markdown]
# # <span style="color:blue">1. Summary Statistic of dataframe:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.005285Z","iopub.execute_input":"2024-02-28T14:37:01.006332Z","iopub.status.idle":"2024-02
df_train.describe().transpose().style.background_gradient(cmap='viridis').format("{:.2f}")

# %% [markdown]
# - **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 a
# - **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'
#

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.134246Z","iopub.execute_input":"2024-02-28T14:37:01.135315Z","iopub.status.idle":"2024-02
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)


# %% [markdown]
# - **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.

# %% [markdown]
# # <span style="color:blue">2. The unique values present in dataset:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.318818Z","iopub.execute_input":"2024-02-28T14:37:01.319310Z","iopub.status.idle":"2024-02
# 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}")


# %% [markdown]
# 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.

# %% [markdown]
# # <span style="color:blue">3. The count of unique value in the NObeyesdad column:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.368984Z","iopub.execute_input":"2024-02-28T14:37:01.369334Z","iopub.status.idle":"2024-02
df_train.groupby('NObeyesdad').count().iloc[:,1]

# %% [markdown]
# - 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".

# %% [markdown]
# # <span style="color:blue">4. Categorical and numerical Variables Analysis:</span>

# %% [markdown]
# # <span style="color:blue">a. Extracting column names for categorical, numerical, and categorical but cardinal variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.405016Z","iopub.execute_input":"2024-02-28T14:37:01.405491Z","iopub.status.idle":"2024-02
# 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)

# %% [markdown]
# - **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.

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.465881Z","iopub.execute_input":"2024-02-28T14:37:01.466356Z","iopub.status.idle":"2024-02
print("Numerical columns:\n", numerical_cols)
print("Categorical columns:\n", categorical_cols)

# %% [markdown]
# # <span style="color:blue">b. Summary Of All Categorical Variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.474289Z","iopub.execute_input":"2024-02-28T14:37:01.474626Z","iopub.status.idle":"2024-02
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)


# %% [markdown]
# # <span style="color:blue">c. Summary Of All Numerical Variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.571024Z","iopub.execute_input":"2024-02-28T14:37:01.571486Z","iopub.status.idle":"2024-02
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))

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

# %% [markdown]
# # Section: 4. Data Preprocessing:

# %% [markdown]
# # <span style="color:blue">1. Typeconversion of dataframe:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.634794Z","iopub.execute_input":"2024-02-28T14:37:01.635160Z","iopub.status.idle":"2024-02
# 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')
```

```
# %% [markdown]
# # <span style="color:blue">2. Renaming the Columns:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.650244Z","iopub.execute_input":"2024-02-28T14:37:01.650705Z","iopub.status.idle":"2024-02
new_column_names = {
    'Gender': 'Gender',
    'Age': 'Age',
    'Height': 'Height',
    '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)


# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.661995Z","iopub.execute_input":"2024-02-28T14:37:01.662502Z","iopub.status.idle":"2024-02
df_train.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.696963Z","iopub.execute_input":"2024-02-28T14:37:01.697439Z","iopub.status.idle":"2024-02
test_sub.head(5)

# %% [markdown] {"execution":{"iopub.status.busy":"2024-02-11T20:35:59.645467Z","iopub.execute_input":"2024-02-11T20:35:59.645800Z","iopub.status.idle":"202
# # <span style="color:blue">3. Detecting Columns with Large or Infinite Values:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.725173Z","iopub.execute_input":"2024-02-28T14:37:01.726697Z","iopub.status.idle":"2024-02
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))

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.740231Z","iopub.execute_input":"2024-02-28T14:37:01.740752Z","iopub.status.idle":"2024-02
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))

# %% [markdown]
# This output indicates that there are no columns in the dataset with infinite or large values.

# %% [markdown]
# # Section:5. Exploratory Data Analysis and Visualisation-EDAV:

# %% [markdown]
# # <span style="color:blue">1. Univariate Analysis:</span>

# %% [markdown]
# # <span style="color:blue">a. Countplots for all Variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:37:01.753743Z","iopub.execute_input":"2024-02-28T14:37:01.755318Z","iopub.status.idle":"2024-02
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()


# %% [markdown]
# # <span style="color:blue">b. Analyzing Individual Variables Using Histogram:</span>
#

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:08.372770Z","iopub.execute_input":"2024-02-28T14:46:08.373677Z","iopub.status.idle":"2024-02
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()

# 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
```

```
# 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()


# %% [markdown]
# # <span style="color:blue">c. KDE Plots of Numerical Columns:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:13.253314Z","iopub.execute_input":"2024-02-28T14:46:13.253786Z","iopub.status.idle":"2024-02
# 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)


# %% [markdown]
# # <span style="color:blue">d. Pie Chart and Barplot for categorical variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:16.823018Z","iopub.execute_input":"2024-02-28T14:46:16.823793Z","iopub.status.idle":"2024-02
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)


# %% [markdown]
# # <span style="color:blue">e. Violin Plot and Box Plot for Numerical variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:20.769735Z","iopub.execute_input":"2024-02-28T14:46:20.770108Z","iopub.status.idle":"2024-02
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)


# %% [markdown]
# # <span style="color:blue">2. Bivariate Analysis:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:25.590177Z","iopub.execute_input":"2024-02-28T14:46:25.590612Z","iopub.status.idle":"2024-02
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()

# %% [markdown]
# # <span style="color:blue">a. Scatter plot: AGE V/s Weight with Obesity Level:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:25.607852Z","iopub.execute_input":"2024-02-28T14:46:25.608331Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Age','Weight','Obesity_Level', df_train)

# %% [markdown]
# # <span style="color:blue">b. Scatter plot: AGE V/s Height with Obesity Level:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:28.435378Z","iopub.execute_input":"2024-02-28T14:46:28.435791Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Age','Height','Obesity_Level',df_train)

# %% [markdown]
# # <span style="color:blue">c. Scatter plot: Height V/s Weight with Obesity Level:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:31.175910Z","iopub.execute_input":"2024-02-28T14:46:31.176281Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Height','Weight','Obesity_Level',df_train)
```

```
# %% [markdown]
# # <span style="color:blue">d. Scatter plot: AGE V/s Weight with Overweighted Family History:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:33.900814Z","iopub.execute_input":"2024-02-28T14:46:33.901663Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Age','Weight','Overweighted Family History',df_train)

# %% [markdown]
# # <span style="color:blue">e. Scatter plot: AGE V/s height with Overweighted Family History:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:36.472004Z","iopub.execute_input":"2024-02-28T14:46:36.472645Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Age','Height','Overweighted Family History',df_train)

# %% [markdown]
# # <span style="color:blue">f. Scatter plot: Height V/s Weight with Overweighted Family History:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:39.008775Z","iopub.execute_input":"2024-02-28T14:46:39.009451Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Height','Weight','Overweighted Family History',df_train)

# %% [markdown]
# # <span style="color:blue">g. Scatter plot: AGE V/s Weight with Transport use:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:41.608491Z","iopub.execute_input":"2024-02-28T14:46:41.608820Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Age','Weight','transport used',df_train)

# %% [markdown]
# # <span style="color:blue">h. Scatter plot: AGE V/s Height with Transport use:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:44.215301Z","iopub.execute_input":"2024-02-28T14:46:44.215677Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Age','Height','transport used',df_train)

# %% [markdown]
# # <span style="color:blue">i. Scatter plot: Height V/s Weight with Transport use:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:46.844851Z","iopub.execute_input":"2024-02-28T14:46:46.845237Z","iopub.status.idle":"2024-02
plot_scatter_relationship('Height','Weight','transport used',df_train)

# %% [markdown]
# # <span style="color:blue">3. Multivariate Analysis:</span>

# %% [markdown]
# # <span style="color:blue">a. Pair Plot of Variables against Obesity Levels:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:46:49.559427Z","iopub.execute_input":"2024-02-28T14:46:49.559759Z","iopub.status.idle":"2024-02
# 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()


# %% [markdown]
# # <span style="color:blue">b. Correlation heatmap for Pearson's correlation coefficient:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:48:42.061372Z","iopub.execute_input":"2024-02-28T14:48:42.061966Z","iopub.status.idle":"2024-02
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')


# %% [markdown]
# # <span style="color:blue">c. Correlation heatmap for Kendall's tau correlation coefficient:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:48:46.926441Z","iopub.execute_input":"2024-02-28T14:48:46.927340Z","iopub.status.idle":"2024-02
# Plot correlation heatmap for Kendall's tau correlation coefficient
plot_correlation_heatmap(df_train_encoded, method='kendall')

# %% [markdown]
# # <span style="color:blue">d. 3D Scatter Plot of Numerical Columns against Obesity Level:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:48:53.538918Z","iopub.execute_input":"2024-02-28T14:48:53.539340Z","iopub.status.idle":"2024-02
# 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()

# %% [markdown]
#  # <span style="color:blue">e. Cluster Analysis:</span>

# %% [markdown]
# # <span style="color:blue">I. K-Means Clustering on Obesity level:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:48:54.675856Z","iopub.execute_input":"2024-02-28T14:48:54.676250Z","iopub.status.idle":"2024-02
# 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)

# %% [markdown]
# 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 t
# - **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 h
# - **Cluster 2**: Relatively balanced distribution across various obesity levels, with no individuals in level 4 and a missing value in level 5. Level 6 ha
#

# %% [markdown]
# # <span style="color:blue">II. PCA Plot of numerical variables against obesity level:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:08.052719Z","iopub.execute_input":"2024-02-28T14:49:08.053107Z","iopub.status.idle":"2024-02
# 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()

# %% [markdown]
# # <span style="color:blue">4. Outlier Analysis:</span>

# %% [markdown]
# # <span style="color:blue">a. Univariate Outlier Analysis:</span>

# %% [markdown]
# # <span style="color:blue">I. Boxplot Outlier Analysis:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:09.382697Z","iopub.execute_input":"2024-02-28T14:49:09.383048Z","iopub.status.idle":"2024-02
# 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')

# %% [markdown]
# # <span style="color:blue">II. Detecting outliers using Z-Score:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:10.939135Z","iopub.execute_input":"2024-02-28T14:49:10.939621Z","iopub.status.idle":"2024-02
# 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')

# %% [markdown]
# # <span style="color:blue">III. Detecting outliers using Interquartile Range (IQR):</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:10.972596Z","iopub.execute_input":"2024-02-28T14:49:10.972980Z","iopub.status.idle":"2024-02
# 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')


# %% [markdown]
# # <span style="color:blue">b. Multivariate Outlier Analysis:</span>

# %% [markdown]
# # <span style="color:blue">I. Detecting Multivariate Outliers Using Mahalanobis Distance:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:11.017627Z","iopub.execute_input":"2024-02-28T14:49:11.018005Z","iopub.status.idle":"2024-02
# 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

# 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)

# %% [markdown]
# # <span style="color:blue">II. Detecting Multivariate Outliers Using Principal Component Analysis (PCA):</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:17.598621Z","iopub.execute_input":"2024-02-28T14:49:17.599057Z","iopub.status.idle":"2024-02
# 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)

# %% [markdown]
# # <span style="color:blue">III. Detecting Cluster-Based Outliers Using KMeans Clustering:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:17.675001Z","iopub.execute_input":"2024-02-28T14:49:17.675555Z","iopub.status.idle":"2024-02
# Select numerical columns for 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)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:27.761699Z","iopub.execute_input":"2024-02-28T14:49:27.762035Z","iopub.status.idle":"2024-02
df_train.drop(columns=['Cluster'], inplace=True)

# %% [markdown]
# # <span style="color:blue">5. Feature Engineering:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:27.773166Z","iopub.execute_input":"2024-02-28T14:49:27.773477Z","iopub.status.idle":"2024-02
# Rename the columns  for train data
test_sub.rename(columns=new_column_names, inplace=True)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:27.785329Z","iopub.execute_input":"2024-02-28T14:49:27.785669Z","iopub.status.idle":"2024-02
test_sub.head(5)

# %% [markdown]
# # <span style="color:blue">a. Encoding Categorical to numerical variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:27.813297Z","iopub.execute_input":"2024-02-28T14:49:27.813978Z","iopub.status.idle":"2024-02
# 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)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:29.520304Z","iopub.execute_input":"2024-02-28T14:49:29.520703Z","iopub.status.idle":"2024-02
df_train.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:29.552391Z","iopub.execute_input":"2024-02-28T14:49:29.552741Z","iopub.status.idle":"2024-02
test_sub.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:29.586310Z","iopub.execute_input":"2024-02-28T14:49:29.586690Z","iopub.status.idle":"2024-02
# 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)

# %% [markdown]
# # <span style="color:blue">b. BMI(Body Mass Index) Calculation:</span>
```

```
# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:31.976576Z","iopub.execute_input":"2024-02-28T14:49:31.976957Z","iopub.status.idle":"2024-02
#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)

# %% [markdown]
# # <span style="color:blue">c. Total Meal Consumed:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:31.987148Z","iopub.execute_input":"2024-02-28T14:49:31.987515Z","iopub.status.idle":"2024-02
# 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']

# %% [markdown]
# # <span style="color:blue">d. Total Activity Frequency Calculation:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:32.000175Z","iopub.execute_input":"2024-02-28T14:49:32.000742Z","iopub.status.idle":"2024-02
# 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']

# %% [markdown]
# # <span style="color:blue">e. Ageing process analysis:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:32.019625Z","iopub.execute_input":"2024-02-28T14:49:32.020014Z","iopub.status.idle":"2024-02
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)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:32.062132Z","iopub.execute_input":"2024-02-28T14:49:32.062613Z","iopub.status.idle":"2024-02
df_train.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:32.098238Z","iopub.execute_input":"2024-02-28T14:49:32.098707Z","iopub.status.idle":"2024-02
test_sub.head(5)

# %% [markdown]
# # Section: 6. Analysis & Prediction Using Machine Learning(ML) Model:

# %% [markdown]
# # <span style="color:blue">1. Feature Importance Analysis and Visualization:</span>

# %% [markdown]
# # <span style="color:blue">a. Feature Importance Analysis  using Random Forest Classifier:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:32.132165Z","iopub.execute_input":"2024-02-28T14:49:32.132540Z","iopub.status.idle":"2024-02
# 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"]))

# %% [markdown]
# # <span style="color:blue">b. Feature Importance Analysis using XGBoost(XGB) Model:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:35.827226Z","iopub.execute_input":"2024-02-28T14:49:35.827565Z","iopub.status.idle":"2024-02
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='blac

# 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()

# %% [markdown]
# # <span style="color:blue">c. Feature Importance Analysis Using (LightGBM) Classifier Model:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:40.609779Z","iopub.execute_input":"2024-02-28T14:49:40.610139Z","iopub.status.idle":"2024-02
# 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()

# %% [markdown]
# # <span style="color:blue">2. Data visualization after Feature Engineering:</span>

# %% [markdown]
# # <span style="color:blue">a. Bar plot of numerical variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:45.063863Z","iopub.execute_input":"2024-02-28T14:49:45.064789Z","iopub.status.idle":"2024-02
# 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()


# %% [markdown]
# # <span style="color:blue">b. PairPlot of Numerical Variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:49:49.072745Z","iopub.execute_input":"2024-02-28T14:49:49.073332Z","iopub.status.idle":"2024-02
# 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()

# %% [markdown]
# # <span style="color:blue">c. Correlation Heatmap of Numerical Variables:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:54:07.789467Z","iopub.execute_input":"2024-02-28T14:54:07.790113Z","iopub.status.idle":"2024-02
# 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()

# %% [markdown]
# # Section: 7. Prediction of Obesity Risk Level Using Machine learning(ML) Models:

# %% [markdown]
# # <span style="color:blue">1. Machine Learning Model Creation: XGBoost and LightGBM and CatBoostClassifier - Powering The Predictions! 🚀</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:54:09.963834Z","iopub.execute_input":"2024-02-28T14:54:09.964723Z","iopub.status.idle":"2024-02
# 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)


# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:54:47.754400Z","iopub.execute_input":"2024-02-28T14:54:47.754721Z","iopub.status.idle":"2024-02
# 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}")

# %% [markdown]
# The reported accuracy of the ensemble model, denoted as `Ensemble Model Accuracy: 0.9080`, signifies a perfect match between the model's predictions and t
#
# However, such high accuracy warrants cautious interpretation. While it may indicate strong predictive performance, it also raises concerns about potential
#
# If this reported accuracy is obtained on a separate test dataset, it indicates that the ensemble model excels in accurately predicting the target variable

# %% [markdown]
# # <span style="color:blue">2. Cutting-edge Machine Learning Model Evaluation: XGBoosting , LightGBM and CatBoost 🤖</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:27.798636Z","iopub.execute_input":"2024-02-28T14:55:27.799035Z","iopub.status.idle":"2024-02
# 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:")
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)

# %% [markdown]
# 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.
```

```
# - 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 instanc

# %% [markdown]
# # <span style="color:blue">3. Finding Best Model Out Of all Model:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:29.893459Z","iopub.execute_input":"2024-02-28T14:55:29.894004Z","iopub.status.idle":"2024-02
# 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
print("Best Model:")
print(best_model)

# %% [markdown]
# Based on the evaluation metrics, the models performed quite similarly, with minor differences in accuracy, precision, recall, and F1-score. The XGBoost mo
#
# 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
#
# Therefore, based on the evaluation results, LightGBM seems to be the best model to move forward with for making predictions on this dataset.

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:29.905821Z","iopub.execute_input":"2024-02-28T14:55:29.906189Z","iopub.status.idle":"2024-02
final_predictions

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:29.917633Z","iopub.execute_input":"2024-02-28T14:55:29.917971Z","iopub.status.idle":"2024-02
print(average_predictions.shape)

# %% [markdown]
# # <span style="color:blue">4. Test Data Preprocessing for Prediction:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:29.932154Z","iopub.execute_input":"2024-02-28T14:55:29.932525Z","iopub.status.idle":"2024-02
test_sub.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:29.967613Z","iopub.execute_input":"2024-02-28T14:55:29.967989Z","iopub.status.idle":"2024-02
test_sub.columns

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:29.977807Z","iopub.execute_input":"2024-02-28T14:55:29.978236Z","iopub.status.idle":"2024-02
df_train.columns

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:29.990035Z","iopub.execute_input":"2024-02-28T14:55:29.990383Z","iopub.status.idle":"2024-02
# 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)


# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:30.048226Z","iopub.execute_input":"2024-02-28T14:55:30.048618Z","iopub.status.idle":"2024-02
test_encoded.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:30.084303Z","iopub.execute_input":"2024-02-28T14:55:30.084669Z","iopub.status.idle":"2024-02
# Make predictions using the LightGBM model
lgbm_predictions_proba = lgbm_model.predict_proba(test_encoded)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:35.367132Z","iopub.execute_input":"2024-02-28T14:55:35.367502Z","iopub.status.idle":"2024-02
final_predictions = np.argmax(lgbm_predictions_proba, axis=1)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:35.389525Z","iopub.execute_input":"2024-02-28T14:55:35.389965Z","iopub.status.idle":"2024-02
# Assuming you want to add the predictions back to the original test DataFrame
test_encoded['Encdd_Obesity_Level_Predictions'] = final_predictions

# %% [markdown]
# # <span style="color:blue">5. Showcase Predicted Encdd_Obesity_Level Values on Test Dataset 📊</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:35.398148Z","iopub.execute_input":"2024-02-28T14:55:35.398486Z","iopub.status.idle":"2024-02
```

```
test_encoded.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:35.432845Z","iopub.execute_input":"2024-02-28T14:55:35.433667Z","iopub.status.idle":"2024-02
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)

# %% [markdown]
# # Section: 8. Conclusion:

# %% [markdown]
# ### Conclusion:
#
# The Prediction of Obesity Risk Level Using Machine Learning (ML) Models project showcases the power of advanced ML techniques, specifically XGBoost and Li
#
# #### 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 fo

# %% [markdown]
# # <span style="color:blue">It's time to make Submission:</span>

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:35.445772Z","iopub.execute_input":"2024-02-28T14:55:35.446108Z","iopub.status.idle":"2024-02
submission = test_encoded[['id', 'NObeyesdad']]

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

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:35.465566Z","iopub.execute_input":"2024-02-28T14:55:35.465939Z","iopub.status.idle":"2024-02
submission.to_csv('/kaggle/working/submission.csv', index = False)

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:35.503736Z","iopub.execute_input":"2024-02-28T14:55:35.504151Z","iopub.status.idle":"2024-02
submission.dtypes

# %% [code] {"execution":{"iopub.status.busy":"2024-02-28T14:55:35.513194Z","iopub.execute_input":"2024-02-28T14:55:35.513562Z","iopub.status.idle":"2024-02
submission.shape

# %% [markdown]
# # Thank You!
```