AUTHOR : OM CHOKSI

Obesity

Obesity, which causes physical and mental problems, is a global health problem with serious consequences. The prevalence of obesity is increasing steadily, and therefore, new research is needed that examines the influencing factors of obesity and how to predict the occurrence of the condition according to these factors.

" https://www.semanticscholar.org/paper/Estimation-of-Obesity-Levels-with-a-Trained-Neural-Ya%C4%9F%C4%B1n-G%C3%BCl%C3%BC/2c1eab51db154493d225c8b86ba885bbaf147a2c "

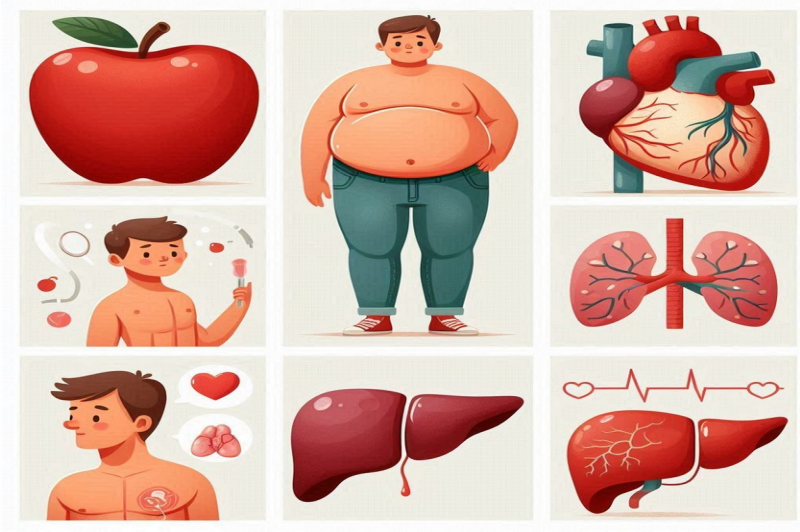
Dataset Information

This dataset include data for the estimation of obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition. The data contains 17 attributes and 2111 records, the records are labeled with the class variable NObesity (Obesity Level), that allows classification of the data using the values of Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II and Obesity Type III. 77% of the data was generated synthetically using the Weka tool and the SMOTE filter, 23% of the data was collected directly from users through a web platform.

Gender: Feature, Categorical, "Gender" Age : Feature, Continuous, "Age" Height: Feature, Continuous Weight: Feature Continuous family_history_with_overweight: Feature, Binary, " Has a family member suffered or suffers from overweight? "

FAVC : Feature, Binary, " Do you eat high caloric food frequently? " FCVC : Feature, Integer, " Do you usually eat vegetables in your meals? " NCP : Feature, Continuous, " How many main meals do you have daily? " CAEC : Feature, Categorical, " Do you eat any food between meals? " SMOKE : Feature, Binary, " Do you smoke? " CH2O: Feature, Continuous, " How much water do you drink daily? " SCC: Feature, Binary, " Do you monitor the calories you eat daily? " FAF: Feature, Continuous, " How often do you have physical activity? " TUE : Feature, Integer, " How much time do you use technological devices such as cell phone, videogames, television, computer and others? "

CALC : Feature, Categorical, " How often do you drink alcohol? " MTRANS : Feature, Categorical, " Which transportation do you usually use? " NObeyesdad : Target, Categorical, "Obesity level"



Aim

- I made this version for my practise , So The Start is Very basic , Data is Pure From Null Values only Some Duplicate There , Which Were Removed.
- The EDA Version Contains Just Visualizations , Not any Observations.
- The Normalization is Done With The Help of Quantile transformer.
- Encoding is Done By Pandas [pd.get_dummies].
- Then The Params are Tunned by Optuna , I did't Add The Optune Code There.

About Data

Dataset Description

The dataset for this competition (both train and test) was generated from a deep learning model trained on the Obesity or CVD risk dataset. Feature distributions are close to, but not exactly the same, as the original. Feel free to use the original dataset as part of this competition, both to explore differences as well as to see whether incorporating the original in training improves model performance.

Note: This dataset is particularly well suited for visualizations, clustering, and general EDA. Show off your skills!

Files

- **train.csv**: the training dataset; NObeyesdad is the categorical target
- **test.csv**: the test dataset; your objective is to predict the class of NObeyesdad for each row
- **sample_submission.csv**: a sample submission file in the correct format

Step 1 | Importing Data & Libraries

```
# Import Basis
import pandas as pd
```

```
import optuna
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
from colorama import Fore, Style, init;
# Import necessary libraries
from IPython.display import display, HTML
from scipy.stats import skew  # Import the skew function
# Import Plotly.go
import plotly.graph_objects as go
# import Subplots
from plotly.subplots import make_subplots
# Ignore warnings
import warnings
warnings.filterwarnings("ignore")
# Model Classifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier, VotingRegressor
from sklearn.model_selection import RepeatedStratifiedKFold,
cross_val_score
from lightgbm import LGBMClassifier
from sklearn.preprocessing import LabelEncoder, MinMaxScaler ,
StandardScaler , QuantileTransformer
import lightgbm as lgb
from sklearn.model_selection import cross_val_score
from catboost import CatBoostClassifier
from sklearn.metrics import *
# Paellete
palette = ["#00B1D2FF", "#FDDB27FF"]
color_palette = sns.color_palette(palette)
# Remove Warnings
import warnings
warnings.filterwarnings("ignore")
# Set the option to display all columns
pd.set_option('display.max_columns', None)
```

Step 2 | Loading Data

```
# Load Submission Data
d_s =
pd.read_csv('/kaggle/input/playground-series-s4e2/sample_submission.cs
v')
# Load test Data
te_d = pd.read_csv('/kaggle/input/playground-series-s4e2/test.csv')
#Train Data
tr_d = pd.read_csv('/kaggle/input/playground-series-s4e2/train.csv')
# Orignal Data
O_D =
```

```
pd.read_csv('/kaggle/input/obesity-levels/ObesityDataSet_raw_and_data_
sinthetic.csv')

# Dropping Id from  Train
tr_d.drop(columns=['id'], inplace=True)
te_d.drop(columns=['id'], inplace=True)

# Concat Train and Original Data
tr_d = pd.concat([tr_d, O_D], ignore_index=True)
```

Step 3 Data overview

Use Case Of this Function

- This Function Will Help to Load and Give a Overview of Data.
- This Functions Takes Data as Inputs and Given all the Information about Data Like Head , shape , Info , Describe , Null Values and Duplicates Values etc.

```
# Text Color
def PrintColor(text: str, color=Fore.CYAN, style=Style.BRIGHT):
    "Prints color outputs using colorama using a text F-string"
    print(style + color + text + Style.RESET_ALL)

# Text For Main Heading
def print_blue_large(text):
    PrintColor(text, Fore.BLUE + Style.BRIGHT)

# Main Heading
def print_boxed_blue_heading(text):
    length = len(text) + 4
    print(f"\n{Style.BRIGHT}{Fore.BLUE}{'='*length}{Style.RESET_ALL}")
    print(f"{Style.BRIGHT}{Fore.BLUE}| {text} |{Style.RESET_ALL}")
    print(f"{Style.BRIGHT}{Fore.BLUE}{'='*length}{Style.RESET_ALL}")

# Function to Overview Data
def data_overview(tr_d, te_d):

    # Display head of the training dataset nicely
    print_boxed_blue_heading("The Head Of Train Dataset is:")
    display(HTML(tr_d.head(5).to_html(index=False).replace('<table
border="1" class="dataframe">', '<table style="border: 2px solid
blue;">')))

    print('\n')

    # Display head of the test dataset nicely
    print_boxed_blue_heading("The Head Of Test Dataset is:")
    display(HTML(te_d.head(5).to_html(index=False).replace('<table
border="1" class="dataframe">', '<table style="border: 2px solid
blue;">')))
```

```
    print('\n')

    # Shapes of Train and Test
    print_boxed_blue_heading("Shape Information:")
    PrintColor(f"The Shape Of Train Data is {tr_d.shape} || No of Rows
is : {tr_d.shape[0]} and Columns is {tr_d.shape[1]}", Fore.CYAN)
    print('\n')
    PrintColor(f"The Shape Of Test Data is {te_d.shape}  || No of Rows
is : {te_d.shape[0]} and Columns is {te_d.shape[1]}", Fore.CYAN)
    print('\n')

    # Info of Both Datasets
    print_boxed_blue_heading("Dataset Information:")
    PrintColor(f"\nThe Info Of Train Dataset", Fore.CYAN)
    tr_d.info()
    PrintColor(f"\nThe Info Of Test Dataset is", Fore.CYAN)
    te_d.info()
    print('\n')

    # Describe Both
    print_boxed_blue_heading("Numerical Summary:")
    PrintColor(f"\nThe Numerical Summary of Train is", Fore.CYAN)
    display(tr_d.describe().style.set_caption("Train Data
Summary").set_table_styles([{'selector': 'caption', 'props':
[('color', 'blue')]}]))
    PrintColor(f"\nThe Numerical Summary of Test is", Fore.CYAN)
    display(te_d.describe().style.set_caption("Test Data
Summary").set_table_styles([{'selector': 'caption', 'props':
[('color', 'blue')]}]))
    print('\n')

    # Null Values in Train and Test
    print_boxed_blue_heading("Null Values:")
    PrintColor("\nNull Values in Train", Fore.CYAN)
    print(tr_d.isnull().sum())
    PrintColor("\nNull Values in Test", Fore.CYAN)
    print(te_d.isnull().sum())
    print('\n')

    # Duplicates Values in Train and Test
    print_boxed_blue_heading("Duplicate Values:")
    PrintColor("\nDuplicates Values in Train", Fore.CYAN)
    print(tr_d.duplicated().sum())
    PrintColor("\nDuplicates Values in Test", Fore.CYAN)
    print(te_d.duplicated().sum())

# Data Overview
data_overview(tr_d,te_d)
```

```
==================================
| The Head Of Train Dataset is: |
==================================

<IPython.core.display.HTML object>



==================================
| The Head Of Test Dataset is: |
==================================

<IPython.core.display.HTML object>



======================
| Shape Information: |
======================
The Shape Of Train Data is (22869, 17) || No of Rows is : 22869 and
Columns is 17


The Shape Of Test Data is (13840, 16)  || No of Rows is : 13840 and
Columns is 16



========================
| Dataset Information: |
========================

The Info Of Train Dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22869 entries, 0 to 22868
Data columns (total 17 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Gender                            22869 non-null  object
 1   Age                               22869 non-null  float64
 2   Height                            22869 non-null  float64
 3   Weight                            22869 non-null  float64
 4   family_history_with_overweight    22869 non-null  object
 5   FAVC                              22869 non-null  object
 6   FCVC                              22869 non-null  float64
 7   NCP                               22869 non-null  float64
 8   CAEC                              22869 non-null  object
 9   SMOKE                             22869 non-null  object
```

```
 10  CH2O                            22869 non-null  float64
 11  SCC                             22869 non-null  object
 12  FAF                             22869 non-null  float64
 13  TUE                             22869 non-null  float64
 14  CALC                            22869 non-null  object
 15  MTRANS                          22869 non-null  object
 16  NObeyesdad                      22869 non-null  object
dtypes: float64(8), object(9)
memory usage: 3.0+ MB

The Info Of Test Dataset is
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13840 entries, 0 to 13839
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          13840 non-null  object
 1   Age                             13840 non-null  float64
 2   Height                          13840 non-null  float64
 3   Weight                          13840 non-null  float64
 4   family_history_with_overweight  13840 non-null  object
 5   FAVC                            13840 non-null  object
 6   FCVC                            13840 non-null  float64
 7   NCP                             13840 non-null  float64
 8   CAEC                            13840 non-null  object
 9   SMOKE                           13840 non-null  object
 10  CH2O                            13840 non-null  float64
 11  SCC                             13840 non-null  object
 12  FAF                             13840 non-null  float64
 13  TUE                             13840 non-null  float64
 14  CALC                            13840 non-null  object
 15  MTRANS                          13840 non-null  object
dtypes: float64(8), object(8)
memory usage: 1.7+ MB



======================
| Numerical Summary: |
======================

The Numerical Summary of Train is

<pandas.io.formats.style.Styler at 0x7e8e54116bf0>


The Numerical Summary of Test is

<pandas.io.formats.style.Styler at 0x7e8dfe545d50>
```

```
================
| Null Values: |
================

Null Values in Train
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64

Null Values in Test
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
dtype: int64



=====================
| Duplicate Values: |
```

```
====================

Duplicates Values in Train
24

Duplicates Values in Test
0
```

Data Have No Null Values and Have Some Duplicates

```
# Drop Duplicates
tr_d.drop_duplicates(inplace=True)
```

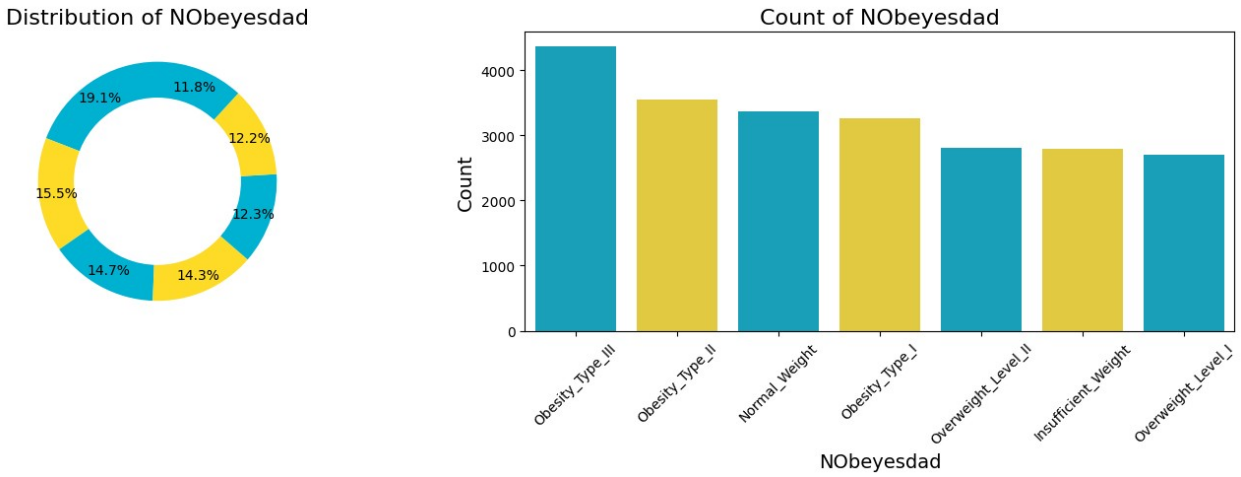
Step 4 | Feature Engineering

```
def Feature_E(df):
    # Feature 1: BMI (Body Mass Index)
    df['BMI'] = df['Weight'] / (df['Height'] / 100)**2

    # Feature 2: Number of meals per day
    df['Meals_Per_Day'] = df['FCVC'] + df['NCP']

    # Feature 3: Total physical activity score
    df['Total_Activity_Score'] = df['FAF'] * df['TUE']

    # Feature 5: Age category (e.g., young, adult, elderly)
    df['Age_Category'] = pd.cut(df['Age'], bins=[0, 18, 60,
float('inf')], labels=['Young', 'Adult', 'Elderly'])

    # Feature 6: Water intake per kg of body weight
    df['Water_Intake_Per_Kg'] = df['CH2O'] / df['Weight']

    return df

# Apply the feature engineering function to your training and testing
datasets
tr_d = Feature_E(tr_d)
te_d = Feature_E(te_d)
```

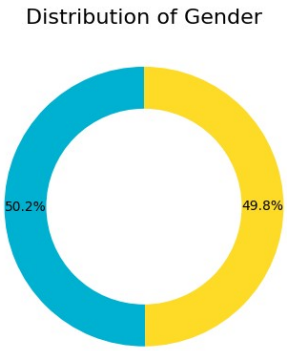
Step 5 | EDA Analysis

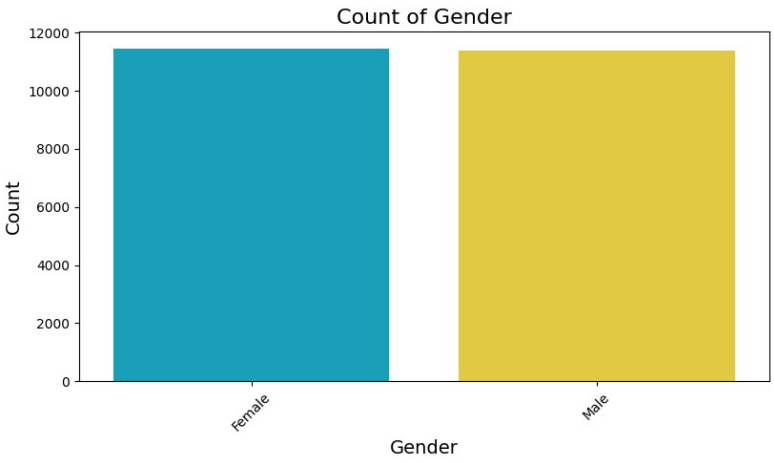
```
# # Function to Plot Single Pie and Bar Plot
def single_plot_distribution(column_name, dataframe):
    # Get the value counts of the specified column
    value_counts = dataframe[column_name].value_counts()

    # Set up the figure with two subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5),
gridspec_kw={'width_ratios': [1, 1]})
```

```
    # Donut pie chart
    pie_colors = palette[0:3]
    ax1.pie(value_counts, autopct='%0.001f%%', startangle=90,
pctdistance=0.85, colors=pie_colors, labels=None)
    centre_circle = plt.Circle((0,0),0.70,fc='white')
    ax1.add_artist(centre_circle)
    ax1.set_title(f'Distribution of {column_name}', fontsize=16)

    # Bar chart
    bar_colors = palette[0:3]
    sns.barplot(x=value_counts.index, y=value_counts.values, ax=ax2,
palette=bar_colors,)
    ax2.set_title(f'Count of {column_name}', fontsize=16)
    ax2.set_xlabel(column_name, fontsize=14)
    ax2.set_ylabel('Count', fontsize=14)

    # Rotate x-axis labels for better readability
    ax2.tick_params(axis='x', rotation=45)

    # Show the plots
    plt.tight_layout()
    plt.show()

# NObeyesdad Ditribution
single_plot_distribution('NObeyesdad',tr_d)
```

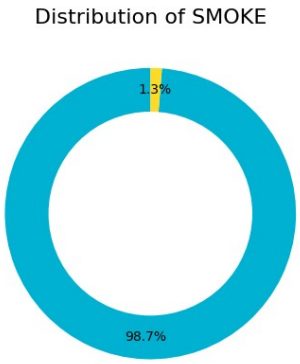


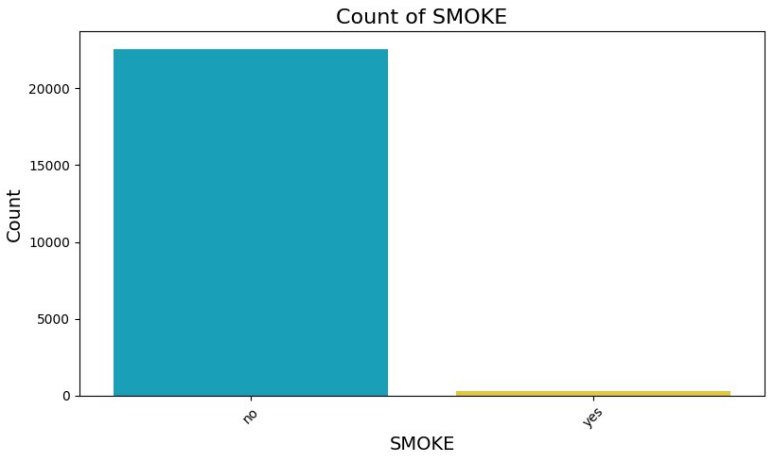

```
# Gender Ditribution
single_plot_distribution('Gender',tr_d)
```



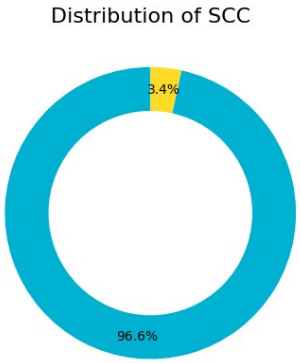


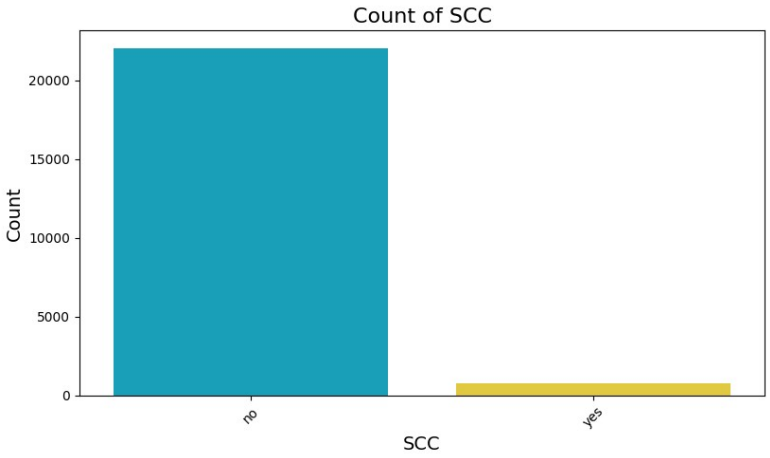

```
# SMOKE Ditribution
single_plot_distribution('SMOKE',tr_d)
```



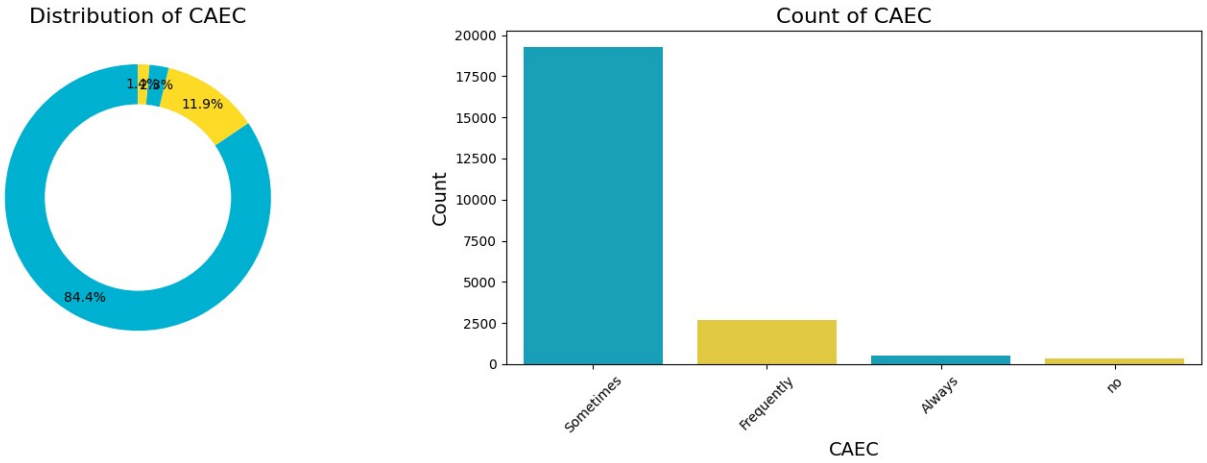



```
# SCC Ditribution
single_plot_distribution('SCC',tr_d)
```

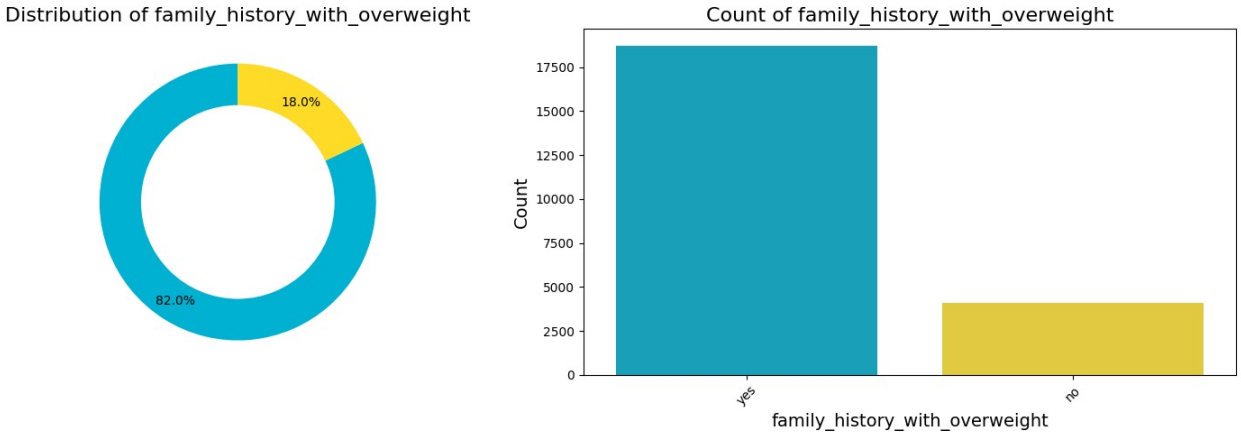




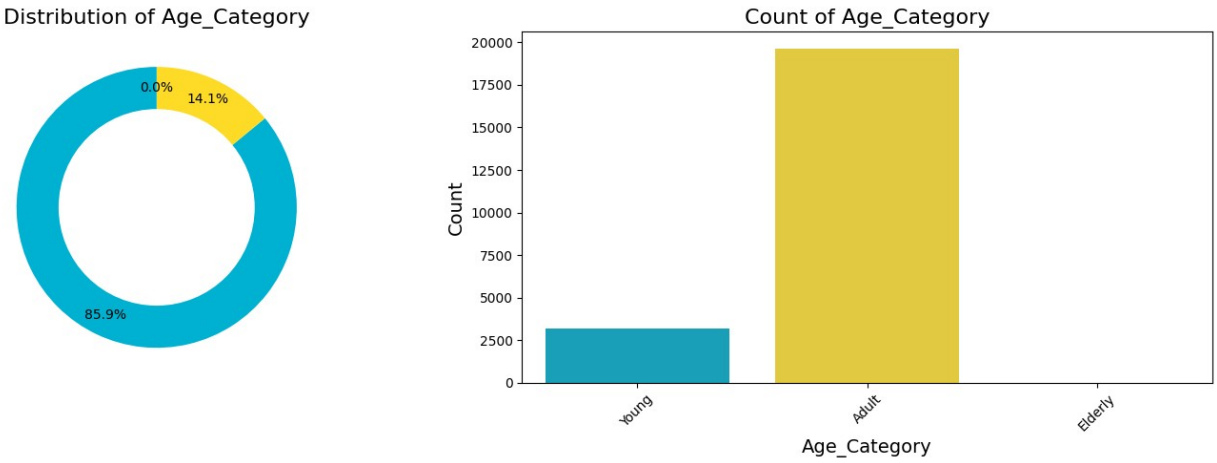
```
# CAEC Ditribution
single_plot_distribution('CAEC',tr_d)
```



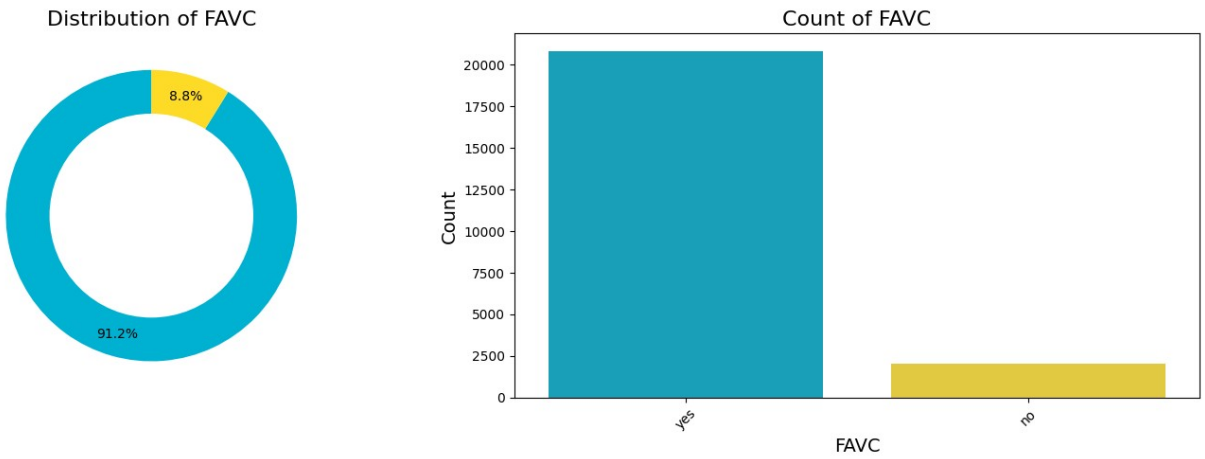

```
# family_history_with_overweight  Ditribution
single_plot_distribution('family_history_with_overweight',tr_d)
```



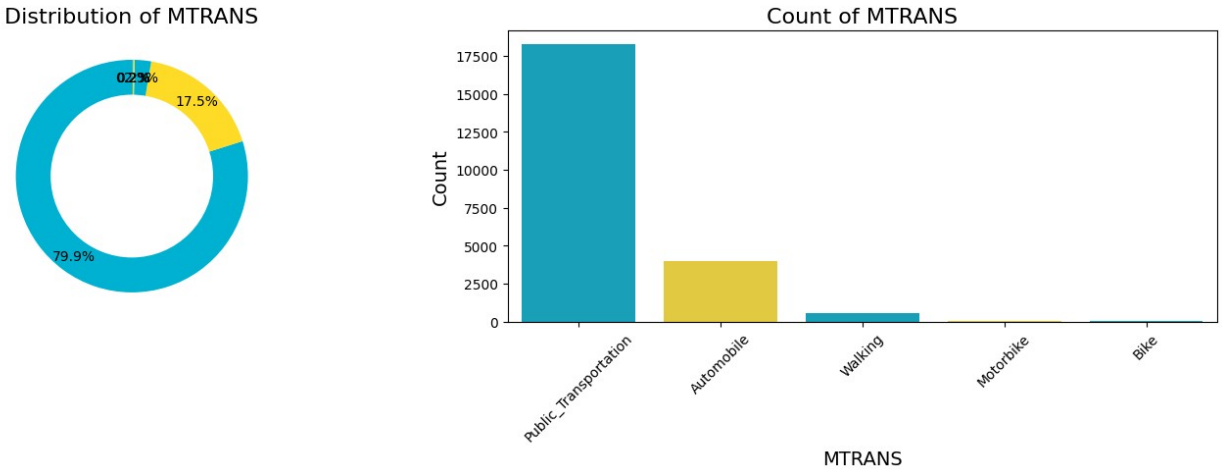

```
# Age_Category Ditribution
single_plot_distribution('Age_Category',tr_d)
```



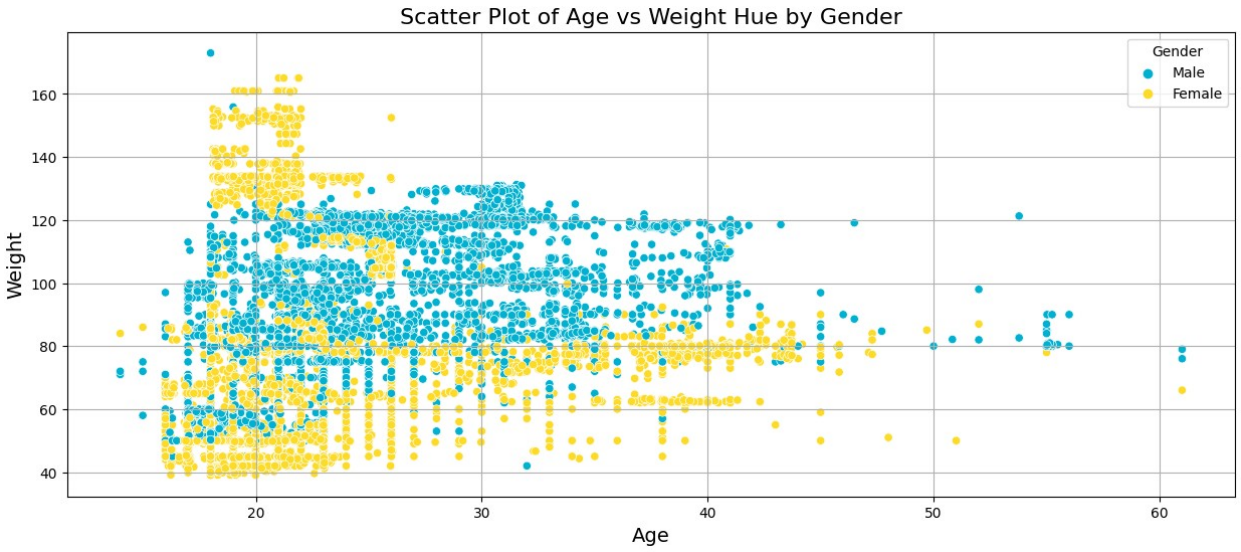

```
# FAVC Ditribution
single_plot_distribution('FAVC',tr_d)
```



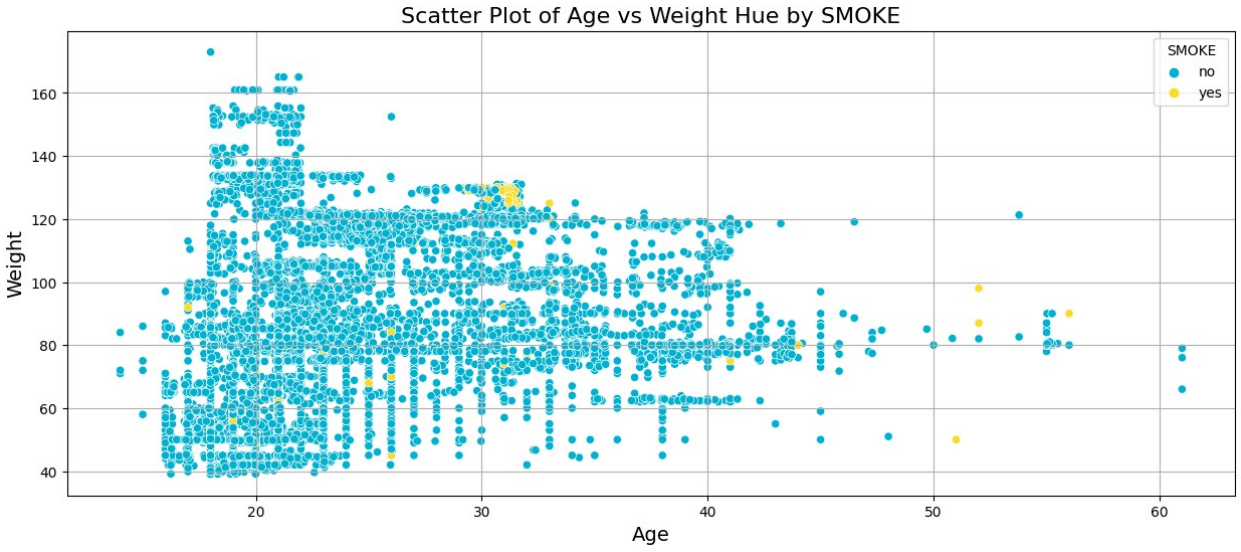

```
# MTRANS Ditribution
single_plot_distribution('MTRANS',tr_d)
```



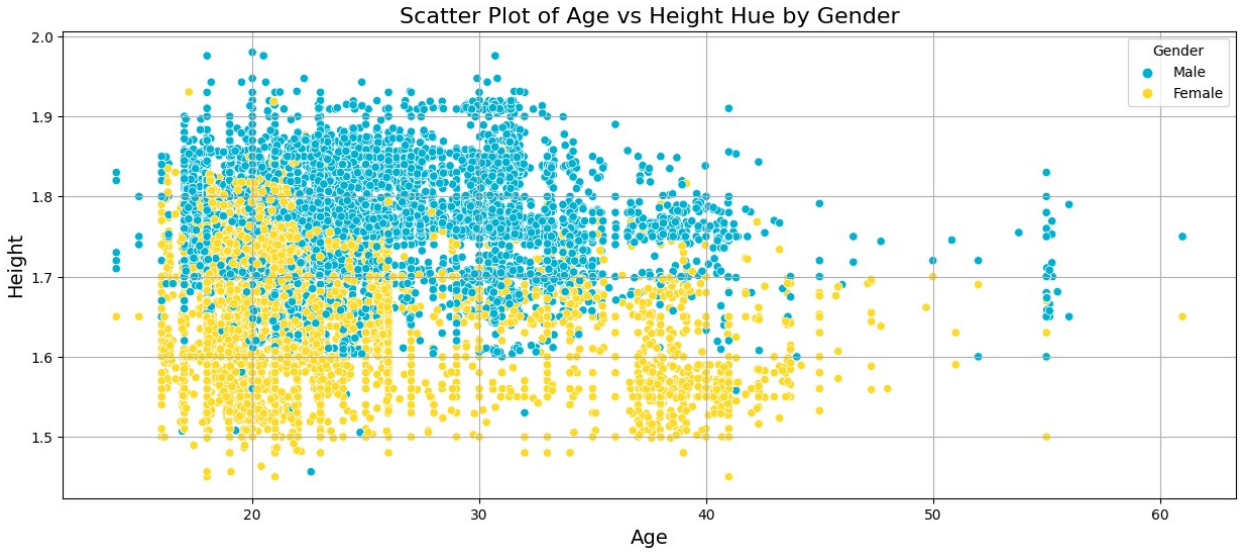
```
# Scatter Plot to Show Realationship Bw 2 Cols
def advanced_scatter_plot(x_column, y_column, target_column,
dataframe):
    plt.figure(figsize=(15, 6))
    sns.scatterplot(x=x_column, y=y_column, hue=target_column,
data=dataframe, palette=palette[0:3])
    plt.title(f'Scatter Plot of {x_column} vs {y_column} Hue by
{target_column}', fontsize=16)
    plt.xlabel(x_column, fontsize=14)
    plt.ylabel(y_column, fontsize=14)
    plt.legend(title=target_column)
    plt.grid(True)
    plt.show()

# Scatter Plot | to Show Age vs Weight RealtionShip
advanced_scatter_plot('Age', 'Weight', 'Gender', tr_d)
```



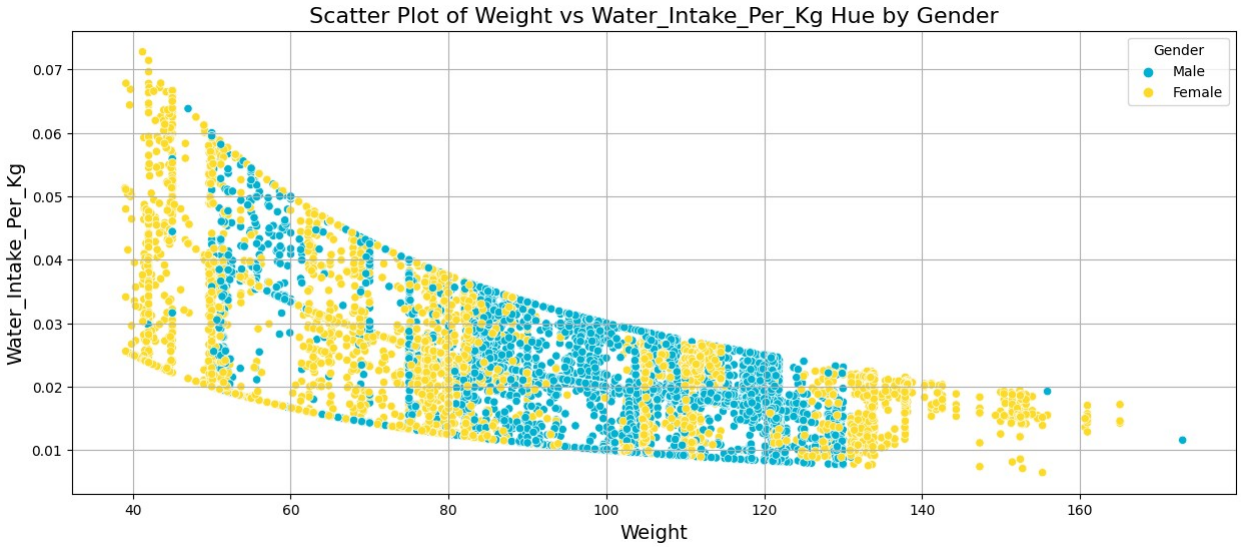

```
# Scatter Plot | to Show Age vs Weight RealtionShip
advanced_scatter_plot('Age', 'Weight', 'SMOKE', tr_d)
```



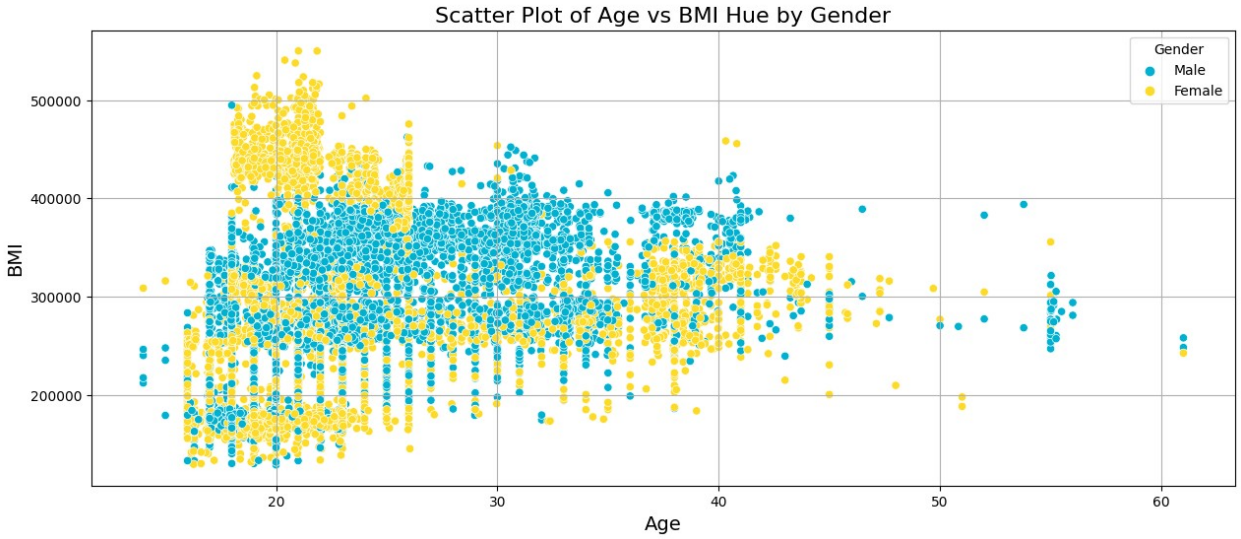

```
# Scatter Plot | to Show Age vs Height RealtionShip
advanced_scatter_plot('Age', 'Height', 'Gender', tr_d)
```



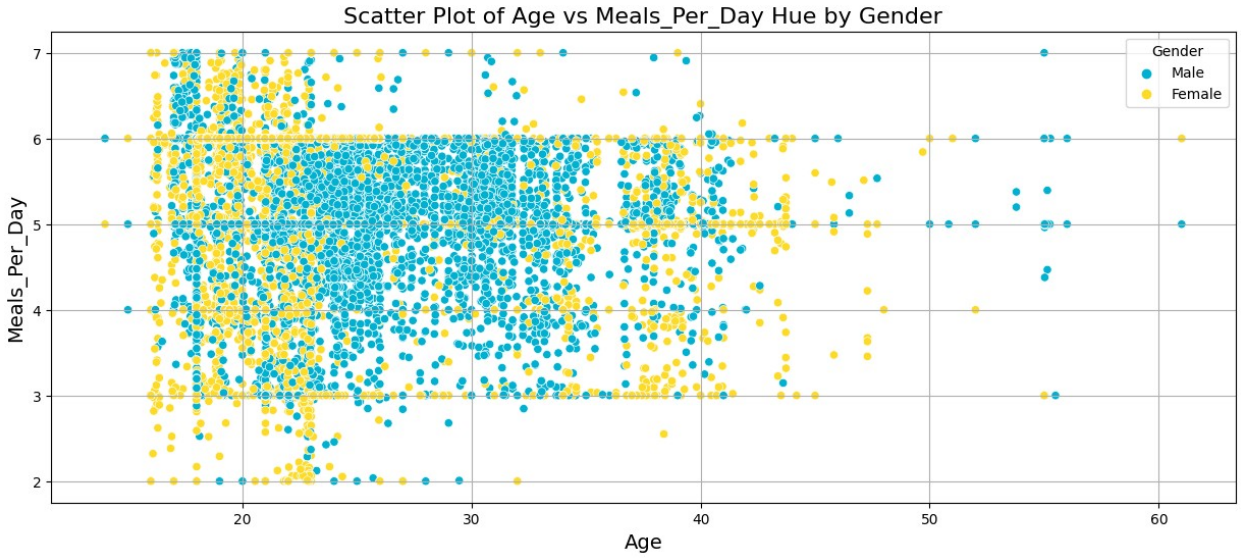

```
# # Scatter Plot | to Show Weight vs Water_Intake_Per_Kg RealtionShip
advanced_scatter_plot('Weight', 'Water_Intake_Per_Kg', 'Gender', tr_d)
```



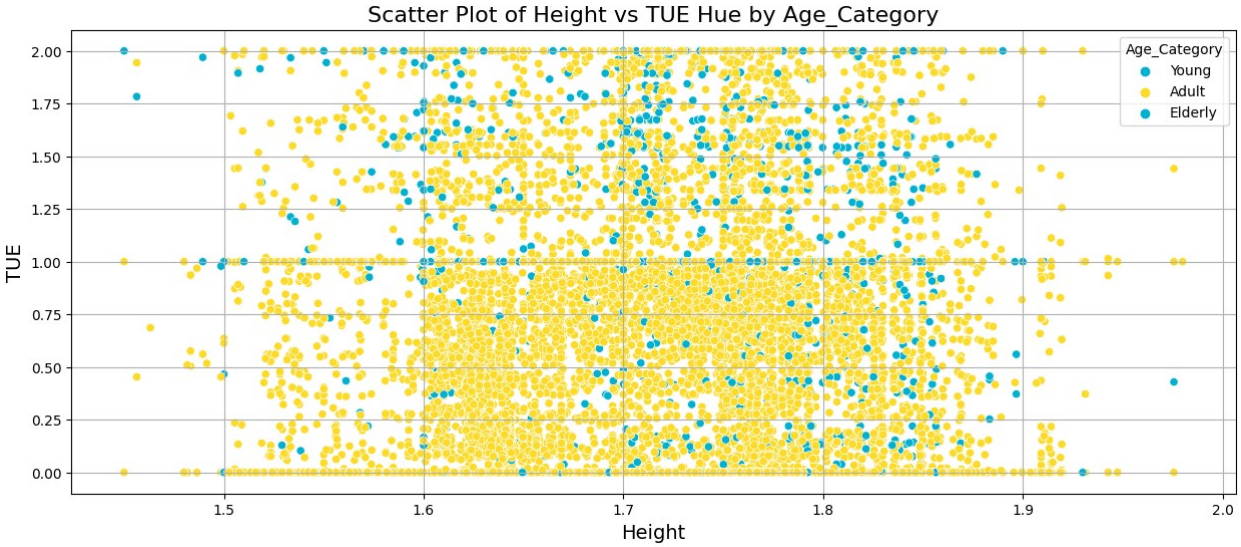

```
# # Scatter Plot | to Show Age vs BMI RealtionShip
advanced_scatter_plot('Age', 'BMI', 'Gender', tr_d)
```




```
# # Scatter Plot | to Show Age vs Meals_Per_Day RealtionShip
advanced_scatter_plot('Age', 'Meals_Per_Day', 'Gender', tr_d)
```

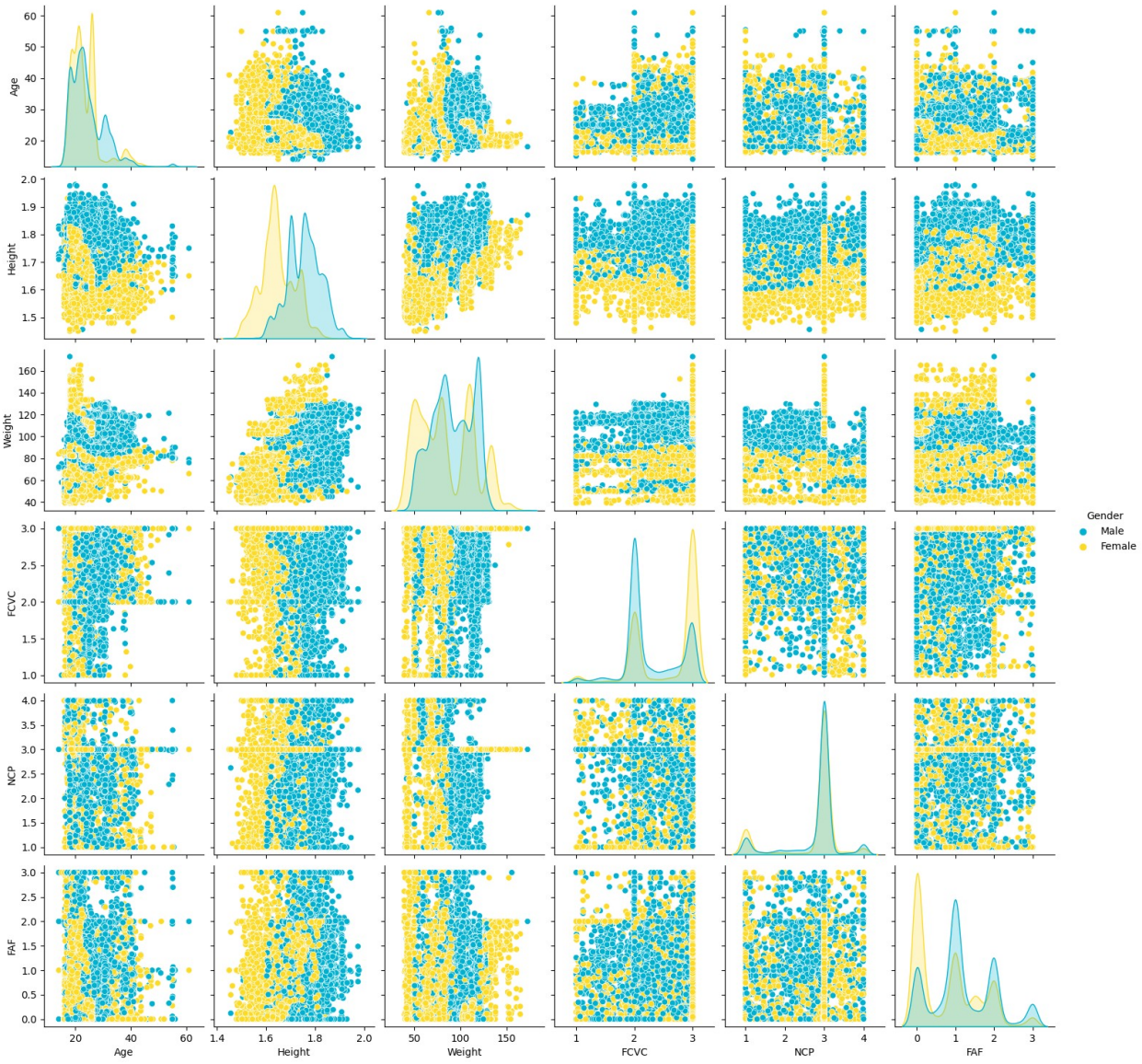



```
# # Scatter Plot | to Show Height vs TUE RealtionShip
advanced_scatter_plot('Height', 'TUE', 'Age_Category', tr_d)
```




```
# Cols to Plot
columns_to_plot = ['Gender', 'Age', 'Height', 'Weight',
       'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'SCC', 'FAF',
       'CALC', 'MTRANS', 'NObeyesdad',
       'Age_Category', ]

# Data Columns
data_to_plot = tr_d[columns_to_plot]

# Create a dictionary to map colors to unique values of the 'Quality'
column
Q_colors = {'Male': palette[0], 'Female': palette[1], 'other': 'gray'}
```

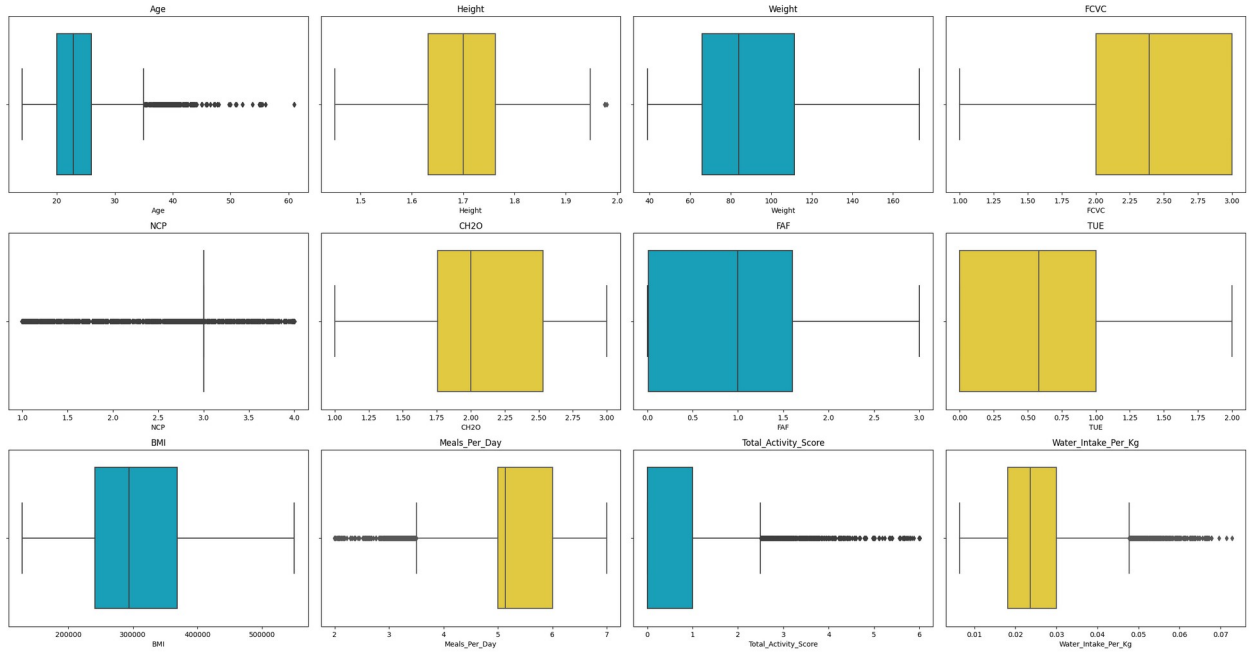
```
# Creating the pairplot with the specified palette for categorical variables
sns.pairplot(data_to_plot, hue='Gender', palette=Q_colors)
plt.show()
```



Step 6 | Outlier Detection

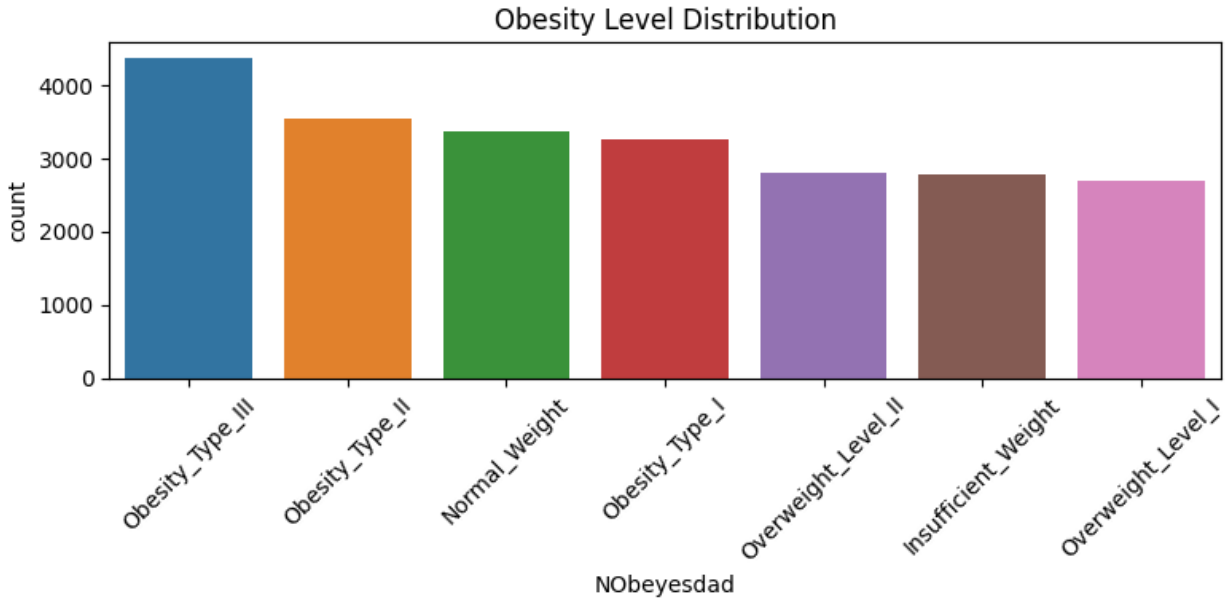
```
# Num _COLS
NUM_COLS_F = [col for col in tr_d.columns if tr_d[col].dtype == 'float']

# Define the number of rows and columns for subplots
num_rows = 4  # 4 rows
```

```
num_cols = 4  # 4 columns

# Create subplots with appropriate titles
fig, axes = plt.subplots(num_rows, num_cols, figsize=(25, 17))

# Paellet
palettes = ["rgb(0, 177, 210)", "rgb(253, 219, 39)"]

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Loop through each numerical column and create a box plot
for i, col in enumerate(NUM_COLS_F[:num_rows * num_cols]):
    sns.boxplot(x=tr_d[col], ax=axes[i], color=palette[i %
len(palette)])
    axes[i].set_title(col)

# Hide empty subplots
for i in range(len(NUM_COLS_F), num_rows * num_cols):
    fig.delaxes(axes[i])

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




```
plt.figure(figsize=(8, 4))
sns.countplot(x='NObeyesdad', data=tr_d,
order=tr_d['NObeyesdad'].value_counts().index)
plt.title('Obesity Level Distribution')
plt.xticks(rotation=45)
```

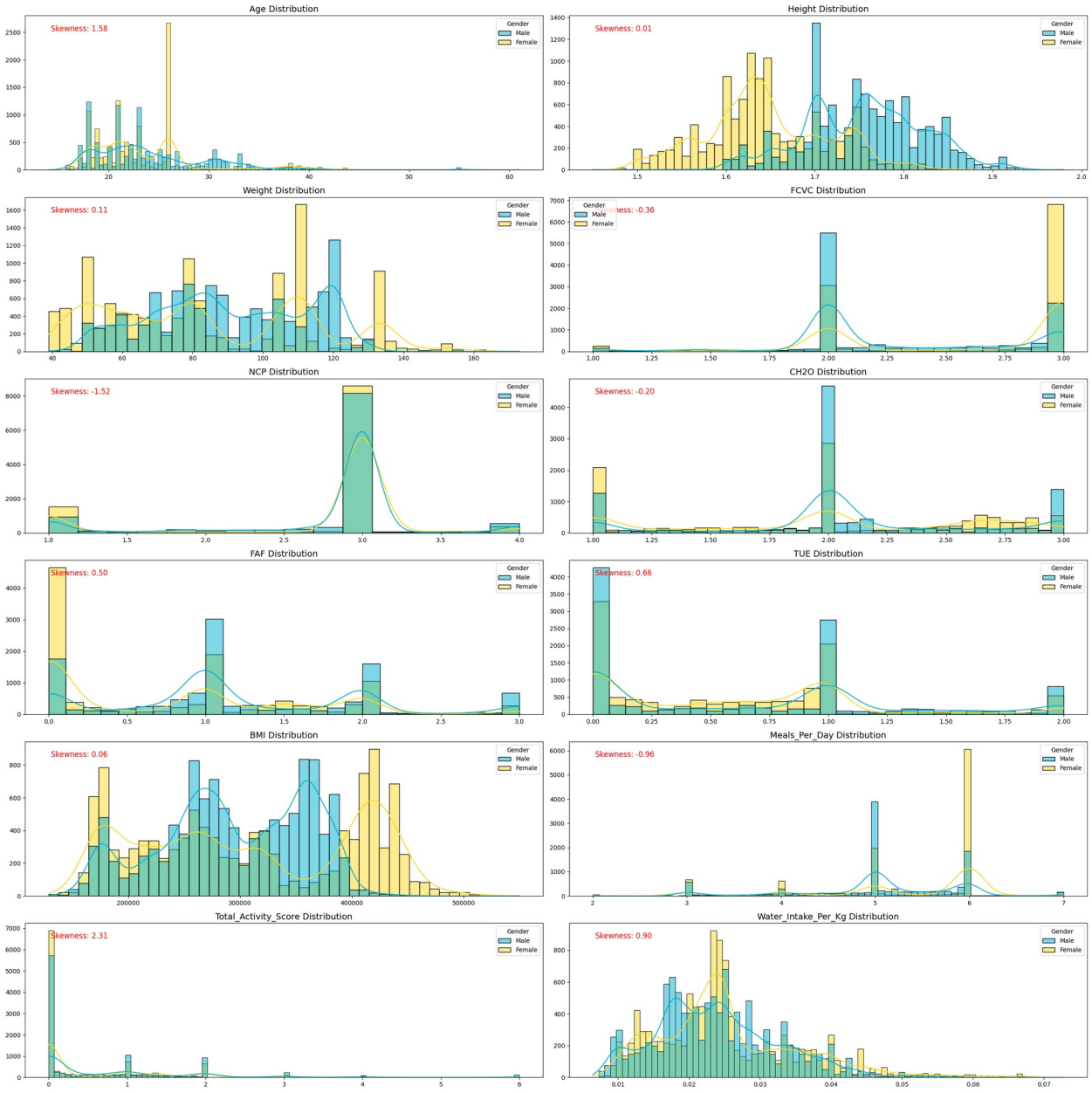
```
plt.tight_layout()
plt.show()
```



Step 7 | Numerical Feature Distribution

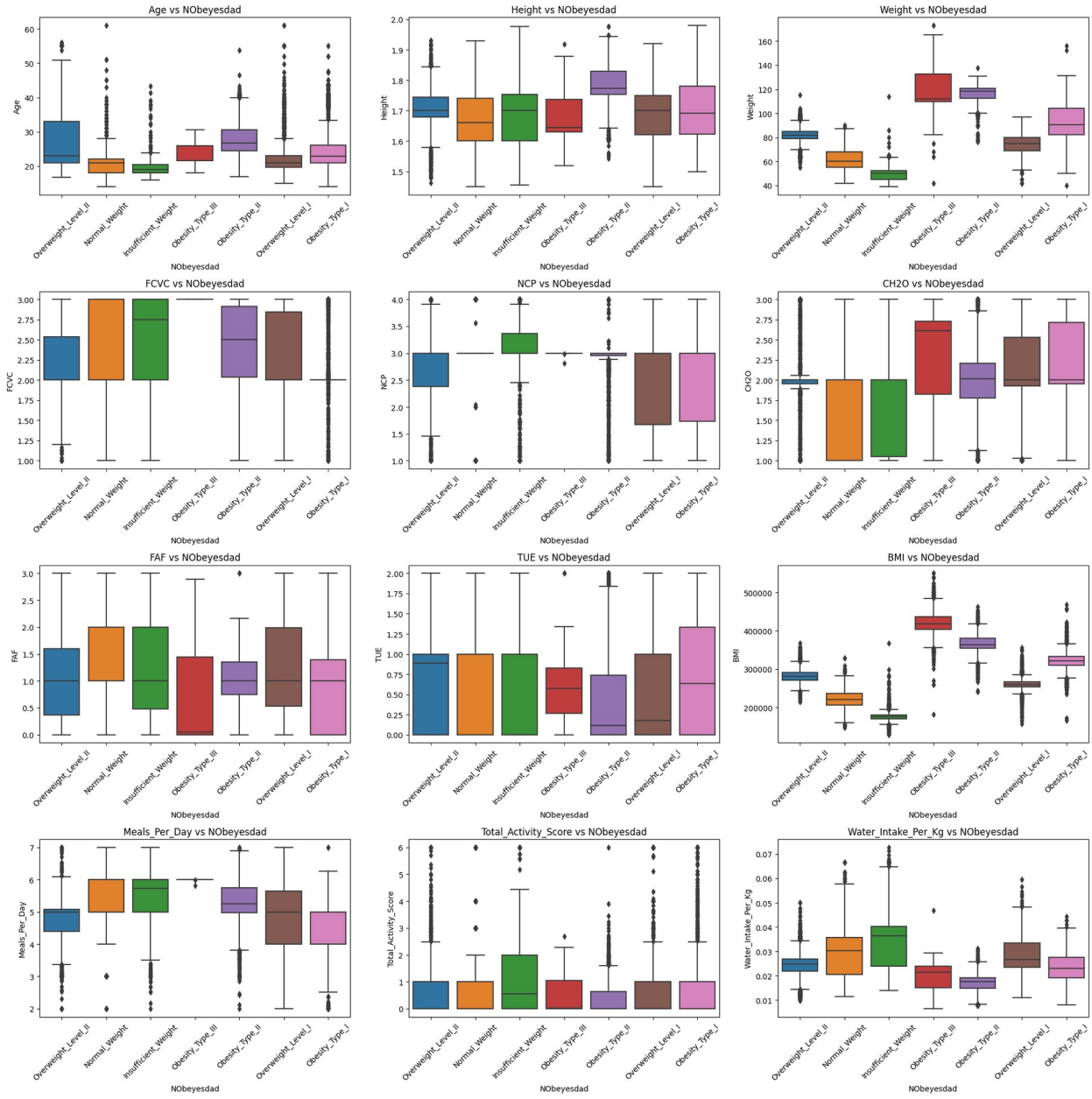
```
# Function to Plot Numerical Distribution
def plot_numerical_distribution_with_hue(data, num_cols,
hue_col='Gender', figsize=(25, 25), dpi=100):
    # Create subplots
    rows = (len(num_cols) + 1) // 2
    fig, ax = plt.subplots(rows, 2, figsize=figsize, dpi=dpi)
    ax = ax.flatten()
        # Loop through each column and plot the distribution with hue
    for i, column in enumerate(num_cols):
        sns.histplot(data=data, x=column, hue=hue_col, ax=ax[i],
kde=True, palette=palette)
        ax[i].set_title(f'{column} Distribution', size=14)
        ax[i].set_xlabel(None)
        ax[i].set_ylabel(None)

        # Calculate skewness
        skewness = skew(data[column].dropna())
        skew_label = f'Skewness: {skewness:.2f}'

        # Add skewness annotation
        ax[i].annotate(skew_label, xy=(0.05, 0.9), xycoords='axes
fraction', fontsize=12, color='red')

    # Remove any extra subplots
    for j in range(len(num_cols), len(ax)):
```

```
        fig.delaxes(ax[j])

    # Set Tight Layout
    plt.tight_layout()

    # Show the plot
    plt.show()

# Cols to Plot
NUM_COLS_F = [col for col in tr_d.columns if tr_d[col].dtype ==
'float']
# Numerical Distribution of Age Vs Fare
plot_numerical_distribution_with_hue(tr_d,NUM_COLS_F,'Gender')
```



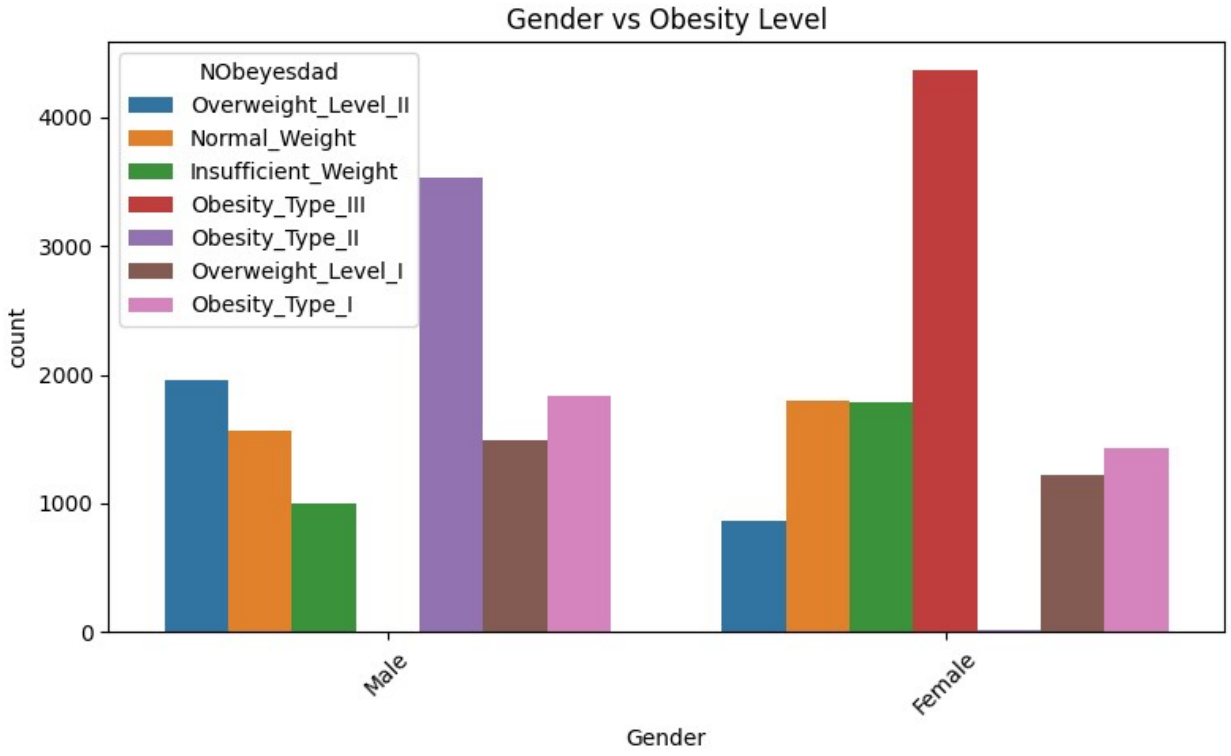

```
num_cols = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF',
'TUE',
            'BMI', 'Meals_Per_Day', 'Total_Activity_Score',
'Water_Intake_Per_Kg']

plt.figure(figsize=(20, 20))
for i, col in enumerate(num_cols, 1):
    plt.subplot(4, 3, i)
    sns.boxplot(x='NObeyesdad', y=col, data=tr_d)
    plt.title(f'{col} vs NObeyesdad')
    plt.xticks(rotation=45)
```

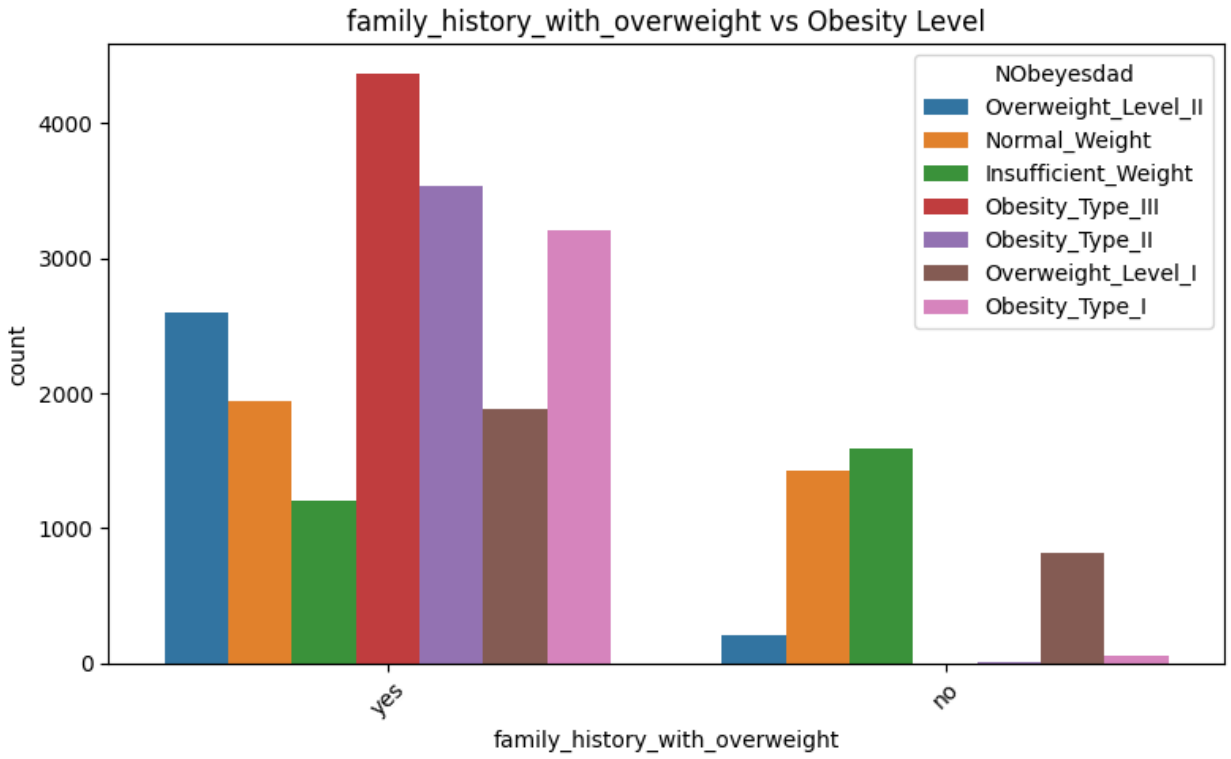
```
plt.tight_layout()
plt.show()
```

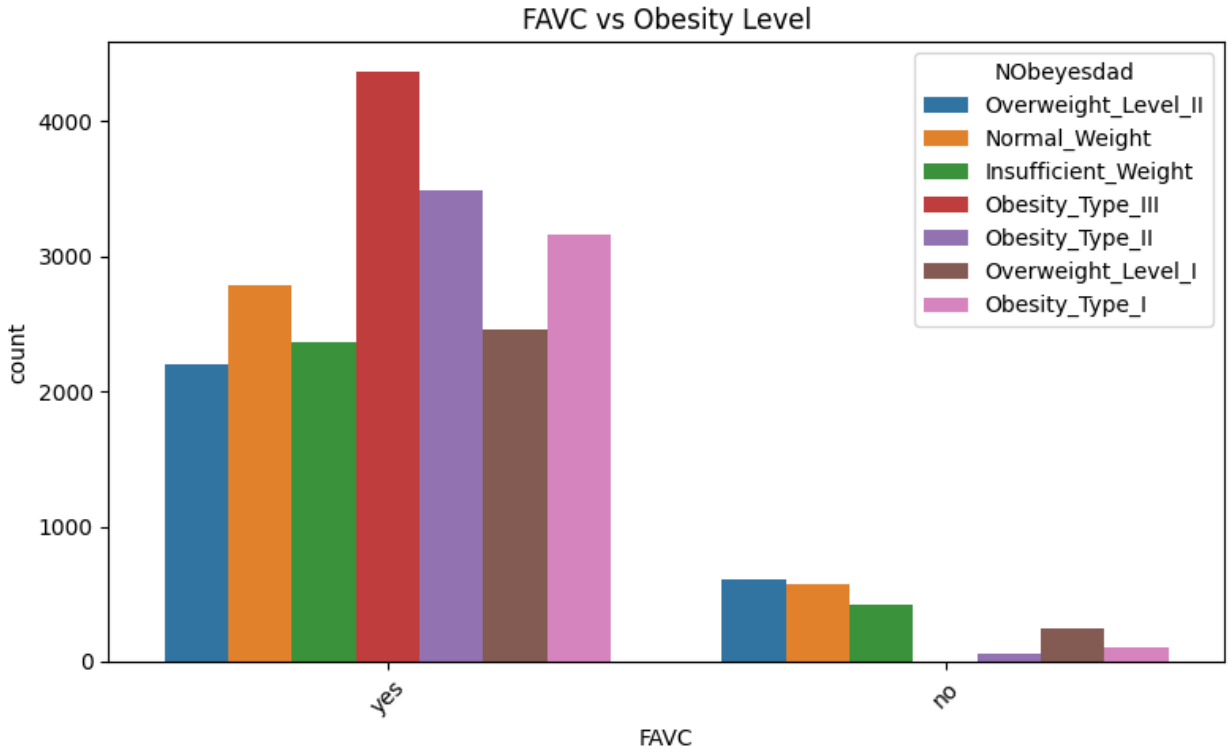


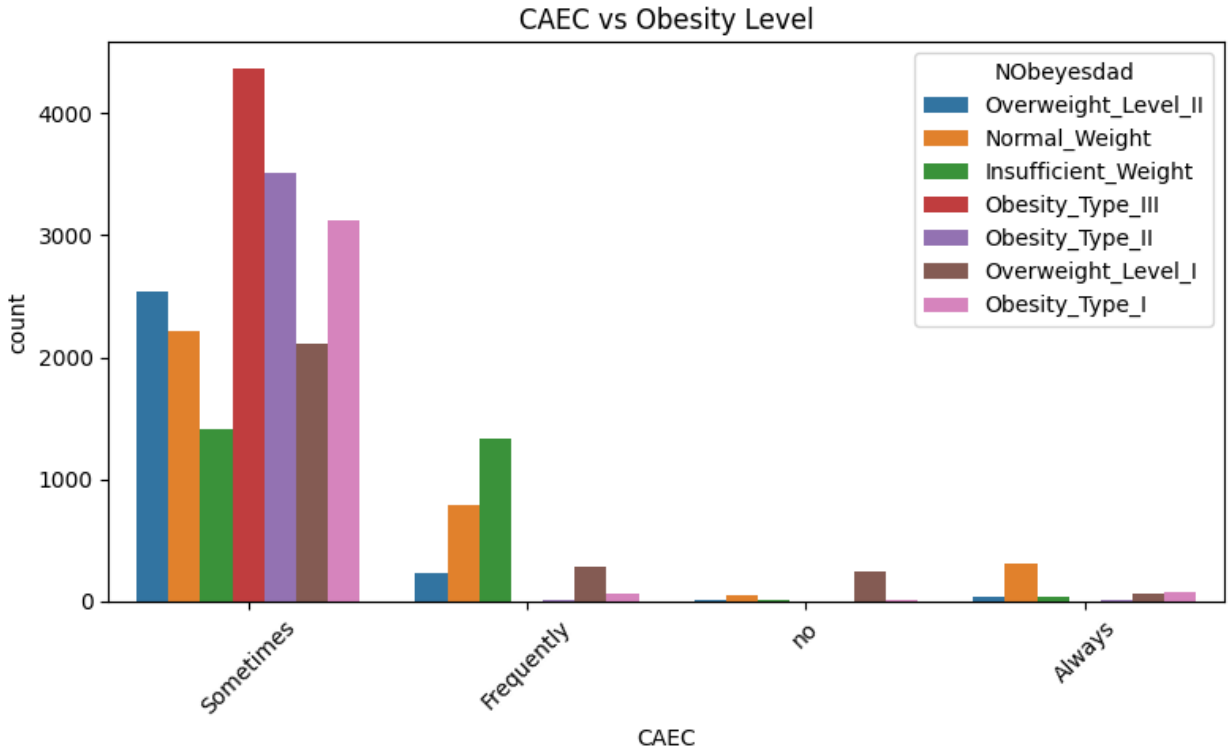

```
cat_cols = [
    'Gender',
    'family_history_with_overweight',
    'FAVC',
    'CAEC',
    'SMOKE',
    'SCC',
    'CALC',
```

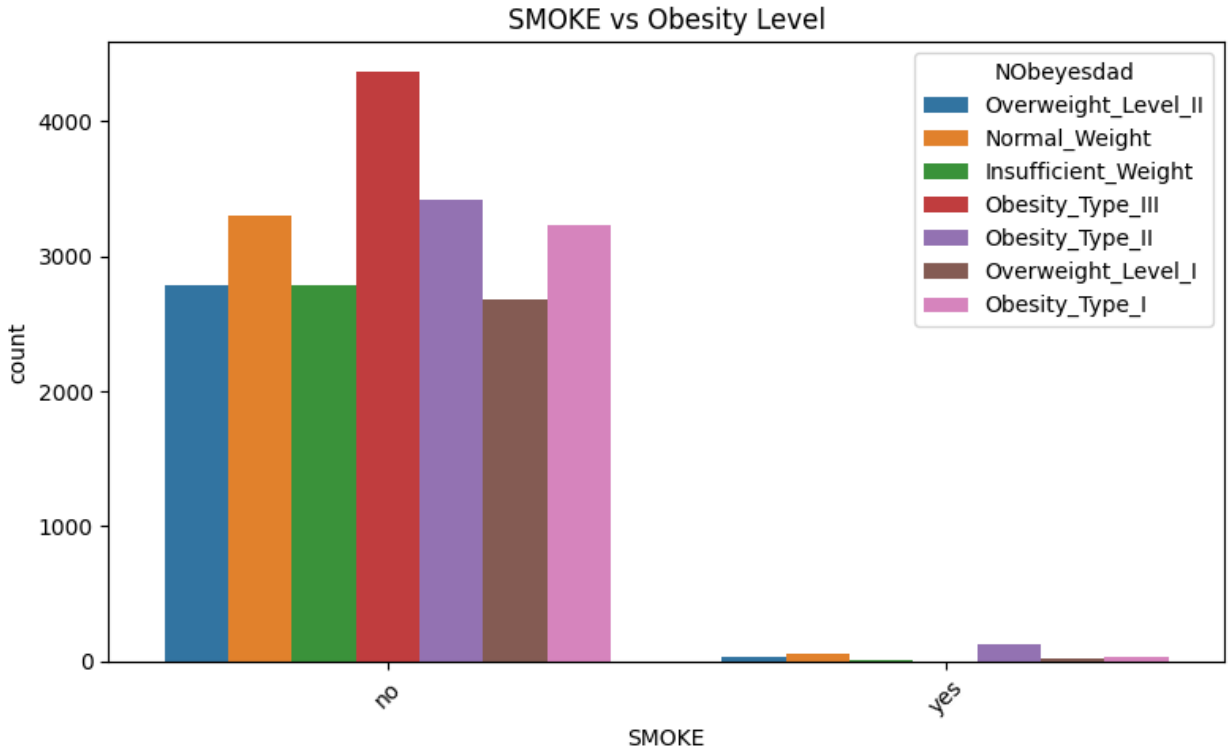
```
    'MTRANS',
    'Age_Category'
]

for col in cat_cols:
    plt.figure(figsize=(8,5))
    sns.countplot(x=col, hue='NObeyesdad', data=tr_d)
    plt.title(f'{col} vs Obesity Level')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

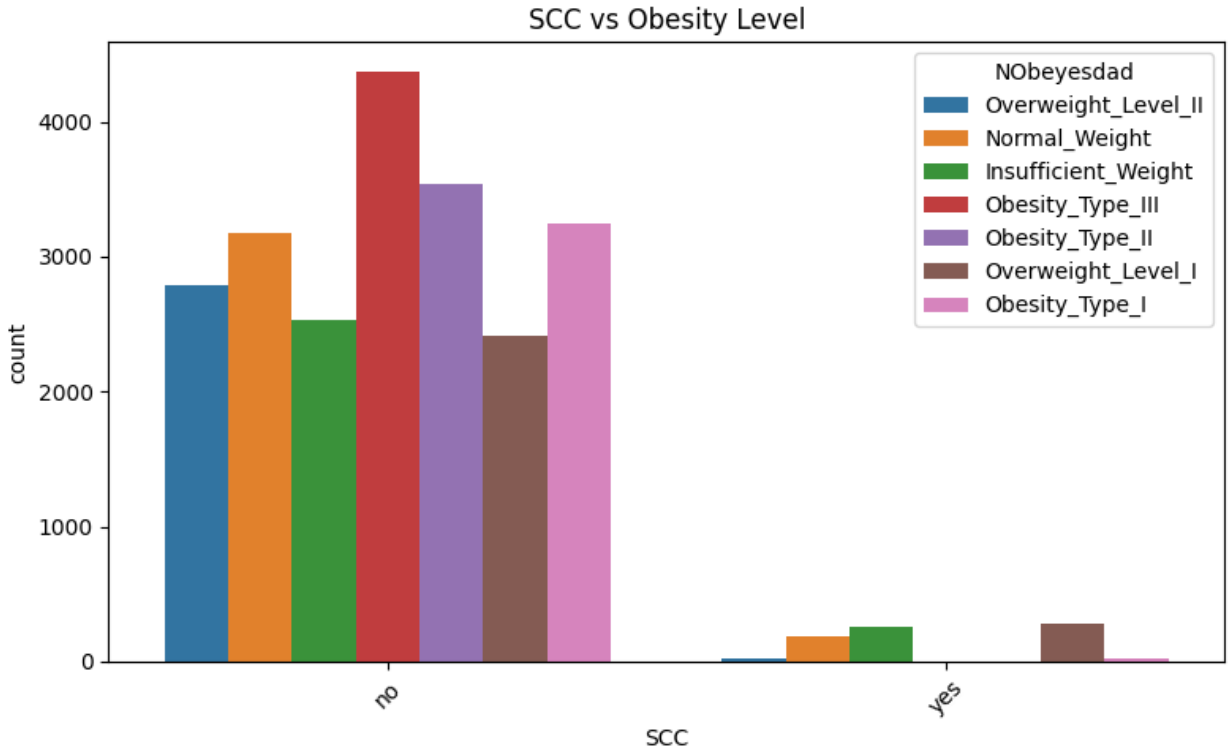


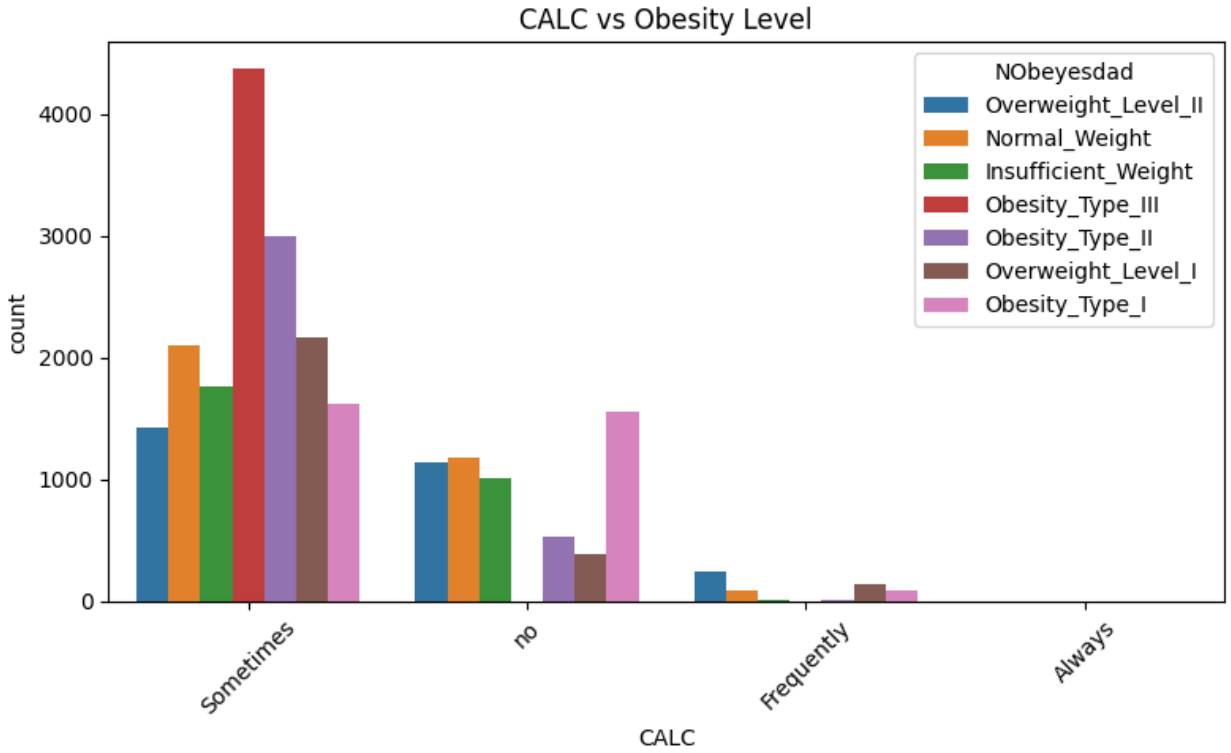

family_history_with_overweight vs Obesity Level
NObeyesdad
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
count
0
1000
2000
3000
4000
yes
no
family_history_with_overweight

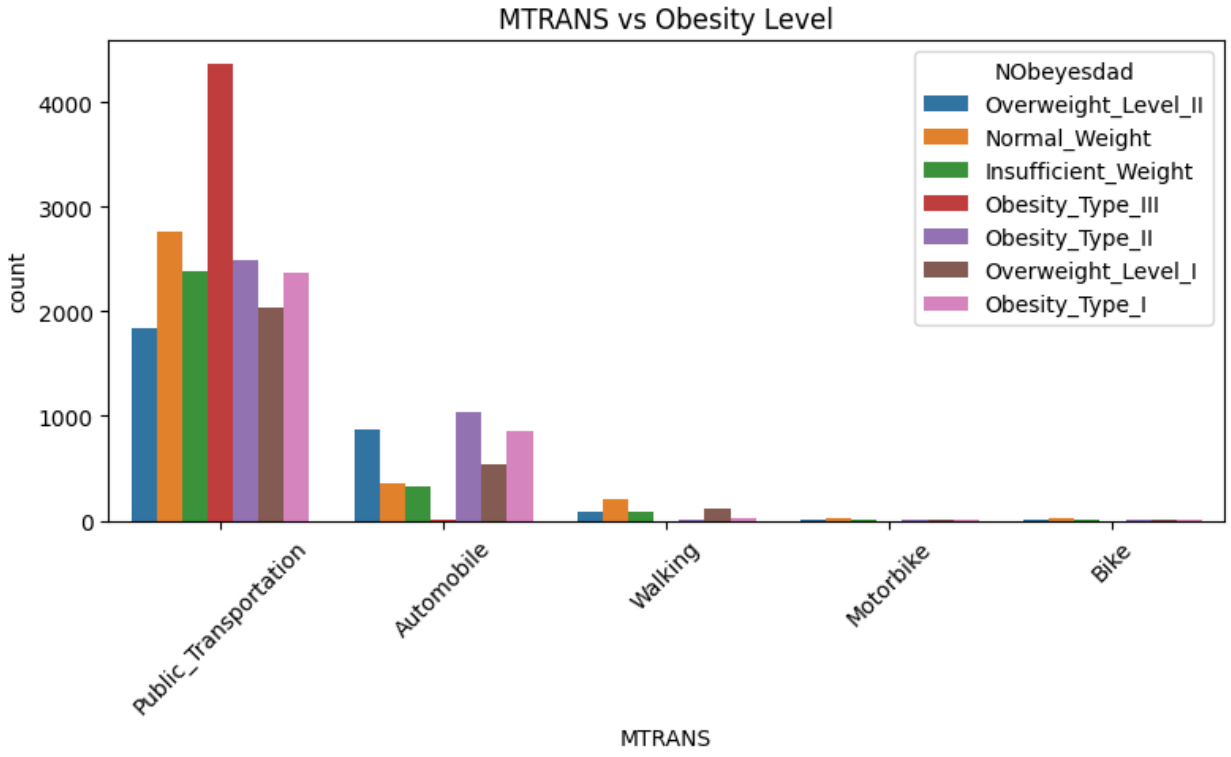

FAVC vs Obesity Level
NObeyesdad
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
count
0
1000
2000
3000
4000
yes
no
FAVC

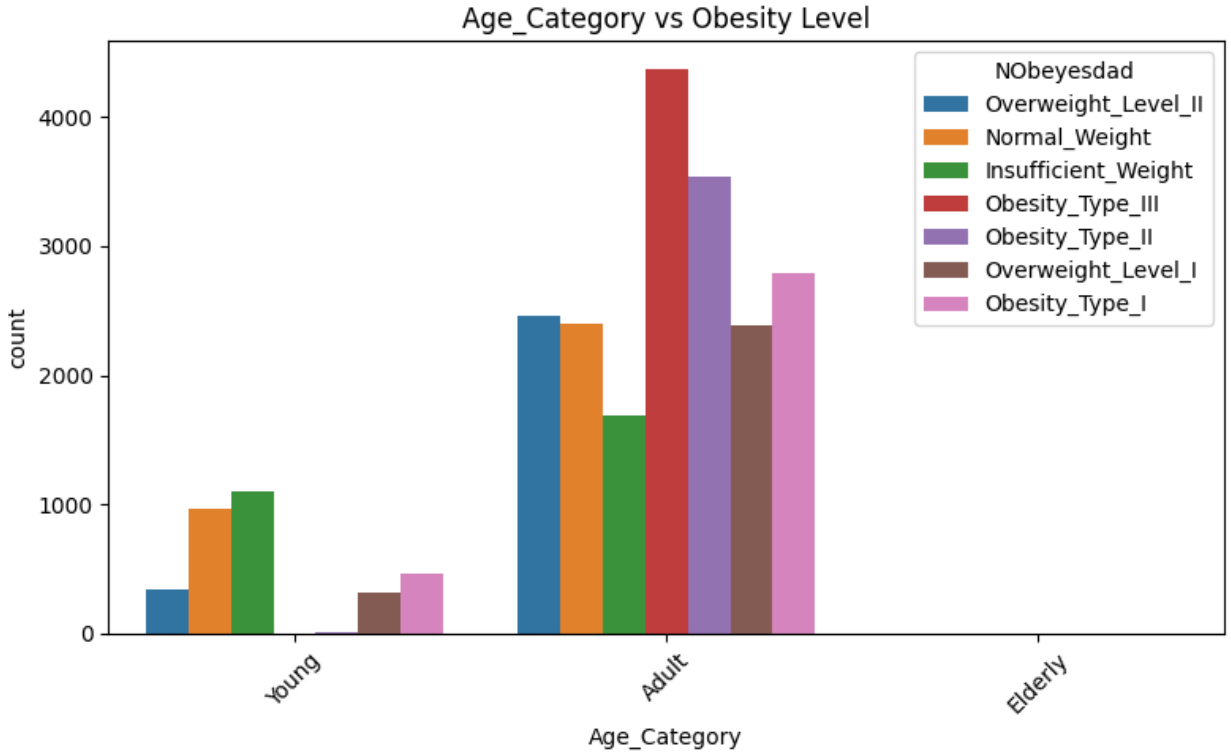

CAEC vs Obesity Level
NObeyesdad
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
count
0
1000
2000
3000
4000
Sometimes
Frequently
no
Always
CAEC


SMOKE vs Obesity Level
NObeyesdad
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
count
0
1000
2000
3000
4000
no
yes
SMOKE


SCC vs Obesity Level
NObeyesdad
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
count
0
1000
2000
3000
4000
no
yes
SCC


CALC vs Obesity Level
NObeyesdad
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
count
0
1000
2000
3000
4000
Sometimes
no
Frequently
Always
CALC


MTRANS vs Obesity Level
NObeyesdad
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
count
0
1000
2000
3000
4000
Public_Transportation
Automobile
Walking
Motorbike
Bike
MTRANS


Age_Category vs Obesity Level
NObeyesdad
Overweight_Level_II
Normal_Weight
Insufficient_Weight
Obesity_Type_III
Obesity_Type_II
Overweight_Level_I
Obesity_Type_I
count
0
1000
2000
3000
4000
Young
Adult
Elderly
Age_Category

Step 7.1 | Normalization

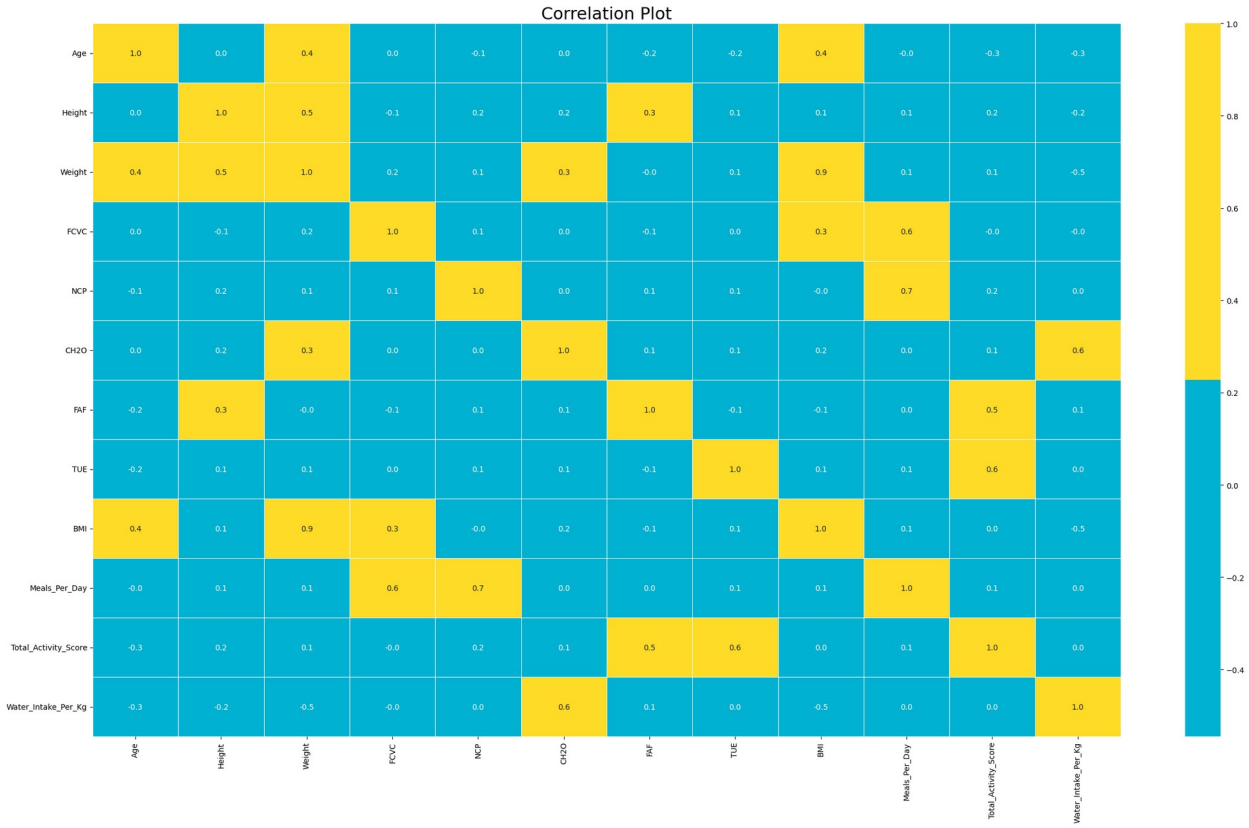
```
# Function to Scale Data
def apply_scaling(data, columns, scaler_type):
    # Check the type of scaler and initialize the appropriate scaler
object
    if scaler_type == 'S':
        scaler = StandardScaler()  # Initialize StandardScaler
    elif scaler_type == 'M':
        scaler = MinMaxScaler()  # Initialize MinMaxScaler
    elif scaler_type == 'Q':
        scaler = QuantileTransformer(output_distribution='normal')  #
Initialize QuantileTransformer
    else:
        raise ValueError("Invalid scaler type. Choose 'S' for
StandardScaler, 'M' for MinMaxScaler, or 'Q' for
QuantileTransformer.")

    # Create a copy of the input data to avoid modifying the original
data
    scaled_data = data.copy()

    # Loop through each column to be scaled
    for col in columns:
        # Apply the scaler to the current column and update the data
with the scaled values
        scaled_data[col] = scaler.fit_transform(scaled_data[[col]])

    # Return the scaled data
    return scaled_data

# Specify columns and scaler type
columns_to_scale = [col for col in tr_d.columns if tr_d[col].dtype ==
'float']
scaler_type = 'Q'

# Apply scaling to training data
tr_d = apply_scaling(tr_d, columns_to_scale, scaler_type)
# Apply the same scaling to testing data
te_d = apply_scaling(te_d, columns_to_scale, scaler_type)
PrintColor('Data Scaled Done')
```

```
Data Scaled Done
```

Step 8 | Correlation Analysis

```
# Select only numeric columns
N_d = tr_d.select_dtypes(include='number')

# Compute the correlation matrix
```

```
correlation_matrix = N_d.corr()

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(25, 15))
sns.heatmap(correlation_matrix, annot=True, cmap=palette, fmt=".1f",
linewidths=0.5)
plt.title('Correlation Plot', fontsize=22)
plt.tight_layout()
plt.show()
```



Step 9 | Encoding

```
# Defining the categorical columns to encode
CAT_COL_E = [
 'Gender',
 'family_history_with_overweight',
 'FAVC',
 'CAEC',
 'SMOKE',
 'SCC',
 'CALC',
 'MTRANS',
]
# Function to Encode Data
```

```
def E_D(data, columns, method='L'):
    encoded_data = data.copy()  # Make a copy of the input data

    if method == 'L':
        # Initialize LabelEncoder
        L_E = LabelEncoder()

        # Encode categorical columns using LabelEncoder
        for col in columns:
            encoded_data[col] = L_E.fit_transform(encoded_data[col])

    elif method == 'D':
        # Create dummy variables for categorical columns
        dummy_cols = pd.get_dummies(encoded_data[columns],
prefix=columns)

        # Concatenate dummy variables with original data
        encoded_data = pd.concat([encoded_data, dummy_cols], axis=1)

        # Drop the original categorical columns
        encoded_data = encoded_data.drop(columns, axis=1)

    else:
        raise ValueError("Invalid method! Please choose either 'L' or
'D'.")

    return encoded_data

# Encoder Train and test
tr_d = E_D(tr_d, CAT_COL_E , 'D')
te_d = E_D(te_d, CAT_COL_E, "D")
PrintColor('Data is Encoded Successfully')
```

```
Data is Encoded Successfully
```

Step 10 | Model Building

```
# # #
================================================================================
=========================================
# # #                          X < y
# #
#================================================================================
=========================================
X_T = tr_d.drop('NObeyesdad', axis=1)
y_T = tr_d['NObeyesdad']

# # #
================================================================================
=========================================
```

```
# # #                              Train < Test Split
# #
#=============================================================================
==========================================
from sklearn.model_selection import train_test_split

X_TR, X_TE, Y_TR, Y_TE = train_test_split(
    X_T, y_T, test_size=0.1, random_state=42, stratify=y_T
)


# # #
=============================================================================
==========================================
# # #                              Shapes <
# #
#=============================================================================
==========================================
PrintColor(f"Training set shape - X: {X_TR.shape}, y: {Y_TR.shape}")
PrintColor(f"Testing set shape - X: {X_TE.shape}, y: {Y_TE.shape}")
```

```
Training set shape - X: (20560, 36), y: (20560,)
Testing set shape - X: (2285, 36), y: (2285,)
```

LightGBM Classifier

Special Note :

- The Params Used are tunned Using Optuna.

Code Working

```
L_BASE = lgb.LGBMClassifier(**lgb_params)
```

This line creates an instance of the LightGBM classifier (`L_BASE`) with the specified parameters (`**lgb_params`).

```
V_CV = cross_val_score(L_BASE,
                       X_T,
                       y_T,
                       scoring='roc_auc',
                       cv=10,
                       n_jobs=-1)
```

Using cross-validation (`cross_val_score`), we evaluate the performance of the LightGBM classifier (`L_BASE`) on the training data (`X_T`, `y_T`) using Accuracy as the scoring metric. We perform 10-fold cross-validation (`cv=10`) and utilize all available CPU cores (`n_jobs=-1`).

```
print_heading(f"The Average Accuracy Of LGB Classifier is :
{V_CV.mean()}")
```

Finally, we print the average ROC AUC score of the LightGBM classifier calculated from the cross-validation results stored in `V_CV.mean()`. The `print_heading` function is used to format the output with a header.

```
# # #
================================================================================
=========================================
# # #                            Params < LGB Classifier
# #
#================================================================================
=========================================
lgb_params = {
 'n_estimators': 899,
 'learning_rate': 0.013003893032117776,
 'max_depth': 18,
 'reg_alpha': 0.9218377389528793,
 'reg_lambda': 0.020694654173173645,
 'num_leaves': 24,
 'subsample': 0.7402011916024158,
 'colsample_bytree': 0.25484261764678784,
 'verbose' : -1
}
# # #
================================================================================
=========================================
# # #                            Train < LGB Classifier
# #
#================================================================================
=========================================
L_BASE = lgb.LGBMClassifier(**lgb_params)
V_CV = cross_val_score(L_BASE,
                       X_T,
                       y_T,
                       scoring='accuracy',
                       cv=15,
                       n_jobs=-1)
# # #
================================================================================
=========================================
# # #                            ROC AUC < LGB Classifier
#
#================================================================================
=========================================
print_boxed_blue_heading(f"The AUCCURACY Of LGB Classifier is :
{V_CV.mean()}")
# # #
================================================================================
=========================================
# # #                            Submission < LGB Classifier
```

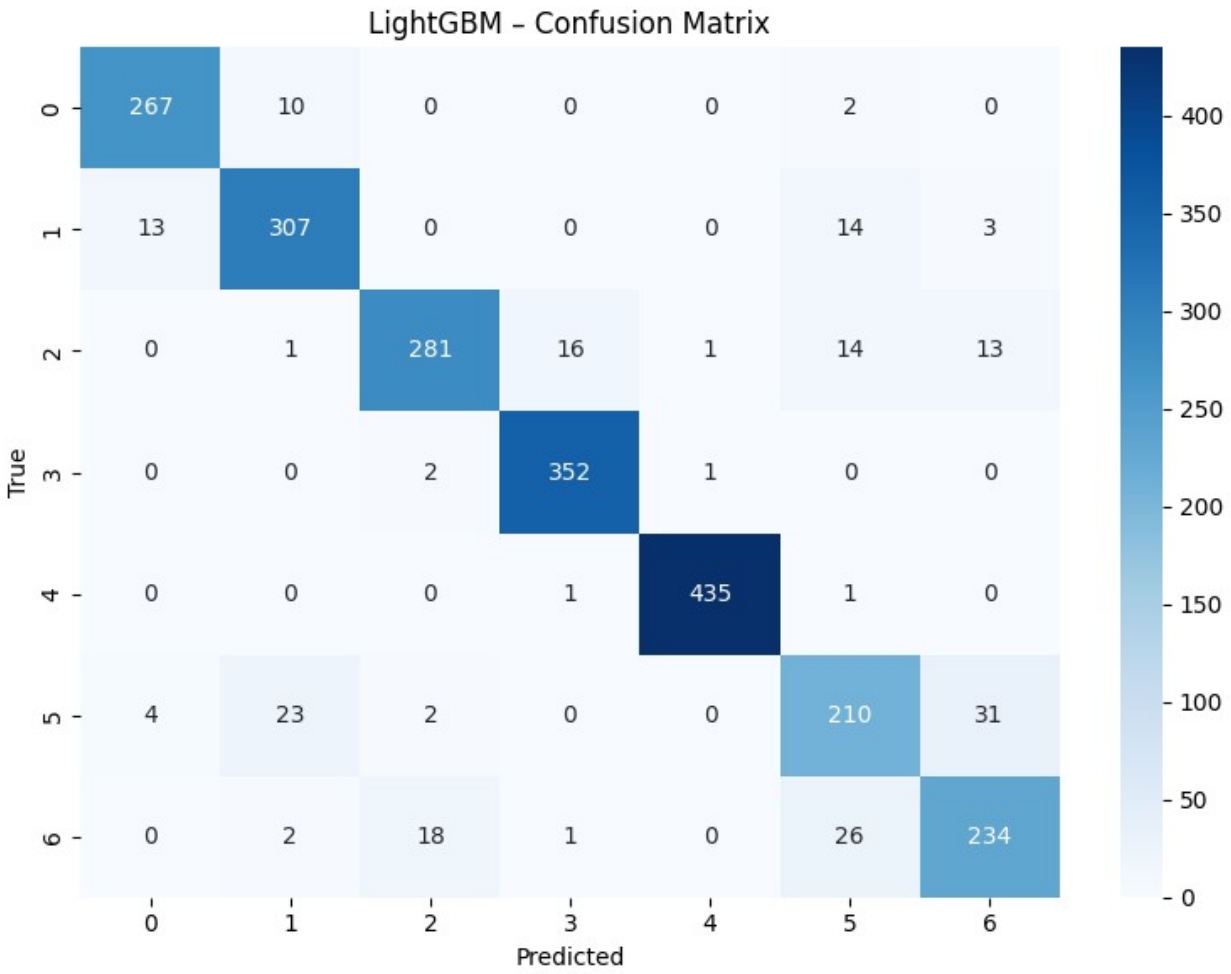
```
#
#=============================================================================
==========================================
# Fit Again
L_BASE.fit(X_T,y_T)
# Pred
T_P = L_BASE.predict(te_d)



# Submission File
SUB = pd.DataFrame({'id': d_s['id'], 'NObeyesdad': T_P})
# Save Submission File
SUB.to_csv('submission_O_R_0.9000.csv', index=False)
PrintColor('Submission File Saved ! Hurray')


============================================================
| The AUCCURACY Of LGB Classifier is : 0.9162179908076165 |
============================================================
Submission File Saved ! Hurray
```

CatBoost Classifier

```
# # #
=============================================================================
==========================================
# # #                          Params < CAT Classifier
# #
#=============================================================================
==========================================
cat_params = {
    'n_estimators': 853,
    'learning_rate': 0.10899577626375372,
    'depth': 7,
    'colsample_bylevel': 0.7340962061535496,
    'random_strength': 6.262882561405091,
    'min_data_in_leaf': 92,
    'verbose': 0,
    'loss_function': 'MultiClass',
    'eval_metric': 'MultiClass',
}

# Specify categorical features
cat_features = ['Age_Category']  # Add the categorical features here

# # #
=============================================================================
==========================================
```

```
# # #                          Train < CAT Classifier
# #
#=============================================================================
=============================================
C_BASE = CatBoostClassifier(**cat_params, cat_features=cat_features)
C_CV = cross_val_score(C_BASE,
                       X_T,
                       y_T,
                       scoring='accuracy',
                       cv=5,
                       n_jobs=-1)
# # #
=============================================================================
=============================================
# # #                          ROC AUC < CAT Classifier
# #
#=============================================================================
=============================================
print_boxed_blue_heading(f"The Accuracy Of CatBoost Classifier is:
{C_CV.mean()}")


============================================================
| The Accuracy Of CatBoost Classifier is: 0.9096082293718538 |
============================================================

from sklearn.metrics import *

def evaluate_model(name, model, X_train, y_train, X_test, y_test):
    # Fit on train
    model.fit(X_train, y_train)

    # Predict
    y_pred = model.predict(X_test)

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    f1_macro = f1_score(y_test, y_pred, average='macro')

    print_boxed_blue_heading(f"{name} — Test Performance")
    print(f"Accuracy     : {acc:.4f}")
    print(f"Macro F1     : {f1_macro:.4f}")
    print("\nClassification Report:\n")
    print(classification_report(y_test, y_pred))

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"{name} — Confusion Matrix")
```

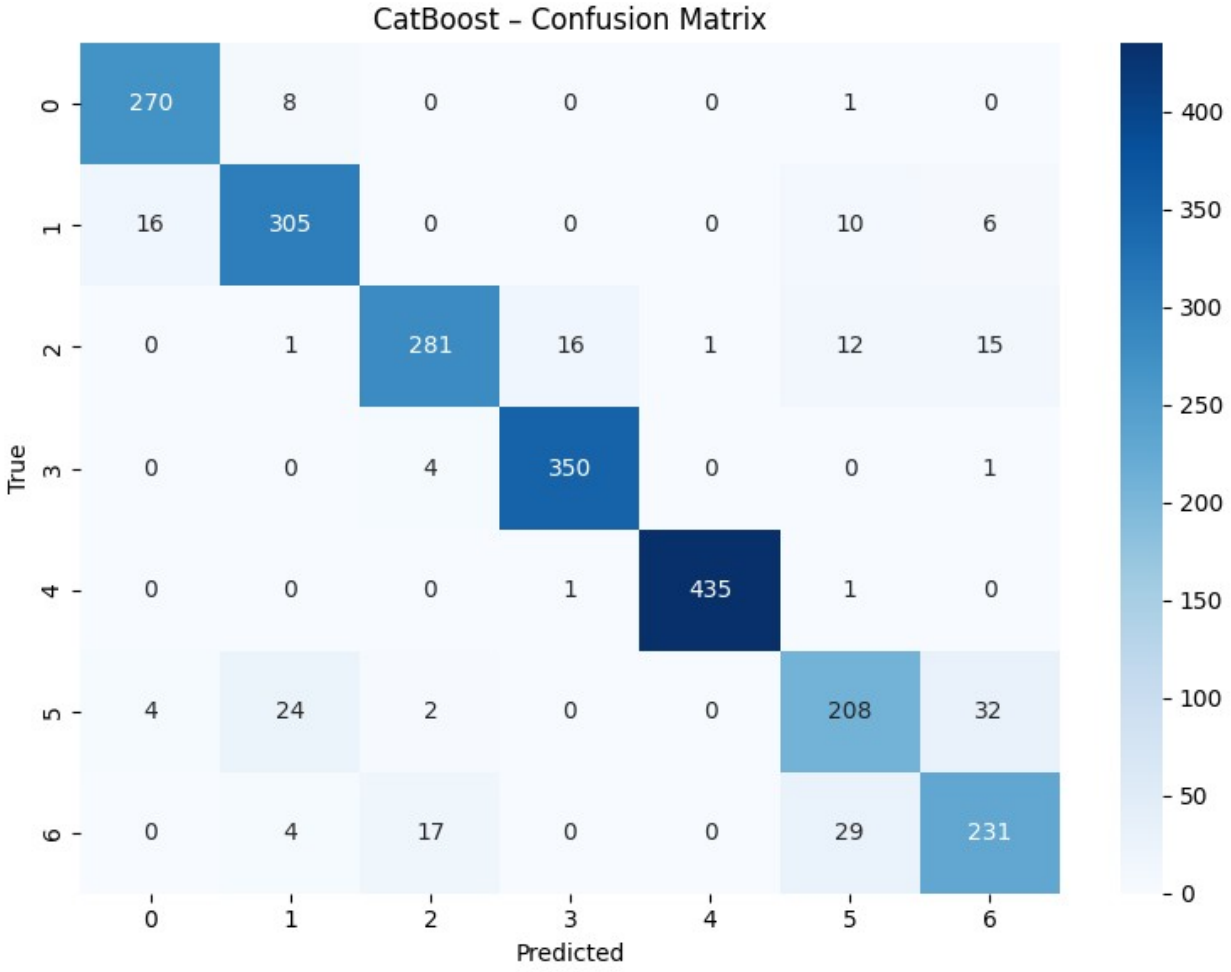
```
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.tight_layout()
    plt.show()

    return acc, f1_macro

# LightGBM
L_BASE = lgb.LGBMClassifier(**lgb_params)
lgb_acc, lgb_f1 = evaluate_model("LightGBM", L_BASE, X_TR, Y_TR, X_TE,
Y_TE)

# CatBoost  <-- no cat_features when data is already one-hot encoded
# Remove extra dummy columns if present
tr_d = tr_d.drop(columns=[col for col in tr_d.columns if
col.startswith("Age_Category_")], errors='ignore')

# CatBoost
C_BASE = CatBoostClassifier(**cat_params,
cat_features=['Age_Category'], verbose=0)
cat_acc, cat_f1 = evaluate_model("CatBoost", C_BASE, X_TR, Y_TR, X_TE,
Y_TE)


==============================
| LightGBM — Test Performance |
==============================
Accuracy     : 0.9129
Macro F1     : 0.9037

Classification Report:

                     precision    recall  f1-score   support

Insufficient_Weight       0.94      0.96      0.95       279
      Normal_Weight       0.90      0.91      0.90       337
     Obesity_Type_I       0.93      0.86      0.89       326
    Obesity_Type_II       0.95      0.99      0.97       355
   Obesity_Type_III       1.00      1.00      1.00       437
 Overweight_Level_I       0.79      0.78      0.78       270
Overweight_Level_II       0.83      0.83      0.83       281

           accuracy                           0.91      2285
          macro avg       0.90      0.90      0.90      2285
       weighted avg       0.91      0.91      0.91      2285
```




```
================================
| CatBoost – Test Performance |
================================
Accuracy     : 0.9103
Macro F1     : 0.9008

Classification Report:
```

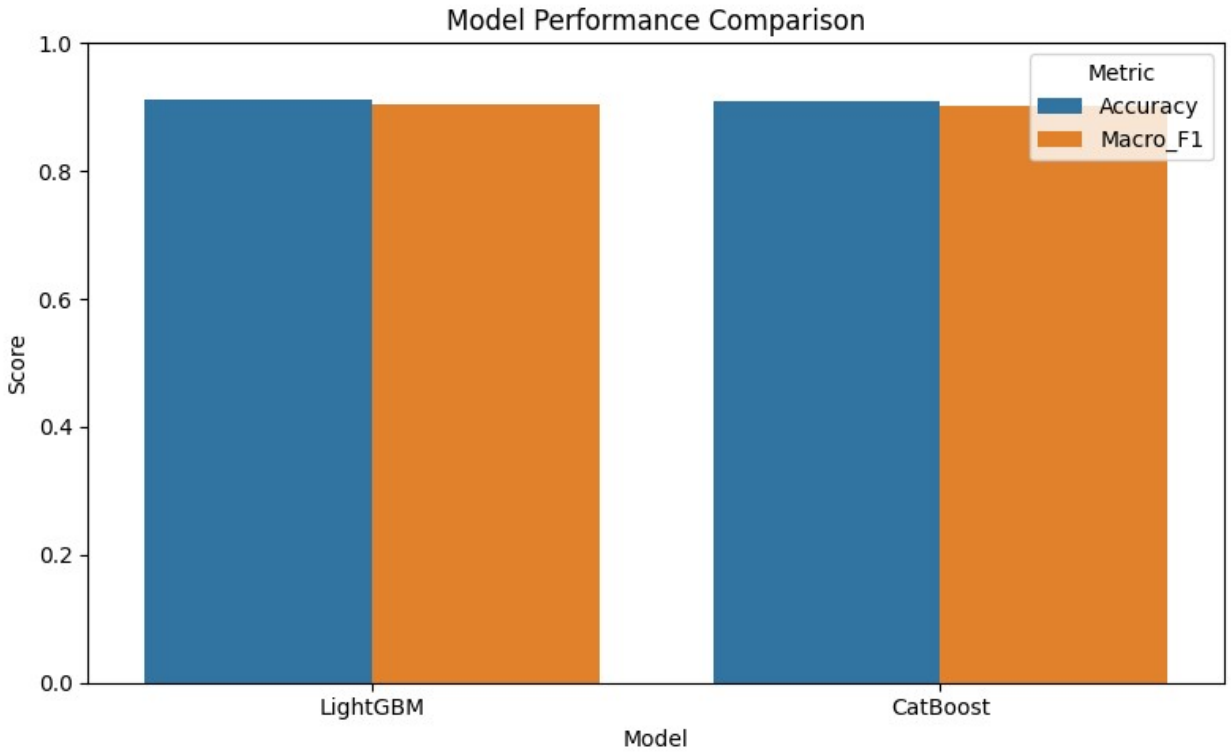
	precision	recall	f1-score	support
Insufficient_Weight	0.93	0.97	0.95	279
Normal_Weight	0.89	0.91	0.90	337
Obesity_Type_I	0.92	0.86	0.89	326
Obesity_Type_II	0.95	0.99	0.97	355
Obesity_Type_III	1.00	1.00	1.00	437
Overweight_Level_I	0.80	0.77	0.78	270
Overweight_Level_II	0.81	0.82	0.82	281
accuracy			0.91	2285

```
           macro avg       0.90      0.90      0.90      2285
        weighted avg       0.91      0.91      0.91      2285
```




```
results = pd.DataFrame({
    'Model': ['LightGBM', 'CatBoost'],
    'Accuracy': [lgb_acc, cat_acc],
    'Macro_F1': [lgb_f1, cat_f1]
})

print_boxed_blue_heading("Model Comparison")
display(results)

plt.figure(figsize=(8,5))
results_melted = results.melt(id_vars='Model', value_vars=['Accuracy',
'Macro_F1'],
                              var_name='Metric', value_name='Score')
sns.barplot(x='Model', y='Score', hue='Metric', data=results_melted)
plt.ylim(0, 1)
```

```
plt.title("Model Performance Comparison")
plt.tight_layout()
plt.show()


====================
| Model Comparison |
====================

      Model  Accuracy  Macro_F1
0  LightGBM  0.912910  0.903748
1  CatBoost  0.910284  0.900750
```




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

models = {
    "LightGBM": lgb.LGBMClassifier(**lgb_params),
    "CatBoost": CatBoostClassifier(**cat_params,
cat_features=['Age_Category'], verbose=0)
}

results = []

for name, model in models.items():
    model.fit(X_TR, Y_TR)
```

```
    preds = model.predict(X_TE)

    acc = accuracy_score(Y_TE, preds)
    f1  = f1_score(Y_TE, preds, average='macro')

    results.append([name, model, acc, f1])
    print(f"{name} -> Accuracy: {acc:.4f} | Macro-F1: {f1:.4f}")

# pick best by macro F1 then accuracy
best = sorted(results, key=lambda x: (x[3], x[2]), reverse=True)[0]
best_name, best_model, best_acc, best_f1 = best

print("\nBest model:", best_name)
print("Accuracy :", best_acc)
print("Macro F1 :", best_f1)

filename = f"best_model_{best_name.replace(' ', '_')}.pkl"
joblib.dump(best_model, filename)
print("Saved as:", filename)
```

```
LightGBM -> Accuracy: 0.9129 | Macro-F1: 0.9037
CatBoost -> Accuracy: 0.9103 | Macro-F1: 0.9008

Best model: LightGBM
Accuracy : 0.912910284463895
Macro F1 : 0.9037477238083376
Saved as: best_model_LightGBM.pkl
```

```
# Sort by highest Macro-F1; if equal, accuracy becomes tie-breaker
best = sorted(results, key=lambda x: (x[3], x[2]), reverse=True)[0]
best_name, best_model, best_acc, best_f1 = best

print("\n================================ ")
print(f" Best Model Detected: {best_name}")
print(f" Accuracy : {best_acc:.4f}")
print(f" Macro-F1 : {best_f1:.4f}")
print("================================\n")
```

```
================================
 Best Model Detected: LightGBM
 Accuracy : 0.9129
 Macro-F1 : 0.9037
================================
```

```
filename = f"best_model_{best_name.replace(' ', '_')}.pkl"
joblib.dump(best_model, filename)
joblib.dump(columns_to_scale, "scaled_columns_list.pkl")
```

```
PrintColor("Best model saved as best_model_obesity.pkl", Fore.GREEN)
print(f"Saved automatically as: {filename}")
```

```
Best model saved as best_model_obesity.pkl
Saved automatically as: best_model_LightGBM.pkl
```

The Final Conclusion