# CHAPTER IV
# RESULTS AND DISCUSSION

This chapter presents the results of data processing conducted in this study. Each finding is discussed in detail to provide a clear overview of the research problem being examined.

## 4.1. Program Implementation

This section describes the program implementation used in this study. The discussion includes the libraries used, data preprocessing stages, the application of class imbalance handling using Random Oversampling (ROS), and the classification process using XGBoost and LightGBM algorithms. This section aims to explain the program workflow applied in data processing and model testing.

### 4.1.1. Library

The program implementation in this study was carried out using the Python programming language in the Google Colab environment. Several libraries were used to support data processing, visualization, class imbalance handling, classification model development, and model evaluation. The pandas and numpy libraries were used for data processing, matplotlib and seaborn were used for visualization, scikit learn was used for data splitting and model evaluation, imblearn was used to apply RandomOverSampler, while xgboost and lightgbm were used to build the classification models. In addition, os, time, warnings, pathlib, and IPython.display were used to support file management, program execution settings, training time measurement, and result display in Google Colab. The list of libraries used is shown in Code 4.1.

**Code 4.1. Libraries Used**

```
import warnings
import html
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from IPython.display import display, HTML
```

**Code 4.1. Libraries Used**

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score,
balanced_accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import RandomOverSampler
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import lightgbm as lgb

warnings.filterwarnings("ignore")
sns.set_theme(style="whitegrid")
```

### 4.1.2. Data Acquisition and Dataset Adjustment

At this stage, data acquisition and dataset adjustment were carried out to ensure that the dataset met the research requirements. The dataset used in this study was the secondary Dhaka Obesity dataset stored in the file Dhaka Obesity.csv. This dataset contains attributes representing respondents' physical conditions, eating habits, lifestyle habits, and modes of transportation, with the initial target variable being Obesity level, which consists of six classes: Insufficient_Weight, Normal_Weight, Overweight, Obesity_Type_I, Obesity_Type_II, and Obesity_Type_III. In the program implementation, the dataset was first read and examined to identify its initial size, attribute names, data types, and target class distribution before adjustment. Based on the data reading results, the initial dataset consisted of 2,182 rows, 17 columns, and 6 initial target classes. The initial class distribution showed that Normal_Weight had the largest number of data, followed by Overweight, Insufficient_Weight, Obesity_Type_I, Obesity_Type_II, and Obesity_Type_III. This condition provided an initial overview that the dataset was not yet in the target format required by the classification standard used in this study.

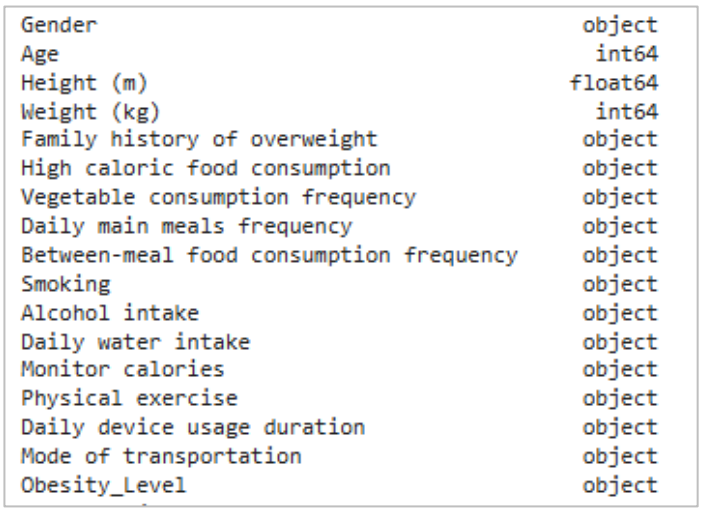
**Code 4.2. Reading the Dataset and Displaying Initial Information**

```
# Membaca dataset
input_filename = "Dhaka Obesity.csv"
df_raw = pd.read_csv(input_filename)

# Validasi kolom target
target_col = "Obesity level"
if target_col not in df_raw.columns:
    raise ValueError(f"Kolom target {target_col} tidak ditemukan.")

# Menampilkan informasi awal dataset
info_awal = pd.DataFrame({
    "Informasi": [
        "Nama file input",
```

**Code 4.2. Reading the Dataset and Displaying Initial Information**

```
        "Jumlah baris awal",
        "Jumlah kolom awal",
        "Jumlah kelas target awal",
        "Nama variabel target"
    ],
    "Nilai": [
        input_filename,
        df_raw.shape[0],
        df_raw.shape[1],
        df_raw[target_col].nunique(),
        target_col
})
display(info_awal)

# Menampilkan cuplikan data dan distribusi kelas awal
display(df_raw.head())
display(df_raw[target_col].value_counts().reset_index())
```

Code 4.2 read the initial dataset from the Dhaka Obesity.csv file, validated important attributes, and displayed the initial dataset information. The displayed information included the input file name, initial number of rows and columns, number of initial target classes, column names, data types of each column, initial target class distribution, and a preview of the first five rows of the dataset. This step was important to provide an overview of the data structure before adjustment. The execution results of Code 4.2 are shown in Figure 4.1 and Figure 4.2. Figure 4.1 presents the initial dataset information, including the dataset size, attribute names, data types, and initial target class distribution. Based on this information, the dataset consists of predictor attributes with numerical and categorical data types, while the target variable Obesity level is categorical.



```
Gender                                   object
Age                                       int64
Height (m)                              float64
Weight (kg)                               int64
Family history of overweight             object
High caloric food consumption            object
Vegetable consumption frequency          object
Daily main meals frequency               object
Between-meal food consumption frequency  object
Smoking                                  object
Alcohol intake                           object
Daily water intake                       object
Monitor calories                         object
Physical exercise                        object
Daily device usage duration              object
Mode of transportation                   object
Obesity_Level                            object
```

**Figure 4.1. Initial Dataset Information**

Based on the initial identification results, it can be seen that the dataset was still in its original form and had not yet undergone dataset adjustment or data cleaning. Therefore, displaying a preview of the initial data was necessary to directly show the dataset contents before the preprocessing stage. This preview shows that each row contains a combination of physical attributes, eating behavior, lifestyle factors, and the initial obesity target.

| Gender | Age | Height (m) | Weight (kg) | Family history of overweight | High caloric food consumption | Vegetable consumption frequency | Daily main meals frequency | Between-meal food consumption frequency | Smoking | Alcohol intake | Daily water intake | Monitor calories | Physical exercise | Daily device usage duration | Mode of transportation | Obesity level |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Male | 29 | 1.65 | 101.0 | Yes | Yes | Sometimes | Three | Frequently | No | I do not drink | Between 1 and 2 L | No | I do not have | More than 5 hours | Private Car | Obesity_Type_II |
| Female | 25 | 1.65 | 53.0 | No | No | Always | Three | Always | No | I do not drink | Between 1 and 2 L | No | Almost Everyday | More than 5 hours | Public Transportation | Normal_Weight |
| Male | 23 | 1.70 | 70.0 | No | No | Always | Between 1-2 | Sometimes | Yes | Sometimes | Less than 1 L | No | I do not have | More than 5 hours | Public Transportation | Normal_Weight |
| Male | 22 | 1.68 | 112.0 | Yes | Yes | Sometimes | Three | Frequently | No | I do not drink | Between 1 and 2 L | No | I do not have | More than 5 hours | Public Transportation | Obesity_Type_II |
| Male | 19 | 1.75 | 67.0 | No | Yes | Always | Three | Sometimes | Yes | I do not drink | Between 1 and 2 L | No | 2 or 4 days | 3-5 hours | Bike | Normal_Weight |

**Figure 4.2. Preview of the First Five Rows of the Dhaka Obesity Dataset**

Figure 4.2 shows that the initial data used the attribute names and class labels from the source dataset. This information became the basis for the next stage: dataset adjustment based on the Indonesian Ministry of Health standard, data cleaning, and variable transformation so the dataset could be used in modeling.

Dataset adjustment was performed because the target variable in the original dataset still followed the default classification from the source dataset, while this study used the Asia Pacific BMI classification adopted by the Indonesian Ministry of Health for the adult population. Therefore, before target relabeling was conducted, adult respondent selection was first performed by removing all respondents under 18 years of age. After that, the obesity target was reconstructed based on the actual BMI calculation from the Height (m) and Weight (kg) attributes. The weight classification used in this study consisted of five classes: Underweight for BMI < 18.5, Normal for BMI 18.5–22.9, Overweight for BMI 23.0–24.9, Obesitas_I for BMI 25.0–29.9, and Obesitas_II for BMI ≥ 30.0. Thus, the target adjustment in this study was not performed merely by renaming the classes, but by recalculating each respondent's nutritional status based on the actual BMI value.

Based on this classification, the first step was to separate underage respondents from adult respondents. This process was necessary to ensure that BMI calculation and target relabeling in the next stage were applied only to data matching the research population. The implementation of age selection is shown in Code 4.3.

**Code 4.3. Selection of Respondents Under 18 Years of Age**

```
# Seleksi responden usia < 18 tahun
df_u18 = df_raw[df_raw["Age"] < 18].copy()
df_adj = df_raw[df_raw["Age"] >= 18].copy()

# Contoh data responden yang dihapus
kolom_contoh = ["Gender", "Age", "Height (m)", "Weight (kg)", "Obesity
level"]
display(df_u18[kolom_contoh].head(5))

# Rekap jumlah data usia < 18 tahun
Tabel_hapus_u18 = (
    df_u18["Age"]
    .value_counts()
    .sort_index()
    .reset_index()
)
Tabel_hapus_u18.columns = ["Keterangan", "Jumlah"]
Tabel_hapus_u18["Keterangan"] = Tabel_hapus_u18["Keterangan"].apply(
    lambda x: f"Usia {x} tahun"
)
ringkasan = pd.DataFrame({
    "Keterangan": [
        "Total data usia < 18 tahun yang dihapus",
        "Total data awal",
        "Total data akhir usia >= 18 tahun"
    ],
    "Jumlah": [
        len(df_u18),
        len(df_raw),
        len(df_adj)
})
display(pd.concat([Tabel_hapus_u18, ringkasan], ignore_index=True))
```

Code 4.3 separated respondents under 18 years of age from adult respondents. The program displayed five examples of removed respondents and generated a summary of the number of eliminated data based on age. This step was carried out to ensure that the dataset used in the next stage only included adult respondents, in accordance with the Asia Pacific BMI classification adopted by the Indonesian Ministry of Health as the reference standard in this study. The execution results of Code 4.3 are shown in Figure 4.3 and Figure 4.4. Figure 4.3 shows five examples of respondents under 18 years of age who were removed from the dataset.

These examples indicate that the removed data came from different initial obesity labels Overweight, Normal_Weight, Obesity_Type_II, and Insufficient_Weight..

| | Gender | Age | Height (m) | Weight (kg) | Obesity level |
|---|---|---|---|---|---|
| 0 | Male | 16 | 1.57 | 73.0 | Overweight |
| 1 | Male | 16 | 1.73 | 58.0 | Normal_Weight |
| 2 | Male | 17 | 1.63 | 95.0 | Obesity_Type_II |
| 3 | Male | 17 | 1.88 | 55.0 | Insufficient_Weight |
| 4 | Female | 15 | 1.68 | 65.0 | Normal_Weight |

**Figure 4.3. Examples of Five Respondents Under 18 Years of Age Removed**

Based on these examples, it can be seen that the elimination process did not remove data from only one target category, but included respondents under 18 years of age from various initial obesity classes. Therefore, this removal was conducted purely based on the age criterion, not based on the target class distribution.

| Description | Count |
|---|---|
| Age 15 years | 8.00 |
| Age 16 years | 15.00 |
| Age 17 years | 9.00 |
| Total data for age < 18 years deleted | 32.00 |
| Percentage of data deleted (%) | 1.47 |
| Total initial data | 2182.00 |
| Total final data (age >= 18 years) | 2150.00 |

**Figure 4.4. Number of Data Under 18 Years of Age Removed**

Figure 4.4 presents the recap of removed data by age, consisting of 8 records from age 15, 15 records from age 16, and 9 records from age 17. Thus, the total eliminated data amounted to 32 respondents, or approximately 1.47% of the entire dataset. After the age selection process, the dataset size changed from 2,182 rows to 2,150 rows. This stage was important to maintain the suitability between the

research data and the classification standard used, so that the relabeling process in the next stage was applied only to the adult population.

After adult respondent data were obtained, the next step was to calculate the Body Mass Index (BMI) value for each respondent and remap the target label based on the Asia Pacific BMI classification. Through this approach, the new target label was determined from the actual BMI value, making it consistent with the classification standard used in this study. Therefore, the old labels from the original dataset were not always retained, because one label could change into a different new label according to the BMI calculation results of each respondent.

**Code 4.4. BMI Calculation and Target Relabeling**

```
# Perhitungan IMT dan relabeling target
df_adj[["Height (m)", "Weight (kg)"]] = df_adj[["Height (m)", "Weight
(kg)"]].apply(
    pd.to_numeric, errors="coerce"
)

df_adj["IMT"] = (df_adj["Weight (kg)"] / (df_adj["Height (m)"] **
2)).round(2)

bins = [ np.inf, 18.5, 23.0, 25.0, 30.0, np.inf]
labels = ["Underweight", "Normal", "Overweight", "Obesitas_I",
"Obesitas_II"]

df_adj["Obesity_Level"] = pd.cut(
    df_adj["IMT"],
    bins=bins,
    labels=labels,
    right=False
)

# Contoh hasil relabeling
contoh_relabel = df_adj[[
    "Gender", "Age", "Height (m)", "Weight (kg)",
    "Obesity level", "IMT", "Obesity_Level"
]].rename(columns={
    "Obesity level": "Label Lama",
    "Obesity_Level": "Label Baru"
}).head(5)

display(contoh_relabel)
```

Code 4.4 was used to calculate the BMI value of each respondent and generate a new target label based on the Asia Pacific BMI classification adopted by the Indonesian Ministry of Health. This program also displayed examples of relabeling results along with the old and new label pairs formed in the dataset. The execution results of Code 4.4 are shown in Figure 4.5 and Figure 4.6. Figure 4.5 shows five examples of target relabeling results. It can be seen that relabeling was

performed based on the actual BMI value, not merely by renaming the classes. For example, a respondent with the initial label Normal_Weight and a BMI of 24.22 was reclassified as Overweight, while a respondent with the initial label Obesity_Type_II and a BMI of 37.10 remained in the Obesitas_II category.

| | Gender | Age | Height (m) | Weight (kg) | Label Lama | IMT | Label Baru |
|---|---|---|---|---|---|---|---|
| 0 | Male | 29 | 1.65 | 101.0 | Obesity_Type_II | 37.10 | Obesitas_II |
| 1 | Female | 25 | 1.65 | 53.0 | Normal_Weight | 19.47 | Normal |
| 2 | Male | 23 | 1.70 | 70.0 | Normal_Weight | 24.22 | Overweight |
| 3 | Male | 22 | 1.68 | 112.0 | Obesity_Type_II | 39.68 | Obesitas_II |
| 4 | Male | 19 | 1.75 | 67.0 | Normal_Weight | 21.88 | Normal |

**Figure 4.5. Examples of Five Data Records After Target Relabeling**

Based on these results, it can be seen that the final target label was determined by the BMI calculation of each respondent. Thus, one old label could be mapped to a different new label according to the respondent's BMI value. To observe the overall impact of target adjustment, the class distribution before and after relabeling was compared. After age selection, the initial target still consisted of six classes: Insufficient_Weight with 376 records, Normal_Weight with 512 records, Overweight with 414 records, Obesity_Type_I with 315 records, Obesity_Type_II with 282 records, and Obesity_Type_III with 251 records. After target adjustment based on the Asia Pacific BMI classification, the class distribution changed into five classes: Underweight with 376 records, Normal with 319 records, Overweight with 193 records, Obesitas_I with 414 records, and Obesitas_II with 848 records. This change shows that the adjustment process not only reduced the number of classes, but also substantially changed the target class composition, especially in the Obesitas_II class, which became the largest class after relabeling. Figure 4.6 presents the target class distribution before and after dataset adjustment. The figure visually shows that the number of target classes decreased from six to five after relabeling based on the Asia Pacific BMI classification. Visually, Obesitas_II became the most dominant class after adjustment, while Normal and Overweight decreased compared to the initial class distribution from the original

dataset. Therefore, the target adjustment process produced a class structure aligned with the focus of this study, namely adult obesity level classification based on the Indonesian Ministry of Health standard.

| | Class Before | Count Before | Class After | Count After |
|---|---|---|---|---|
| 0 | Insufficient_Weight | 376 | Underweight | 376.0 |
| 1 | Normal_Weight | 512 | Normal | 319.0 |
| 2 | Overweight | 414 | Overweight | 193.0 |
| 3 | Obesity_Type_I | 315 | Obesitas_I | 414.0 |
| 4 | Obesity_Type_II | 292 | Obesitas_II | 848.0 |
| 5 | Obesity_Type_III | 251 | NaN | NaN |

**Figure 4.6. Changes in Target Class Distribution**

In the final stage, the adjusted dataset was saved as DhakaObesity_TargetAsiaPasifik.xlsx to be used in the subsequent preprocessing and modeling stages. The final dataset summary showed that the output file contained 2,150 rows, 17 columns, and a new target variable named Obesity_Level. The preview of the first five rows of the final dataset showed that the predictor attributes were retained, while the old target variable Obesity level had been replaced with the new target variable Obesity_Level. Therefore, the data acquisition and dataset adjustment stage produced a dataset ready to be used in the next stages, including data cleaning, encoding, training and testing data splitting, class imbalance handling, and model development..

### 4.1.3. Data Preprocessing

At this stage, data preprocessing was carried out to prepare the dataset with adjusted target labels before it was used in the modeling process. The preprocessing stages in this study included data cleaning, data encoding, feature and target separation, and training and testing data splitting. The processed dataset was DhakaObesity_TargetAsiaPasifik.xlsx, which initially consisted of 2,150 rows and 17 columns. After all preprocessing stages were completed, the dataset became more consistent, numerical, and ready to be used in the model development stage.

a. Data Cleaning

The data cleaning stage was conducted to ensure that the dataset with adjusted target labels was consistent, well structured, and ready for the next stage. In this study, data cleaning was not performed by removing certain attributes or replacing zero values with mean values. Instead, it focused on four main stages: standardizing column names, standardizing category values in categorical variables, checking missing values, and detecting and removing duplicate data. This approach was selected because all attributes in the dataset were still relevant to the research objectives, while the main issues found were inconsistencies in naming, variations in category writing, and the presence of duplicate data.

The first step in data cleaning was standardizing column names to make them shorter, more consistent, and easier to use in data processing with Python. The original column names in the dataset still contained spaces, parentheses, and long phrases, so they needed to be converted into shorter variable codes. For example, the Height (m) column was changed to HEIGHT, Weight (kg) was changed to WEIGHT, and Family history of overweight was changed to FHO. This standardization aimed to simplify variable calling in the program code and maintain consistent naming throughout the analysis stages.

**Code 4.5. Dataset Column Name Standardization**

```
# Penyeragaman nama kolom
rename_columns = {
    "Gender": "GENDER",
    "Age": "AGE",
    "Height (m)": "HEIGHT",
    "Weight (kg)": "WEIGHT",
    "Family history of overweight": "FHO",
    "High caloric food consumption": "HCFC",
    "Vegetable consumption frequency": "VCF",
    "Daily main meals frequency": "DMMF",
    "Between meal food consumption frequency": "BMFC",
    "Smoking": "SMOKE",
    "Alcohol intake": "ALCOHOL",
    "Daily water intake": "DWI",
    "Monitor calories": "MC",
    "Physical exercise": "PE",
    "Daily device usage duration": "DDU",
    "Mode of transportation": "MOT",
    "Obesity_Level": "OBESITY"
}

df = df_raw.rename(columns=rename_columns).copy()
df.head()
```

Code 4.5 was used to standardize the dataset column names into a shorter and more consistent format. This standardization was carried out so that each attribute name could be used more easily during preprocessing, data analysis, and machine learning modeling without changing the meaning of each variable. The results of column name standardization are shown in Figure 4.7.

| | Nama Kolom Asli | Kode Variabel |
|---|---|---|
| 0 | Gender | GENDER |
| 1 | Age | AGE |
| 2 | Height (m) | HEIGHT |
| 3 | Weight (kg) | WEIGHT |
| 4 | Family history of overweight | FHO |
| 5 | High caloric food consumption | HCFC |
| 6 | Vegetable consumption frequency | VCF |
| 7 | Daily main meals frequency | DMMF |
| 8 | Between-meal food consumption frequency | BMFC |
| 9 | Smoking | SMOKE |
| 10 | Alcohol intake | ALCOHOL |
| 11 | Daily water intake | DWI |
| 12 | Monitor calories | MC |
| 13 | Physical exercise | PE |
| 14 | Daily device usage duration | DDU |
| 15 | Mode of transportation | MOT |
| 16 | Obesity_Level | OBESITY |

**Figure 4.7. Results of Column Name Standardization**

After the column names were standardized, the next stage was cleaning the category values in categorical variables. This process was necessary because several categories had the same meaning but were written in different formats. For example, the categories “Between 1 2” and “Between 1  2” needed to be standardized into Between_1_2, while “I do not drink” was changed into I_do_not_drink. Without standardization, the system could interpret labels that should have the same meaning as different categories.

**Code 4.6. Category Value Standardization in Categorical Variables**

```
# Standarisasi isi kategori variabel kategorikal
cat_cols = df.select_dtypes(include="object").columns
```

**Code 4.6. Category Value Standardization in Categorical Variables**

```
df[cat_cols] = df[cat_cols].apply(
    lambda s: s.astype(str)
              .str.strip()
              .str.replace("[--]", " ", regex=True)
              .str.replace(r"\s* \s*", "_", regex=True)
              .str.replace(" ", "_", regex=False)
)
# Perbaikan penulisan kategori tertentu
df = df.replace({
    "three": "Three",
    "More_than_three": "More_than_Three"
})
display(df.head())
```

Code 4.6 was used to standardize category values in categorical variables so that there were no writing differences for categories with the same meaning. This standardization process was carried out to maintain data consistency, ensuring that each category could be correctly recognized during preprocessing and modeling. Examples of the category standardization results are shown in Figure 4.8..

| Index | Attribute | Before Standardization | After Standardization |
|---|---|---|---|
| 0 | DMMF | Between 1-2 | Between_1_2 |
| 1 | DMMF | More than Three | More_than_Three |
| 2 | DWI | Between 1 and 2 L | Between_1_and_2_L |
| 3 | PE | Almost Everyday | Almost_Everyday |
| 4 | DDU | More than 5 hours | More_than_5_hours |
| 5 | MOT | Public Transportation | Public_Transportation |
| 6 | ALCOHOL | I do not drink | I_do_not_drink |

**Figure 4.8. Categorical Value Standardization**

The next stage was checking missing values to ensure that there were no empty values in the dataset after column name standardization and category value standardization had been performed. This checking process was important so that encoding and model training would not be affected by incomplete data. Code 4.7 was used to check missing values.

**Code 4.7. Missing Value Checking**

```
# Pengecekan missing value
missing_summary = df.isnull().sum().reset_index()
missing_summary.columns = ["Kolom", "Jumlah Missing Value"]
print(missing_summary)
```

Based on the checking results, all columns in the dataset had a missing value count of 0. This indicates that the dataset did not contain missing data and could proceed to the next preprocessing stage without imputation. The missing value checking results are shown in Figure 4.9.

**Missing Value Count per Column**

| | | |
|---|---|---|
| **0** | GENDER | 0 |
| **1** | AGE | 0 |
| **2** | HEIGHT | 0 |
| **3** | WEIGHT | 0 |
| **4** | FHO | 0 |
| **5** | HCFC | 0 |
| **6** | VCF | 0 |
| **7** | DMMF | 0 |
| **8** | BMFC | 0 |
| **9** | SMOKE | 0 |
| **10** | ALCOHOL | 0 |
| **11** | DWI | 0 |
| **12** | MC | 0 |
| **13** | PE | 0 |
| **14** | DDU | 0 |
| **15** | MOT | 0 |
| **16** | OBESITY | 0 |

**Figure 4.9. Missing Value Checking Results**

The final stage in data cleaning was detecting and removing duplicate data. Duplicate data refers to rows that have completely identical values to other rows. The presence of duplicate data can make the data distribution less proportional and may cause the model to repeatedly learn certain patterns. Therefore, duplicate data needed to be identified and removed before the analysis process was continued..

**Code 4.8. Duplicate Data Detection and Removal**

```
# Deteksi data duplikat
rows_before, cols_before = df.shape
dup_count = int(df.duplicated().sum())
dup_mask_all = df.duplicated(keep=False)

# Menghitung grup duplikat dan jumlah baris yang terlibat
dup_groups_df = (
    df.groupby(list(df.columns), dropna=False)
```

**Code 4.8. Duplicate Data Detection and Removal**

```
      .size()
      .reset_index(name="Frekuensi")
)
dup_groups_only = dup_groups_df[dup_groups_df["Frekuensi"] > 1].copy()
dup_group_count = int(dup_groups_only.shape[0])
dup_rows_involved = int(dup_groups_only["Frekuensi"].sum())

# Menghapus data duplikat
df_clean = df.drop_duplicates().reset_index(drop=True)
rows_after, cols_after = df_clean.shape
dup_after = int(df_clean.duplicated().sum())
print("Jumlah duplikasi sebelum dihapus:", dup_count)
print("Jumlah duplikasi sesudah dihapus:", dup_after)
print("Ukuran data setelah cleaning:", df_clean.shape)
```

The checking results showed that before removal, there were 25 duplicate rows, with 49 rows involved in 24 duplicate groups. After the removal process, the dataset size changed from 2,150 rows to 2,125 rows, and no duplicate data remained in the dataset. The summary of duplicate data removal is shown in Figure 4.10.

| Condition | Number of Rows | Number of Columns | Number of Duplicates | Number of Duplicated Rows | Number of Duplicated Groups |
|---|---|---|---|---|---|
| Before Deletion | 2150 | 17 | 25 | 49 | 24 |
| After Deletion | 2125 | 17 | 0 | 0 | 0 |

**Figure 4.10. Duplicate Data Summary**

After duplicate data were removed, the cleaned dataset was displayed again to show the condition of the data after cleaning. The final cleaning results showed that the column names had been standardized, category writing was consistent, no missing values were found, and duplicate rows had been removed.

b. Data Encoding

The data encoding stage was carried out after the cleaning process was completed, with the aim of converting categorical variables into numerical form so that they could be processed by the XGBoost and LightGBM models. In this study, the encoding method was adjusted according to the characteristics of each variable. Binary variables such as GENDER, FHO, HCFC, SMOKE, and MC were encoded using label encoding. Ordinal variables such as VCF, DMMF, BMFC, ALCOHOL, DWI, PE, and DDU were encoded using ordinal encoding. Meanwhile, the MOT

variable was encoded using one hot encoding. Numerical variables such as AGE, HEIGHT, and WEIGHT were not encoded because they were already in numerical form from the beginning. In addition to the feature variables, the OBESITY target variable was also converted into numerical form so that it could be used in multiclass classification. In this study, the target was mapped as Underweight = 0, Normal = 1, Overweight = 2, Obesitas_I = 3, and Obesitas_II = 4. The encoding results showed that the categorical data, which were originally in text form, were successfully converted into numerical data without removing the meaning of each variable.

**Code 4.9. Data Encoding Process**

```
# Proses encoding data
df_encoded = df_clean.copy()

encoding_order = {
    "GENDER": ["Male", "Female"],
    "FHO": ["No", "Yes"],
    "HCFC": ["No", "Yes"],
    "SMOKE": ["No", "Yes"],
    "MC": ["No", "Yes"],
    "VCF": ["Never", "Sometimes", "Always"],
    "DMMF": ["Between_1_2", "Three", "More_than_Three"],
    "BMFC": ["No", "Sometimes", "Frequently", "Always"],
    "ALCOHOL": ["I_do_not_drink", "Sometimes", "Frequently", "Always"],
    "DWI": ["Less_than_a_liter", "Between_1_and_2_L", "More_than_2_L"],
    "PE": ["I_do_not_have", "1_or_2_days", "2_or_4_days", "4_or_5_days",
"Almost_Everyday"],
    "DDU": ["0_2_hours", "3_5_hours", "More_than_5_hours"],
    "OBESITY": ["Underweight", "Normal", "Overweight", "Obesitas_I",
"Obesitas_II"]

for col, order in encoding_order.items():
    df_encoded[col] = pd.Categorical(
        df_encoded[col],
        categories=order,
        ordered=True
    ).codes

df_encoded = pd.get_dummies(
    df_encoded,
    columns=["MOT"],
    prefix="MOT",
    dtype=int
display(df_encoded.head())
```

Code 4.9 was used to convert all categorical variables into numerical form according to their data types. In the code, binary variables were encoded using label encoding, ordinal variables were encoded using ordinal encoding, the MOT variable was transformed using one hot encoding, and the OBESITY target was mapped into

five numerical classes. The results of this process showed that data originally in text category form had been converted into numerical data and were ready for the next stage. After the encoding process was completed, the transformed dataset was checked again. The checking results showed that all columns in the encoded dataset had a missing value count of 0, so no missing data problems were found after transformation. In addition, the program output also showed changes in the data structure before and after encoding, including the addition of new columns from one hot encoding for the MOT variable, namely MOT_Bike, MOT_Private_Car, MOT_Public_Transportation, MOT_Rickshaw, and MOT_Walking.

| | GENDER | FHO | HCFC | VCF | DMMF | BMFC | ALCOHOL | DWI | PE | DDU | OBESITY | MOT Bike | MOT Private Car | MOT Public Transportation | MOT Rickshaw | MOT Walking |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 1 | 1 | 1 | 2 | 0 | 1 | 0 | 2 | 4 | 0 | 1 | 0 | 0 | 0 |
| **1** | 1 | 0 | 0 | 2 | 1 | 3 | 0 | 1 | 4 | 2 | 1 | 0 | 0 | 1 | 0 | 0 |
| **2** | 0 | 0 | 0 | 2 | 0 | 1 | 1 | 0 | 0 | 2 | 2 | 0 | 0 | 1 | 0 | 0 |
| **3** | 0 | 1 | 1 | 1 | 1 | 2 | 0 | 1 | 0 | 2 | 4 | 0 | 0 | 1 | 0 | 0 |
| **4** | 0 | 0 | 1 | 2 | 1 | 1 | 0 | 1 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

**Figure 4.11. Example of Encoded Data**

Figure 4.11 shows an example of encoded data, namely the conversion of several categorical attributes into numerical values. The results show that variables such as GENDER, FHO, HCFC, VCF, DMMF, BMFC, ALCOHOL, DWI, PE, and DDU were successfully converted into numerical form, while the MOT variable was split into several binary columns. Therefore, the encoded dataset was already in a suitable format to be used in the training and testing data split stage. This encoding stage was important so that all attributes could be processed by the XGBoost and LightGBM algorithms, which require numerical input.

c. Pembagian Data Latih dan Data Uji Training and Testing Data Split

The training and testing data split stage was carried out after the cleaning and encoding processes were completed. At this stage, the preprocessed dataset was divided into two main parts, namely features and target. The features consisted of all encoded attributes used as model input variables, while the target was taken from

the OBESITY column, which represented the obesity level class. Data splitting was performed using the stratified train test split technique. This technique was used to maintain the proportion of each target class in both the training and testing data, so that classes such as Underweight, Normal, Overweight, Obesitas_I, and Obesitas_II remained proportionally represented in both subsets. Stratification was important because the dataset had an imbalanced class distribution, so data splitting needed to be controlled to make the model evaluation results more representative. In this study, the training and testing data split was not only used as a preprocessing stage, but also as part of the model testing scheme. Therefore, variations in data split ratios are discussed further in the comparison of testing schemes subsection. Thus, this section only explains the general mechanism for separating features, target, training data, and testing data before balancing and modeling were carried out.

### 4.1.4. Data Balancing Using Random Oversampling

The data balancing stage was carried out after splitting the training and testing data to handle class imbalance in the training set. Class imbalance occurs when the number of samples in each class is not proportional, causing the model to learn more from the majority class and perform less optimally on minority classes. In this study, Random Oversampling (ROS) was used by randomly duplicating minority class samples until their distribution became balanced with the majority class. ROS does not generate synthetic data, but increases minority class representation by copying existing samples.

ROS was applied only to the training data, not to the entire dataset or testing data, to prevent data leakage. Therefore, the testing data remained in their original distribution, allowing the evaluation results to reflect the model's generalization ability more objectively. ROS was applied after data splitting in each testing scenario. The class distribution before and after ROS, along with its effect on model performance, is discussed further in the subsection comparing the baseline model with Random Oversampling.

**Code 4.10. Random Oversampling Implementation**

```
# Implementasi Random Oversampling pada data latih ros =
RandomOverSampler(random_state=RANDOM_STATE) X_train_ros, y_train_ros =
ros.fit_resample(X_train, y_train)
```

Code 4.10 was used to apply Random Oversampling to the training data. The RandomOverSampler object was created using the random_state parameter to ensure that the oversampling process could be reproduced. Next, the fit_resample() function was applied to X_train and y_train to produce X_train_ros and y_train_ros as the new balanced training data. The ROS balanced training data were then used in the XGBoost and LightGBM model development stage. Meanwhile, the testing data retained their original distribution and were not subjected to oversampling, so model evaluation was still conducted on unmodified data.

### 4.1.5. Model Development

The model development stage was carried out after the dataset had undergone preprocessing, training and testing data splitting, and the application of Random Oversampling to the training data. At this stage, the balanced training data were used as input to train the obesity level classification models. The models used in this study were XGBoost and LightGBM. Both algorithms are decision tree based gradient boosting methods that are suitable for tabular data and capable of learning nonlinear relationships between variables.

In the program implementation, the models were developed to solve a multiclass classification problem with five target classes, namely Underweight, Normal, Overweight, Obesitas_I, and Obesitas_II. The OBESITY target had previously been encoded into numerical values from 0 to 4, allowing the models to process the target data in a suitable format. The training data balanced using Random Oversampling were used in the training process, while the testing data were maintained in their original condition to measure the models' ability to classify unseen data.

**Code 4.11. XGBoost and LightGBM Model Initialization**

```
# Fungsi inisialisasi model
def get_model_default(model_name):
    if model_name == "XGBoost":
        return XGBClassifier(
            objective="multi:softprob",
            num_class=5,
            eval_metric="mlogloss",
            random_state=RANDOM_STATE,
            n_jobs= 1,
            tree_method="hist",
            verbosity=0
        )
```

**Code 4.11. XGBoost and LightGBM Model Initialization**

```
if model_name == "LightGBM":
    return LGBMClassifier(
        objective="multiclass",
        num_class=5,
        random_state=RANDOM_STATE,
        n_jobs= 1,
        verbose= 1,
        subsample_freq=1
    )

raise ValueError("Model tidak dikenali.")
```

Code 4.11 was used to initialize the XGBoost and LightGBM classification models. The get_model_default() function receives the name of the selected model as input. If the selected model is XGBoost, the program creates an XGBClassifier object. If the selected model is LightGBM, the program creates an LGBMClassifier object. In the XGBoost model, the objective = "multi:softprob" parameter was used because this study involved multiclass classification. The num_class = 5 parameter indicates that the classification target consists of five classes, while eval_metric = "mlogloss" was used to monitor the loss value during training. In the LightGBM model, the objective = "multiclass" and num_class = 5 parameters were used so that the model could perform classification on five target classes.

In addition, the random_state parameter was used to ensure that the training results could be reproduced, n_jobs = 1 was used to utilize all available CPU cores, and verbose was used to control the display of the training process. At this stage, the code only shows the general process of model initialization. The testing of parameter variations and the selection of the best configuration are discussed specifically in the hyperparameter comparison subsection. After the models were initialized, the training process was carried out using the Random Oversampling balanced training data. The trained models were then used to predict the classes in the testing data. The prediction results were subsequently evaluated to determine the performance of each model in classifying obesity levels.

### 4.1.6. Model Evaluation

The model evaluation stage was carried out after the model development and training processes were completed. At this stage, the models trained using

Random Oversampling balanced training data were used to predict the classes in the testing data. The testing data were not subjected to oversampling, so the evaluation results still reflected the models' ability to recognize data with the original distribution. Model evaluation in this study did not only use accuracy, but also precision macro, recall macro, F1 macro, balanced accuracy, classification report, confusion matrix, training validation loss graph, and train test gap. These metrics were selected because this study addressed multiclass classification with an imbalanced class distribution, so model performance assessment could not rely only on accuracy.

Accuracy was used to measure the proportion of correct predictions among all testing data. Precision macro was used to calculate the average prediction precision across all classes with equal weight. Recall macro was used to measure the average ability of the model to identify actual data in each class. F1 macro was used to evaluate the balance between precision and recall across all classes. Meanwhile, balanced accuracy was used as an important metric because it calculates the average recall of each class, making it more suitable for evaluating model performance on data with an imbalanced class distribution. In addition to aggregate metrics, this study also used a classification report to observe the precision, recall, F1 score, and support values for each class. A confusion matrix was used to examine the pattern of correct and incorrect predictions for each target class. Through the confusion matrix, it can be identified which classes were predicted well and which classes were still frequently misclassified as other classes. The training validation loss graph was used to observe the model learning pattern during the boosting process, while the train test gap was used to examine the performance difference between the training and testing data.

**Code 4.12. Model Training and Initial Evaluation**

```
# Fungsi evaluasi model
def evaluate_predictions(y_true, y_pred):
    return {
        "Accuracy": accuracy_score(y_true, y_pred),
        "Precision_macro": precision_score(
            y_true, y_pred, average="macro", zero_division=0
        ),
        "Recall_macro": recall_score(
            y_true, y_pred, average="macro", zero_division=0
        ),
        "F1_macro": f1_score(
```

**Code 4.12. Model Training and Initial Evaluation**

```
            y_true, y_pred, average="macro", zero_division=0
        ),
        "Balanced_Accuracy": balanced_accuracy_score(y_true, y_pred)
    }

# Evaluasi hasil prediksi model
test_metrics = evaluate_predictions(y_test, y_test_pred)

report_text = classification_report(
    y_test,
    y_test_pred,
    labels=LABEL_ORDER_NUM,
    target_names=LABEL_ORDER_TEXT,
    digits=4,
    zero_division=0
)
cm = confusion_matrix(
    y_test,
    y_test_pred,
    labels=LABEL_ORDER_NUM
)
```

Code 4.12 was used to calculate the model evaluation metrics based on the prediction results on the testing data. The evaluate_predictions() function produced accuracy, precision macro, recall macro, F1 macro, and balanced accuracy values. In addition, classification_report() was used to display the detailed performance of each class, while confusion_matrix() was used to generate the prediction error matrix among classes. The evaluation results from this code were used as the basis for comparing the performance of XGBoost and LightGBM in the testing schemes. However, the detailed discussion of the evaluation results is not presented in this subsection, but explained in the subsection comparing the testing schemes.

## 4.2. Comparison of Testing Schemes

This subsection discusses the comparison of testing schemes in the obesity level classification models. Testing was conducted to determine the effect of data split ratio variations and hyperparameter settings on model performance. The models used in this study were XGBoost and LightGBM, with Random Oversampling (ROS) applied as a method for handling class imbalance in the training data. The dataset used was the result of preprocessing and encoding from the previous stage, consisting of 2,125 data records and 21 columns. The target variable in this study was OBESITY, which consisted of five classes: Underweight,

Normal, Overweight, Obesitas_I, and Obesitas_II.

Model evaluation was conducted using several metrics, namely accuracy, precision, recall, F1 macro, and balanced accuracy. Accuracy was used to measure the overall prediction correctness of the model, while precision, recall, and F1 macro were used to evaluate model performance for each class. In this study, F1 macro and balanced accuracy were the main focus because both metrics are more suitable for evaluating multiclass classification, especially when the class distribution is not fully balanced. Therefore, model performance was not only assessed based on overall accuracy, but also based on the model's ability to recognize each target class more evenly.

The testing scheme was carried out in stages. The first stage was testing data split ratios using three variations: 60:40, 70:30, and 80:20. After the data were divided into training and testing data, ROS was applied only to the training data to make the class distribution more balanced. The testing data were maintained in their original condition so that the evaluation results could reflect the model's ability to classify unseen data. The next stage was stepwise manual hyperparameter testing on learning_rate, max_depth, n_estimators, subsample, and colsample_bytree. The testing was conducted one parameter at a time, so the best value from the previous parameter was used in the next parameter test. Therefore, the final configuration obtained was the result of stepwise selection based on evaluation metrics, not a global optimization of all parameter combinations. The manual testing scheme of the main models in this study is shown in Table 4.1..

**Table 4.1. Main Model Manual Testing Scheme**

| Model | Split Ratio | Learning Rate | Max Depth | n_estimators | Subsample | colsample_bytree |
|---|---|---|---|---|---|---|
| XGBoost ROS | 60:40 | 0.01 \| 0.05 \| 0.10 | 3 \| 5 \| 7 | 100 \| 200 \| 300 | 0.70 \| 0.80 \| 0.90 | 0.70 \| 0.80 \| 0.90 |
| XGBoost ROS | 70:30 | 0.01 \| 0.05 \| 0.10 | 3 \| 5 \| 7 | 100 \| 200 \| 300 | 0.70 \| 0.80 \| 0.90 | 0.70 \| 0.80 \| 0.90 |
| XGBoost ROS | 80:20 | 0.01 \| 0.05 \| 0.10 | 3 \| 5 \| 7 | 100 \| 200 \| 300 | 0.70 \| 0.80 \| 0.90 | 0.70 \| 0.80 \| 0.90 |
| LightGBM ROS | 60:40 | 0.01 \| 0.05 \| 0.10 | 3 \| 5 \| 7 | 100 \| 200 \| 300 | 0.70 \| 0.80 \| 0.90 | 0.70 \| 0.80 \| 0.90 |

| Model | Split Ratio | Learning Rate | Max Depth | n_estimators | Subsample | colsample_bytree |
|---|---|---|---|---|---|---|
| LightGBM ROS | 70:30 | 0.01 \| 0.05 \| 0.10 | 3 \| 5 \| 7 | 100 \| 200 \| 300 | 0.70 \| 0.80 \| 0.90 | 0.70 \| 0.80 \| 0.90 |
| LightGBM ROS | 80:20 | 0.01 \| 0.05 \| 0.10 | 3 \| 5 \| 7 | 100 \| 200 \| 300 | 0.70 \| 0.80 \| 0.90 | 0.70 \| 0.80 \| 0.90 |

### 4.2.1. Testing of Data Split Ratios on XGBoost + ROS

Testing the data split ratios on the XGBoost + ROS model was conducted to determine the effect of training and testing proportions on obesity level classification performance. The tested ratios were 80:20, 70:30, and 60:40. Random Oversampling was applied only to the training data after splitting, while the testing data remained in their original condition to ensure objective evaluation on unseen data. Each scenario was evaluated using a confusion matrix, classification report, training and validation loss graph, as well as accuracy, precision macro, recall macro, F1 macro, and balanced accuracy. The results were used to select the best split ratio for the stepwise manual hyperparameter testing stage.

**XGBoost + ROS Split 80:20**

In the first scenario, the dataset was divided into 80% training data and 20% testing data. This resulted in 1,700 training records and 425 testing records. After Random Oversampling was applied, the training data increased to 3,385 records. The 80:20 ratio provided the largest training portion compared to other scenarios.

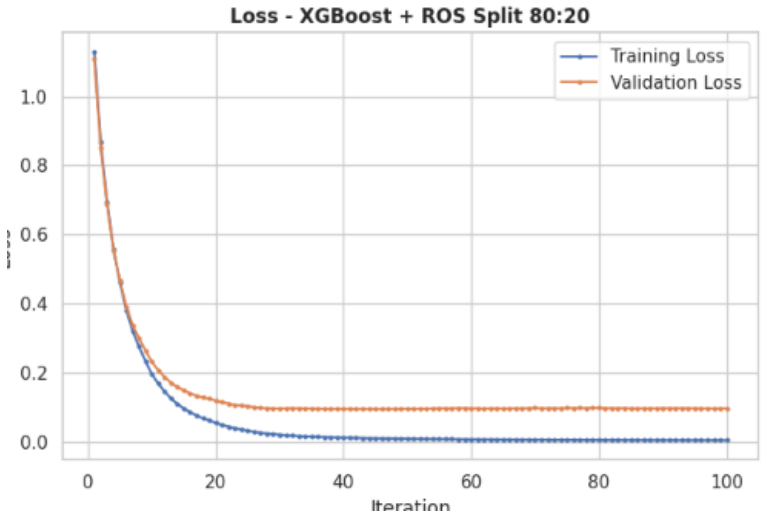



**Figure 4.12. Train Test Graph of XGBoost + ROS Split 80:20**

Figure 4.12 shows the training and validation loss graph of the XGBoost + ROS model using the 80:20 split ratio. The decreasing loss pattern indicates that the model learned the training data well while maintaining stable performance on the testing data. The relatively small gap between training and validation loss shows good generalization ability without excessive overfitting. Therefore, the 80:20 split was considered suitable for testing the XGBoost + ROS model.

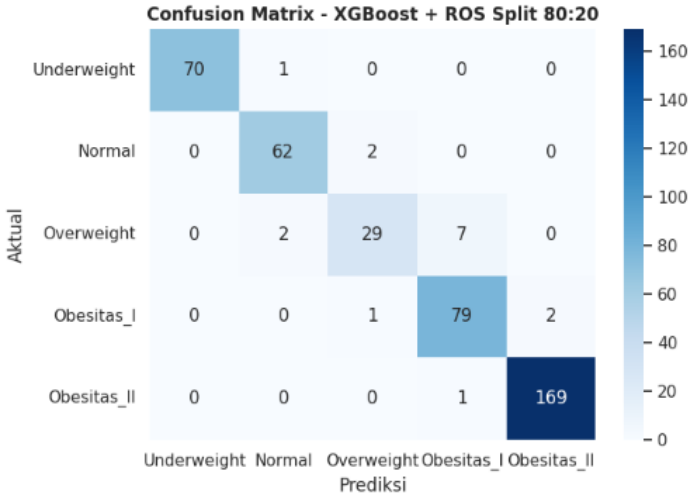



**Figure 4.13. Confusion Matrix of XGBoost + ROS Split 80:20**

Figure 4.13 shows the confusion matrix of the XGBoost + ROS model using the 80:20 split ratio. The model correctly predicted 70 Underweight, 61 Normal, 30 Overweight, 79 Obesitas_I, and 169 Obesitas_II data. Most testing data were classified correctly, although errors still occurred in the Overweight class, which overlapped with the Normal and Obesitas_I classes.

```
Model              : XGBoost + ROS
Test Type          : Split Ratio Test
Split Used         : 80:20
Test Parameter     : Split Ratio
Split Ratio        : 80:20
Testing Accuracy   : 0.962353

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

   Underweight       1.0000    0.9859    0.9929        71
        Normal       0.9538    0.9688    0.9612        64
    Overweight       0.9062    0.7632    0.8286        38
 Obesity_Type_I      0.9080    0.9634    0.9349        82
Obesity_Type_II      0.9883    0.9941    0.9912       170

      accuracy                         0.9624       425
     macro avg       0.9513    0.9351    0.9418       425
  weighted avg       0.9622    0.9624    0.9616       425
```

**Figure 4.14. Classification Report of XGBoost + ROS Split 80:20**

Figure 4.14 presents the classification report of the XGBoost + ROS model using the 80:20 split ratio. Based on the classification report, the model achieved an accuracy of 0.962353. The precision macro value of 0.951289 indicates that, on average, the model had good ability to produce correct positive predictions across all classes. The recall macro value of 0.935071 shows that the model was also able to identify most actual data in each class. Meanwhile, the F1 macro value of 0.941767 indicates a good balance between precision and recall across classes. In terms of class level performance, classes with larger amounts of data tended to show more stable performance. The Obesitas_II class showed strong results because most data in this class were correctly predicted. The Underweight class also showed good performance because its classification errors were relatively low. However, the Overweight class still requires attention because it had a tendency to be misclassified into adjacent classes, particularly Normal and Obesitas_I. This condition is reasonable because the Overweight class is located between normal weight and obesity categories, so its data characteristics may be similar to both classes.

**XGBoost + ROS Split 70:30**

In the second scenario, the dataset was divided using a 70:30 ratio, consisting of 1,487 training data and 638 testing data. After Random Oversampling was applied, the number of training data increased to 2,965 records. Compared to the 80:20 split, this ratio had a larger testing set, but fewer training data for the learning process.

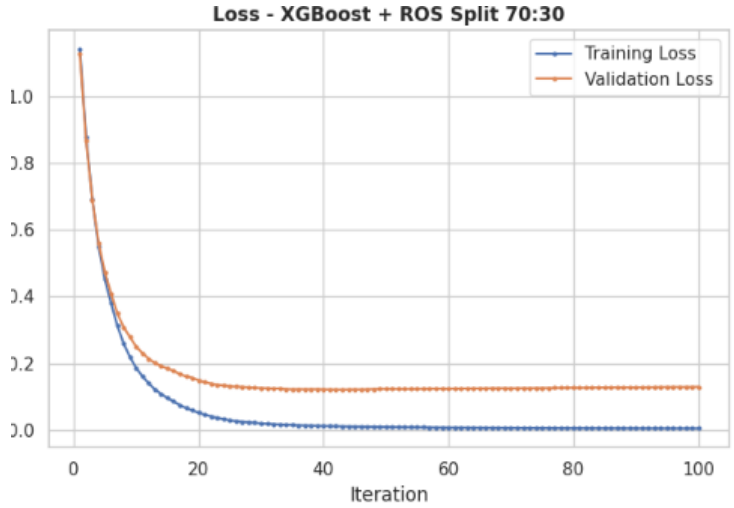



**Figure 4.15. Train Test Graph of XGBoost + ROS Split 70:30**

Figure 4.15 shows the training loss and validation loss graph of the XGBoost + ROS model using the 70:30 split ratio. This graph was used to observe the stability of model learning on the training and testing data. In this scenario, the model still showed stable performance. However, the slightly lower evaluation metrics compared to the 80:20 split indicate that the reduced amount of training data could affect the quality of model learning.

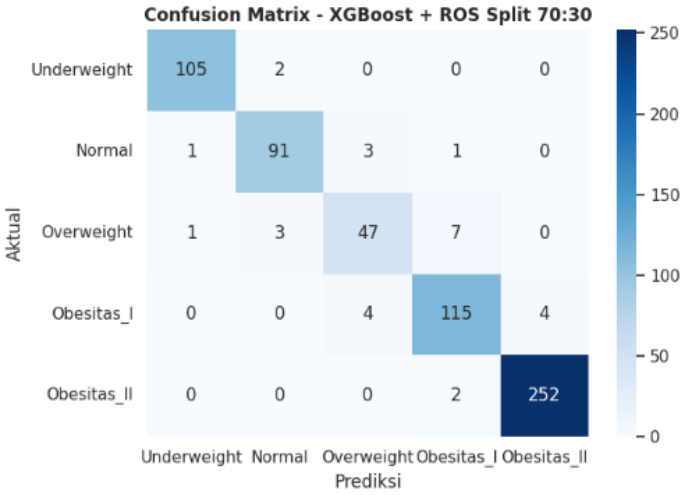



**Figure 4.16. Confusion Matrix of XGBoost + ROS Split 70:30**

Figure 4.16 shows the confusion matrix of the XGBoost + ROS model using the 70:30 split ratio. Most testing data were classified correctly, as indicated by the dominant values on the main diagonal. However, compared to the 80:20 split, prediction errors increased, especially in the Overweight class, which was still frequently misclassified as Normal and Obesitas_I.

```
Model              : XGBoost + ROS
Test Type          : Split Ratio Test
Split Ratio        : 70:30
Test Parameter     : Split Ratio
Ratio Used         : 70:30
Testing Accuracy   : 0.956113

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

   Underweight       0.9813    0.9813    0.9813       107
  Normal Weight      0.9479    0.9479    0.9479        96
    Overweight       0.8704    0.8103    0.8393        58
 Obesity Class I     0.9200    0.9350    0.9274       123
Obesity Class II     0.9844    0.9921    0.9882       254

      accuracy                         0.9561       638
     macro avg       0.9408    0.9333    0.9368       638
  weighted avg       0.9556    0.9561    0.9557       638
```

**Figure 4.17. Classification Report of XGBoost + ROS Split 70:30**

Figure 4.17 shows the classification report of the XGBoost + ROS model using the 70:30 split ratio. The classification report shows that the model achieved an accuracy of 0.956113, precision macro of 0.940794, recall macro of 0.933331, F1 macro of 0.936833, and balanced accuracy of 0.933331. These values indicate that the model still performed well, although there was a slight decrease compared to the 80:20 split scenario. This decrease indicates that the reduced number of training data may affect the model's ability to learn data patterns optimally. However, the relatively high F1 macro and balanced accuracy values show that the model was still able to maintain classification performance across all target classes.

**XGBoost + ROS Split 60:40**

In the third scenario, the dataset was divided using a 60:40 ratio, with 60% used as training data and 40% as testing data. The number of training data in this scenario was 1,275 records, while the testing data consisted of 850 records. After Random Oversampling was applied, the number of training data increased to 2,540 records. This ratio had the largest amount of testing data compared to the previous two scenarios, but the smallest amount of training data used to build the model.

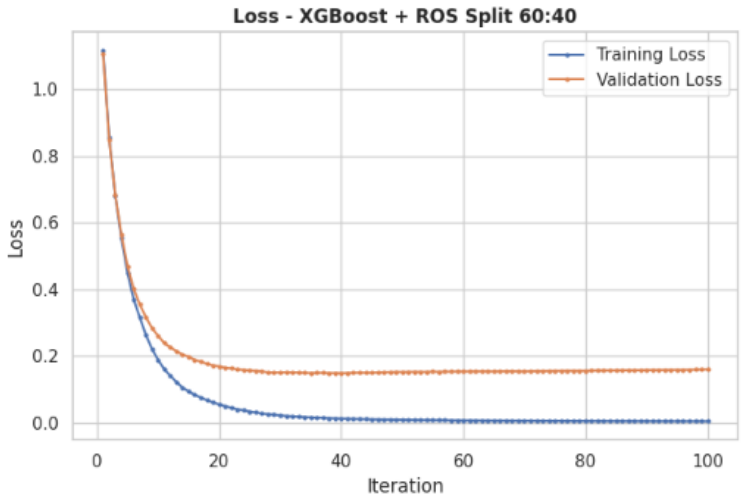



**Figure 4.18. Train Test Graph of XGBoost + ROS Split 60:40**

Figure 4.18 shows the training loss and validation loss graph of the XGBoost + ROS model using the 60:40 split ratio. This graph was used to compare model performance on the training and testing data. In this scenario, the model still showed a learning process, but the evaluation results indicated that testing performance was lower than the other two split ratios. This indicates that a smaller amount of training data may limit the model's ability to form strong classification patterns. If the gap

between training loss and validation loss appears wider, this condition needs attention because it may indicate differences in model performance between the training and testing data.

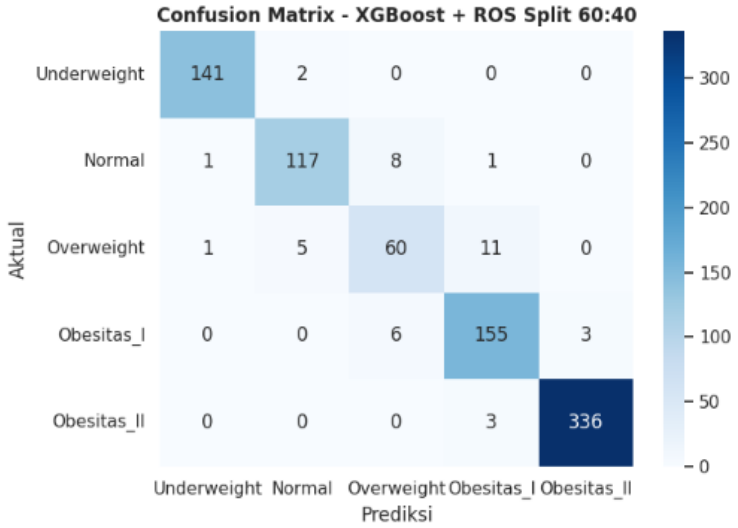



**Figure 4.19. Confusion Matrix of XGBoost + ROS Split 60:40**

Figure 4.19 shows the confusion matrix results of the XGBoost + ROS model using the 60:40 split ratio. Based on the confusion matrix, the model was still able to classify most testing data correctly, but the number of prediction errors was higher than in the 80:20 and 70:30 splits. These errors mainly occurred in the Overweight class, which was still often misclassified as Normal or Obesitas_I because its characteristics were relatively close to those classes.

```
Model               : XGBoost + ROS
Type of Test        : Split Ratio Test
Split Ratio Used    : 60:40
Test Parameters     : Split Ratio
Ratio Used          : 60:40
Testing Accuracy    : 0.951765

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

   Underweight     0.9860    0.9860    0.9860       143
        Normal     0.9435    0.9213    0.9323       127
    Overweight     0.8108    0.7792    0.7947        77
     Obesity_I     0.9118    0.9451    0.9281       164
    Obesity_II     0.9912    0.9912    0.9912       339

      accuracy                         0.9518       850
     macro avg     0.9287    0.9246    0.9265       850
  weighted avg     0.9515    0.9518    0.9515       850
```

**Figure 4.20. Classification Report of XGBoost + ROS Split 60:40**

Figure 4.20 shows the classification report of the XGBoost + ROS model using the 60:40 split ratio. The classification report shows that the model achieved an accuracy of 0.951765, precision macro of 0.928658, recall macro of 0.924553, F1 macro of 0.926456, and balanced accuracy of 0.924553. These values still indicate good classification performance, but they were the lowest compared to the other two split scenarios. This shows that the smaller amount of training data in the 60:40 split limited the variation of patterns that the model could learn. Although Random Oversampling was able to balance the class distribution in the training data, the oversampling process still depended on information from the original training data. Therefore, when the amount of training data was smaller, the model's ability to recognize patterns among classes also became more limited.

**Table 4.2. Comparison of XGBoost + ROS Split Ratios**

| Description | Split 80:20 | Split 70:30 | Split 60:40 |
|---|---|---|---|
| Accuracy | 0.962353 | 0.956113 | 0.951765 |
| Precision Macro | 0.951289 | 0.940794 | 0.928658 |
| Recall Macro | 0.935071 | 0.933331 | 0.924553 |
| F1 Macro | 0.941767 | 0.936833 | 0.926456 |
| Balanced Accuracy | 0.935071 | 0.933331 | 0.924553 |
| Train Time | 0.348381 | 0.319956 | 0.400045 |
| Status | Selected | Not Selected | Not Selected |

Based on Table 4.2, the XGBoost + ROS model achieved the best performance using the 80:20 split ratio. This is shown by an accuracy of 0.962353, precision macro of 0.951289, recall macro of 0.935071, F1 macro of 0.941767, and balanced accuracy of 0.935071. These values were the highest compared to the 70:30 and 60:40 splits. In the 70:30 split, the model achieved an accuracy of 0.956113, F1 macro of 0.936833, and balanced accuracy of 0.933331. Meanwhile, in the 60:40 split, the model achieved an accuracy of 0.951765, F1 macro of 0.926456, and balanced accuracy of 0.924553.

These results indicate that the 80:20 ratio provided the most suitable data composition for the XGBoost + ROS model in the data split ratio testing stage. The

larger amount of training data in the 80:20 ratio helped the model learn the patterns of each class more effectively, while the 20% testing data remained sufficient to evaluate the model's generalization ability. Compared to the 70:30 and 60:40 splits, the 80:20 split produced a better performance balance based on F1 macro and balanced accuracy. Therefore, the 80:20 split ratio was selected as the best ratio for the XGBoost + ROS model. This ratio was used as the reference for the next testing stage because it produced the highest overall performance and showed more stable classification ability across all target classes. Although classification errors were still found in the Overweight class, the testing results show that the XGBoost + ROS model with the 80:20 split had the most optimal performance compared to the other data split scenarios..

### 4.2.2. Testing of Data Split Ratios on LightGBM + ROS

Testing the data split ratios on the LightGBM + ROS model was conducted to determine the effect of variations in the number of training and testing data on the performance of the LightGBM model in obesity level classification. At this stage, the LightGBM model was tested using three data split ratios: 80:20, 70:30, and 60:40. Random Oversampling was applied only to the training data after the data splitting process, while the testing data were maintained in their original condition so that the evaluation results could reflect the model's ability to classify data that had not undergone balancing. Each split ratio was evaluated using a confusion matrix, classification report, training loss and validation loss graph, and evaluation metrics consisting of accuracy, precision macro, recall macro, F1 macro, and balanced accuracy. These metrics were used so that model performance was not only assessed based on overall prediction accuracy, but also based on the model's balanced ability to recognize each target class. Therefore, the best split ratio could be determined based on performance stability and the model's ability to recognize minority classes.

**LightGBM + ROS Split 80:20**

In the first scenario, the dataset was divided using an 80:20 ratio, where 80% of the data were used as training data and 20% as testing data. The number of training data in this scenario was 1,700 records, while the testing data consisted of

425 records. After Random Oversampling was applied to the training data, the number of training data increased to 3,385 records. This ratio provided a larger portion of training data compared to the other two ratios, allowing the model to learn classification patterns in each target class from more data. A larger training portion is important because the model has more examples to recognize the relationship between input features and obesity level categories. Therefore, the 80:20 split was expected to provide better learning ability while still maintaining sufficient testing data for model evaluation.

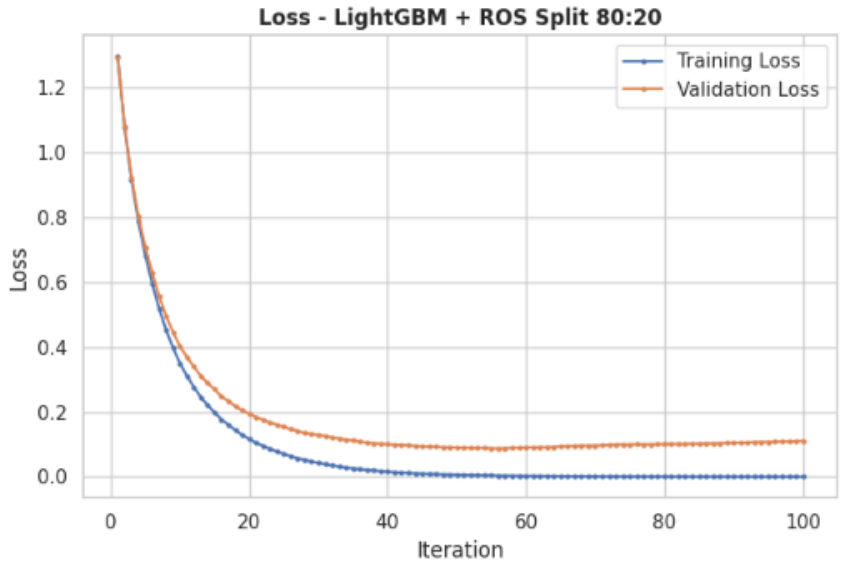



**Figure 4.21. Train Test Graph of LightGBM + ROS Split 80:20**

Figure 4.21 shows the training loss and validation loss graph of the LightGBM + ROS model using the 80:20 split ratio. The graph was used to observe the stability of the model learning process during training. In this scenario, the training loss and validation loss patterns showed a stable learning process. If both curves decrease and the gap between them is not too wide, the model can be considered able to learn from the training data without losing its generalization ability on the testing data. Based on the train test graph, the LightGBM + ROS model with the 80:20 split showed stable performance. There was no indication of excessive overfitting because the performance on the training and testing data remained balanced. This is consistent with the evaluation metrics, which show that the 80:20 split produced the highest accuracy, F1 macro, and balanced accuracy compared to the other two split ratios.

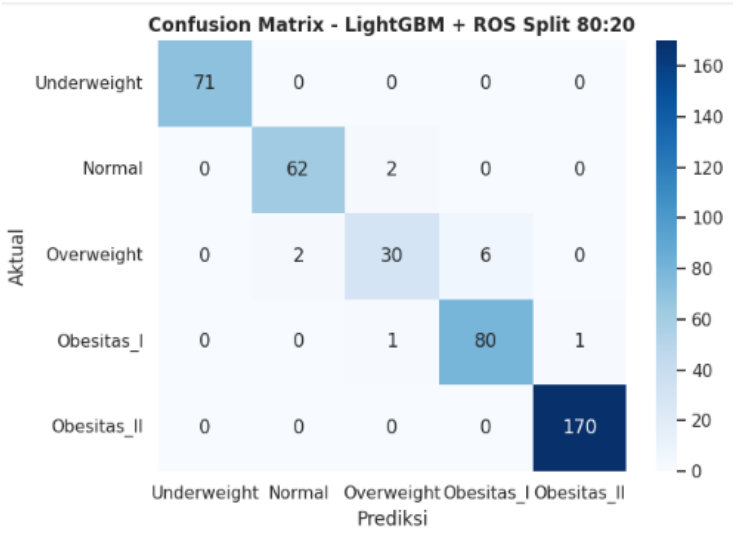



**Figure 4.22. Confusion Matrix of LightGBM + ROS Split 80:20**

Figure 4.22 shows the evaluation results of the LightGBM + ROS model using the 80:20 split ratio. Based on the confusion matrix, most testing data were successfully classified according to their actual classes, as indicated by the dominant values on the main diagonal. The Obesitas_II class showed stable prediction results because it had the largest number of data and most of its samples were correctly recognized. In addition, the Underweight and Obesitas_I classes also showed good classification patterns with relatively few prediction errors. However, classification errors still occurred in the Overweight class because this class is located between Normal and Obesitas_I, so its characteristics tend to be close to both classes and are more easily misclassified.

```
Model              : LightGBM + ROS
Test Type          : Split Ratio Test
Split Used         : 80:20
Test Parameters    : Split Ratio
Split Ratio Used   : 80:20
Testing Accuracy   : 0.971765

=== Classification Report (Testing) ===

               precision    recall  f1-score   support

  Underweight     1.0000    1.0000    1.0000        71
       Normal     0.9688    0.9688    0.9688        64
   Overweight     0.9091    0.7895    0.8451        38
    Obesity_I     0.9302    0.9756    0.9524        82
   Obesity_II     0.9942    1.0000    0.9971       170

     accuracy                         0.9718       425
    macro avg     0.9604    0.9468    0.9527       425
 weighted avg     0.9714    0.9718    0.9711       425
```

**Figure 4.23. Classification Report of LightGBM + ROS Split 80:20**

Figure 4.23 shows that the LightGBM + ROS model with the 80:20 split achieved an accuracy of 0.971765, precision macro of 0.960445, recall macro of 0.946767, F1 macro of 0.952654, and balanced accuracy of 0.946767. These values indicate that the model was able to produce very good and relatively balanced predictions across all classes. In addition, the high F1 macro value shows a good balance between precision and recall in multiclass classification. Based on the support values, the Obesitas_II class had the largest number of testing data, while Overweight had smaller support and still required attention because its characteristics were close to Normal and Obesitas_I. Overall, LightGBM + ROS with the 80:20 split was able to recognize most target classes very well.

**LightGBM + ROS Split 70:20**

In the second scenario, the dataset was divided using a 70:30 ratio, where 70% of the data were used as training data and 30% as testing data. The number of training data in this scenario was 1,487 records, while the testing data consisted of 638 records. After Random Oversampling was applied, the number of training data increased to 2,965 records. Compared to the 80:20 ratio, this scenario had a larger amount of testing data, but fewer training data for the learning process.

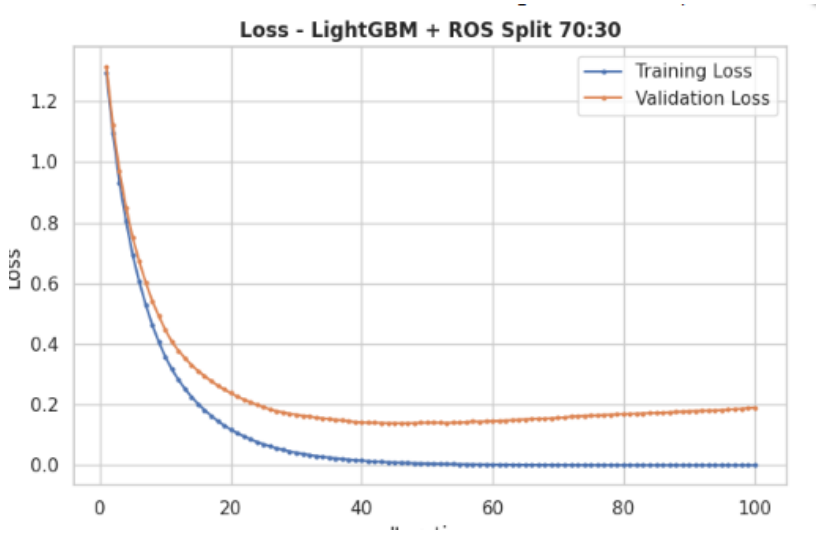



**Figure 4.24. Train Test Graph of LightGBM + ROS Split 70:30**

Figure 4.24 hows the training and validation loss graph of the LightGBM + ROS model using the 70:30 split ratio. The graph indicates that the model still had a fairly stable learning process, with no strong sign of overfitting because the gap between training and validation loss was not too large. However, the evaluation

metrics were lower than those of the 80:20 split. This shows that fewer training data reduced the model's ability to learn optimal classification patterns.

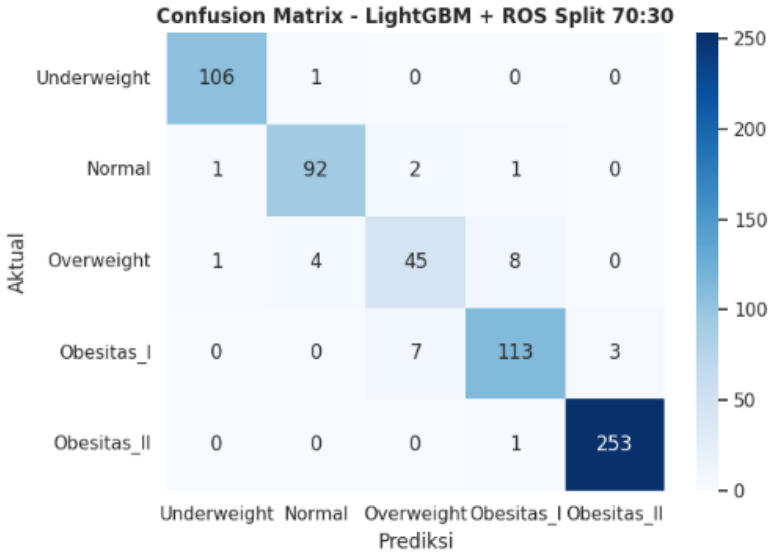



**Figure 4.25. Confusion Matrix of LightGBM + ROS Split 70:30**

Figure 4.25 shows the confusion matrix of the LightGBM + ROS model using the 70:30 split ratio. Most testing data were still classified correctly, as shown by the dominant values on the main diagonal. The Obesitas_II class had the most stable prediction results, while errors still occurred in the Overweight class because its characteristics were close to the Normal and Obesitas_I classes. Compared to the 80:20 split, the 70:30 split produced more prediction errors, indicating that fewer training data can affect the model's ability to distinguish similar classes.

```
Model              : LightGBM + ROS
Test Type          : Split Ratio Test
Split Used         : 70:30
Test Parameters    : Split Ratio
Split Ratio Used   : 70:30
Testing Accuracy   : 0.954545

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

   Underweight     0.9815    0.9907    0.9860       107
        Normal     0.9485    0.9583    0.9534        96
    Overweight     0.8333    0.7759    0.8036        58
     Obesity_I     0.9187    0.9187    0.9187       123
    Obesity_II     0.9883    0.9961    0.9922       254

      accuracy                         0.9545       638
     macro avg     0.9340    0.9279    0.9388       638
  weighted avg     0.9536    0.9545    0.9540       638
```

**Figure 4.26. Classification Report of LightGBM + ROS Split 70:30**

Based on Figure 4.26, the classification report of the LightGBM + ROS model using the 70:30 split ratio shows an accuracy of 0.954545, precision macro of 0.934050, recall macro of 0.927922, F1 macro of 0.930768, and balanced accuracy of 0.927922. These results indicate that the model still had good classification performance, although it decreased compared to the 80:20 split scenario. In terms of class level performance, classes with larger amounts of data tended to have more stable precision, recall, and F1 score values. In contrast, the Overweight class showed lower performance because its original data size was smaller. This indicates that although Random Oversampling balanced the training data, the amount and variation of the original data still affected the model's ability to learn the characteristics of minority classes.

**LightGBM + ROS Split 60:40**

In the third scenario, the dataset was divided using a 60:40 ratio, where 60% of the data were used as training data and 40% as testing data. In this split, the training data consisted of 1,275 records, while the testing data reached 850 records. After Random Oversampling was applied to the training data, the number of training data increased to 2,540 records. The 60:40 ratio had the largest amount of testing data compared to the previous two scenarios, providing a wider evaluation space. However, the amount of training data used to build the model was the smallest among all data split scenarios.

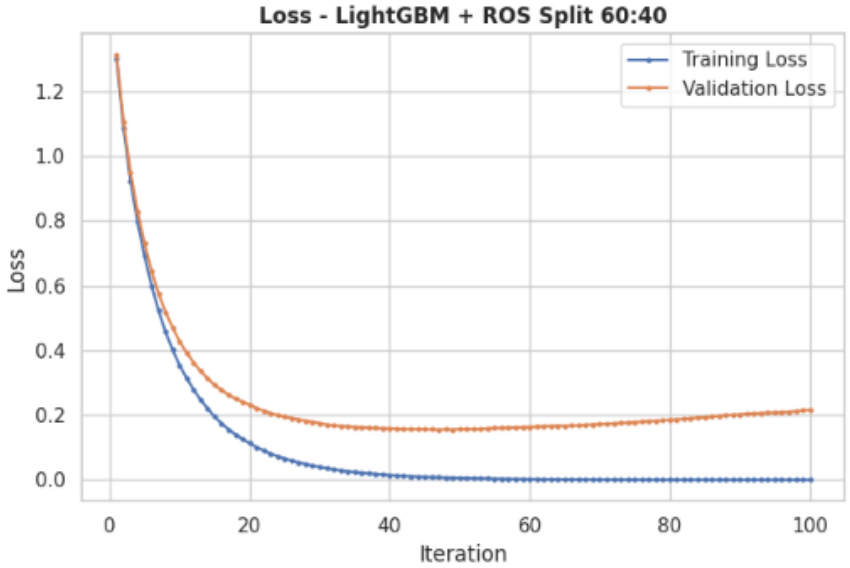



**Figure 4.27. Train Test Graph of LightGBM + ROS Split 60:40**

Figure 4.27 shows the training loss and validation loss graph of the LightGBM + ROS model using the 60:40 split ratio. This graph was used to observe model stability during the learning process. In this scenario, the model still showed the ability to learn from the training data. However, because the amount of training data was smaller, the model's ability to form strong classification patterns became more limited. If there is a gap between training loss and validation loss, this condition indicates that the model has not been able to maintain generalization performance as well as scenarios with larger training data.

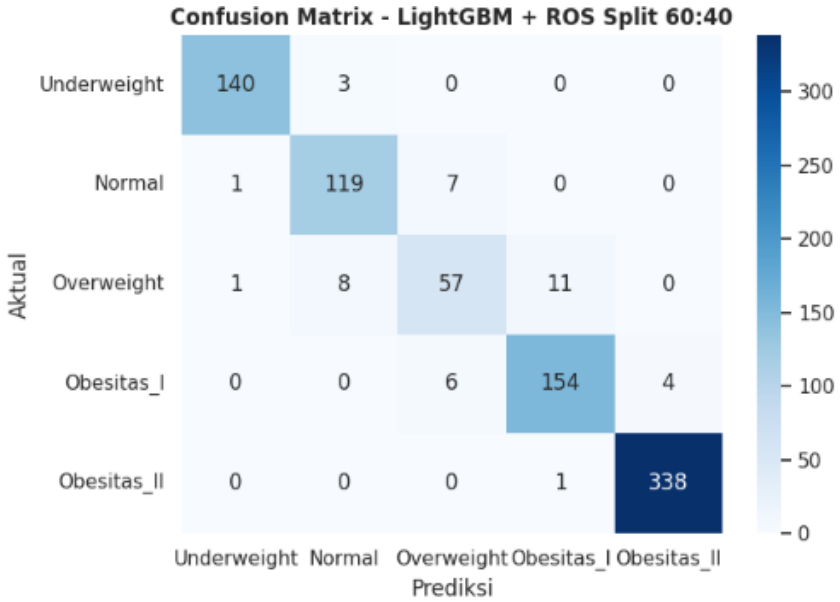



**Figure 4.28. Confusion Matrix of LightGBM + ROS Split 60:40**

Figure 4.28 shows the evaluation results of the LightGBM + ROS model using the 60:40 split ratio. Based on the confusion matrix, the model was still able to produce correct predictions for most testing data, as indicated by the dominant values on the main diagonal. However, compared to the 80:20 and 70:30 splits, classification errors in the 60:40 split were higher. These errors mainly occurred in classes with similar characteristics, especially the Overweight class, which was still often misclassified as Normal or Obesitas_I. This was caused by the smaller amount of original data in the Overweight class and its feature patterns that were located between those two classes. In addition, the reduced amount of training data in the 60:40 split made the information that could be learned by the model more limited, resulting in lower performance compared to the other two ratios.

```
Model              : LightGBM + ROS
Test Type          : Split Ratio Test
Split Used         : 60:40
Test Parameter     : Split Ratio
Split Ratio        : 60:40
Testing Accuracy   : 0.950588

=== Classification Report (Testing) ===

                    precision    recall  f1-score   support

     Underweight       0.9859    0.9790    0.9825       143
          Normal       0.9154    0.9370    0.9261       127
      Overweight       0.8143    0.7403    0.7755        77
   Obesity Type I      0.9277    0.9390    0.9333       164
  Obesity_Type_II      0.9883    0.9971    0.9927       339

        accuracy                           0.9586       850
       macro avg      0.9263    0.9185    0.9228       850
    weighted avg      0.9496    0.9506    0.9499       850
```

**Figure 4.29. Classification Report of LightGBM + ROS Split 60:40**

Based on Figure 4.29, the classification report shows that the LightGBM + ROS model with the 60:40 split achieved an accuracy of 0.950588, precision macro of 0.926320, recall macro of 0.918473, F1 macro of 0.922006, and balanced accuracy of 0.918473. Although the model still performed well, this result was the lowest compared to the 80:20 and 70:30 splits. The Overweight class had the lowest recall and F1 score because it had fewer data and characteristics close to other classes. This indicates that Overweight was the most difficult class to recognize in the obesity classification evaluation.

**Table 4.3. Comparison of LightGBM + ROS Split Ratios**

| Description | Split 80:20 | Split 70:30 | Split 60:40 |
|---|---|---|---|
| Accuracy | 0.971765 | 0.954545 | 0.950588 |
| Precision Macro | 0.960445 | 0.934050 | 0.926320 |
| Recall Macro | 0.946767 | 0.927922 | 0.918473 |
| F1 Macro | 0.952654 | 0.930768 | 0.922006 |
| Balanced Accuracy | 0.946767 | 0.927922 | 0.918473 |
| Train Time | 3.465019 | 0.691267 | 0.479419 |
| Status | Selected | Not Selected | Not Selected |

Based on Table 4.3, the 80:20 split ratio produced the best performance for the LightGBM + ROS model. This is shown by an accuracy of 0.971765, precision macro of 0.960445, recall macro of 0.946767, F1 macro of 0.952654, and balanced accuracy of 0.946767. These values were the highest compared to the 70:30 and 60:40 split ratios. In the 70:30 split, the model achieved an accuracy of 0.954545, F1 macro of 0.930768, and balanced accuracy of 0.927922. Meanwhile, in the 60:40 split, the model achieved an accuracy of 0.950588, F1 macro of 0.922006, and balanced accuracy of 0.918473. This comparison shows that model performance decreased when the portion of training data was reduced. Although the testing data became larger, the smaller amount of training data gave the model more limited information to learn the target class patterns. Therefore, the 80:20 split ratio was selected as the best data split ratio for the LightGBM + ROS model because it provided the best balance between the amount of training and testing data. The larger training data helped the model learn the patterns of each class more optimally, while 20% testing data remained sufficient to measure the model's generalization ability. Based on the F1 macro and balanced accuracy values, the 80:20 split also showed more balanced performance across all target classes.

### 4.2.3. Learning Rate Testing

Learning_rate testing is the first stage in stepwise manual hyperparameter testing. This parameter controls the contribution of each new tree in correcting the prediction errors of the previous model. A learning_rate value that is too small can make the learning process slower, while a larger value can accelerate learning but still needs to be monitored to avoid excessive fitting to the training data. At this stage, the tested learning_rate values were 0.01, 0.05, and 0.10 on the XGBoost + ROS and LightGBM + ROS models using the best split ratio of 80:20.

**XGBoost + ROS learning_rate 0,01**

In the first scenario, the XGBoost + ROS model was tested using a learning_rate value of 0.01. This value was the smallest in the testing range, so the model learning process became more gradual because each new tree only gave a small update. As a result, the model required more iterations to improve its performance.

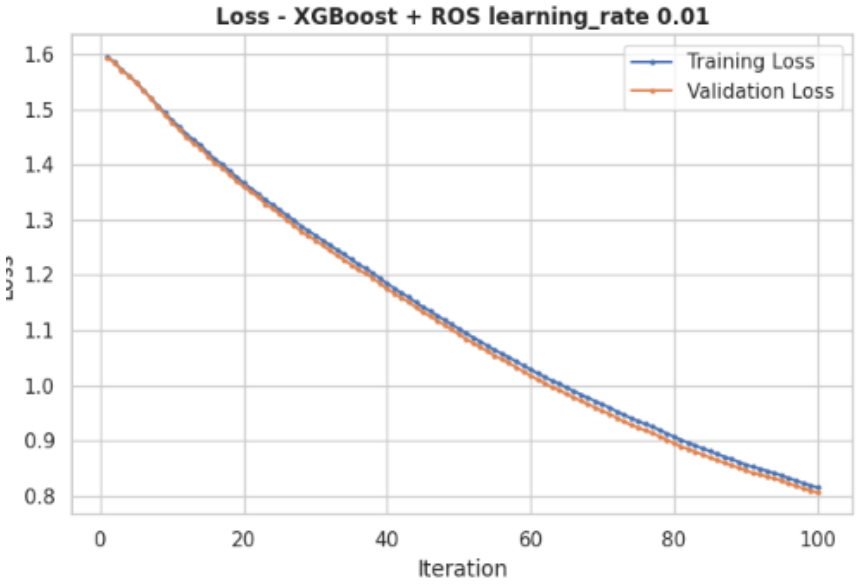



**Figure 4.30. Train Test Graph of XGBoost + ROS learning_rate 0.01**

Figure 4.30 shows the training loss and validation loss graph of the XGBoost + ROS model with learning_rate 0.01. A small learning_rate value caused the loss reduction to occur more slowly because model updates were performed gradually. This pattern indicates that the model was still learning stably, but had not yet reached optimal performance.

**Figure 4.31. Confusion Matrix of XGBoost + ROS learning_rate 0.01**

Based on Figure 4.31, the XGBoost + ROS model with learning_rate 0.01 still produced correct predictions, but the number of prediction errors was higher than those obtained using learning_rate 0.05 and 0.10. The main diagonal values

were not yet dominant, indicating that the model had not optimally recognized the target class patterns, especially in adjacent classes such as Overweight, Normal, and Obesitas_I.

```
Model              : XGBoost + ROS
Test Type          : Learning Rate Test
Split Used         : 80:20
Test Parameter     : Learning Rate
Learning Rate Used : 0.01
Testing Accuracy   : 0.917647

=== Classification Report (Testing) ===

                    precision    recall  f1-score   support

     Underweight       0.9859    0.9859    0.9859        71
          Normal       0.8507    0.8906    0.8702        64
      Overweight       0.6538    0.4474    0.5312        38
   Obesity_Type_I      0.8478    0.9512    0.8966        82
  Obesity_Type_II      0.9941    0.9882    0.9912       170

        accuracy                         0.9176       425
       macro avg       0.8665    0.8527    0.8550       425
    weighted avg       0.9125    0.9176    0.9127       425
```

**Figure 4.32. Classification Report of XGBoost + ROS learning_rate 0.01**

Figure 4.32 presents the classification report of the XGBoost + ROS model with learning_rate 0.01. Based on the evaluation results, the model achieved an accuracy of 0.917647, F1 macro of 0.855019, and balanced accuracy of 0.852673. These values indicate that the model performance was still lower than the other two learning_rate values. The low F1 macro and balanced accuracy values show that the model had not yet achieved balanced performance across all target classes. In terms of class level performance, classes with clearer patterns could still be recognized fairly well, while classes with adjacent characteristics tended to be more difficult to predict. The Overweight class became one of the classes that required attention because it had the potential to obtain lower recall and F1 score values than other classes. This condition indicates that learning_rate 0.01 was still too small to produce an optimal learning process for the XGBoost + ROS model.

**XGBoost + ROS learning_rate 0,05**

In the second scenario, the XGBoost + ROS model was tested using a learning_rate value of 0.05. This value was higher than learning_rate 0.01, allowing the model to perform learning updates with larger steps. This test was conducted to determine whether increasing the learning_rate value could improve model performance.

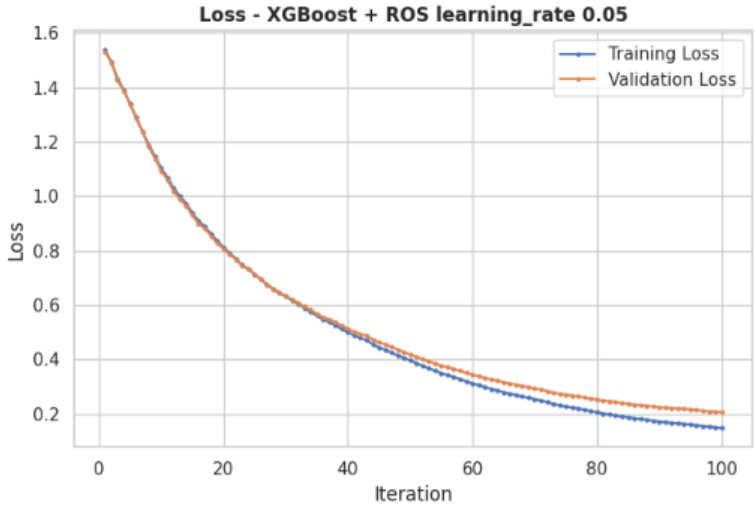



**Figure 4.33. Train Test Graph of XGBoost + ROS learning_rate 0.05**

Figure 4.33 shows the training loss and validation loss graph of the XGBoost + ROS model with learning_rate 0.05. Compared to learning_rate 0.01, the loss reduction appeared better because the model updates were more effective. The relatively small gap between training loss and validation loss indicates that the model still had fairly good generalization ability.

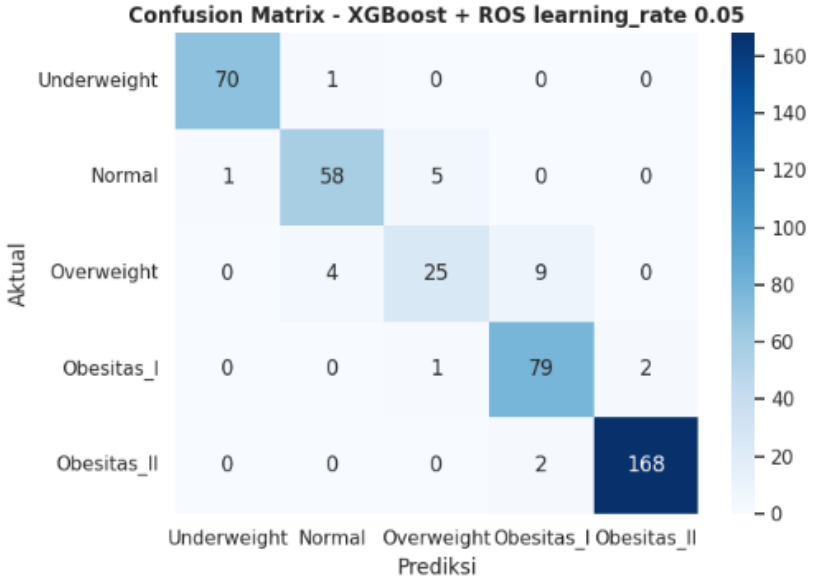



**Figure 4.34. Confusion Matrix of XGBoost + ROS learning_rate 0.05**

Based on Figure 4.34, the XGBoost + ROS model with learning_rate 0.05 performed better than learning_rate 0.01. The dominant main diagonal values show that the model correctly recognized more testing data. However, errors occurred in adjacent classes, especially Overweight, which overlaps with Normal and Obesitas_I.

```
Model                : XGBoost + ROS
Test Type            : Learning Rate Test
Split Used           : 80:20
Test Parameters      : Learning Rate
Learning Rate Used   : 0.05
Testing Accuracy     : 0.941176

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

   Underweight       0.9859    0.9859    0.9859        71
        Normal       0.9206    0.9062    0.9134        64
    Overweight       0.8065    0.6579    0.7246        38
     Obesity_I       0.8778    0.9634    0.9186        82
    Obesity_II       0.9882    0.9882    0.9882       170

      accuracy                         0.9412       425
     macro avg       0.9158    0.9003    0.9062       425
  weighted avg       0.9401    0.9412    0.9396       425
```

**Figure 4.35. Classification Report of XGBoost + ROS learning_rate 0.05**

Based on the classification report in Figure 4.35, the model achieved an accuracy of 0.941176, F1 macro of 0.906156, and balanced accuracy of 0.900342. These results improved compared to learning_rate 0.01, showing better overall prediction and class performance. However, learning_rate 0.05 was still not the best value, because the model achieved more optimal results when the learning_rate was increased to 0.10.

**XGBoost + ROS learning_rate 0,10**

In the third scenario, the XGBoost + ROS model was tested using a learning_rate value of 0.10. This value was the largest in the learning_rate testing range used in this study. This test aimed to determine whether a higher learning_rate could produce better performance than the values 0.01 and 0.05.

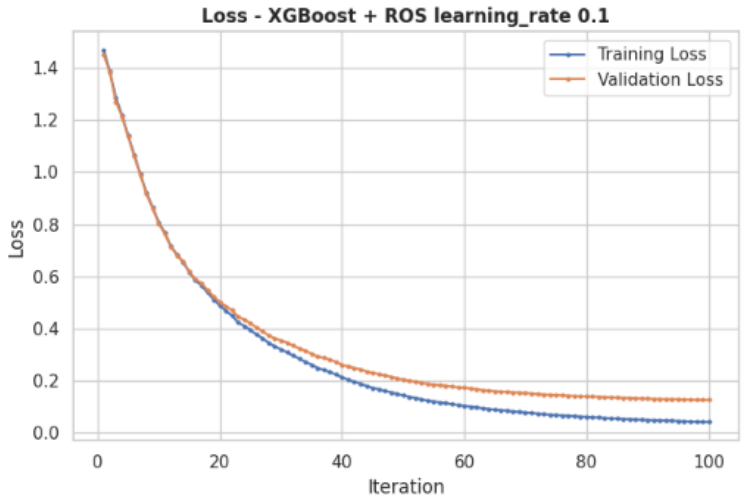



**Figure 4.36. Train Test Graph of XGBoost + ROS learning_rate 0.10**

Figure 4.36 shows the training loss and validation loss graph of the XGBoost + ROS model with learning_rate 0.10. The graph shows a stable decrease in loss, with a gap between training loss and validation loss that was not too large. Since it produced the best metrics, learning_rate 0.10 was used as the fixed parameter in the next testing stage, namely max_depth.

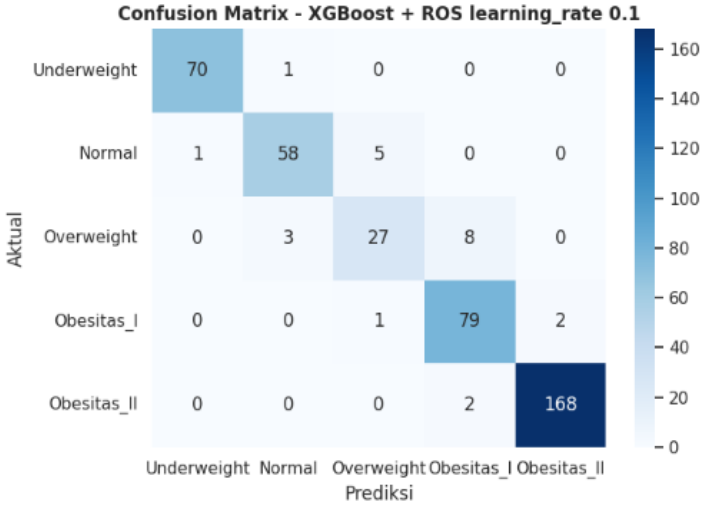



**Figure 4.37. Confusion Matrix of XGBoost + ROS learning_rate 0.10**

Based on Figure 4.37, the XGBoost + ROS model with learning_rate 0.10 achieved the best performance among the tested scenarios. The dominant main diagonal values indicate that more testing data were classified correctly. Although errors still occurred in the Overweight class, learning_rate 0.10 gave the best overall classification result.

```
Model              : XGBoost + ROS
Test Type          : Learning Rate Test
Split Used         : 80:20
Test Parameter     : Learning Rate
Learning Rate Used : 0.1
Testing Accuracy   : 0.945882

===  Classification Report (Testing) ===

                    precision    recall    f1-score    support

     Underweight         0.9859    0.9859    0.9859         71
          Normal         0.9355    0.9062    0.9206         64
      Overweight         0.8182    0.7165    0.7666         38
  Obesity_Type_I         0.8876    0.9634    0.9240         82
 Obesity_Type_II         0.9882    0.9882    0.9882        170

        accuracy                             0.9459        425
       macro avg         0.9231    0.9189    0.9159        425
    weighted avg         0.9453    0.9459    0.9449        425
```

**Figure 4.38. Classification Report of XGBoost + ROS learning_rate 0.10**

Based on the classification report in Figure 4.38, the model achieved an accuracy of 0.945882, F1 macro of 0.915865, and balanced accuracy of 0.910868. These values were the highest compared to learning_rate 0.01 and 0.05 in the XGBoost + ROS model, indicating that learning_rate 0.10 provided a more effective learning process. The higher F1 macro value shows that the balance between precision and recall across classes improved, while the increase in balanced accuracy indicates that the model was able to recognize the target classes more evenly. Therefore, learning_rate 0.10 was selected as the best value in the learning_rate testing stage for the XGBoost + ROS model.

**LightGBM + ROS learning_rate 0,01**

In the fourth scenario, the LightGBM + ROS model was tested using a learning_rate value of 0.01. Similar to XGBoost, this value was the smallest in the testing range, so the learning process was performed more slowly and gradually.

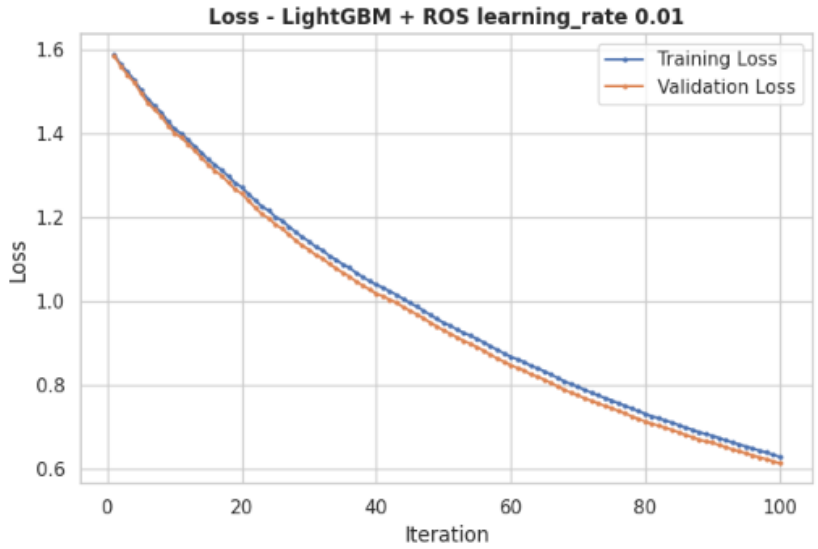



**Figure 4.39. Train Test Graph of LightGBM + ROS learning_rate 0.01**

Figure 4.39 shows the training loss and validation loss graph of the LightGBM + ROS model with learning_rate 0.01. At a small learning_rate value, the loss reduction tended to occur more slowly. The model still showed a learning process, but the resulting performance was not yet optimal. If the training loss and validation loss curves still decrease slowly, the model still requires more effective updates to achieve higher performance. Therefore, learning_rate 0.01 was not selected as the best configuration.

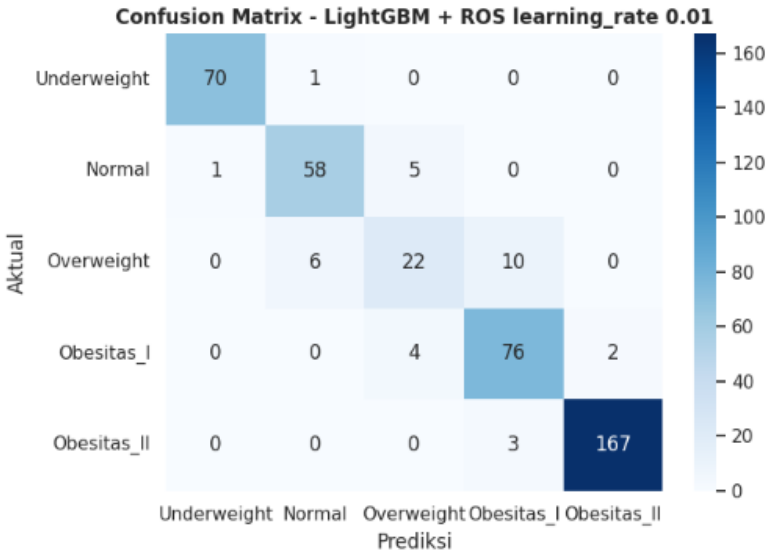



**Figure 4.40. Confusion Matrix of LightGBM + ROS learning_rate 0.01**

Based on Figure 4.40, the LightGBM + ROS model with learning_rate 0.01 showed performance that was not yet optimal compared to learning_rate 0.05 and 0.10. In the confusion matrix, correct predictions were still found in several classes, but classification errors were higher and the main diagonal values were not yet highly dominant. Errors mainly occurred in classes with adjacent characteristics, such as Overweight with Normal or Obesitas_I.

```
Model               : LightGBM + ROS
Test Type           : Learning Rate Test
Split Used          : 80:20
Test Parameters     : Learning Rate
Learning Rate Used  : 0.01
Testing Accuracy    : 0.924786

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

   Underweight       0.9859    0.9859    0.9859        71
        Normal       0.8923    0.9062    0.8992        64
    Overweight       0.7097    0.5789    0.6377        38
     Obesity_I       0.8539    0.9268    0.8889        82
    Obesity_II       0.9882    0.9824    0.9853       170

      accuracy                         0.9247       425
     macro avg       0.8860    0.8761    0.8794       425
  weighted avg       0.9226    0.9247    0.9227       425
```

**Figure 4.41. Classification Report of LightGBM + ROS learning_rate 0.01**

Based on the classification report in Figure 4.41, the model achieved an accuracy of 0.924706, F1 macro of 0.879392, and balanced accuracy of 0.876059. These values indicate that the model performance was still lower than those

obtained using learning_rate 0.05 and 0.10. Thus, learning_rate 0.01 was not sufficiently optimal in helping LightGBM learn obesity level classification patterns with the number of iterations used. In terms of class level metrics, classes with larger amounts of data tended to have more stable performance, while classes with adjacent characteristics still had the potential to show lower performance. The F1 macro value, which was lower than the other two scenarios, indicates that balanced performance across classes had not yet been achieved optimally at learning_rate 0.01.

**LightGBM + ROS learning_rate 0,05**

In the fifth scenario, the LightGBM + ROS model was tested using a learning_rate value of 0.05. This value was used to observe the improvement in model performance when the learning step was increased compared to learning_rate 0.01. A higher learning_rate allows the model to update its learning process more strongly in each iteration. Therefore, this scenario was important to determine whether a moderate learning_rate could produce better classification results.

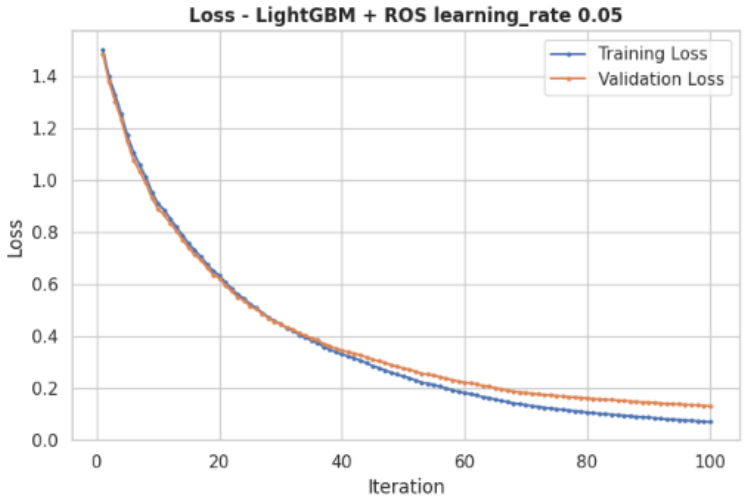



**Figure 4.42. Train Test Graph of LightGBM + ROS learning_rate 0.05**

Figure 4.42 shows the training loss and validation loss graph of the LightGBM + ROS model with learning_rate 0.05. The graph indicates that the model experienced a better learning process compared to learning_rate 0.01. If both loss curves decrease with a gap that is not too large, the model can be considered stable and does not show an indication of excessive overfitting. However, based on the evaluation metrics, the performance of learning_rate 0.05 still did not exceed learning_rate 0.10.

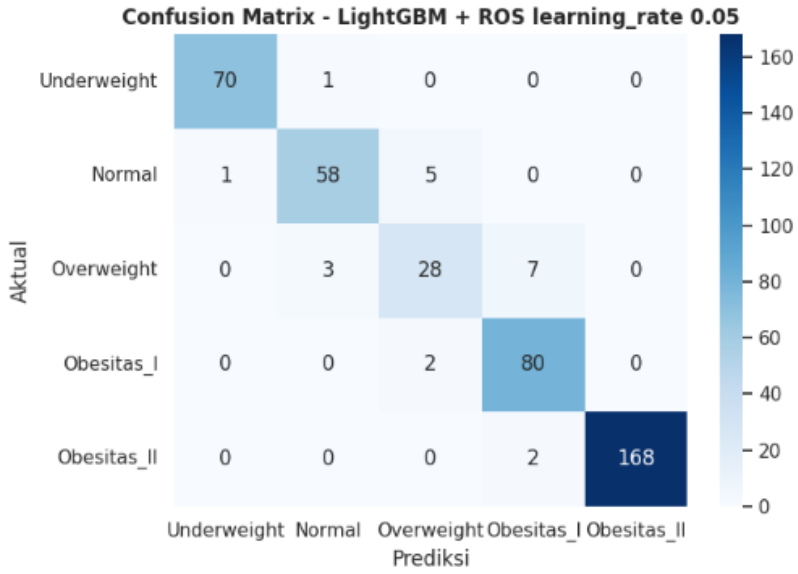



**Figure 4.43. Confusion Matrix of LightGBM + ROS learning_rate 0.05**

Based on Figure 4.43, the LightGBM + ROS model with learning_rate 0.05 showed improved performance compared to the learning_rate 0.01 scenario. In the confusion matrix, the number of correct predictions on the main diagonal appeared more dominant, indicating that the model was able to recognize more testing data according to their actual classes. However, classification errors still appeared in several classes, especially classes with adjacent data patterns. The Overweight class remained one of the classes that required attention because it tended to be more difficult to clearly separate from Normal and Obesitas_I.

```
Model              : LightGBM + ROS
Type of Test       : Learning Rate Test
Split Ratio Used   : 80:20
Test Parameters    : learning_rate
Ratio Used         : 0.05
Testing Accuracy   : 0.950588

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

   Underweight     0.9859    0.9859    0.9859        71
        Normal     0.9355    0.9062    0.9206        64
    Overweight     0.8000    0.7368    0.7671        38
     Obesity_I     0.8989    0.9756    0.9357        82
    Obesity_II     1.0000    0.9882    0.9941       170

      accuracy                         0.9586       425
     macro avg     0.9241    0.9186    0.9287       425
  weighted avg     0.9505    0.9506    0.9581       425
```

**Figure 4.44. Classification Report of LightGBM + ROS learning_rate 0.05**

Based on the classification report in Figure 4.44, the model achieved an accuracy of 0.950588, F1 macro of 0.920686, and balanced accuracy of 0.918571. These values show an improvement in performance compared to learning_rate 0.01, indicating that learning_rate 0.05 was more effective in helping the model learn data patterns. However, learning_rate 0.05 was not the best value for the LightGBM + ROS model because its accuracy, F1 macro, and balanced accuracy were still lower than those of learning_rate 0.10. Therefore, learning_rate 0.05 showed good performance, but did not provide the most optimal results in the learning_rate testing stage.

**LightGBM + ROS learning_rate 0,10**

In the sixth scenario, the LightGBM + ROS model was tested using a learning_rate value of 0.10. This value was the largest in the learning_rate testing range used in this study.

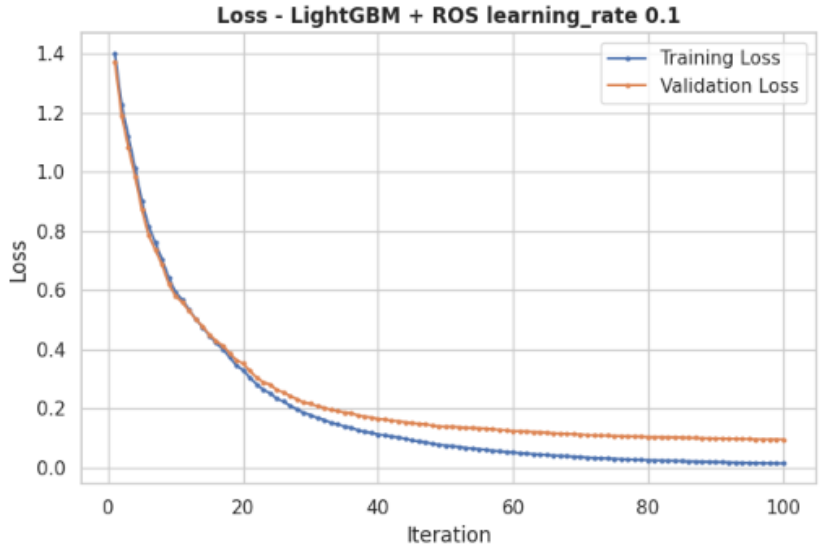



**Figure 4.45. Train Test Graph of LightGBM + ROS learning_rate 0.10**

Figure 4.45 shows the training loss and validation loss graph of the LightGBM + ROS model with learning_rate 0.10. The graph shows that the learning process was more effective than those obtained using smaller learning_rate values. If both training loss and validation loss decrease and do not show a large gap, the model can be considered to have good learning stability. Based on the evaluation metrics, learning_rate 0.10 became the best configuration and was fixed for the next testing stage, namely max_depth.

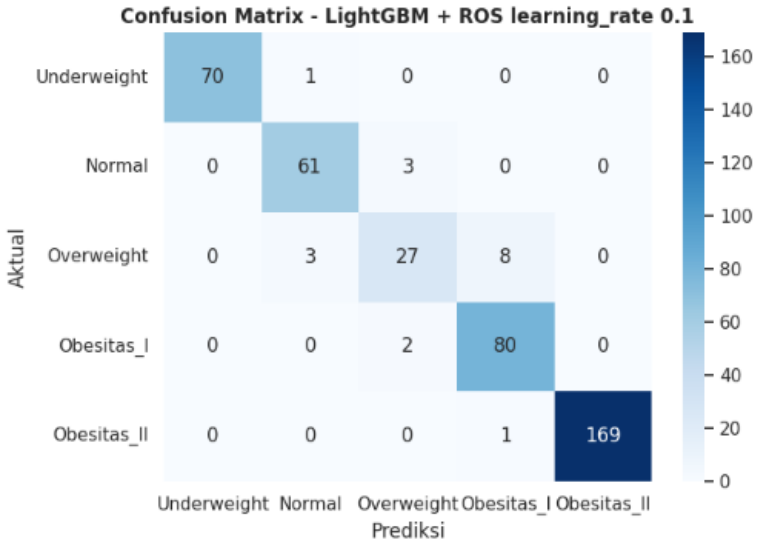



**Figure 4.46. Confusion Matrix of LightGBM + ROS learning_rate 0.10**

Based on Figure 4.46, the LightGBM + ROS model with learning_rate 0.10 showed the best performance compared to the two previous scenarios. The main diagonal values in the confusion matrix appeared more dominant, indicating that the model produced more correct predictions on the testing data. Prediction errors could still occur in the Overweight class because its characteristics were close to Normal and Obesitas_I. Overall, learning_rate 0.10 enabled LightGBM + ROS to learn the target patterns more effectively and stably.

```
Model                : LightGBM + ROS
Type of Test         : Learning Rate Test
Split Used           : 80:20
Parameter Tested     : learning_rate
Value Used           : 0.1
Testing Accuracy     : 0.957647

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

 Underweight       1.0000    0.9859    0.9929        71
      Normal       0.9385    0.9531    0.9457        64
  Overweight       0.8438    0.7105    0.7714        38
   Obesity_I       0.8989    0.9756    0.9357        82
  Obesity_II       1.0000    0.9941    0.9971       170

    accuracy                         0.9576       425
   macro avg       0.9362    0.9239    0.9286       425
weighted avg       0.9573    0.9576    0.9566       425
```

**Figure 4.47. Classification Report of LightGBM + ROS learning_rate 0.10**

Based on the classification report in Figure 4.47, the model achieved an accuracy of 0.957647, F1 macro of 0.928559, and balanced accuracy of 0.923859.

These values were the highest in the learning_rate testing stage for the LightGBM + ROS model, indicating that learning_rate 0.10 provided the best performance compared to learning_rate 0.01 and 0.05. The higher F1 macro value shows that the balance between precision and recall across all classes improved, while the increase in balanced accuracy indicates that the model was able to maintain more even performance across each target class. Therefore, learning_rate 0.10 was selected as the best value for the LightGBM + ROS model.

**Table 4.4. Comparison of Learning Rate on XGBoost**

| Description | XGBoost + ROS 0.01 | XGBoost + ROS 0.05 | XGBoost + ROS 0.10 |
|---|---|---|---|
| Accuracy | 0.917647 | 0.941176 | 0.945882 |
| Precision Macro | 0.866483 | 0.915803 | 0.923091 |
| Recall Macro | 0.852673 | 0.900342 | 0.910868 |
| F1 Macro | 0.855019 | 0.906156 | 0.915865 |
| Balanced Accuracy | 0.852673 | 0.900342 | 0.910868 |
| Train Time | 0.493518 | 0.427671 | 0.416708 |
| Status | Not Selected | Not Selected | Selected |

**Table 4.5. Comparison of Learning Rate on LightGBM**

| Description | LightGBM + ROS 0.01 | LightGBM + ROS 0.05 | LightGBM + ROS 0.10 |
|---|---|---|---|
| Accuracy | 0.924706 | 0.950588 | 0.957647 |
| Precision Macro | 0.886000 | 0.924055 | 0.936218 |
| Recall Macro | 0.876059 | 0.918571 | 0.923859 |
| F1 Macro | 0.879392 | 0.920686 | 0.928559 |
| Balanced Accuracy | 0.876059 | 0.918571 | 0.923859 |
| Train Time | 0.439324 | 0.385521 | 0.428518 |
| Status | Not Selected | Not Selected | Selected |

Based on Tables 4.4 and 4.5, the evaluation results show that learning_rate 0.10 consistently produced the most optimal performance for both models within the tested range. In the XGBoost + ROS model, this parameter setting resulted in an accuracy of 0.945882, an F1 macro of 0.915865, and a balanced accuracy of 0.910868. These results exceeded the performance obtained using learning_rate 0.05, where the model recorded an accuracy of 0.941176, an F1 macro of 0.906156, and a balanced accuracy of 0.900342. A lower performance was observed at learning_rate 0.01, with an accuracy of 0.917647, an F1 macro of 0.855019, and a balanced accuracy of 0.852673.

A similar trend was also found in the LightGBM + ROS model. The use of learning_rate 0.10 generated the highest performance, with an accuracy of 0.957647, an F1 macro of 0.928559, and a balanced accuracy of 0.923859. Meanwhile, learning_rate 0.05 produced slightly lower results, with an accuracy of 0.950588, an F1 macro of 0.920686, and a balanced accuracy of 0.918571. The lowest performance was obtained at learning_rate 0.01, where the model achieved an accuracy of 0.924706, an F1 macro of 0.879392, and a balanced accuracy of 0.876059.

These findings indicate that a very small learning_rate value did not provide the most effective learning process for either model. At learning_rate 0.01, model updates were relatively limited, which contributed to lower classification performance. When the value was increased to 0.05, both models showed improvement, and the strongest results were achieved at learning_rate 0.10. Therefore, learning_rate 0.10 was selected as the best parameter value for both XGBoost + ROS and LightGBM + ROS. This value was then used as the fixed parameter in the next hyperparameter testing stage, namely max_depth, because it produced the highest accuracy, F1 macro, and balanced accuracy compared with the other tested values.

### 4.2.4. Max Depth Testing

Max_depth testing represents the second stage of the stepwise manual hyperparameter testing process after the optimal learning_rate value had been determined. The max_depth parameter defines how deep the decision trees are allowed to grow during the learning process. A larger max_depth value enables the

model to capture more complex data patterns; however, an excessively deep tree may increase the possibility of overfitting because the model can become too adjusted to the training data. In this stage, three max_depth values, namely 3, 5, and 7, were tested on both the XGBoost + ROS and LightGBM + ROS models. The testing was conducted using the previously selected 80:20 split ratio. The learning_rate parameter was kept constant at 0.10, while the remaining parameters followed the initial configuration, consisting of n_estimators = 100, subsample = 0.80, and colsample_bytree = 0.80. Each testing scenario was assessed using several evaluation outputs, including the confusion matrix, classification report, training loss graph, and validation loss graph. In addition, model performance was compared based on accuracy, precision macro, recall macro, F1 macro, and balanced accuracy.

**XGBoost + ROS max_depth 3**

In the first scenario, the XGBoost + ROS model was tested using max_depth 3. This value was the smallest tree depth in the testing range, resulting in a simpler tree structure. Limited depth can reduce the risk of overfitting, but it may also limit the model's ability to learn complex feature relationships. Therefore, the classification performance in this scenario had the potential to be less optimal if the data patterns were sufficiently diverse.

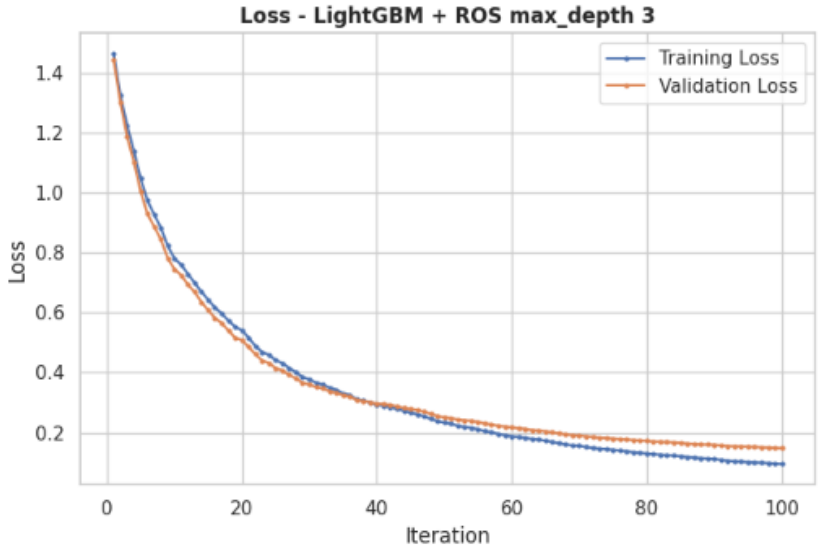



**Figure 4.48. Train Test Graph of XGBoost + ROS max_depth 3**

Figure 4.48 shows the training loss and validation loss graph of the XGBoost + ROS model with max_depth 3. A lower max_depth value can reduce the risk of

overfitting, but it also limits the model's ability to learn complex patterns. Max_depth 3 did not provide the best performance.

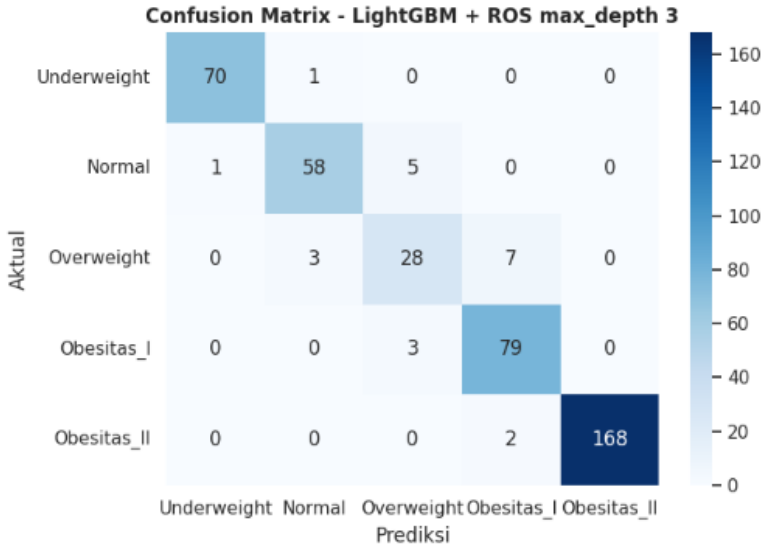



**Figure 4.49. Confusion Matrix of XGBoost + ROS max_depth 3**

Based on Figure 4.49, the XGBoost + ROS model with max_depth 3 still showed good performance, but it was not optimal compared to max_depth 5 and 7. The main diagonal values were not yet highly dominant, indicating that the model had not fully captured the classification patterns optimally. Errors still occurred in adjacent classes, especially Overweight, which had the potential to be misclassified as Normal or Obesitas_I.

```
Model               : XGBoost + ROS
Test Type           : Maximum Depth Test
Split Used          : 80:20
Test Parameter      : Maximum Depth
Max Depth Used      : 3
Testing Accuracy    : 0.936471

=== Classification Report (Testing) ===

                    precision    recall  f1-score   support

      Underweight       0.9859    0.9859    0.9859        71
           Normal       0.9206    0.9062    0.9134        64
       Overweight       0.7812    0.6579    0.7143        38
   Obesity_Type_I       0.8587    0.9634    0.9088        82
  Obesity_Type_II       0.9940    0.9765    0.9852       170

         accuracy                         0.9365       425
        macro avg       0.9081    0.8980    0.9014       425
     weighted avg       0.9365    0.9365    0.9354       425
```

**Figure 4.50. Classification Report of XGBoost + ROS max_depth 3**

Based on the classification report in Figure 4.50, the model achieved an accuracy of 0.936471, precision macro of 0.908102, recall macro of 0.897989, F1 macro of 0.901359, and balanced accuracy of 0.897989. These values indicate that the model still had good performance, but it was not the best result in the max_depth testing stage. The lower F1 macro and balanced accuracy values compared to max_depth 5 and 7 indicate that balanced performance across classes had not yet been achieved optimally. This condition suggests that the model still had difficulty distinguishing several classes with adjacent characteristics. This may occur because a tree structure that is too shallow is not strong enough to separate data patterns in those classes. Therefore, max_depth 3 was not selected as the best configuration for the XGBoost + ROS model.

**XGBoost + ROS max_depth 5**

In the second scenario, the XGBoost + ROS model was tested using max_depth 5. This value was the initial value used before max_depth testing was conducted. A tree depth of 5 provides a more complex model structure compared to max_depth 3, allowing the model to have greater ability to learn data patterns.

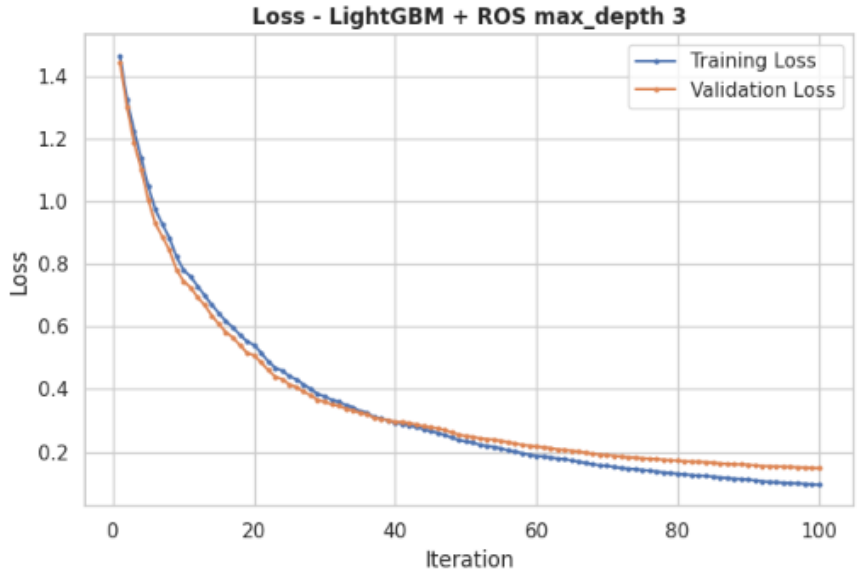



**Figure 4.51. Train Test Graph of XGBoost + ROS max_depth 5**

Figure 4.51 shows the training loss and validation loss graph of the XGBoost + ROS model with max_depth 5. This graph was used to observe the stability of model performance during training. In this scenario, the model showed good

stability and was able to learn data patterns more optimally than max_depth 3. Thus, max_depth 5 provided improved performance compared to the lower tree depth.

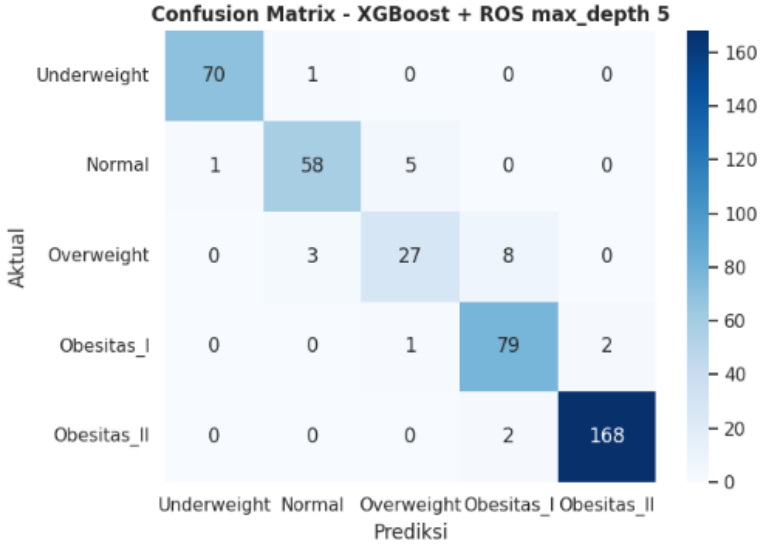



**Figure 4.52. Confusion Matrix of XGBoost + ROS max_depth 5**

Based on Figure 4.52, the XGBoost + ROS model with max_depth 5 showed better correct prediction values on the main diagonal compared to max_depth 3. This indicates that increasing the tree depth helped the model recognize more testing data according to their actual classes. However, prediction errors were still found in adjacent classes, especially Overweight, which had the potential to be misclassified as Normal and Obesitas_I.

```
Model              : XGBoost + ROS
Test Type          : Max Depth Test
Split Used         : 80:20
Test Parameters    : Max Depth
Max Depth Used     : 5
Testing Accuracy   : 0.945882

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

    Underweight     0.9859    0.9859    0.9859        71
         Normal     0.9355    0.9062    0.9286        64
     Overweight     0.8182    0.7105    0.7686        38
      Obesity_I     0.8876    0.9634    0.9240        82
     Obesity_II     0.9882    0.9882    0.9882       170

       accuracy                         0.9459       425
      macro avg     0.9231    0.9109    0.9159       425
   weighted avg     0.9453    0.9459    0.9449       425
```

**Figure 4.53. Classification Report of XGBoost + ROS max_depth 5**

Based on the classification report in Figure 4.53, the model achieved an accuracy of 0.945882, precision macro of 0.923091, recall macro of 0.910868, F1 macro of 0.915865, and balanced accuracy of 0.910868. These values show an improvement in performance compared to max_depth 3. The increase in accuracy indicates that the overall number of correct predictions increased, while the improvements in F1 macro and balanced accuracy show that the model began to provide more balanced performance across all target classes. This result indicates that increasing the tree depth gave the model better ability to learn data patterns and distinguish characteristics between classes. However, the performance of max_depth 5 was still lower than max_depth 7, so this configuration was not selected as the best in the max_depth testing stage for the XGBoost + ROS model.

**XGBoost + ROS max_depth 7**

In the third scenario, the XGBoost + ROS model was tested using max_depth 7. This value was the largest tree depth in the max_depth testing range. With a deeper tree structure, the model had greater ability to capture complex relationships among features. This condition allowed the model to learn data patterns in more detail, so the classification process could be performed more effectively.

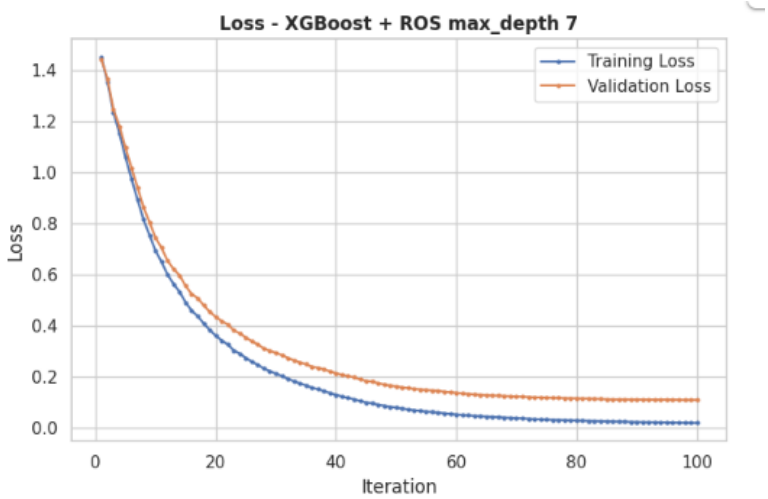



**Figure 4.54. Train Test Graph of XGBoost + ROS max_depth 7**

Figure 4.54 shows the training loss and validation loss graph of the XGBoost + ROS model with max_depth 7. This graph was used to observe learning stability when model complexity increased. In this scenario, the model obtained the best evaluation metrics, indicating that max_depth 7 helped the model learn data patterns

better without showing excessive overfitting. Therefore, max_depth 7 was used as the fixed parameter in the next testing stage, namely n_estimators.

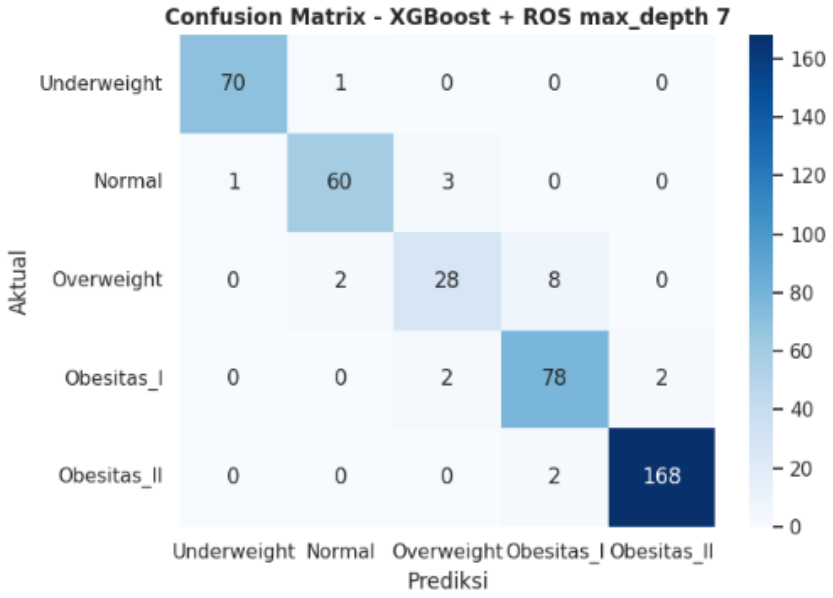



**Figure 4.55. Confusion Matrix of XGBoost + ROS max_depth 7**

Figure 4.55 shows the confusion matrix of the XGBoost + ROS model with max_depth 7. The values on the main diagonal appeared more dominant compared to max_depth 3 and 5, indicating that the model produced more correct predictions on the testing data. Although errors still occurred in the Overweight class, overall, the model showed better classification performance.

```
Model               : XGBoost + ROS
Type of Test        : Maximum Depth Test
Split Ratio Used    : 80:20
Test Parameters     : max_depth
Value Used          : 7
Testing Accuracy    : 0.950588

=== Classification Report (Testing) ===

                  precision     recall    f1-score    support

   Underweight        0.9859     0.9859      0.9859         71
        Normal        0.9524     0.9375      0.9449         64
    Overweight        0.8485     0.7368      0.7887         38
     Obesity_I        0.8864     0.9512      0.9176         82
    Obesity_II        0.9882     0.9882      0.9882        170

      accuracy                             0.9586        425
     macro avg        0.9323     0.9199      0.9251        425
  weighted avg        0.9503     0.9506      0.9499        425
```

**Figure 4.56. Classification Report of XGBoost + ROS max_depth 7**

Figure 4.56 presents the classification report of the XGBoost + ROS model with max_depth 7. Based on the evaluation results, the model achieved an accuracy of 0.950588, precision macro of 0.932276, recall macro of 0.919942, F1 macro of 0.925082, and balanced accuracy of 0.919942. These values were the highest compared to max_depth 3 and 5 in the XGBoost + ROS model. The higher F1 macro value indicates that the balance between precision and recall across all target classes improved. The increased balanced accuracy also shows that the model was able to recognize the target classes more evenly. Therefore, max_depth 7 was selected as the best value in the max_depth testing stage for the XGBoost + ROS model.

**LightGBM + ROS max_depth 3**

In the fourth scenario, the LightGBM + ROS model was tested using max_depth 3 as the lowest tree depth. This value produced a simpler model structure, which could reduce the risk of overfitting, but it also limited the model's ability to learn complex feature relationships. Therefore, max_depth 3 had the potential to produce less optimal classification performance across all target classes.

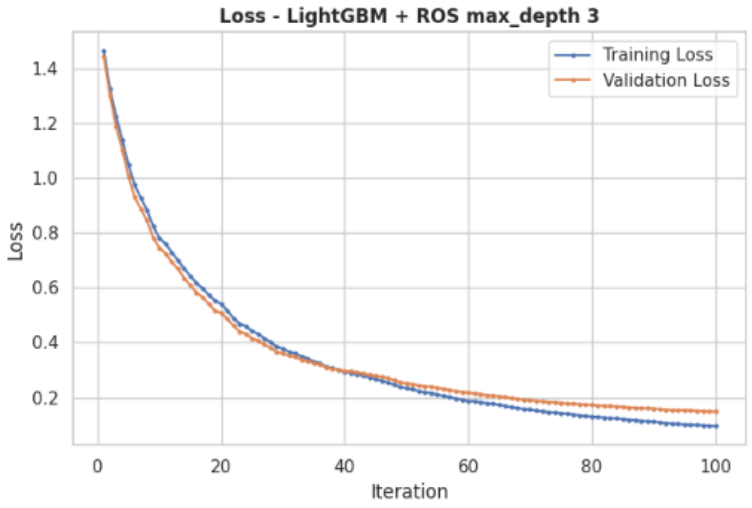



**Figure 4.57. Train Test Graph of LightGBM + ROS max_depth 3**

Figure 4.57 shows the training loss and validation loss graph of the LightGBM + ROS model with max_depth 3. This graph was used to observe performance stability during training. At a low tree depth, the model tended to be more stable because its complexity was limited. However, the performance was not yet optimal, indicating that the model still required a deeper tree structure to learn complex data patterns. Therefore, max_depth 3 was not the best configuration at this stage.

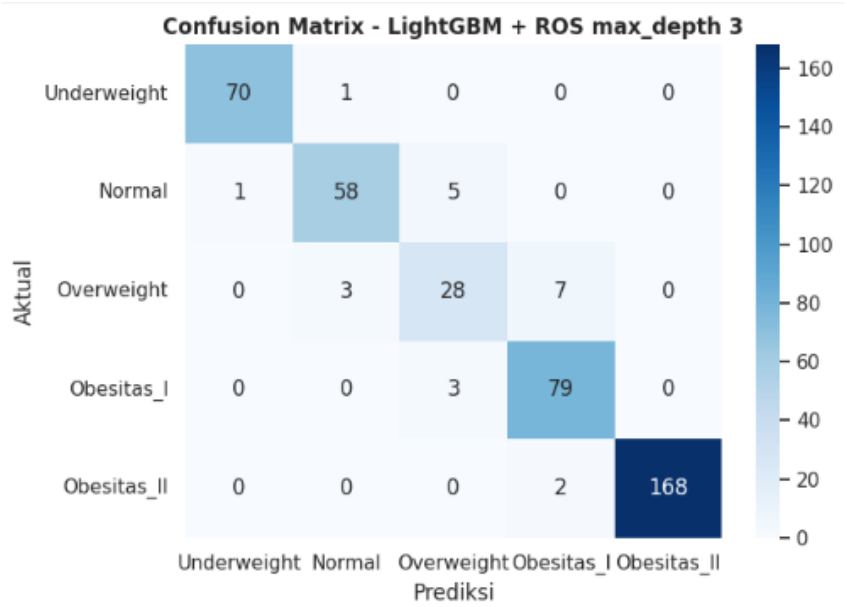



**Figure 4.58. Confusion Matrix of LightGBM + ROS max_depth 3**

Based on Figure 4.58, the LightGBM + ROS model with max_depth 3 already showed fairly good classification performance, but it was not the best compared to higher max_depth values. The main diagonal values indicate that the model was able to recognize most classes, although prediction errors still occurred in several classes. The Overweight class remained relatively difficult to predict because its characteristics were close to Normal and Obesitas_I.

```
Model              : LightGBM + ROS
Type of Test       : Maximum Depth Test
Split Used         : 80:20
Parameter Tested   : max_depth
Value Used         : 3
Testing Accuracy   : 0.948235

=== Classification Report (Testing) ===

                precision    recall    f1-score   support

  Underweight       0.9859    0.9859      0.9859        71
       Normal       0.9355    0.9062      0.9286        64
   Overweight       0.7778    0.7368      0.7568        38
    Obesity_I       0.8977    0.9634      0.9294        82
   Obesity_II       1.0000    0.9882      0.9941       170

     accuracy                           0.9482       425
    macro avg       0.9194    0.9161      0.9174       425
 weighted avg       0.9483    0.9482      0.9480       425
```

**Figure 4.59. Classification Report of LightGBM + ROS max_depth 3**

Based on the classification report in Figure 4.59, the model achieved an accuracy of 0.948235, precision macro of 0.919381, recall macro of 0.916132, F1 macro of 0.917360, and balanced accuracy of 0.916132. These values indicate that LightGBM + ROS with max_depth 3 already had fairly good performance, but it was not the best result in the max_depth testing stage. The lower F1 macro and balanced accuracy values compared to max_depth 5 and 7 indicate that model performance could still be improved by using a deeper tree structure. Therefore, max_depth 3 was not selected as the best configuration for the LightGBM + ROS model.

**LightGBM + ROS max_depth 5**

In the fifth scenario, the LightGBM + ROS model was tested using max_depth 5. This value provides a more complex tree structure compared to max_depth 3, giving the model a greater opportunity to recognize more complex relationships among features. With a higher tree depth, the model was expected to learn data patterns in more detail without increasing model complexity excessively.

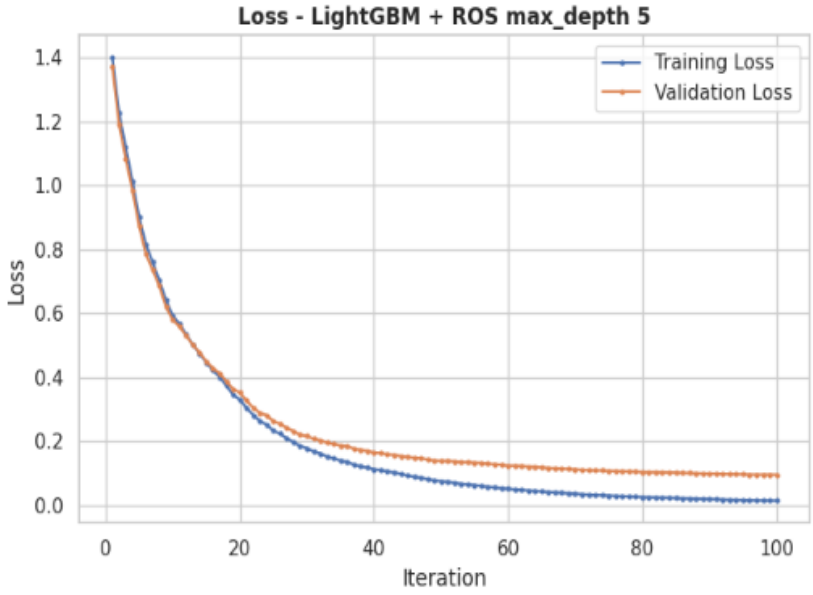



**Figure 4.60. Train Test Graph of LightGBM + ROS max_depth 5**

Figure 4.60 shows the training loss and validation loss graph of the LightGBM + ROS model with max_depth 5. The graph shows that the model was able to learn data patterns better than max_depth 3. If the training loss and validation loss curves decrease consistently and do not have a large gap, the model can be

considered stable. However, based on the evaluation metrics, the performance of max_depth 5 still did not exceed max_depth 7.

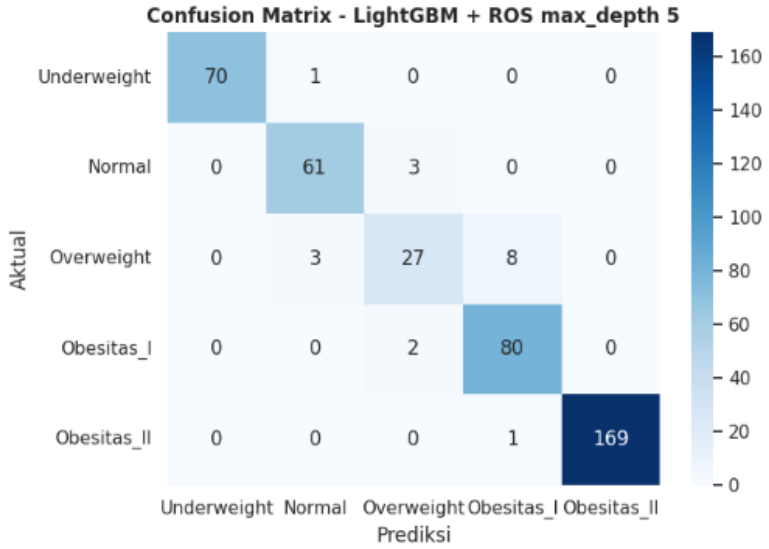



**Figure 4.61. Confusion Matrix of LightGBM + ROS max_depth 5**

Figure 4.61 shows the confusion matrix results of the LightGBM + ROS model with max_depth 5. Based on the confusion matrix, the number of correct predictions on the main diagonal appeared better than max_depth 3, indicating that increasing the tree depth helped the model classify testing data more accurately. However, several classification errors still appeared, especially in classes with adjacent characteristic boundaries.

```
Model              : LightGBM + ROS
Type of Test       : max_depth Test
Split Used         : 80:20
Test Parameter     : max_depth
Value Used         : 5
Testing Accuracy   : 0.957647

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

 Underweight       1.0000    0.9859    0.9929        71
      Normal       0.9385    0.9531    0.9457        64
  Overweight       0.8438    0.7105    0.7714        38
  Obesitys_I       0.8989    0.9756    0.9357        82
 Obesitys_II       1.0000    0.9941    0.9971       170

    accuracy                         0.9576       425
   macro avg       0.9362    0.9239    0.9286       425
weighted avg       0.9573    0.9576    0.9566       425
```

**Figure 4.62. Classification Report of LightGBM + ROS max_depth 5**

Figure 4.62 shows the classification report of the LightGBM + ROS model with max_depth 5. Based on the classification report, the model achieved an accuracy of 0.957647, precision macro of 0.936218, recall macro of 0.923859, F1 macro of 0.928559, and balanced accuracy of 0.923859. These values show an improvement in performance compared to max_depth 3, especially in precision macro, recall macro, F1 macro, and balanced accuracy. However, this performance was still lower than max_depth 7, so max_depth 5 was not selected as the best configuration in the max_depth testing stage for the LightGBM + ROS model.

**LightGBM + ROS max_depth 7**

In the sixth scenario, the LightGBM + ROS model was tested using max_depth 7. This value was the largest in the max_depth testing range and provided greater ability for the model to form deeper trees. With a higher tree depth, the model could learn relationships among features and more complex data patterns during training. This condition was expected to improve the model's ability to distinguish the characteristics of each class, resulting in better classification performance.

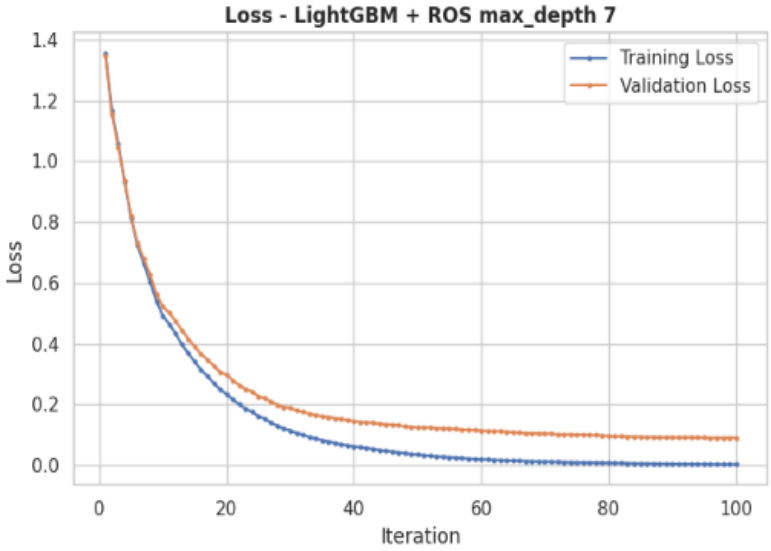



**Figure 4.63. Train Test Graph of LightGBM + ROS max_depth 7**

Figure 4.63 shows the training loss and validation loss graph of the LightGBM + ROS model with max_depth 7. This graph was used to evaluate model stability after tree complexity was increased. In this scenario, the model obtained the best performance based on the evaluation metrics. The balanced decreasing pattern of training loss and validation loss indicates that the model was able to learn

data patterns well without showing excessive overfitting. Therefore, max_depth 7 was selected as the best parameter and fixed for the next testing stage.

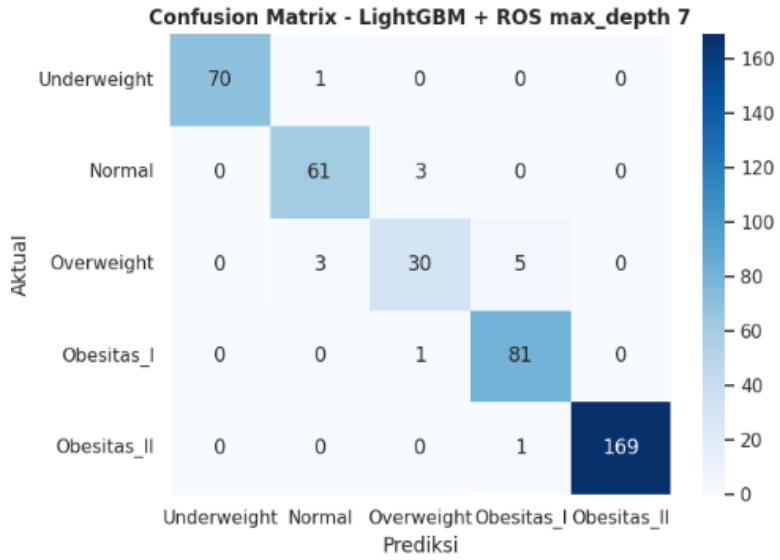



**Figure 4.64. Confusion Matrix of LightGBM + ROS max_depth 7**

Based on Figure 4.64, the LightGBM + ROS model with max_depth 7 showed the best performance compared to max_depth 3 and 5. The correct prediction values on the main diagonal appeared more dominant, indicating that the model was able to classify more testing data according to their actual classes. Prediction errors were still found in the Overweight class, but the number was relatively lower than those obtained using smaller max_depth values.

```
Model              : LightGBM + ROS
Type of Test       : Max_Depth Testing
Split Used         : 80:20
Parameter Tested   : max_depth
Value(s) Used      : 7
Testing Accuracy   : 0.967059

=== Classification Report (Testing) ===

                precision    recall   f1-score   support

  Underweight      1.0000    0.9859     0.9929        71
       Normal      0.9385    0.9531     0.9457        64
   Overweight      0.8824    0.7895     0.8333        38
    Obesity_I      0.9310    0.9878     0.9586        82
   Obesity_II      1.0000    0.9941     0.9971       170

     accuracy                         0.9671       425
    macro avg      0.9504    0.9421     0.9455       425
 weighted avg      0.9669    0.9671     0.9666       425
```

**Figure 4.65. Classification Report of LightGBM + ROS max_depth 7**

Based on the classification report in Figure 4.65, the model achieved an accuracy of 0.967059, precision macro of 0.950370, recall macro of 0.942087, F1 macro of 0.945522, and balanced accuracy of 0.942087. These values were the highest compared to max_depth 3 and 5 in the LightGBM + ROS model. The high F1 macro and balanced accuracy values indicate that the model was able to maintain more balanced performance across all target classes. This result shows that increasing the tree depth to max_depth 7 helped the model learn more complex data patterns without significantly reducing its generalization ability. Therefore, max_depth 7 was selected as the best value for the LightGBM + ROS model.

**Table 4.6. Comparison of Max Depth on XGBoost**

| Description | XGBoost + ROS max_depth 3 | XGBoost + ROS max_depth 5 | XGBoost + ROS max_depth 7 |
|---|---|---|---|
| Accuracy | 0.936471 | 0.945882 | 0.950588 |
| Precision Macro | 0.908102 | 0.923091 | 0.932276 |
| Recall Macro | 0.897989 | 0.910868 | 0.919942 |
| F1 Macro | 0.901359 | 0.915865 | 0.925082 |
| Balanced Accuracy | 0.897989 | 0.910868 | 0.919942 |
| Train Time | 0.347131 | 0.398280 | 0.485342 |
| Status | Not Selected | Not Selected | Selected |

**Table 4.7. Comparison of Max Depth on LightGBM**

| Description | LightGBM + ROS max_depth 3 | LightGBM + ROS max_depth 5 | LightGBM + ROS max_depth 7 |
|---|---|---|---|
| Accuracy | 0.948235 | 0.957647 | 0.967059 |
| Precision Macro | 0.919381 | 0.936218 | 0.950370 |
| Recall Macro | 0.916132 | 0.923859 | 0.942087 |
| F1 Macro | 0.917360 | 0.928559 | 0.945522 |
| Balanced Accuracy | 0.916132 | 0.923859 | 0.942087 |
| Train Time | 0.262707 | 0.434399 | 0.458504 |

| Description | LightGBM + ROS max_depth 3 | LightGBM + ROS max_depth 5 | LightGBM + ROS max_depth 7 |
|---|---|---|---|
| Status | Not Selected | Not Selected | Selected |

Based on Tables 4.6 and 4.7, max_depth 7 showed the most optimal performance among the tested values for both models. In the XGBoost + ROS model, this configuration obtained an accuracy of 0.950588, precision macro of 0.932276, recall macro of 0.919942, F1 macro of 0.925082, and balanced accuracy of 0.919942. Compared with this result, max_depth 5 produced slightly lower performance, with an accuracy of 0.945882, F1 macro of 0.915865, and balanced accuracy of 0.910868. Meanwhile, max_depth 3 resulted in the lowest performance, with an accuracy of 0.936471, F1 macro of 0.901359, and balanced accuracy of 0.897989.

A similar pattern was observed in the LightGBM + ROS model. The best result was also obtained when max_depth was set to 7, with an accuracy of 0.967059, precision macro of 0.950370, recall macro of 0.942087, F1 macro of 0.945522, and balanced accuracy of 0.942087. When max_depth 5 was applied, the model reached an accuracy of 0.957647, F1 macro of 0.928559, and balanced accuracy of 0.923859. The performance decreased further at max_depth 3, where the model obtained an accuracy of 0.948235, F1 macro of 0.917360, and balanced accuracy of 0.916132.

These results indicate that increasing the tree depth from 3 to 7 helped both models learn obesity classification patterns more effectively. At max_depth 3, the tree structure was relatively shallow, so the models were less capable of representing more complex relationships in the data. The use of max_depth 5 improved the performance, but the results had not reached the highest values. Max_depth 7 provided the most suitable balance between model complexity and generalization ability on the testing data. Therefore, max_depth 7 was selected as the optimal value for both XGBoost + ROS and LightGBM + ROS. This parameter was then fixed and used in the next hyperparameter testing stage, namely n_estimators testing, because it produced the highest accuracy, precision macro, recall macro, F1 macro, and balanced accuracy among the tested max_depth values.

### 4.2.5. n_estimators Testing

The n_estimators testing stage was conducted after the best learning_rate and max_depth values had been determined. This parameter controls the number of trees constructed during the model training process. Increasing the n_estimators value can help the model capture data patterns more effectively; however, it may also require longer computational time and increase model complexity. In this stage, the tested n_estimators values were 100, 200, and 300 for both the XGBoost + ROS and LightGBM + ROS models. The evaluation used the 80:20 data split, with learning_rate fixed at 0.10 and max_depth fixed at 7. Meanwhile, subsample and colsample_bytree were set to 0.80. Each testing scenario was evaluated using a confusion matrix, classification report, training validation loss graph, and several performance metrics, including accuracy, precision macro, recall macro, F1 macro, and balanced accuracy.

**XGBoost + ROS n_estimator 100**

In the first scenario, the XGBoost + ROS model was tested using n_estimators 100. This value was the initial number of estimators used in the previous testing stage. With fewer trees, the model had lower complexity and relatively faster training time.

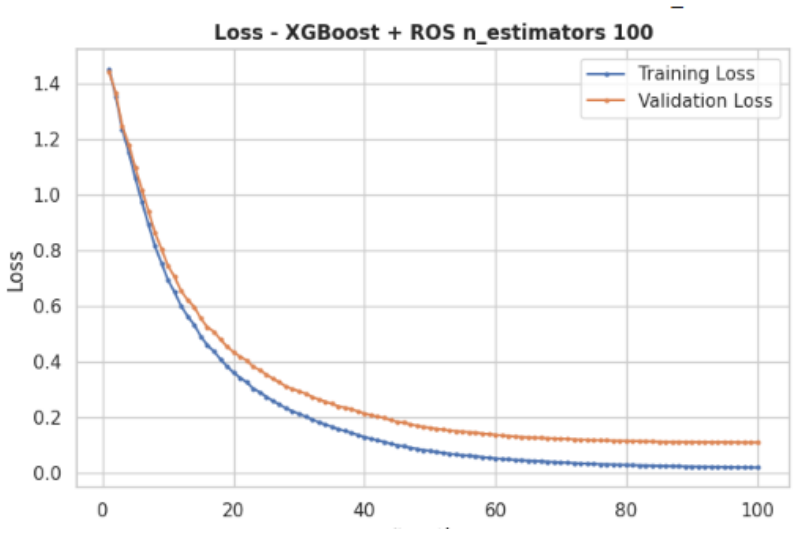



**Figure 4.66. Train Test Graph of XGBoost + ROS n_estimators 100**

Figure 4.66 shows the training loss and validation loss graph of the XGBoost + ROS model with n_estimators 100. This graph was used to observe the stability of the model learning process. With 100 estimators, the model was already able to

learn well, but there was still room for performance improvement. Based on the evaluation metrics, n_estimators 100 was not the best configuration.

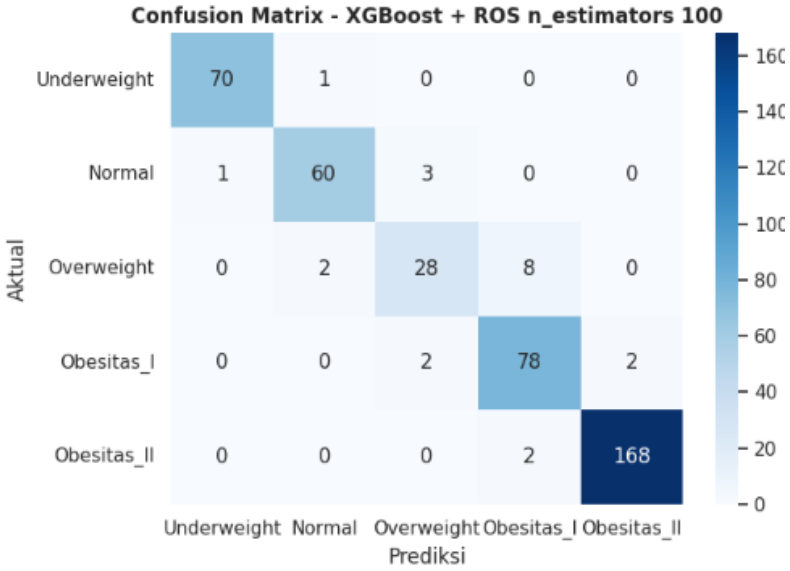



**Figure 4.67. Confusion Matrix of XGBoost + ROS n_estimators 100**

Figure 4.67 shows the confusion matrix of the XGBoost + ROS model with n_estimators 100. The values on the main diagonal indicate that most testing data were predicted correctly. However, classification errors still occurred in adjacent classes, especially Overweight, Normal, and Obesitas_I. This condition shows that n_estimators 100 was already fairly good, but it did not provide the most optimal results.

```
Model                   : XGBoost + ROS
Test Type               : Number of Estimators Test
Split Used              : 80:20
Test Parameter          : Number of Estimators
Number of Estimators Used : 100
Testing Accuracy        : 0.950588

=== Classification Report (Testing) ===

                    precision    recall  f1-score   support

      Underweight       0.9859    0.9859    0.9859        71
           Normal       0.9524    0.9375    0.9449        64
       Overweight       0.8485    0.7368    0.7887        38
   Obesity_Type_I       0.8864    0.9512    0.9176        82
  Obesity_Type_II       0.9882    0.9882    0.9882       170

         accuracy                         0.9586       425
        macro avg       0.9323    0.9199    0.9251       425
     weighted avg       0.9503    0.9506    0.9499       425
```

**Figure 4.68. Classification Report of XGBoost + ROS n_estimators 100**

Figure 4.68 presents the classification report of the XGBoost + ROS model with n_estimators 100. Based on the evaluation results, the model achieved an accuracy of 0.950588, precision macro of 0.932276, recall macro of 0.919942, F1 macro of 0.925082, and balanced accuracy of 0.919942. These values indicate that the model had good classification performance. However, the F1 macro and balanced accuracy values at n_estimators 100 were still lower than those obtained at n_estimators 200. This shows that 100 estimators were not fully optimal in helping the model learn data patterns across all target classes. Therefore, n_estimators 100 was not selected as the best configuration for the XGBoost + ROS model.

**XGBoost + ROS n_estimator 200**

In the second scenario, the XGBoost + ROS model was tested using n_estimators 200. This value provided more trees than n_estimators 100, giving the model a greater opportunity to correct prediction errors from the previous learning process.

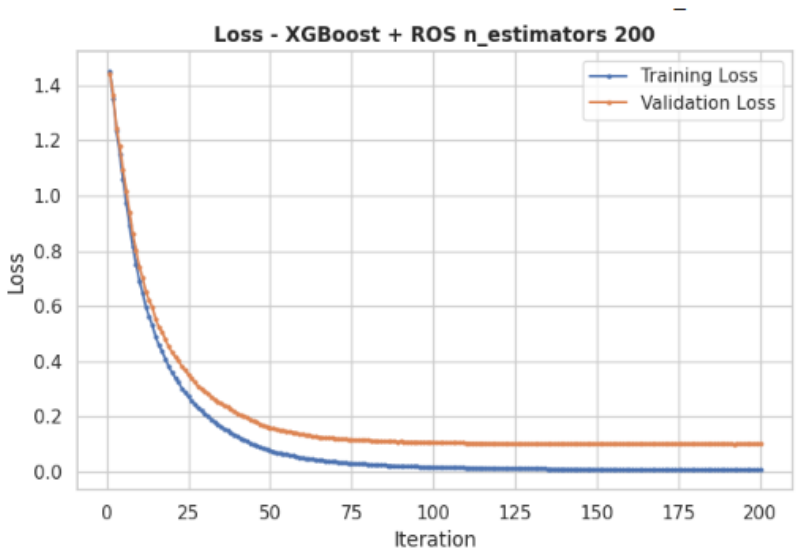



**Figure 4.69. Train Test Graph of XGBoost + ROS n_estimators 200**

Figure 4.69 shows the training and validation loss graph of the XGBoost + ROS model with n_estimators 200. The model showed better learning performance than n_estimators 100, with decreasing loss and no large gap between training and validation loss. Therefore, n_estimators 200 was selected as the best configuration and fixed for the next testing stage.

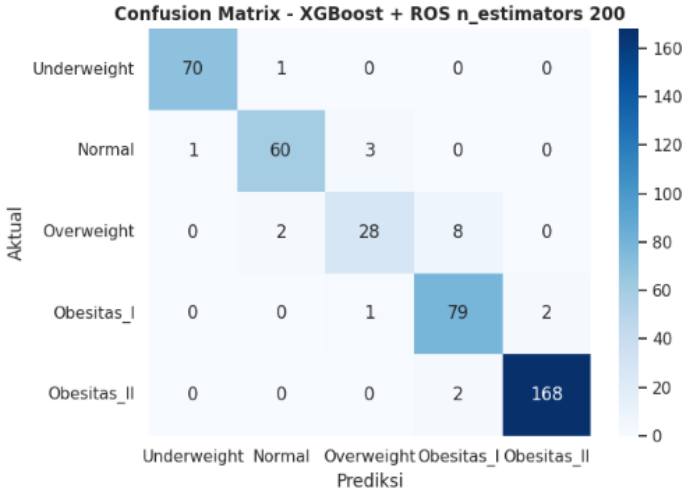



**Figure 4.70. Confusion Matrix of XGBoost + ROS n_estimators 200**

Figure 4.70 shows the confusion matrix of the XGBoost + ROS model with n_estimators 200. The main diagonal values were more dominant than those at n_estimators 100, indicating more correct predictions. Although errors still occurred in adjacent classes, the overall classification pattern improved.

```
Model                : XGBoost + ROS
Test Type            : N Estimators Test
Split Used           : 80:20
Test Parameters      : N Estimators
N Estimators Used    : 200
Testing Accuracy     : 0.952941

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

    Underweight     0.9859    0.9859    0.9859        71
         Normal     0.9524    0.9375    0.9449        64
     Overweight     0.8750    0.7368    0.8080        38
      Obesity_I     0.8876    0.9634    0.9240        82
     Obesity_II     0.9882    0.9882    0.9882       170

       accuracy                         0.9529       425
      macro avg     0.9378    0.9224    0.9286       425
   weighted avg     0.9529    0.9529    0.9521       425
```

**Figure 4.71. Classification Report of XGBoost + ROS n_estimators 200**

Figure 4.71 presents the classification report of the XGBoost + ROS model with n_estimators 200. The model achieved an accuracy of 0.952941, precision macro of 0.937834, recall macro of 0.922382, F1 macro of 0.928602, and balanced accuracy of 0.922382. These results were higher than n_estimators 100, showing that n_estimators 200 improved class recognition. Therefore, n_estimators 200 was selected as the best value in this stage.

**XGBoost + ROS n_estimator 300**

In the third scenario, the XGBoost + ROS model was tested using n_estimators 300, the largest value in the testing range. Although more estimators can improve pattern learning, they may increase training time and model complexity.

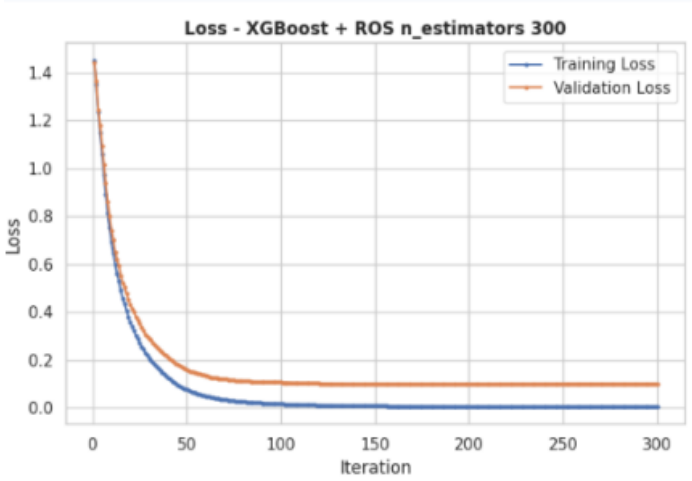



**Figure 4.72. Train Test Graph of XGBoost + ROS n_estimators 300**

Figure 4.72 shows the training and validation loss graph of the XGBoost + ROS model with n_estimators 300. Increasing the estimators to 300 did not improve the evaluation metrics compared to n_estimators 200. Therefore, this value was less efficient because it increased computational time without improving model performance.

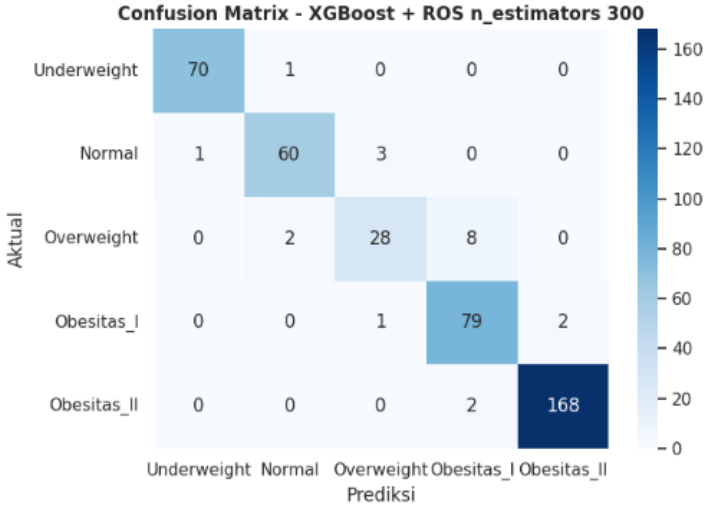



**Figure 4.73. Confusion Matrix of XGBoost + ROS n_estimators 300**

Figure 4.73 shows the confusion matrix of the XGBoost + ROS model with n_estimators 300. Based on the confusion matrix pattern, the model was still able to produce good correct predictions on the main diagonal. However, the resulting prediction pattern did not provide better performance than n_estimators 200. Prediction errors in classes with adjacent characteristics could still occur.

```
Model               : XGBoost + ROS
Type of Test        : Number of Estimators Test
Split Ratio Used    : 80:20
Test Parameter      : n_estimators
Value Used          : 300
Testing Accuracy    : 0.952941

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

   Underweight       0.9859    0.9859    0.9859        71
        Normal       0.9524    0.9375    0.9449        64
    Overweight       0.8750    0.7368    0.8000        38
     Obesity_I       0.8876    0.9634    0.9240        82
    Obesity_II       0.9882    0.9882    0.9882       170

      accuracy                         0.9529       425
     macro avg       0.9378    0.9224    0.9286       425
  weighted avg       0.9529    0.9529    0.9521       425
```

**Figure 4.74. Classification Report of XGBoost + ROS n_estimators 300**

Figure 4.74 presents the classification report of the XGBoost + ROS model with n_estimators 300. Based on the evaluation results, the model achieved an accuracy of 0.952941, precision macro of 0.937834, recall macro of 0.922382, F1 macro of 0.928602, and balanced accuracy of 0.922382. These values were the same as those obtained using n_estimators 200. Although n_estimators 300 produced the same performance as n_estimators 200, it was not selected because it required much longer training time. The training time for n_estimators 300 was 4.168282 seconds, while n_estimators 200 only required 0.915617 seconds. Thus, n_estimators 200 was more efficient because it produced the same performance with lower training time.

**LightGBM + ROS n_estimators 100**

In the fourth scenario, the LightGBM + ROS model was tested using n_estimators 100. This value was the initial number of estimators used before n_estimators testing was conducted. In the LightGBM model, the number of estimators determines the number of boosting trees built to gradually correct prediction errors.

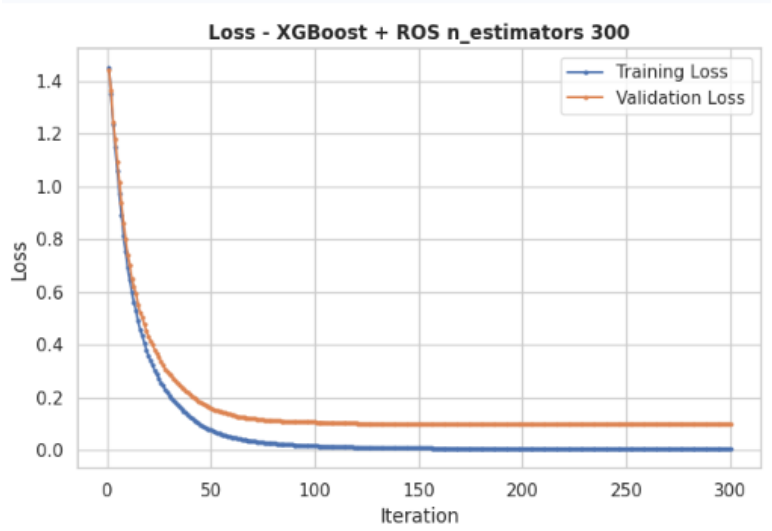



**Figure 4.75. Train Test Graph of LightGBM + ROS n_estimators 100**

Figure 4.75 shows the training loss and validation loss graph of the LightGBM + ROS model with n_estimators 100. This graph was used to observe the stability of the learning process at the initial estimator value. In this scenario, the model was able to learn the data patterns well, although the evaluation metrics showed that n_estimators 100 was not yet the most optimal configuration. Therefore, further testing with a larger number of estimators was needed to determine whether model performance could be improved.

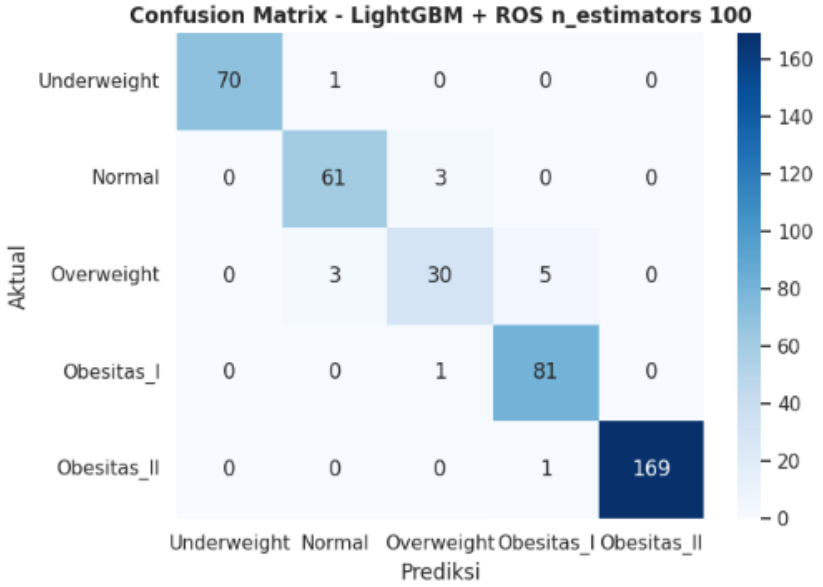



**Figure 4.76. Confusion Matrix of LightGBM + ROS n_estimators 100**

Figure 4.76 shows the confusion matrix of the LightGBM + ROS model with n_estimators 100. Based on the confusion matrix pattern, the model produced

dominant correct predictions on the main diagonal. This indicates that LightGBM + ROS with n_estimators 100 already had good classification performance. However, several classification errors still occurred in certain classes, especially classes with adjacent characteristics.

```
Model              : LightGBM + ROS
Type of Test       : Number of Estimators Test
Split Used         : 80:20
Parameter Tested   : Number of Estimators
Value Used         : 100
Testing Accuracy   : 0.967859

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

 Underweight     1.0000    0.9859    0.9929        71
      Normal     0.9385    0.9531    0.9457        64
  Overweight     0.8824    0.7895    0.8333        38
   Obesity_I     0.9310    0.9878    0.9586        82
  Obesity_II     1.0000    0.9941    0.9971       170

    accuracy                         0.9671       425
   macro avg     0.9504    0.9421    0.9455       425
weighted avg     0.9669    0.9671    0.9666       425
```

**Figure 4.77. Classification Report of LightGBM + ROS n_estimators 100**

Figure 4.77 presents the classification report of the LightGBM + ROS model with n_estimators 100. Based on the evaluation results, the model achieved an accuracy of 0.967059, precision macro of 0.950370, recall macro of 0.942087, F1 macro of 0.945522, and balanced accuracy of 0.942087. These values indicate that the model had very good performance. Although the accuracy at n_estimators 100 was the same as at n_estimators 200, the precision macro, recall macro, F1 macro, and balanced accuracy values were still lower than those obtained using n_estimators 200. This shows that n_estimators 100 was not the most optimal configuration for the LightGBM + ROS model.

**LightGBM + ROS n_estimators 200**

In the fifth scenario, the LightGBM + ROS model was tested using n_estimators 200. This value provided more trees than n_estimators 100, giving the model a greater opportunity to improve its classification ability. With more estimators, the model could learn data patterns more thoroughly during the training process. This scenario was used to determine whether increasing the number of trees could produce better performance than the previous setting.

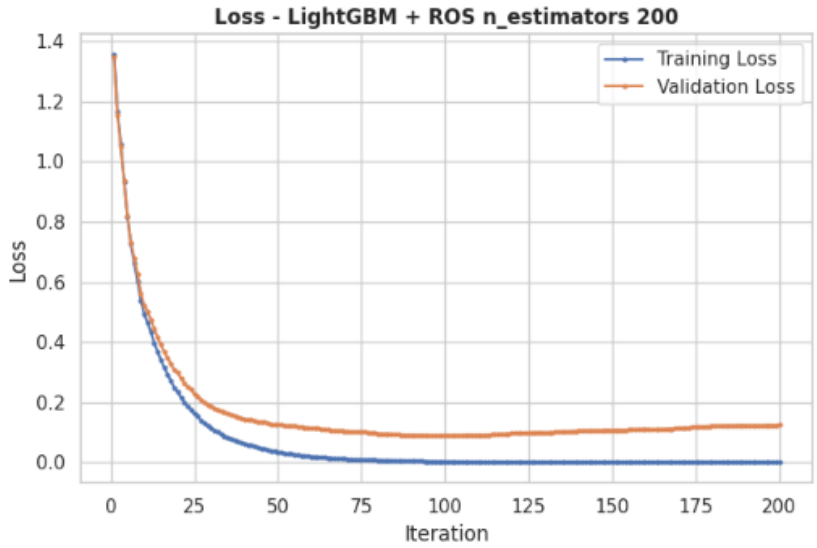



**Figure 4.78. Train Test Graph of LightGBM + ROS n_estimators 200**

Figure 4.78 shows the training loss and validation loss graph of the LightGBM + ROS model with n_estimators 200. The graph shows that the model was able to learn stably. If training loss and validation loss decrease in a balanced manner without a large gap, the model can be considered to have good generalization ability. Based on the evaluation results, n_estimators 200 was fixed as the best parameter for the next testing stage.

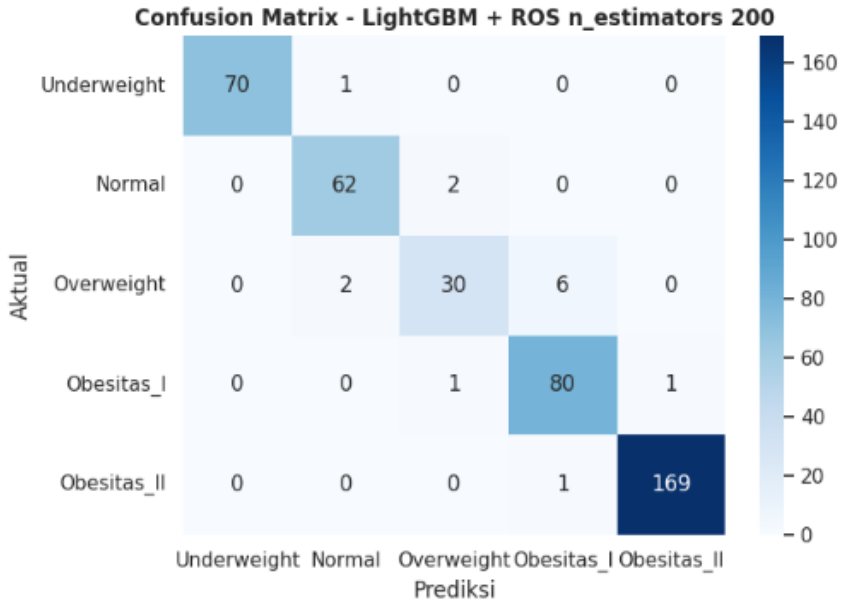



**Figure 4.79. Confusion Matrix of LightGBM + ROS n_estimators 200**

Figure 4.79 shows the confusion matrix of the LightGBM + ROS model with n_estimators 200. Based on the confusion matrix pattern, the main diagonal values appeared dominant and showed that most testing data were classified

correctly. Classification errors could still occur in certain classes, especially Overweight, but overall, the model showed better performance than n_estimators 100.

```
Model             : LightGBM + ROS
Type of Test      : n_estimators Test
Split Used        : 80:20
Test Parameter    : n_estimators
Value Used        : 200
Testing Accuracy  : 0.967059

=== Classification Report (Testing) ===
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Underweight | 1.0000 | 0.9859 | 0.9929 | 71 |
| Normal | 0.9538 | 0.9688 | 0.9612 | 64 |
| Overweight | 0.9891 | 0.7895 | 0.8451 | 38 |
| Obesitys_I | 0.9195 | 0.9756 | 0.9467 | 82 |
| Obesitys_II | 0.9941 | 0.9941 | 0.9941 | 170 |
| accuracy | | | 0.9671 | 425 |
| macro avg | 0.9553 | 0.9428 | 0.9488 | 425 |
| weighted avg | 0.9670 | 0.9671 | 0.9665 | 425 |

**Figure 4.80. Classification Report of LightGBM + ROS n_estimators 200**

Figure 4.80 presents the classification report of the LightGBM + ROS model with n_estimators 200. In this scenario, the model achieved an accuracy of 0.967059, precision macro of 0.955319, recall macro of 0.942773, F1 macro of 0.948016, and balanced accuracy of 0.942773. These values were the best results in the n_estimators testing stage for the LightGBM + ROS model. Although the accuracy was the same as n_estimators 100, the precision macro, recall macro, F1 macro, and balanced accuracy values at n_estimators 200 were higher. This indicates that n_estimators 200 provided more balanced performance across all target classes. Therefore, n_estimators 200 was selected as the best value for the LightGBM + ROS model.

**LightGBM + ROS n_estimators 300**

In the sixth scenario, the LightGBM + ROS model was tested using n_estimators 300. This value was the largest number of estimators in the testing range. This test was conducted to determine whether further increasing the number of estimators could improve model performance.

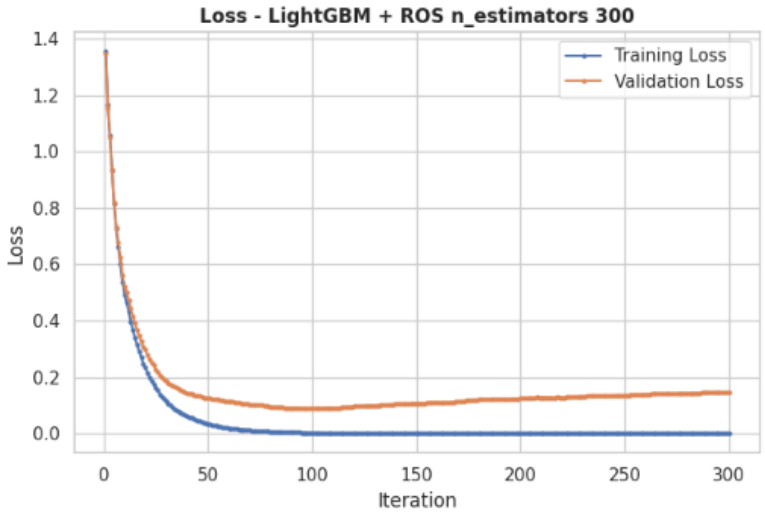



**Figure 4.81. Train Test Graph of LightGBM + ROS n_estimators 300**

Figure 4.81 shows the training loss and validation loss graph of the LightGBM + ROS model with n_estimators 300. The graph was used to observe model stability when the number of estimators was increased. In this scenario, although the model was still able to learn from the training data, the evaluation results showed a decrease in performance compared to n_estimators 200. Therefore, n_estimators 300 did not provide more optimal results.

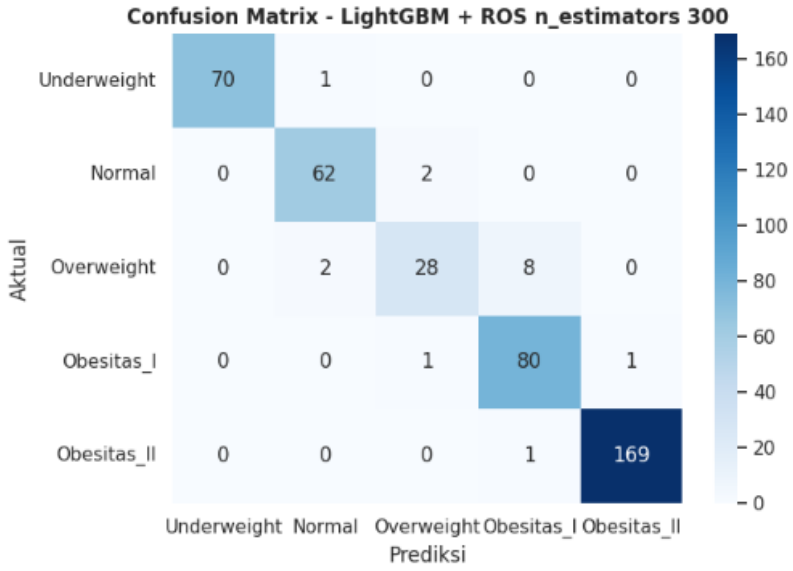



**Figure 4.82. Confusion Matrix of LightGBM + ROS n_estimators 300**

Figure 4.82 shows the confusion matrix of the LightGBM + ROS model with n_estimators 300. Based on the confusion matrix pattern, the model was still able to produce many correct predictions on the main diagonal. However, compared to n_estimators 200, the model performance decreased. This indicates that

increasing the number of estimators to 300 did not improve the performance of the LightGBM + ROS model.

```
Model              : LightGBM + ROS
Type of Test       : N_Estimators Testing
Split Used         : 80:20
Parameter Tested   : N_Estimators
Value(s) Used      : 300
Testing Accuracy   : 0.962353

=== Classification Report (Testing) ===
                precision    recall  f1-score   support

   Underweight     1.0000    0.9859    0.9929        71
        Normal     0.9538    0.9688    0.9612        64
    Overweight     0.9032    0.7368    0.8116        38
     Obesity_I     0.8989    0.9756    0.9357        82
    Obesity_II     0.9941    0.9941    0.9941       170

      accuracy                         0.9624       425
     macro avg     0.9500    0.9322    0.9391       425
  weighted avg     0.9625    0.9624    0.9614       425
```

**Figure 4.83. Classification Report of LightGBM + ROS n_estimators 300**

Figure 4.83 presents the classification report of the LightGBM + ROS model with n_estimators 300. Based on the evaluation results, the model achieved an accuracy of 0.962353, precision macro of 0.950013, recall macro of 0.932247, F1 macro of 0.939106, and balanced accuracy of 0.932247. These values were lower than those obtained using n_estimators 200. The decrease in F1 macro and balanced accuracy indicates that increasing the number of estimators to 300 did not always improve model performance. An excessive number of estimators can make the model more complex and may not provide additional benefits on the testing data. Therefore, n_estimators 300 was not selected as the best configuration.

**Table 4.8. Comparison of n_estimators on XGBoost**

| Description | XGBoost + ROS n_estimators 100 | XGBoost + ROS n_estimators 200 | XGBoost + ROS n_estimators 300 |
|---|---|---|---|
| Accuracy | 0.950588 | 0.952941 | 0.952941 |
| Precision Macro | 0.932276 | 0.937834 | 0.937834 |
| Recall Macro | 0.919942 | 0.922382 | 0.922382 |
| F1 Macro | 0.925082 | 0.928602 | 0.928602 |
| Balanced Accuracy | 0.919942 | 0.922382 | 0.922382 |

| Description | XGBoost + ROS n_estimators 100 | XGBoost + ROS n_estimators 200 | XGBoost + ROS n_estimators 300 |
|---|---|---|---|
| Train Time | 0.481177 | 0.915617 | 4.168282 |
| Status | Not Selected | Selected | Not Selected |

**Table 4.9. Comparison of n_estimators on LightGBM**

| Description | LightGBM + ROS n_estimators 100 | LightGBM + ROS n_estimators 200 | LightGBM + ROS n_estimators 300 |
|---|---|---|---|
| Accuracy | 0.967059 | 0.967059 | 0.962353 |
| Precision Macro | 0.950370 | 0.955319 | 0.950013 |
| Recall Macro | 0.942087 | 0.942773 | 0.932247 |
| F1 Macro | 0.945522 | 0.948016 | 0.939106 |
| Balanced Accuracy | 0.942087 | 0.942773 | 0.932247 |
| Train Time | 2.212992 | 2.774208 | 1.268871 |
| Status | Not Selected | Selected | Not Selected |

Based on Tables 4.8 and 4.9, n_estimators 200 gave the best performance for both XGBoost + ROS and LightGBM + ROS. In XGBoost + ROS, this value achieved an accuracy of 0.952941, precision macro of 0.937834, recall macro of 0.922382, F1 macro of 0.928602, and balanced accuracy of 0.922382. Although n_estimators 300 produced the same metrics, it required longer training time, so n_estimators 200 was considered more efficient. In LightGBM + ROS, n_estimators 200 also produced the best result, with an accuracy of 0.967059, precision macro of 0.955319, recall macro of 0.942773, F1 macro of 0.948016, and balanced accuracy of 0.942773. This value performed better than n_estimators 100 and 300. Therefore, n_estimators 200 was selected as the best value for both models and used in the next stage, namely subsample testing.

### 4.2.6. Subsample Testing

Subsample testing is the fourth stage after obtaining the best learning_rate, max_depth, and n_estimators values. The subsample parameter controls the

proportion of training data used to build each tree. A smaller value can reduce overfitting, but too small a value may limit the information learned by the model. In this stage, subsample values of 0.70, 0.80, and 0.90 were tested on XGBoost + ROS and LightGBM + ROS using the 80:20 split. The fixed parameters were learning_rate = 0.10, max_depth = 7, n_estimators = 200, and colsample_bytree = 0.80. Each scenario was evaluated using a confusion matrix, classification report, training validation loss graph, accuracy, precision macro, recall macro, F1 macro, and balanced accuracy.

**XGBoost + ROS Subsample 0,7**

In the first scenario, the XGBoost + ROS model was tested using subsample 0.70, meaning that 70% of the training data were used in building each tree. Using only a portion of the training data aimed to increase the variation in the model learning process and reduce dependence on the entire training dataset.

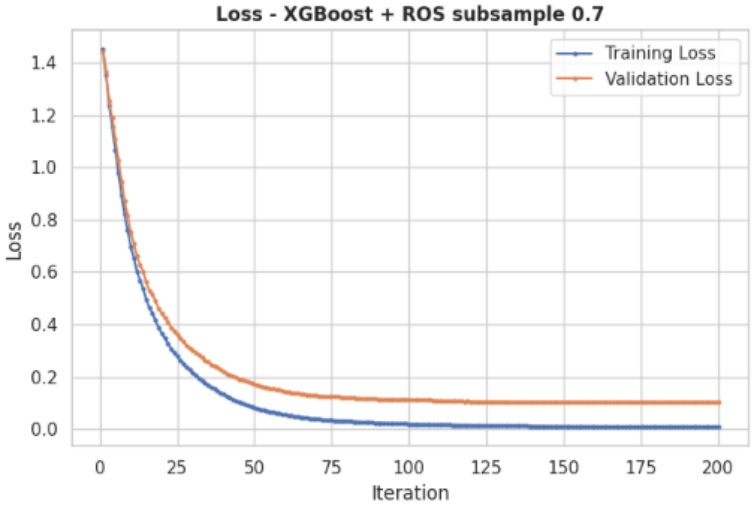



**Figure 4.84. Train Test Graph of XGBoost + ROS subsample 0.70**

Figure 4.84 shows the training loss and validation loss graph of the XGBoost + ROS model with subsample 0.70. This graph was used to observe the stability of the model learning process. In this scenario, the model showed good learning ability. However, based on the evaluation metrics, the performance of subsample 0.70 was still slightly lower than subsample 0.90. Therefore, subsample 0.70 was not the most optimal configuration.

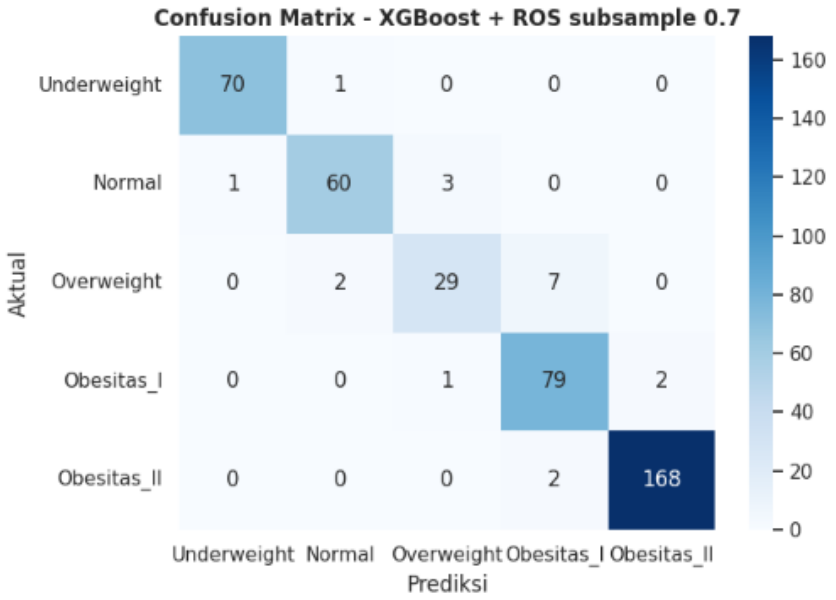



**Figure 4.85. Confusion Matrix of XGBoost + ROS subsample 0.70**

Figure 4.85 shows the confusion matrix of the XGBoost + ROS model with subsample 0.70. The values on the main diagonal appeared dominant, indicating that most testing data were classified according to their actual classes. However, prediction errors still occurred in adjacent classes, especially Overweight, Normal, and Obesitas_I. This condition shows that subsample 0.70 was able to produce good performance, although it was not fully optimal.

```
Model                : XGBoost + ROS
Test Type            : Subsample Test
Split Used           : 80:20
Test Parameter       : Subsample
Subsample Used       : 0.7
Testing Accuracy     : 0.955294

=== Classification Report (Testing) ===

                    precision    recall  f1-score   support

      Underweight       0.9859    0.9859    0.9859        71
           Normal       0.9524    0.9375    0.9449        64
       Overweight       0.8788    0.7632    0.8169        38
       Obesitas_I       0.8977    0.9634    0.9294        82
      Obesitas_II       0.9882    0.9882    0.9882       170

         accuracy                         0.9553       425
        macro avg       0.9406    0.9276    0.9331       425
     weighted avg       0.9552    0.9553    0.9547       425
```

**Figure 4.86. Classification Report of XGBoost + ROS subsample 0.70**

Figure 4.86 presents the classification report of the XGBoost + ROS model with subsample 0.70. Based on the evaluation results, the model achieved an accuracy of 0.955294, precision macro of 0.940609, recall macro of 0.927645, F1 macro of 0.933069, and balanced accuracy of 0.927645. These values indicate that the model was able to perform classification well. Although the accuracy and precision macro values at subsample 0.70 were relatively high, the balanced accuracy value was still lower than that of subsample 0.90. This shows that the model's ability to recognize all classes evenly was still not as good as in the subsample 0.90 scenario. Therefore, subsample 0.70 was not selected as the best configuration for the XGBoost + ROS model.

**XGBoost + ROS Subsample 0,8**

In the second scenario, the XGBoost + ROS model was tested using subsample 0.80. This value was the initial value used before subsample testing was conducted. With subsample 0.80, the model used 80% of the training data in each tree building process.

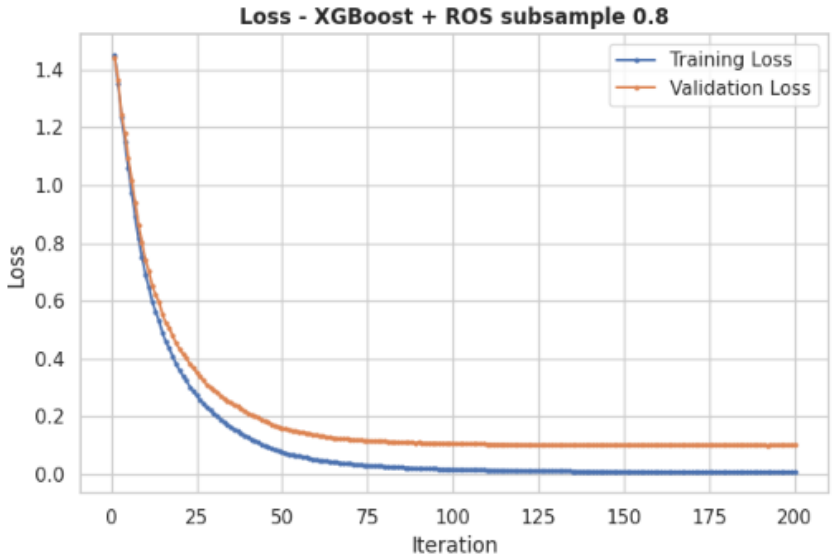



**Figure 4.87. Train Test Graph of XGBoost + ROS subsample 0.80**

Figure 4.87 shows the training loss and validation loss graph of the XGBoost + ROS model with subsample 0.80. The graph illustrates the model learning process during training. If training loss and validation loss decrease with a gap that is not too large, the model can be considered stable. However, based on the evaluation metrics, subsample 0.80 did not provide the best performance compared to the other candidates.

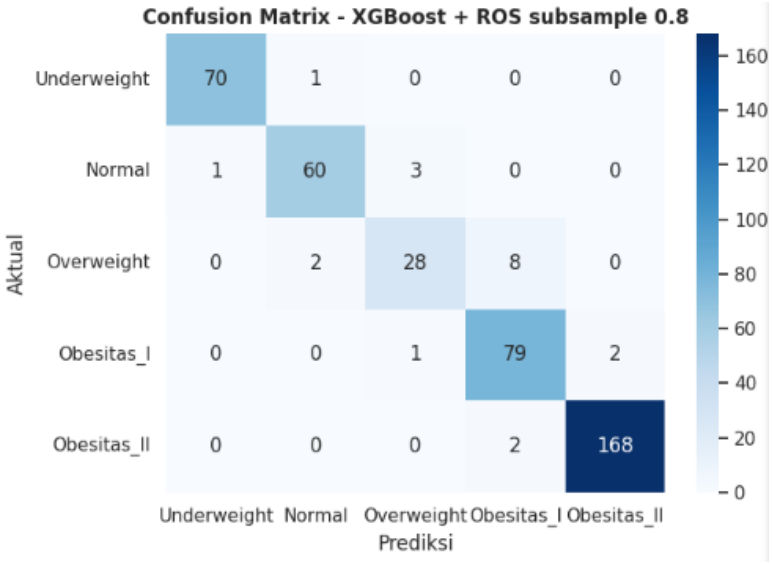



**Figure 4.88. Confusion Matrix of XGBoost + ROS subsample 0.80**

Figure 4.88 shows the confusion matrix of the XGBoost + ROS model with subsample 0.80. Based on the confusion matrix pattern, the model was still able to correctly predict most testing data. The values on the main diagonal remained dominant compared to the values outside the diagonal. However, the obtained performance was not better than subsample 0.70 and 0.90. Prediction errors still appeared in classes with adjacent characteristics.

```
Model               : XGBoost + ROS
Test Type           : Subsample Test
Split Used          : 80:20
Test Parameters     : Subsample
Subsample Used      : 0.8
Testing Accuracy    : 0.952941

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

     Underweight     0.9859    0.9859    0.9859        71
          Normal     0.9524    0.9375    0.9449        64
      Overweight     0.8750    0.7368    0.8080        38
       Obesity_I     0.8876    0.9634    0.9240        82
      Obesity_II     0.9882    0.9882    0.9882       170

        accuracy                         0.9529       425
       macro avg     0.9378    0.9224    0.9286       425
    weighted avg     0.9529    0.9529    0.9521       425
```

**Figure 4.89. Classification Report of XGBoost + ROS subsample 0.80**

Figure 4.89 presents the classification report of the XGBoost + ROS model with subsample 0.80. In this scenario, the model achieved an accuracy of 0.952941,

precision macro of 0.937834, recall macro of 0.922382, F1 macro of 0.928602, and balanced accuracy of 0.922382. These values indicate that the model still had good performance, but it was the lowest compared to the other two subsample values in the XGBoost + ROS model. The lower F1 macro and balanced accuracy values show that subsample 0.80 had not provided the best balanced performance across all target classes. Therefore, this value was not selected as the best configuration in the subsample testing stage.

**XGBoost + ROS subsample 0.90**

In the third scenario, the XGBoost + ROS model was tested using subsample 0.90. This value indicates that 90% of the training data were used in building each tree. With a larger data proportion, the model obtained more information during the learning process, while still maintaining variation because not all data were used in each iteration. This condition was expected to improve the model's ability to learn data patterns while maintaining generalization ability on the testing data.

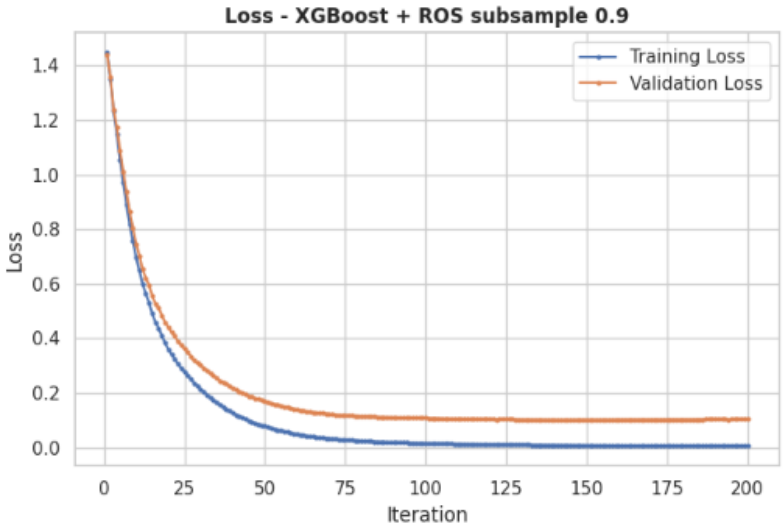



**Figure 4.90. Train Test Graph XGBoost + ROS Subsample 0,9**

Figure 4.90 shows the training loss and validation loss graph of the XGBoost + ROS model with subsample 0.90. This graph was used to observe model stability after increasing the proportion of training data used in each tree. In this scenario, the model obtained the best evaluation metrics. If training loss and validation loss show a balanced decreasing pattern, the model can be considered to have a stable learning process. Therefore, subsample 0.90 was used as the fixed parameter for the next testing stage, namely colsample_bytree.

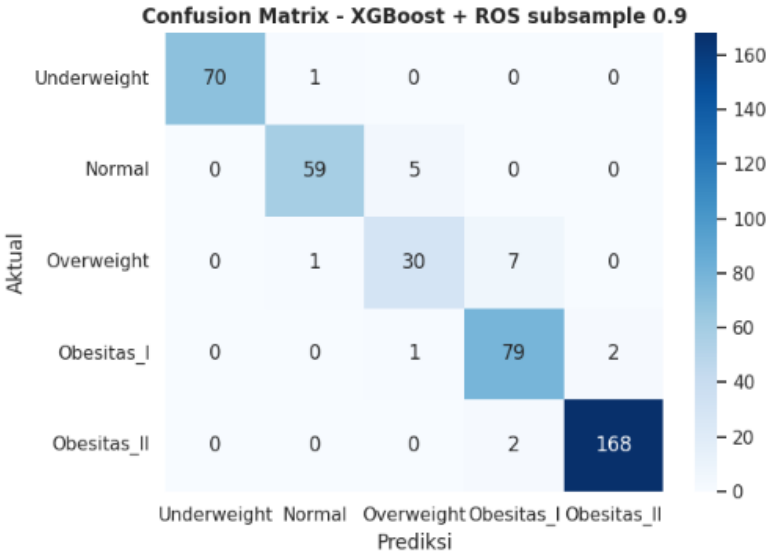



**Figure 4.91. Confusion Matrix of XGBoost + ROS subsample 0.90**

Figure 4.91 shows the confusion matrix of the XGBoost + ROS model with subsample 0.90. Based on the confusion matrix pattern, the values on the main diagonal appeared dominant and showed that the model produced many correct predictions. Prediction errors could still occur in certain classes, especially Overweight, but overall, the classification pattern showed better results compared to subsample 0.70 and 0.80.

```
Model              : XGBoost + ROS
Type of Test       : Subsample Test
Split Ratio Used   : 80:20
Test Parameter     : subsample
Value Used         : 0.9
Testing Accuracy   : 0.955294

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

   Underweight     1.0000    0.9859    0.9929        71
        Normal     0.9672    0.9219    0.9440        64
    Overweight     0.8333    0.7895    0.8108        38
     Obesity_I     0.8977    0.9634    0.9294        82
    Obesity_II     0.9882    0.9882    0.9882       170

      accuracy                         0.9553       425
     macro avg     0.9373    0.9298    0.9331       425
  weighted avg     0.9557    0.9553    0.9551       425
```

**Figure 4.92. Classification Report of XGBoost + ROS subsample 0.90**

Figure 4.92 presents the classification report of the XGBoost + ROS model with subsample 0.90. Based on the evaluation results, the model achieved an

accuracy of 0.955294, precision macro of 0.937302, recall macro of 0.929783, F1 macro of 0.933073, and balanced accuracy of 0.929783. The F1 macro and balanced accuracy values in this scenario were the highest compared to the other subsample values in the XGBoost + ROS model. Although the accuracy was the same as subsample 0.70, subsample 0.90 had higher recall macro, F1 macro, and balanced accuracy values. This shows that subsample 0.90 was better in maintaining the model's ability to recognize all target classes evenly. Therefore, subsample 0.90 was selected as the best value in the subsample testing stage for the XGBoost + ROS model.

**LightGBM + ROS Subsample 0,7**

In the fourth scenario, the LightGBM + ROS model was tested using subsample 0.70. This value indicates that the model used 70% of the training data in each tree building process. This test was conducted to determine whether using only a portion of the training data could improve model generalization.

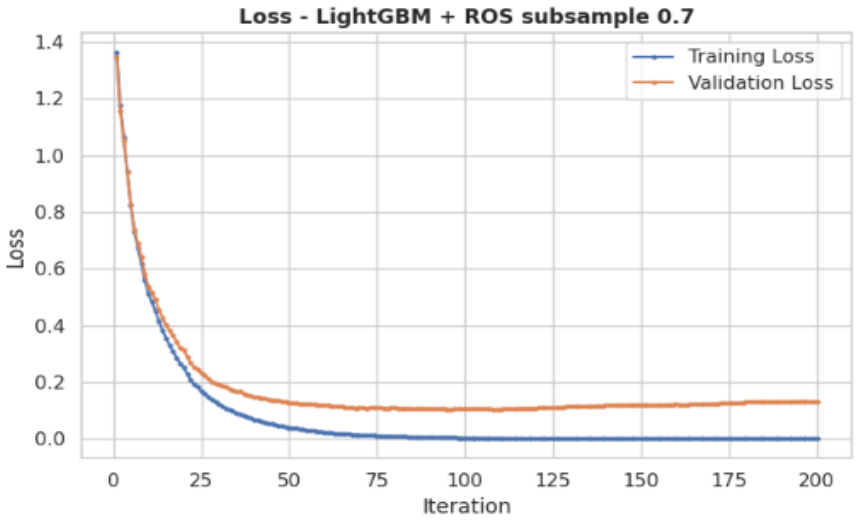



**Figure 4.93. Train Test Graph of LightGBM + ROS subsample 0.70**

Figure 4.93 shows the training loss and validation loss graph of the LightGBM + ROS model with subsample 0.70. The graph shows the model learning process during training. At this value, the model was still able to learn from the training data, but the evaluation results showed that its performance was not as good as subsample 0.80. Therefore, subsample 0.70 was not the most optimal configuration.

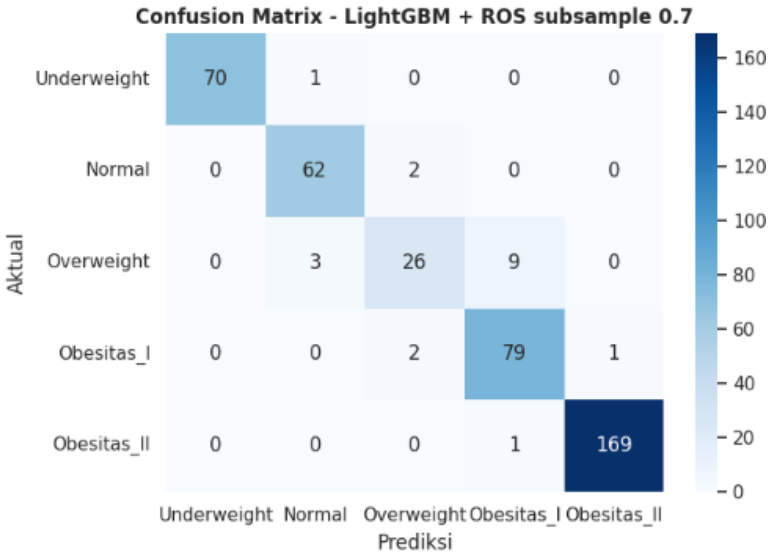



**Figure 4.94. Confusion Matrix of LightGBM + ROS subsample 0.70**

Figure 4.94 shows the confusion matrix of the LightGBM + ROS model with subsample 0.70. Based on the confusion matrix pattern, the model was still able to produce correct predictions for most testing data. However, prediction errors were still found in several classes, especially those with adjacent characteristic boundaries. Compared to subsample 0.80 and 0.90, the performance of subsample 0.70 was still lower.

```
Model              : LightGBM + ROS
Type of Test       : Subsample Test
Split Used         : 80:20
Parameter Tested   : Subsample
Value Used         : 0.7
Testing Accuracy   : 0.955294

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

 Underweight       1.0000    0.9859    0.9929        71
      Normal       0.9394    0.9688    0.9538        64
  Overweight       0.8667    0.6842    0.7647        38
   Obesity_I       0.8876    0.9634    0.9240        82
  Obesity_II       0.9941    0.9941    0.9941       170

    accuracy                         0.9553       425
   macro avg       0.9376    0.9193    0.9259       425
weighted avg       0.9549    0.9553    0.9538       425
```

**Figure 4.95. Classification Report of LightGBM + ROS subsample 0.70**

Figure 4.95 presents the classification report of the LightGBM + ROS model with subsample 0.70. Based on the evaluation results, the model achieved an

accuracy of 0.955294, precision macro of 0.937564, recall macro of 0.919282, F1 macro of 0.925911, and balanced accuracy of 0.919282. These values indicate that the model had good performance, but it was not the best result. The F1 macro and balanced accuracy values at subsample 0.70 were lower than those obtained at subsample 0.80 and 0.90. This shows that using 70% of the training data was not sufficiently optimal in helping LightGBM recognize all target classes evenly. Therefore, subsample 0.70 was not selected as the best configuration.

**LightGBM + ROS Subsample 0,8**

In the fifth scenario, the LightGBM + ROS model was tested using subsample 0.80. This value was the initial value used in the previous stage. With subsample 0.80, the model used 80% of the training data in building each tree. Using a portion of the training data in each iteration aimed to maintain variation among trees so that the model could still learn data patterns well without excessively increasing complexity.

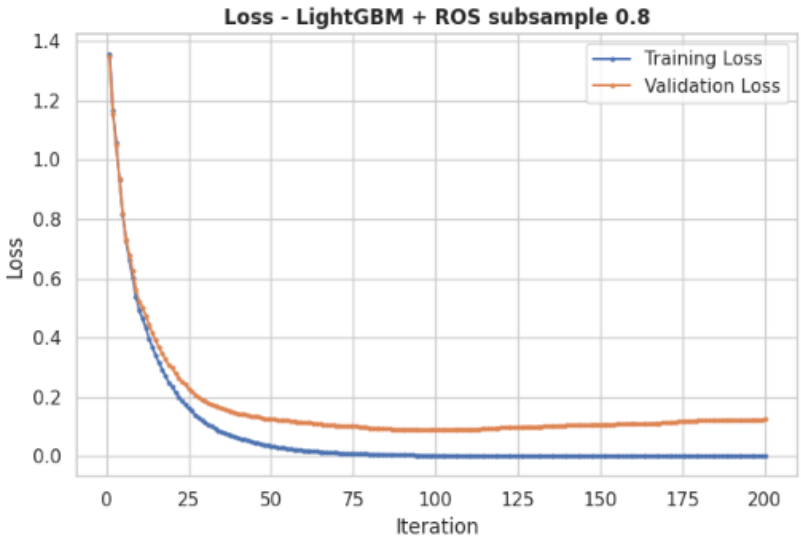



**Figure 4.96. Train Test Graph of LightGBM + ROS subsample 0.80**

Figure 4.96 shows the training loss and validation loss graph of the LightGBM + ROS model with subsample 0.80. The graph illustrates the stability of the model learning process. If training loss and validation loss decrease consistently without a large gap, the model can be considered to have good generalization ability. Based on the evaluation metrics, subsample 0.80 was fixed as the best parameter for the next testing stage.

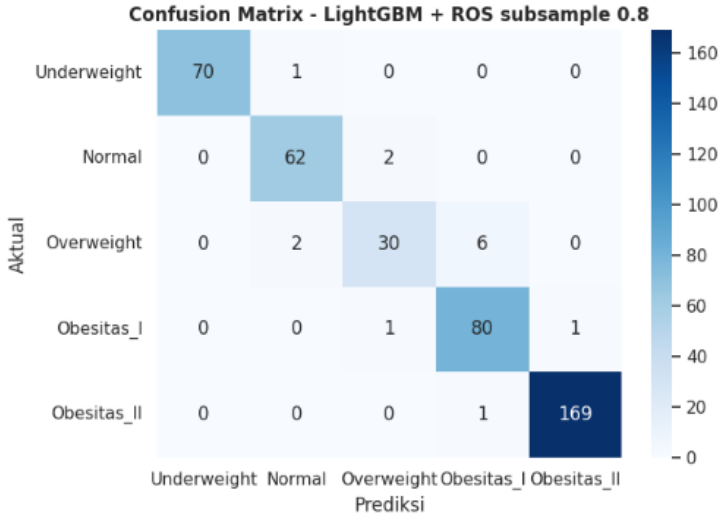



**Figure 4.97. Confusion Matrix of LightGBM + ROS subsample 0.80**

Figure 4.97 shows the confusion matrix of the LightGBM + ROS model with subsample 0.80. Based on the confusion matrix pattern, the correct prediction values on the main diagonal appeared dominant. This indicates that most testing data were classified according to their actual classes. Prediction errors could still appear in certain classes, especially Overweight, but overall, the model showed the best performance compared to the other subsample candidates.

```
Model              : LightGBM + ROS
Type of Test       : Subsample Test
Split Used         : 80:20
Test Parameter     : Subsample
Value Used         : 0.8
Testing Accuracy   : 0.967059

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

 Underweight     1.0000    0.9859    0.9929        71
      Normal     0.9538    0.9688    0.9612        64
  Overweight     0.9091    0.7895    0.8451        38
  Obesitys_I     0.9195    0.9756    0.9467        82
 Obesitys_II     0.9941    0.9941    0.9941       170

    accuracy                         0.9671       425
   macro avg     0.9553    0.9428    0.9488       425
weighted avg     0.9670    0.9671    0.9665       425
```

**Figure 4.98. Classification Report of LightGBM + ROS subsample 0.80**

Figure 4.98 presents the classification report of the LightGBM + ROS model with subsample 0.80. The model achieved an accuracy of 0.967059, precision macro of 0.955319, recall macro of 0.942773, F1 macro of 0.948016, and balanced

accuracy of 0.942773. These results were the best in this stage, so subsample 0.80 was selected for the LightGBM + ROS model.

**LightGBM + ROS Subsample 0,9**

In the sixth scenario, the LightGBM + ROS model was tested using subsample 0.90, where 90% of the training data were used to build each tree. This value used a larger data portion than subsample 0.70 and 0.80.

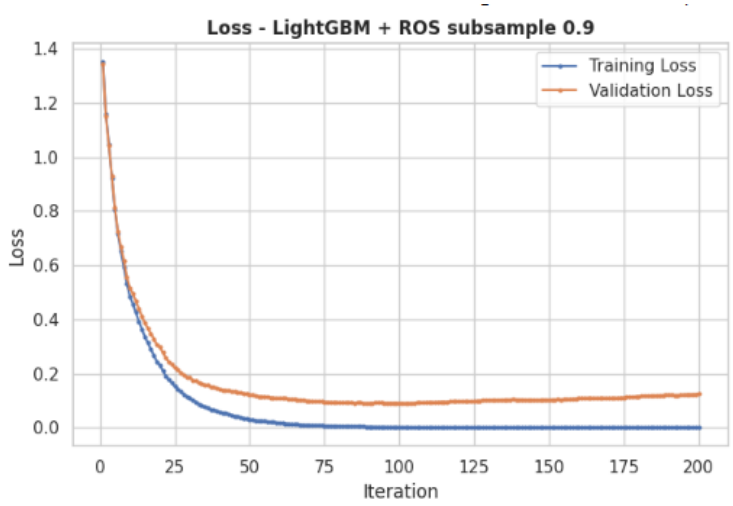



**Figure 4.99. Train Test Graph of LightGBM + ROS subsample 0.90**

Figure 4.99 shows the training and validation loss graph of the LightGBM + ROS model with subsample 0.90. The model still showed stable performance, but the results did not exceed subsample 0.80. Therefore, subsample 0.90 was not selected as the best configuration.

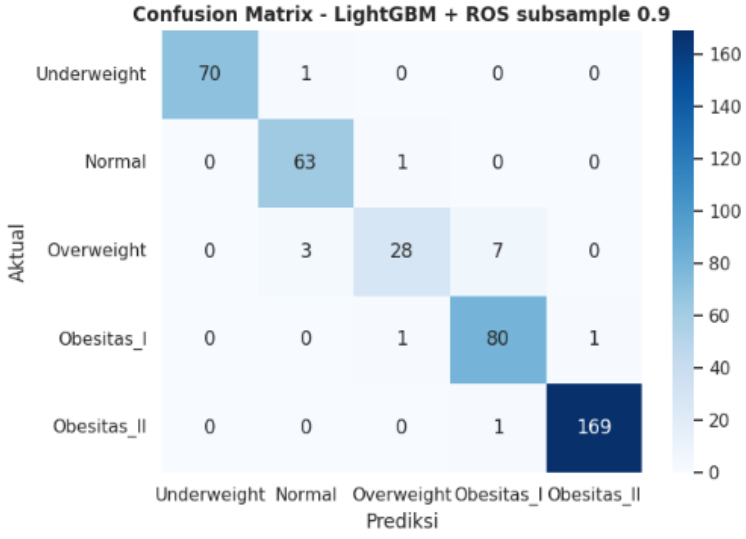



**Figure 4.100. Confusion Matrix of LightGBM + ROS subsample 0.90**

Figure 4.100 shows the confusion matrix of the LightGBM + ROS model with subsample 0.90. Based on the confusion matrix pattern, most testing data were still correctly predicted, as shown by the dominant values on the main diagonal. This indicates that the model still had good classification ability. However, compared to subsample 0.80, the number of correct predictions was slightly lower, and classification errors were still found in several classes, especially those with adjacent characteristics or boundaries. This condition indicates that increasing the subsample value to 0.90 did not improve performance compared to the previous configuration.

```
Model              : LightGBM + ROS
Type of Test       : Subsample Testing
Split Used         : 80:20
Parameter Tested   : Subsample
Value(s) Used      : 0.9
Testing Accuracy   : 0.964786

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

   Underweight     1.0000    0.9859    0.9929        71
        Normal     0.9403    0.9844    0.9618        64
    Overweight     0.9333    0.7368    0.8235        38
     Obesity_I     0.9891    0.9756    0.9412        82
    Obesity_II     0.9941    0.9941    0.9941       170

      accuracy                         0.9647       425
     macro avg     0.9554    0.9354    0.9427       425
  weighted avg     0.9652    0.9647    0.9636       425
```

**Figure 4.101. Classification Report of LightGBM + ROS subsample 0.90**

Figure 4.101 presents the classification report of the LightGBM + ROS model with subsample 0.90. Based on the evaluation results, the model achieved an accuracy of 0.964706, precision macro of 0.955368, recall macro of 0.935372, F1 macro of 0.942713, and balanced accuracy of 0.935372. These values indicate that the model performance was still high, but lower than subsample 0.80 in terms of accuracy, recall macro, F1 macro, and balanced accuracy. Although precision macro at subsample 0.90 was slightly higher, its balanced accuracy and F1 macro values were lower than those of subsample 0.80. This shows that subsample 0.80 was better in maintaining balanced performance across classes. Therefore, subsample 0.90 was not selected as the best configuration for the LightGBM + ROS model.

**Table 4.10. Comparison of Subsample on XGBoost**

| Description | XGBoost + ROS subsample 0.70 | XGBoost + ROS subsample 0.80 | XGBoost + ROS subsample 0.90 |
|---|---|---|---|
| Accuracy | 0.955294 | 0.952941 | 0.955294 |
| Precision Macro | 0.940609 | 0.937834 | 0.937302 |
| Recall Macro | 0.927645 | 0.922382 | 0.929783 |
| F1 Macro | 0.933069 | 0.928602 | 0.933073 |
| Balanced Accuracy | 0.927645 | 0.922382 | 0.929783 |
| Train Time | 1.236850 | 0.811226 | 0.770310 |
| Status | Not Selected | Not Selected | Selected |

**Table 4.11. Comparison of Subsample on LightGBM**

| Description | LightGBM + ROS subsample 0.70 | LightGBM + ROS subsample 0.80 | LightGBM + ROS subsample 0.90 |
|---|---|---|---|
| Accuracy | 0.955294 | 0.967059 | 0.964706 |
| Precision Macro | 0.937564 | 0.955319 | 0.955368 |
| Recall Macro | 0.919282 | 0.942773 | 0.935372 |
| F1 Macro | 0.925911 | 0.948016 | 0.942713 |
| Balanced Accuracy | 0.919282 | 0.942773 | 0.935372 |
| Train Time | 0.917748 | 0.898841 | 1.036017 |
| Status | Not Selected | Selected | Not Selected |

Based on Tables 4.10 and 4.11, the best subsample value for the XGBoost + ROS model was 0.90. This value produced an accuracy of 0.955294, precision macro of 0.937302, recall macro of 0.929783, F1 macro of 0.933073, and balanced accuracy of 0.929783. Although the accuracy at subsample 0.90 was the same as subsample 0.70, the recall macro, F1 macro, and balanced accuracy values at subsample 0.90 were higher. In addition, the training time at subsample 0.90 was also lower than subsample 0.70. Therefore, subsample 0.90 was selected as the best

configuration for XGBoost + ROS. In the LightGBM + ROS model, the best subsample value was 0.80. This value produced an accuracy of 0.967059, precision macro of 0.955319, recall macro of 0.942773, F1 macro of 0.948016, and balanced accuracy of 0.942773. These values were higher than subsample 0.70 and 0.90 in terms of accuracy, recall macro, F1 macro, and balanced accuracy. Although precision macro at subsample 0.90 was slightly higher, subsample 0.80 was still selected because it provided more balanced overall performance.

The testing results show that the optimal subsample value was not always the same for each model. XGBoost + ROS achieved the best performance at subsample 0.90, while LightGBM + ROS achieved the best performance at subsample 0.80. Therefore, the best subsample value for each model was fixed and used in the next hyperparameter testing stage, namely colsample_bytree testing..

### 4.2.7. Colsample_bytree Testing

Colsample_bytree testing is the fifth stage after the best learning_rate, max_depth, n_estimators, and subsample values were obtained. This parameter is used to control the proportion of features used in building each tree. A smaller colsample_bytree value can reduce the model's dependence on all features and help minimize the risk of overfitting. However, if the value is too small, important information from certain features may not be utilized optimally. At this stage, the tested colsample_bytree values were 0.70, 0.80, and 0.90 on the XGBoost + ROS and LightGBM + ROS models using the best split ratio of 80:20. In XGBoost + ROS, the fixed parameters were learning_rate = 0.10, max_depth = 7, n_estimators = 200, and subsample = 0.90. Meanwhile, in LightGBM + ROS, the fixed parameters were learning_rate = 0.10, max_depth = 7, n_estimators = 200, and subsample = 0.80. Each scenario was evaluated using a confusion matrix, classification report, training validation loss graph, and evaluation metrics consisting of accuracy, precision macro, recall macro, F1 macro, and balanced accuracy.

**XGBoost + ROS colsample_bytree 0,7**

In the first scenario, the XGBoost + ROS model was tested using colsample_bytree 0.70, meaning that 70% of the features were used in building each

tree. Using only a portion of the features aimed to prevent the model from depending too heavily on all features, so the learning process could become more varied and the risk of overfitting could be reduced. Therefore, the model was expected to capture data patterns more effectively and maintain good generalization ability.

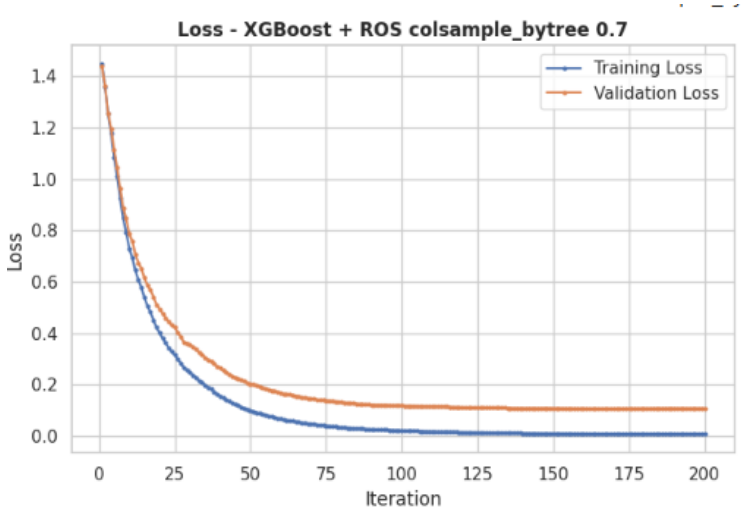



**Figure 4.102. Train Test Graph of XGBoost + ROS colsample_bytree 0.70**

Figure 4.102 shows the training loss and validation loss graph of the XGBoost + ROS model with colsample_bytree 0.70. This graph was used to observe the stability of the model learning process. In this scenario, the model still showed good learning ability, but the evaluation metrics did not exceed those obtained using colsample_bytree 0.90. Therefore, this value was not the most optimal configuration.

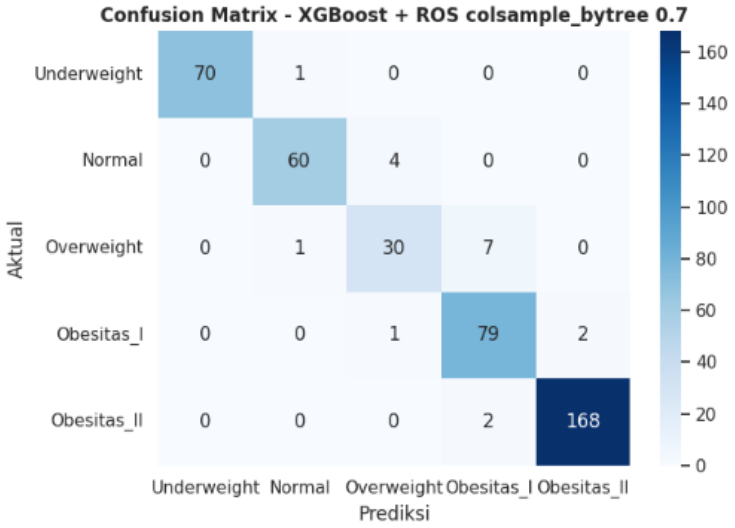



**Figure 4.103. Confusion Matrix of XGBoost + ROS colsample_bytree 0.70**

Figure 4.103 shows the confusion matrix of the XGBoost + ROS model with colsample_bytree 0.70. The values on the main diagonal appeared dominant, indicating that most testing data were classified correctly. However, prediction errors still occurred in adjacent classes, especially Overweight, which had the potential to be misclassified as Normal or Obesitas_I. In general, colsample_bytree 0.70 produced good performance, but it was not the best result.

```
Model                : XGBoost + ROS
Test Type            : Colsample by Tree Test
Split Used           : 80:20
Test Parameter       : Colsample by Tree
Colsample by Tree Used : 0.7
Testing Accuracy     : 0.957647

=== Classification Report (Testing) ===

                     precision    recall  f1-score   support

      Underweight       1.0000    0.9859    0.9929        71
           Normal       0.9677    0.9375    0.9524        64
       Overweight       0.8571    0.7895    0.8219        38
   Obesity_Type_I       0.8977    0.9634    0.9294        82
  Obesity_Type_II       0.9882    0.9882    0.9882       170

         accuracy                         0.9576       425
        macro avg       0.9422    0.9329    0.9378       425
     weighted avg       0.9579    0.9576    0.9574       425
```

**Figure 4.104. Classification Report of XGBoost + ROS colsample_bytree 0.70**

Figure 4.104 presents the classification report of the XGBoost + ROS model with colsample_bytree 0.70. Based on the evaluation results, the model achieved an accuracy of 0.957647, precision macro of 0.942169, recall macro of 0.932908, F1 macro of 0.936971, and balanced accuracy of 0.932908. These values indicate that the model had good classification ability across all target classes. However, the F1 macro and balanced accuracy values at colsample_bytree 0.70 were still lower than those obtained at colsample_bytree 0.90. This shows that using 70% of the features in each tree was not sufficiently optimal to produce the best performance for the XGBoost + ROS model. Therefore, colsample_bytree 0.70 was not selected as the best configuration.

**XGBoost + ROS colsample_bytree 0,8**

In the second scenario, the XGBoost + ROS model was tested using colsample_bytree 0.80. This value was the initial value used before the

colsample_bytree testing stage was conducted. With a value of 0.80, the model used 80% of the features in building each tree.

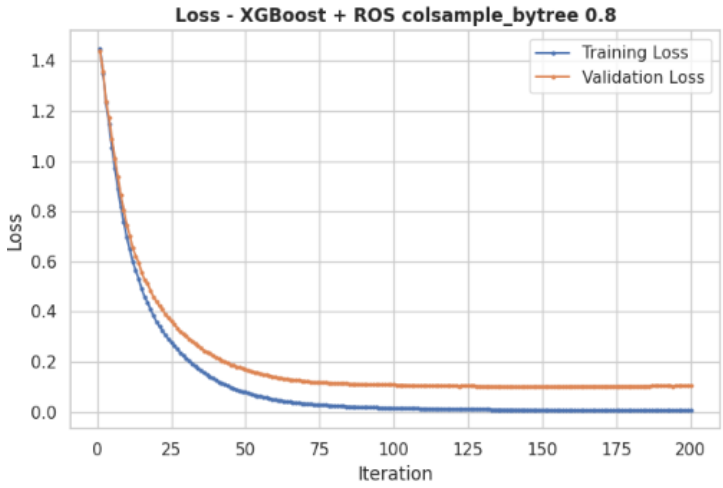



**Figure 4.105. Train Test Graph of XGBoost + ROS colsample_bytree 0.80**

Figure 4.105 shows the training loss and validation loss graph of the XGBoost + ROS model with colsample_bytree 0.80. Both curves tended to decrease with a gap that was not too large, indicating that the model learning process was stable and that the model was able to generalize well. However, based on the evaluation metrics, this configuration did not provide the best performance compared to the other colsample_bytree candidates.

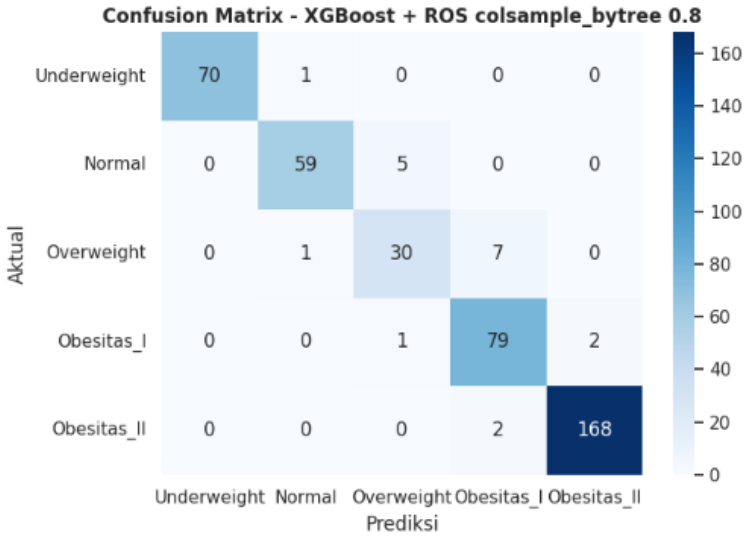



**Figure 4.106. Confusion Matrix of XGBoost + ROS colsample_bytree 0.80**

Figure 4.106 shows the confusion matrix of the XGBoost + ROS model with colsample_bytree 0.80. Based on the confusion matrix pattern, the model was still able to produce correct predictions for most testing data. The values on the main

diagonal remained more dominant than those outside the diagonal. However, compared to colsample_bytree 0.70 and 0.90, the performance at 0.80 was not the best. Classification errors were still observed in several classes with adjacent characteristics.

```
Model                : XGBoost + ROS
Test Type            : Colsample bytree Test
Split Used           : 80:20
Test Parameters      : Colsample bytree
Colsample bytree Used : 0.8
Testing Accuracy     : 0.955294

=== Classification Report (Testing) ===

                   precision    recall  f1-score   support

    Underweight       1.0000    0.9859    0.9929        71
         Normal       0.9672    0.9219    0.9440        64
     Overweight       0.8333    0.7895    0.8188        38
      Obesity_I       0.8977    0.9634    0.9294        82
     Obesity_II       0.9882    0.9882    0.9882       170

       accuracy                         0.9553       425
      macro avg       0.9373    0.9298    0.9331       425
   weighted avg       0.9557    0.9553    0.9551       425
```

**Figure 4.107. Classification Report of XGBoost + ROS colsample_bytree 0.80**

Figure 4.107 presents the classification report of the XGBoost + ROS model with colsample_bytree 0.80. In this scenario, the model achieved an accuracy of 0.955294, precision macro of 0.937302, recall macro of 0.929783, F1 macro of 0.933073, and balanced accuracy of 0.929783. These values indicate that the model still had good performance, but it was the lowest compared to the other two colsample_bytree values in the XGBoost + ROS model. The lower F1 macro and balanced accuracy values show that colsample_bytree 0.80 did not provide the best balanced performance across all target classes. Therefore, this value was not selected as the best configuration in the colsample_bytree testing stage for the XGBoost + ROS model.

**XGBoost + ROS colsample_bytree 0,9**

In the third scenario, the XGBoost + ROS model was tested using colsample_bytree 0.90. This value indicates that 90% of the features were used in building each tree. With a larger feature proportion, the model obtained more information from the input variables in each tree, while still maintaining variation because not all features were used completely.

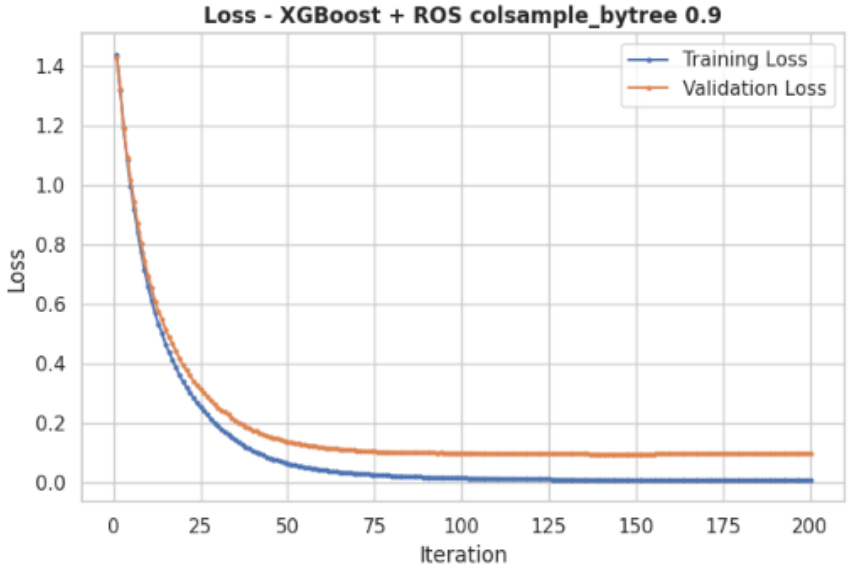



**Figure 4.108. Train Test Graph of XGBoost + ROS colsample_bytree 0.90**

Figure 4.108 shows the training loss and validation loss graph of the XGBoost + ROS model with colsample_bytree 0.90. This graph was used to observe model stability after increasing the proportion of features used in each tree. In this scenario, the model obtained the best evaluation metrics. If training loss and validation loss show a balanced decreasing pattern, the model can be considered to have a stable learning process. Therefore, colsample_bytree 0.90 was used as the best parameter in the final XGBoost + ROS combination.

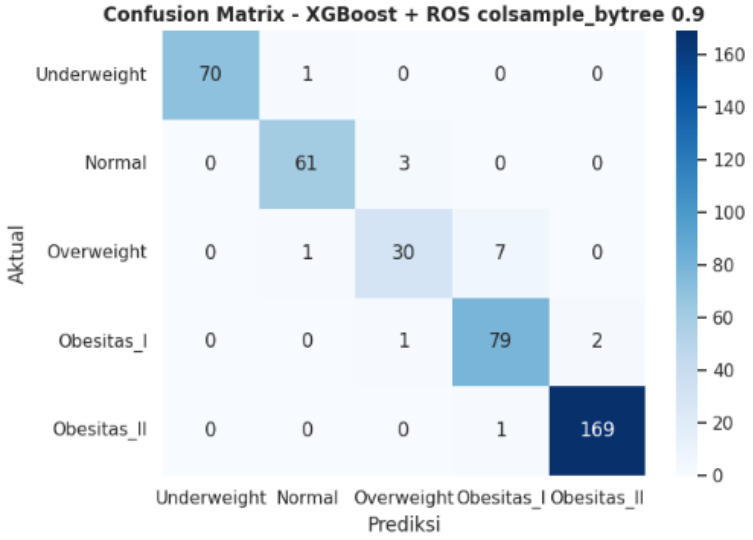



**Figure 4.109. Confusion Matrix of XGBoost + ROS colsample_bytree 0.90**

Figure 4.109 shows the confusion matrix of the XGBoost + ROS model with colsample_bytree 0.90. Based on the confusion matrix pattern, the values on the

main diagonal appeared dominant and showed that the model produced many correct predictions. Prediction errors could still occur in certain classes, especially Overweight, but overall, the classification pattern showed better results compared to colsample_bytree 0.70 and 0.80.

```
Model              : XGBoost + ROS
Type of Test       : Colsample Bytree Test
Split Ratio Used   : 80:20
Test Parameter     : colsample_bytree
Value Used         : 0.9
Testing Accuracy   : 0.962353

=== Classification Report (Testing) ===

                  precision    recall  f1-score   support

   Underweight       1.0000    0.9859    0.9929        71
        Normal       0.9683    0.9531    0.9686        64
    Overweight       0.8824    0.7895    0.8333        38
     Obesity_I       0.9080    0.9634    0.9349        82
    Obesity_II       0.9883    0.9941    0.9912       170

      accuracy                         0.9624       425
     macro avg       0.9494    0.9372    0.9426       425
  weighted avg       0.9623    0.9624    0.9619       425
```

**Figure 4.110. Classification Report of XGBoost + ROS colsample_bytree 0.90**

Figure 4.110 presents the classification report of the XGBoost + ROS model with colsample_bytree 0.90. Based on the evaluation results, the model achieved an accuracy of 0.962353, precision macro of 0.949391, recall macro of 0.937209, F1 macro of 0.942597, and balanced accuracy of 0.937209. These values were the highest results in the colsample_bytree testing stage for the XGBoost + ROS model. The higher accuracy indicates that the overall number of correct predictions increased. In addition, the higher F1 macro and balanced accuracy values show that the model was able to provide more balanced performance across all target classes. Therefore, colsample_bytree 0.90 was selected as the best value for the XGBoost + ROS model.

**LightGBM + ROS colsample_bytree 0,7**

In the fourth scenario, the LightGBM + ROS model was tested using colsample_bytree 0.70. This value indicates that 70% of the features were used in building each tree. This test was conducted to determine whether using only a portion of the features could help the model produce better generalization.

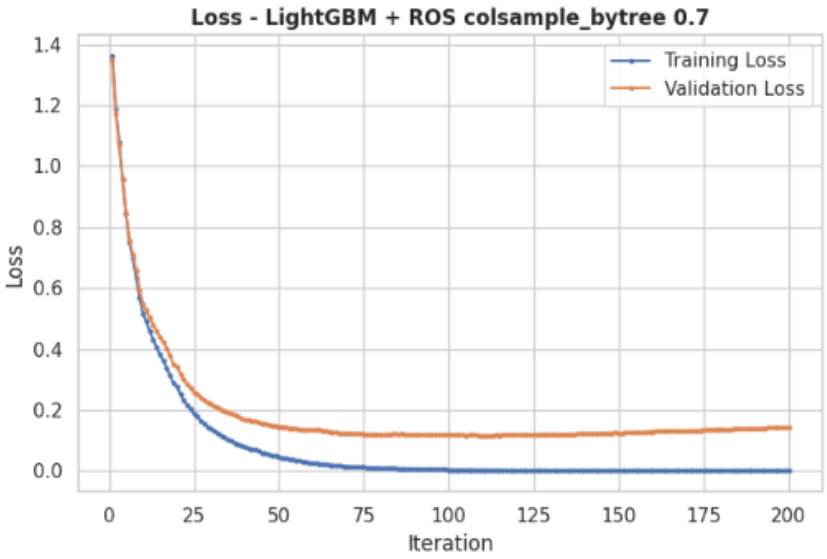



**Figure 4.111. Train Test Graph of LightGBM + ROS colsample_bytree 0.70**

Figure 4.111 shows the training loss and validation loss graph of the LightGBM + ROS model with colsample_bytree 0.70. The graph shows the model learning process during training. At this value, the model was still able to learn from the training data, but the evaluation results showed that its performance was not as good as colsample_bytree 0.80. Therefore, colsample_bytree 0.70 was not the most optimal configuration.

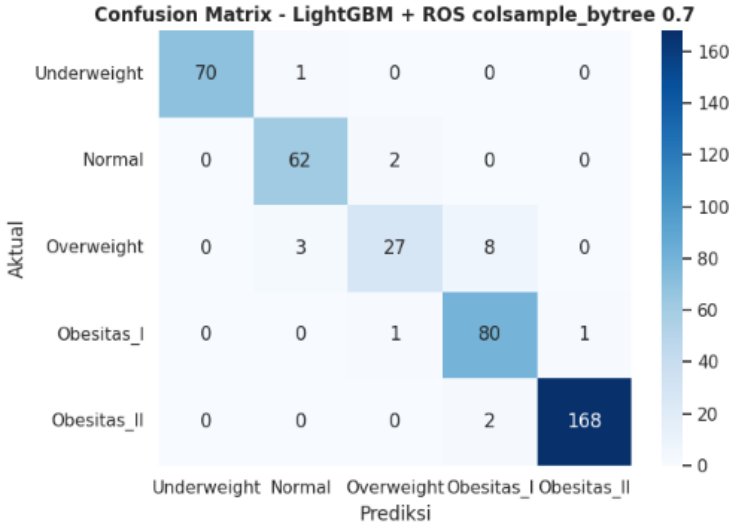



**Figure 4.112. Confusion Matrix of LightGBM + ROS colsample_bytree 0.70**

Figure 4.112 shows the confusion matrix of the LightGBM + ROS model with colsample_bytree 0.70. Based on the confusion matrix pattern, the model was still able to produce correct predictions for most testing data. However, prediction errors were still found in several classes, especially those with adjacent

characteristic boundaries. Compared to colsample_bytree 0.80 and 0.90, the performance at 0.70 was still lower.

```
Model              : LightGBM + ROS
Type of Test       : Column Sample by Tree Test
Split Used         : 80:20
Parameter Tested   : colsample_bytree
Value Used         : 0.7
Testing Accuracy   : 0.957647

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

  Underweight     1.0000    0.9859    0.9929        71
       Normal     0.9394    0.9688    0.9538        64
   Overweight     0.9000    0.7105    0.7941        38
    Obesity_I     0.8889    0.9756    0.9302        82
   Obesity_II     0.9941    0.9882    0.9912       170

     accuracy                         0.9576       425
    macro avg     0.9445    0.9258    0.9325       425
 weighted avg     0.9581    0.9576    0.9565       425
```

**Figure 4.113. Classification Report LightGBM + ROS colsamplebytree 0.70**

Figure 4.113 presents the classification report of the LightGBM + ROS model with colsample_bytree 0.70. Based on the evaluation results, the model achieved an accuracy of 0.957647, precision macro of 0.944473, recall macro of 0.925807, F1 macro of 0.932451, and balanced accuracy of 0.925807. These values show that the model still had good performance, but it was not the best result in the colsample_bytree testing stage. The F1 macro and balanced accuracy values at colsample_bytree 0.70 were lower than those at colsample_bytree 0.80 and 0.90. This indicates that using 70% of the features was not sufficiently optimal to help LightGBM + ROS recognize all target classes evenly. Therefore, colsample_bytree 0.70 was not selected as the best configuration.

**LightGBM + ROS colsample_bytree 0,8**

In the fifth scenario, the LightGBM + ROS model was tested using colsample_bytree 0.80. This value was the initial value and also the value used in the previous stage. With colsample_bytree 0.80, the model used 80% of all features in each decision tree building process. This relatively high feature proportion allowed the model to utilize more information available in the dataset to recognize classification patterns. In addition, this setting was expected to maintain a balance

between the model's ability to capture relationships among features and reduce dependence on all features simultaneously, so that model performance could remain optimal.

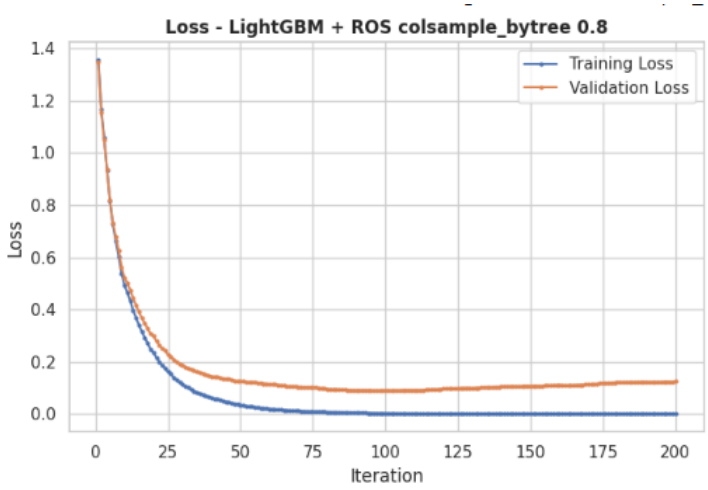



**Figure 4.114. Train Test Graph of LightGBM + ROS colsample_bytree 0.80**

Figure 4.114 shows the training loss and validation loss graph of the LightGBM + ROS model with colsample_bytree 0.80. Both curves showed a stable learning process with a loss gap that was not too large. Based on the evaluation metrics, colsample_bytree 0.80 was used as the best parameter in the final LightGBM + ROS combination.

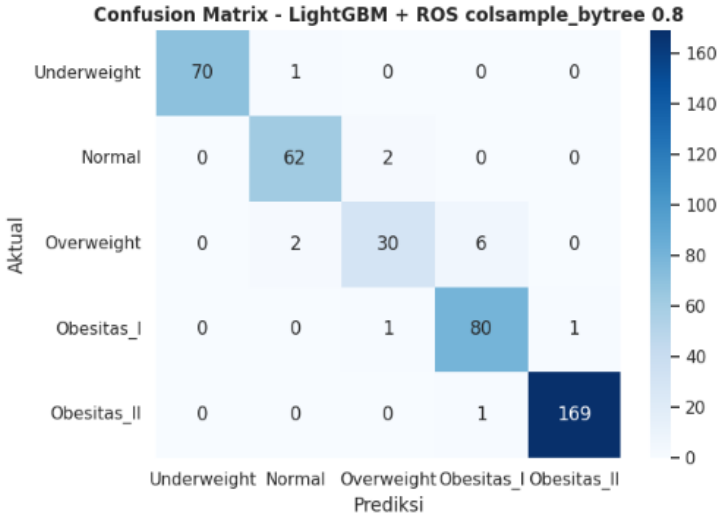



**Figure 4.115. Confusion Matrix of LightGBM + ROS colsample_bytree 0.80**

Figure 4.115 shows the confusion matrix of the LightGBM + ROS model with colsample_bytree 0.80. The main diagonal values appeared dominant, indicating that most testing data were classified according to their actual classes.

Although errors were still found in the Overweight class, overall, this configuration showed the best performance compared to the other colsample_bytree candidates.

```
Model              : LightGBM + ROS
Type of Test       : Colsample_bytree Test
Split Used         : 80:20
Test Parameter     : colsample_bytree
Value Used         : 0.8
Testing Accuracy   : 0.967059

=== Classification Report (Testing) ===

                precision    recall  f1-score   support

 Underweight     1.0000    0.9859    0.9929        71
      Normal     0.9538    0.9688    0.9612        64
  Overweight     0.9091    0.7895    0.8451        38
  Obesitys_I     0.9195    0.9756    0.9467        82
 Obesitys_II     0.9941    0.9941    0.9941       170

    accuracy                         0.9671       425
   macro avg     0.9553    0.9428    0.9488       425
weighted avg     0.9670    0.9671    0.9665       425
```

**Figure 4.116. Classification Report LightGBM + ROS colsample_bytree 0.80**

Figure 4.116 presents the classification report of the LightGBM + ROS model with colsample_bytree 0.80. In this scenario, the model achieved an accuracy of 0.967059, precision macro of 0.955319, recall macro of 0.942773, F1 macro of 0.948016, and balanced accuracy of 0.942773. These values were the best results in the colsample_bytree testing stage for the LightGBM + ROS model. The high F1 macro and balanced accuracy values show that colsample_bytree 0.80 was able to provide a good balance between the model's ability to produce correct predictions and recognize actual data in each class. Therefore, colsample_bytree 0.80 was selected as the best value for the LightGBM + ROS model.

**LightGBM + ROS colsample_bytree 0,9**

In the sixth scenario, the LightGBM + ROS model was tested using colsample_bytree 0.90. This value indicates that 90% of the features were used in building each decision tree. Compared to colsample_bytree 0.70 and 0.80, the value of 0.90 provided a larger feature proportion, giving the model broader information in determining node splits at each iteration.

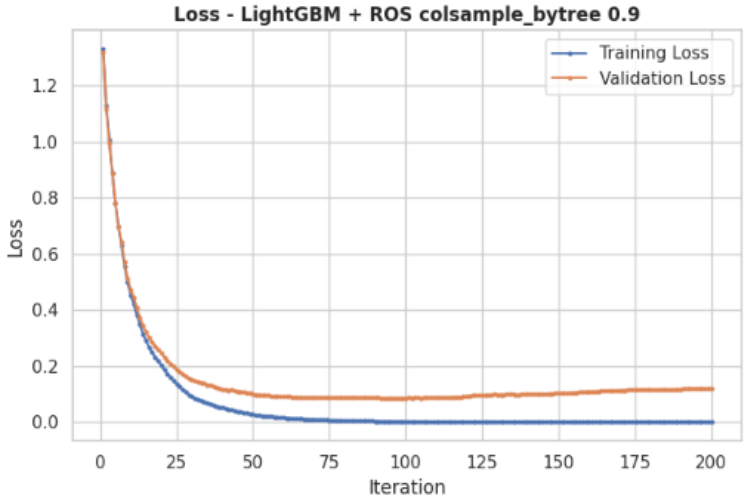



**Figure 4.117. Train Test Graph of LightGBM + ROS colsample_bytree 0.90**

Figure 4.117 shows the training loss and validation loss graph of the LightGBM + ROS model with colsample_bytree 0.90. This graph was used to observe whether increasing the feature proportion in each tree produced better results. In this scenario, the model still showed stable performance, but the evaluation results did not exceed colsample_bytree 0.80. Therefore, colsample_bytree 0.90 was not the best configuration.

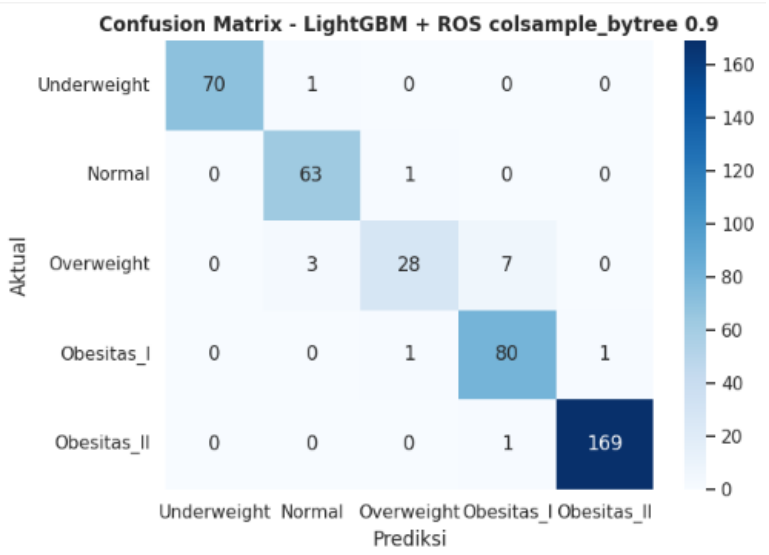



**Figure 4.118. Confusion Matrix of LightGBM + ROS colsample_bytree 0.90**

Figure 4.118 shows the confusion matrix of the LightGBM + ROS model with colsample_bytree 0.90. Based on the confusion matrix pattern, the model was still able to produce good correct predictions on the main diagonal. However, the evaluation results showed that the performance of colsample_bytree 0.90 did not

exceed colsample_bytree 0.80. Prediction errors could still occur in certain classes, especially those with adjacent characteristics.

```
Model              : LightGBM + ROS
Type of Test       : Colsample_Bytree Testing
Split Used         : 80:20
Parameter Tested   : colsample_bytree
Value(s) Used      : 0.9
Testing Accuracy   : 0.964786

=== Classification Report (Testing) ===
                precision    recall  f1-score   support

   Underweight     1.0000    0.9859    0.9929        71
        Normal     0.9403    0.9844    0.9618        64
    Overweight     0.9333    0.7368    0.8235        38
     Obesity_I     0.9891    0.9756    0.9412        82
    Obesity_II     0.9941    0.9941    0.9941       170
      accuracy                         0.9647       425
     macro avg     0.9554    0.9354    0.9427       425
  weighted avg     0.9652    0.9647    0.9636       425
```

**Figure 4.119. Classification Report LightGBM + ROS colsample_bytree 0.90**

Figure 4.119 presents the classification report of the LightGBM + ROS model with colsample_bytree 0.90. The model achieved an accuracy of 0.964706, precision macro of 0.955368, recall macro of 0.935372, F1 macro of 0.942713, and balanced accuracy of 0.935372. These values were still high, but lower than colsample_bytree 0.80 in terms of accuracy, recall macro, F1 macro, and balanced accuracy. Although the precision macro value was slightly higher, colsample_bytree 0.80 was better in maintaining balanced performance across classes. Therefore, colsample_bytree 0.90 was not selected as the best configuration.

**Table 4.12. Comparison of Colsample_bytree on XGBoost**

| Description | XGBoost + ROS colsample_bytree 0.70 | XGBoost + ROS colsample_bytree 0.80 | XGBoost + ROS colsample_bytree 0.90 |
|---|---|---|---|
| Accuracy | 0.957647 | 0.955294 | 0.962353 |
| Precision Macro | 0.942169 | 0.937302 | 0.949391 |
| Recall Macro | 0.932908 | 0.929783 | 0.937209 |
| F1 Macro | 0.936971 | 0.933073 | 0.942597 |

| Description | XGBoost + ROS colsample_bytree 0.70 | XGBoost + ROS colsample_bytree 0.80 | XGBoost + ROS colsample_bytree 0.90 |
|---|---|---|---|
| Balanced Accuracy | 0.932908 | 0.929783 | 0.937209 |
| Train Time | 0.859211 | 0.838060 | 0.802957 |
| Status | Not Selected | Not Selected | Selected |

**Table 4.13. Comparison of Colsample_bytree on LightGBM**

| Keterangan | LightGBM + ROS colsample_bytree 0,7 | LightGBM + ROS colsample_bytree 0,8 | LightGBM + ROS colsample_bytree 0,9 |
|---|---|---|---|
| Accuracy | 0,957647 | 0,967059 | 0,964706 |
| Precision Macro | 0,944473 | 0,955319 | 0,955368 |
| Recall Macro | 0,925807 | 0,942773 | 0,935372 |
| F1 Macro | 0,932451 | 0,948016 | 0,942713 |
| Balanced Accuracy | 0,925807 | 0,942773 | 0,935372 |
| Train Time | 0,899487 | 0,944062 | 0,884321 |
| Status | Tidak Dipilih | Dipilih | Tidak Dipilih |

Based on Tables 4.12 and 4.13, the best colsample_bytree value for the XGBoost + ROS model was 0.90. This value produced an accuracy of 0.962353, precision macro of 0.949391, recall macro of 0.937209, F1 macro of 0.942597, and balanced accuracy of 0.937209. These values were higher than colsample_bytree 0.70 and 0.80 across all main evaluation metrics. In addition, the training time at colsample_bytree 0.90 was also lower than the other two candidates. Therefore, colsample_bytree 0.90 was selected as the best configuration for XGBoost + ROS. In the LightGBM + ROS model, the best colsample_bytree value was 0.80. This value produced an accuracy of 0.967059, precision macro of 0.955319, recall macro of 0.942773, F1 macro of 0.948016, and balanced accuracy of 0.942773. These values were higher than colsample_bytree 0.70 and 0.90 in terms of accuracy, recall macro, F1 macro, and balanced accuracy. Although the precision macro value at

colsample_bytree 0.90 was slightly higher, colsample_bytree 0.80 was still selected because it provided more balanced overall performance.

After all manual hyperparameter testing stages were completed, one best hyperparameter combination was obtained for each model based on the highest evaluation metric values at each testing stage. The selected hyperparameter value from one stage was used as a fixed parameter in the next testing stage until the final configuration was obtained. The best hyperparameter combinations were then used in the final training and evaluation process of the XGBoost + Random Oversampling (ROS) and LightGBM + Random Oversampling (ROS) models.

**Table 4.14. Selected Manual Split and Hyperparameters**

| Hyperparameter | XGBoost + ROS | LightGBM + ROS |
|---|---|---|
| Split Data | 80:20 | 80:20 |
| learning_rate | 0,10 | 0,10 |
| max_depth | 7 | 7 |
| n_estimators | 200 | 200 |
| subsample | 0,90 | 0,80 |
| colsample_bytree | 0,90 | 0,80 |

### 4.2.8. Final Evaluation of XGBoost + ROS and LightGBM + ROS

After all stages of manual hyperparameter testing were completed, the best configuration for each model, namely XGBoost + ROS and LightGBM + ROS, was obtained. These best configurations were then used as the basis for the final evaluation to examine model performance after the application of Random Oversampling. At this stage, the discussion focuses on comparing the final results of both models based on the main evaluation metrics, including accuracy, precision macro, recall macro, F1 macro, balanced accuracy, and training time.

**Table 4.15. Final Evaluation of XGBoost + ROS and LightGBM + ROS**

| Description | LightGBM + ROS | XGBoost + ROS |
|---|---|---|
| Accuracy | 0.967059 | 0.962353 |

| Description | LightGBM + ROS | XGBoost + ROS |
|---|---|---|
| Precision Macro | 0.955319 | 0.949391 |
| Recall Macro | 0.942773 | 0.937209 |
| F1 Macro | 0.948016 | 0.942597 |
| Balanced Accuracy | 0.942773 | 0.937209 |
| Train Time | 3.941868 | 0.842885 |
| Status | Best | |

Based on Table 4.15, LightGBM + ROS achieved the best performance compared to XGBoost + ROS using the best manual hyperparameter configuration. LightGBM + ROS obtained an accuracy of 0.967059, while XGBoost + ROS obtained an accuracy of 0.962353. On 425 testing data, LightGBM + ROS produced 411 correct predictions, while XGBoost + ROS produced 409 correct predictions. This difference indicates that LightGBM + ROS was slightly superior in overall classification accuracy. In addition to accuracy, LightGBM + ROS also achieved higher values in precision macro, recall macro, F1 macro, and balanced accuracy. The precision macro value of 0.955319 shows that LightGBM + ROS had better prediction precision across all target classes. The recall macro and balanced accuracy values of 0.942773 indicate that this model was better able to recognize actual data in each class more evenly. This is important because this study involved multiclass classification with an imbalanced class distribution.

The F1 macro value of LightGBM + ROS, which was 0.948016, was also higher than that of XGBoost + ROS, which was 0.942597. This indicates that LightGBM + ROS had a better balance between precision and recall across all classes. Although the training time of LightGBM + ROS was longer than that of XGBoost + ROS, the improvement in all main evaluation metrics shows that this model was more optimal for use in this study. Therefore, based on the final evaluation results, LightGBM + Random Oversampling was selected as the best model. This model was chosen because it achieved the highest values in all main evaluation metrics, produced more correct predictions, and showed more balanced classification ability across all target classes.

### 4.2.9. XGBoost and LightGBM without Random Oversampling

This test analyzed the performance of XGBoost and LightGBM without Random Oversampling (ROS). The training data remained imbalanced, while the testing data used the same distribution as previous scenarios. This test was used to measure the baseline ability of each model and compare the effect of ROS in the next analysis stage.

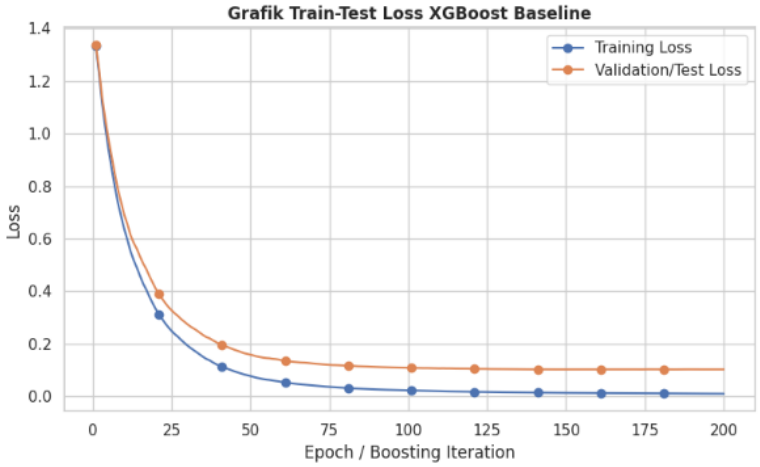



**Figure 4.120. Train Test Graph of XGBoost without ROS**

Figure 4.120 shows the training and validation loss graph of the XGBoost model without ROS. The model still learned fairly stably, as both losses tended to decrease. However, the gap between training and testing performance indicates that the model's generalization was less optimal. This may occur because the imbalanced training data made the model more likely to learn classes with larger sample sizes.

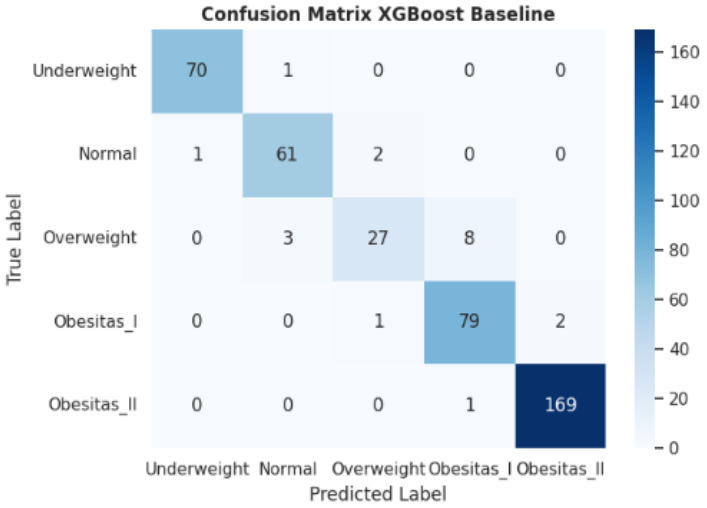



**Figure 4.121. Confusion Matrix of XGBoost without ROS**

Figure 4.121 shows the confusion matrix of the XGBoost model without ROS. Based on the confusion matrix, most testing data were classified correctly, as indicated by the dominant values on the main diagonal. However, classification errors were still observed in several classes, especially the Overweight class. The Overweight class tended to be misclassified as Normal and Obesitas_I because it has adjacent characteristics. In addition, the smaller number of Overweight data compared to other classes may have made the model less optimal in recognizing the pattern of this class. This shows that without the application of ROS, XGBoost still had difficulty maintaining balanced classification performance in minority classes.

```
Scenario          : XGBoost Baseline
Model             : XGBoost
ROS               : No
Split             : 80:20
Accuracy          : 0.955294
Precision Macro   : 0.942082
Recall Macro      : 0.921420
F1-Macro          : 0.929277
Balanced Accuracy : 0.921420
Train Time        : 0.576828 seconds

=== Classification Report (Testing) ===
              precision    recall  f1-score   support

 Underweight   0.985915  0.985915  0.985915        71
      Normal   0.938462  0.953125  0.945736        64
  Overweight   0.900000  0.710526  0.794118        38
   Obesity_I   0.897727  0.963415  0.929412        82
  Obesity_II   0.988304  0.994118  0.991202       170

    accuracy                       0.955294       425
   macro avg   0.942082  0.921420  0.929277       425
weighted avg   0.955028  0.955294  0.953929       425
```

**Figure 4.122. Classification Report of XGBoost without ROS**

Figure 4.122 presents the classification report of the XGBoost model without ROS. Based on the evaluation results, the model achieved an accuracy of 0.955294, precision macro of 0.942082, recall macro of 0.921420, F1 macro of 0.929277, and balanced accuracy of 0.921420. The high accuracy value shows that XGBoost without ROS was still able to produce a large number of correct predictions across the overall testing data. However, the lower recall macro and balanced accuracy values compared to the XGBoost + ROS scenario indicate that the model's ability to recognize all classes evenly was not yet optimal. Therefore,

XGBoost without ROS can be considered to have good baseline performance, but it was still not fully optimal in handling class imbalance.

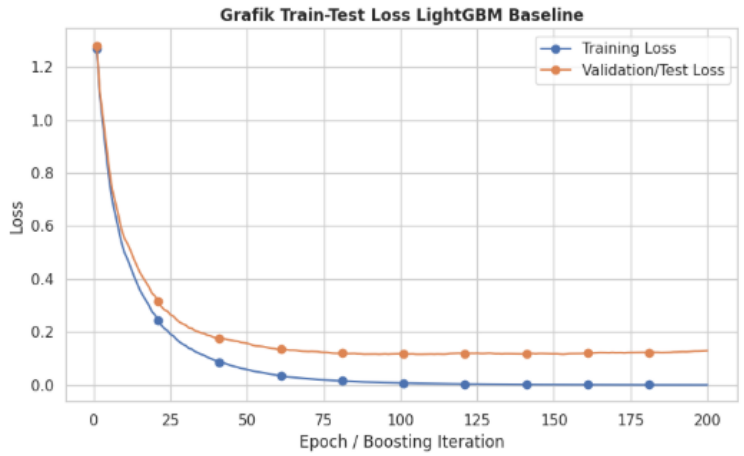



**Figure 4.123. Train Test Graph of LightGBM without ROS**

Figure 4.123 shows the training loss and validation loss graph of the LightGBM model without ROS. This graph was used to observe the stability of the LightGBM learning process on training data with its original distribution. Based on the graph pattern, LightGBM without ROS still showed a stable learning process because the training loss and validation loss decreased. However, there was still a gap between the training data and testing data. This gap indicates that the model had not fully achieved optimal generalization on imbalanced data. This condition shows that the class distribution in the training data still affected the model's ability to recognize all target classes.

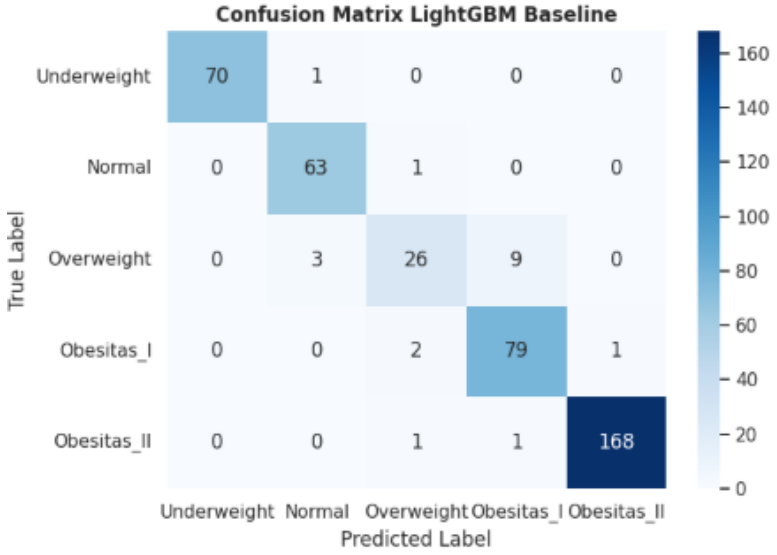



**Figure 4.124. Confusion Matrix of LightGBM without ROS**

Figure 4.124 shows the confusion matrix of the LightGBM model without ROS. Based on the confusion matrix, most testing data were classified correctly. However, classification errors still occurred in classes with adjacent characteristics, especially the Overweight class. The Overweight class still had the potential to be predicted as Normal or Obesitas_I. This indicates that without the application of Random Oversampling, LightGBM was still not fully optimal in distinguishing minority classes or classes with closely related characteristic boundaries.

```
Scenario          : LightGBM Baseline
Model             : LightGBM
ROS               : No
Split             : 80:20
Accuracy          : 0.955294
Precision Macro   : 0.937738
Recall Macro      : 0.921230
F1-Macro          : 0.926915
Balanced Accuracy : 0.921230
Train Time        : 0.609013 seconds

=== Classification Report (Testing) ===
              precision    recall  f1-score   support

 Underweight   1.000000  0.985915  0.992908        71
      Normal   0.940299  0.984375  0.961832        64
  Overweight   0.866667  0.684211  0.764706        38
  Obesity_I    0.887640  0.963415  0.923977        82
 Obesity_II    0.994083  0.988235  0.991150       170

    accuracy                       0.955294       425
   macro avg   0.937738  0.921230  0.926915       425
weighted avg   0.955042  0.955294  0.953822       425
```

**Figure 4.125. Classification Report of LightGBM without ROS**

Figure 4.125 presents the classification report of the LightGBM model without ROS. Based on the evaluation results, the model achieved an accuracy of 0.955294, precision macro of 0.937738, recall macro of 0.921230, F1 macro of 0.926915, and balanced accuracy of 0.921230. These values indicate that LightGBM without ROS had fairly good overall classification performance. However, the F1 macro and balanced accuracy values show that performance balance across classes was not yet optimal. This indicates that although LightGBM was able to produce high accuracy, the model still required class imbalance handling so that performance across each class could become more balanced.

**Table 4.16. Comparison Results of XGBoost and LightGBM without ROS**

| Description | XGBoost without ROS | LightGBM without ROS |
|---|---|---|
| Accuracy | 0.955294 | 0.955294 |
| Precision Macro | 0.942082 | 0.937738 |
| Recall Macro | 0.921420 | 0.921230 |
| F1 Macro | 0.929277 | 0.926915 |
| Balanced Accuracy | 0.921420 | 0.921230 |
| Train Time | 0.576828 | 0.609013 |
| Status | Baseline | Baseline |

Based on Table 4.16, XGBoost without ROS and LightGBM without ROS achieved the same accuracy value of 0.955294. This shows that both models had relatively balanced classification ability when viewed only from the total number of correct predictions. However, when analyzed using macro based metrics, XGBoost without ROS showed slightly better performance than LightGBM without ROS. XGBoost achieved a precision macro of 0.942082, recall macro of 0.921420, F1 macro of 0.929277, and balanced accuracy of 0.921420. Meanwhile, LightGBM achieved a precision macro of 0.937738, recall macro of 0.921230, F1 macro of 0.926915, and balanced accuracy of 0.921230.

These differences indicate that in the scenario without ROS, XGBoost had a slightly better ability to maintain balanced performance across classes compared to LightGBM. However, the performance gap between the two models was not substantial. Both models still showed the same weakness, namely classification errors in the Overweight class. This class became the most difficult to recognize because it had fewer data and was located between the Normal and Obesitas_I classes. Therefore, the testing results without ROS show that both models still required a class imbalance handling strategy to improve classification ability across all target classes. Overall, the testing without Random Oversampling shows that XGBoost and LightGBM were already able to produce fairly good baseline performance. However, the recall macro, F1 macro, and balanced accuracy values, which were still lower than those in the ROS scenario, indicate that the imbalanced

class distribution still affected the balance of model performance. Thus, the results in this subsection serve as the comparison basis for analyzing the effect of Random Oversampling on improving the performance of XGBoost and LightGBM in the following subsection.

### 4.2.10. Analysis and Determination of the Best Model

This subsection discusses the final analysis used to determine the best model based on four main scenarios, namely XGBoost without Random Oversampling, XGBoost with Random Oversampling, LightGBM without Random Oversampling, and LightGBM with Random Oversampling. The determination of the best model was not only based on accuracy, but also considered precision macro, recall macro, F1 macro, balanced accuracy, error patterns in the confusion matrix, and the effect of Random Oversampling on performance balance across classes. This approach was used because this study involved multiclass classification with an imbalanced class distribution. Therefore, the best model should be able to produce high performance while maintaining prediction ability for each target class.

**Table 4.17. Comparison of Models without ROS and with ROS**

| Description | XGBoost without ROS | XGBoost + ROS | LightGBM without ROS | LightGBM + ROS |
|---|---|---|---|---|
| Accuracy | 0.955294 | 0.962353 | 0.955294 | 0.967059 |
| Precision Macro | 0.942082 | 0.949391 | 0.937738 | 0.955319 |
| Recall Macro | 0.921420 | 0.937209 | 0.921230 | 0.942773 |
| F1 Macro | 0.929277 | 0.942597 | 0.926915 | 0.948016 |
| Balanced Accuracy | 0.921420 | 0.937209 | 0.921230 | 0.942773 |
| Train Time | 0.576828 | 0.731692 | 0.609013 | 0.802853 |
| Status | Without ROS | Improved | Without ROS | Best |

Based on Table 4.17, LightGBM + ROS achieved the highest performance across all main evaluation metrics. This model obtained an accuracy of 0.967059, precision macro of 0.955319, recall macro of 0.942773, F1 macro of 0.948016, and balanced accuracy of 0.942773. These values were higher than those of the other

three scenarios. This indicates that LightGBM + ROS was not only superior in terms of the overall number of correct predictions, but also better in maintaining prediction precision and class recognition ability more evenly.

In multiclass classification with an imbalanced data distribution, accuracy alone is not sufficient to determine the best model. Therefore, macro based metrics and balanced accuracy became important considerations. Precision macro was used to observe the average prediction precision across all classes, while recall macro and balanced accuracy were used to evaluate the model's ability to recognize actual data from each class. F1 macro was used to examine the balance between precision and recall across all target classes. Based on these metrics, LightGBM + ROS showed the most balanced performance because it achieved the highest values in precision macro, recall macro, F1 macro, and balanced accuracy.

When compared with the scenarios without ROS, the application of Random Oversampling improved the performance of both models. In XGBoost, accuracy increased from 0.955294 to 0.962353, F1 macro increased from 0.929277 to 0.942597, and balanced accuracy increased from 0.921420 to 0.937209. In LightGBM, accuracy increased from 0.955294 to 0.967059, F1 macro increased from 0.926915 to 0.948016, and balanced accuracy increased from 0.921230 to 0.942773. These improvements show that Random Oversampling helped the models recognize classes that were previously underrepresented in the training data..

**Table 4.18. Performance Difference after Applying ROS**

| Model | Delta Accuracy | Delta Precision Macro | Delta Recall Macro | Delta F1 Macro | Delta Balanced Accuracy |
|---|---|---|---|---|---|
| XGBoost | 0.007059 | 0.007309 | 0.015789 | 0.013320 | 0.015789 |
| LightGBM | 0.011765 | 0.017581 | 0.021543 | 0.021101 | 0.021543 |

Based on Table 4.18, the effect of Random Oversampling on LightGBM was greater than on XGBoost. LightGBM achieved an accuracy improvement of 0.011765, while XGBoost improved by 0.007059. The increase in F1 macro for LightGBM was also higher, at 0.021101, compared to XGBoost at 0.013320. In addition, the increase in balanced accuracy for LightGBM was 0.021543, which

was higher than XGBoost at 0.015789. These results indicate that LightGBM was more responsive to the application of Random Oversampling, especially in improving balanced performance across classes.

The confusion matrix analysis also supports these findings. Based on Figure 4.123 and Figure 4.124, the application of ROS to XGBoost increased the number of correct predictions from 406 to 409 out of 425 testing data. Improvement was also observed in the Overweight class, which was previously more difficult to predict because it had fewer samples and characteristics close to the Normal and Obesitas_I classes. In the XGBoost scenario without ROS, the Overweight class was still more frequently misclassified into adjacent classes. After ROS was applied, the number of correct predictions in this class increased, indicating that balancing the training data helped the model recognize minority class patterns more effectively.

A similar pattern was also observed in LightGBM. Based on Figure 4.129 and Figure 4.130, the number of correct predictions increased from 406 in the LightGBM without ROS scenario to 411 in the LightGBM + ROS scenario. This was the highest number of correct predictions among all testing scenarios. Classification errors were still found in the Overweight class, but the number of errors was relatively lower after the application of ROS. This shows that LightGBM + ROS had better ability to distinguish classes with adjacent characteristics.

In terms of performance balance across classes, LightGBM + ROS was also the strongest model. The recall macro and balanced accuracy values of 0.942773 indicate that the model was able to recognize each target class more evenly. This is important because the classes in this study did not have the same number of data. The Overweight class had fewer samples than other classes, making it more difficult for the model to recognize. With the application of ROS, the training data distribution became more balanced, giving the model a better opportunity to learn minorityclass patterns.

Although LightGBM + ROS had a slightly higher training time than the other scenarios, which was 0.802853 seconds, this training time was still relatively small and proportional to the performance improvement achieved. In the context of this study, the main priority was not only training speed, but also the model's ability

to produce accurate and balanced predictions across all target classes. Therefore, improvements in accuracy, F1 macro, and balanced accuracy became more important considerations in determining the best model.

Based on the overall analysis, Random Oversampling was proven to have a positive effect on the performance of XGBoost and LightGBM. This effect was not only reflected in the improvement of accuracy, but also in the increase of recall macro, F1 macro, and balanced accuracy. This shows that ROS helped improve classification balance across classes, especially in classes with fewer data. Between the two models, LightGBM benefited more from the application of ROS because it produced greater metric improvements than XGBoost. Therefore, the best model in this study was LightGBM + Random Oversampling. This model was selected because it achieved the highest values across all main evaluation metrics, produced the highest number of correct predictions based on the confusion matrix, showed better performance balance across classes, and provided the greatest performance improvement after the application of ROS. Therefore, LightGBM + ROS was used as the final model in the obesity level classification system implementation in this study.

### 4.3. System Interface Design Implementation Results

After the best model was obtained through the testing process, the next stage was to implement the research results into a simple Streamlit based web system. This implementation was carried out to show how the obesity level classification model can be used through an interface that is easier for users to operate. Through this system, users can enter individual characteristics and lifestyle habit data, obtain obesity category prediction results, and view the model output in a more informative format. The system interface implementation in this study refers to the design described in Chapter III, consisting of the Login, Register, Home, Input Form, and Prediction Result pages.

#### 4.3.1. Login Page Implementation Results

The Login page is the initial system display that appears when users first open the application. This page is used to enter an email and password before users can access the system. The page provides an email input field, a password input

field, a Login button, and a Register button to navigate to the registration page if the user does not yet have an account. The implementation of this page aims to provide a more structured access flow before users use the obesity prediction feature.

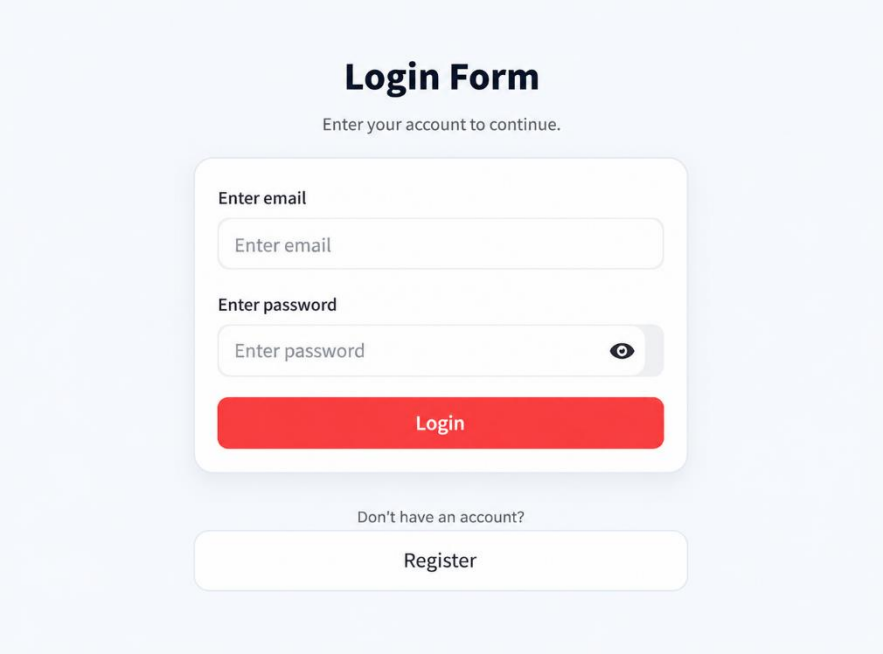



**Figure 4.126. Login Page Implementation Result**

Based on Figure 4.126, the Login page is designed simply with a focus on the user authentication process. The center of the page displays the Form Login title, a short description for entering the system, an email field, a password field, a Login button, and a Register button. With this display, users can immediately understand the page function and easily log in to the system.

### 4.3.2. Register Page Implementation Results

After users click the Register button on the Login page, the system displays the Register page. This page is used to create a new account by entering an email, password, and password confirmation. After the registration process is successful, the system displays a short notification indicating that registration has been completed, and then users are directed back to the Login page to access the system. The implementation of this page completes the basic usage flow before users access the available prediction feature.

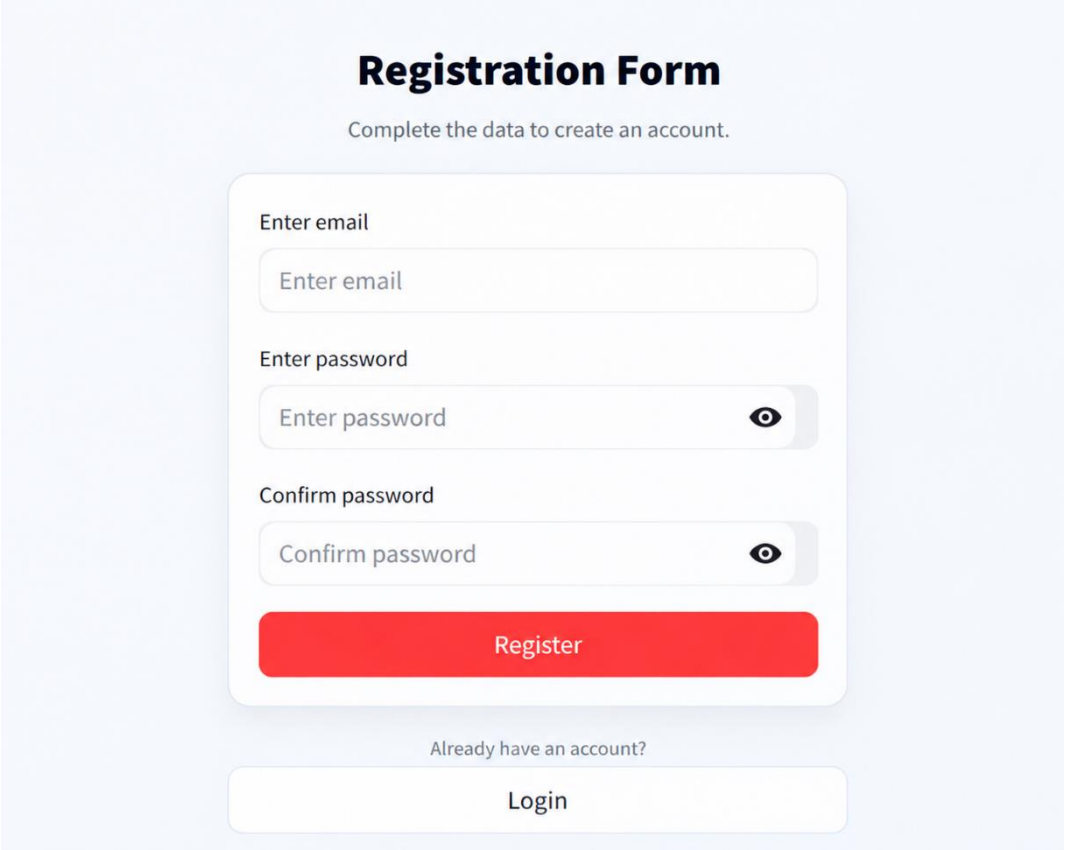



**Figure 4.127. Register Page Implementation Result**

Based on Figure 4.127, the Register page displays the Form Registrasi title and three main input components, namely email, password, and password confirmation. At the bottom of the page, there is a Register button to save the registration data and a Login button to return to the authentication page. Therefore, this page allows new users to create an account before using the system.

### 4.3.3. Home Page Implementation Results

After users successfully log in, the system displays the Home page. In the latest version, this page is designed more concisely by displaying the system title in the header, followed by one main section containing an explanation of the obesity prediction system, the purpose of using the system, a statement that the system output is only informative, and an explanation of the five obesity categories based on the Asia Pacific BMI classification, namely Underweight, Normal, Overweight, Obesity I, and Obesity II. At the bottom of the page, there is a Start Prediction Test button to navigate to the data input page and a Logout button to exit the system.

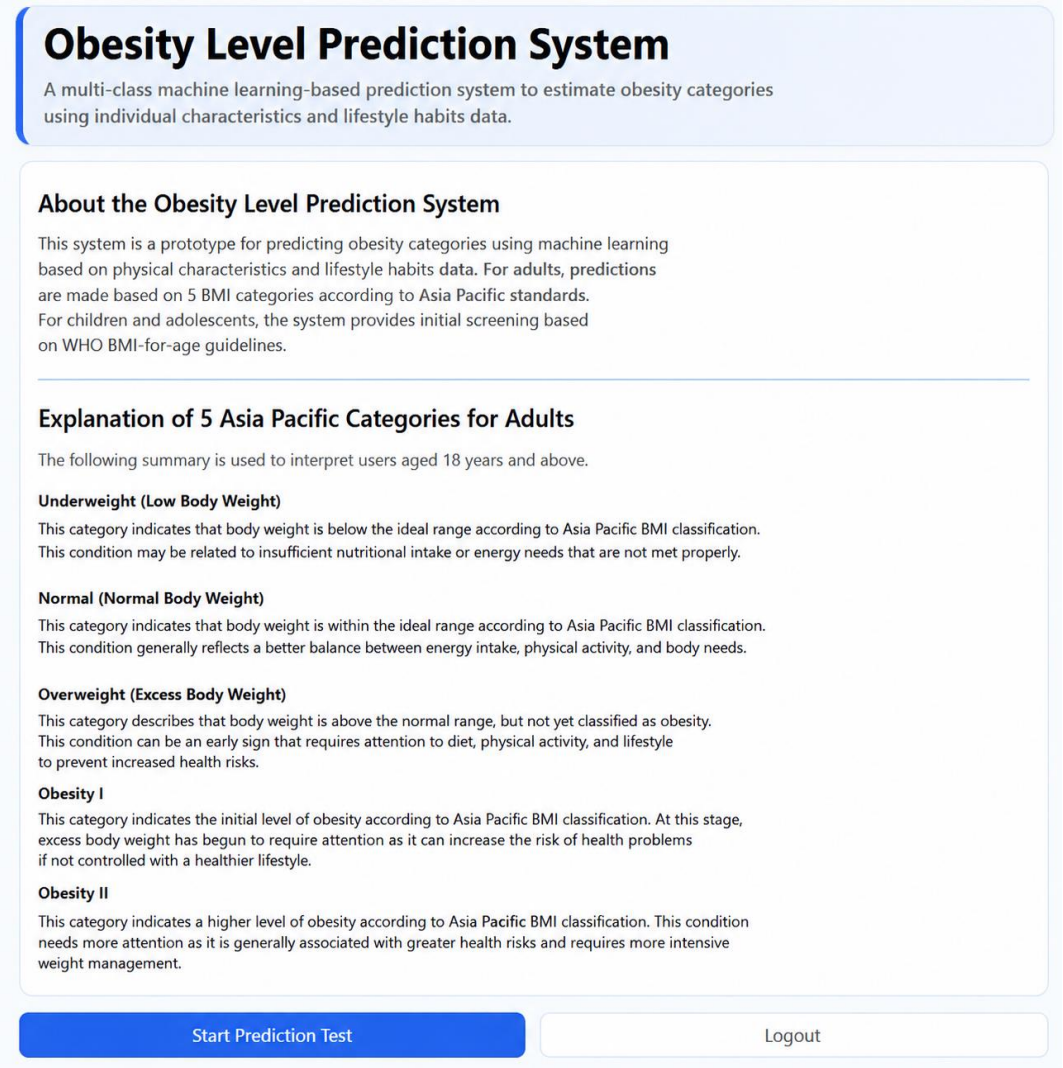



**Figure 4.128. Home Page Implementation Result**

Based on Figure 4.128, the Home page displays the system title Obesity Level Prediction System along with a short subtitle at the top. After that, the system displays one main card entitled About the Obesity Level Prediction System, which contains a brief explanation of the system function and the informative nature of the prediction results. In the same section, the five Asia Pacific BMI categories are also presented directly in a more concise arrangement. These category explanations help users understand the meaning of each obesity level before using the prediction feature. The page also provides a Start Prediction Test button that directs users to the input form and a Logout button to end the session. With this display, the Home page functions not only as the initial navigation page, but also as a brief educational medium for users before they enter their data.

### 4.3.4. Input Form Page Implementation Result

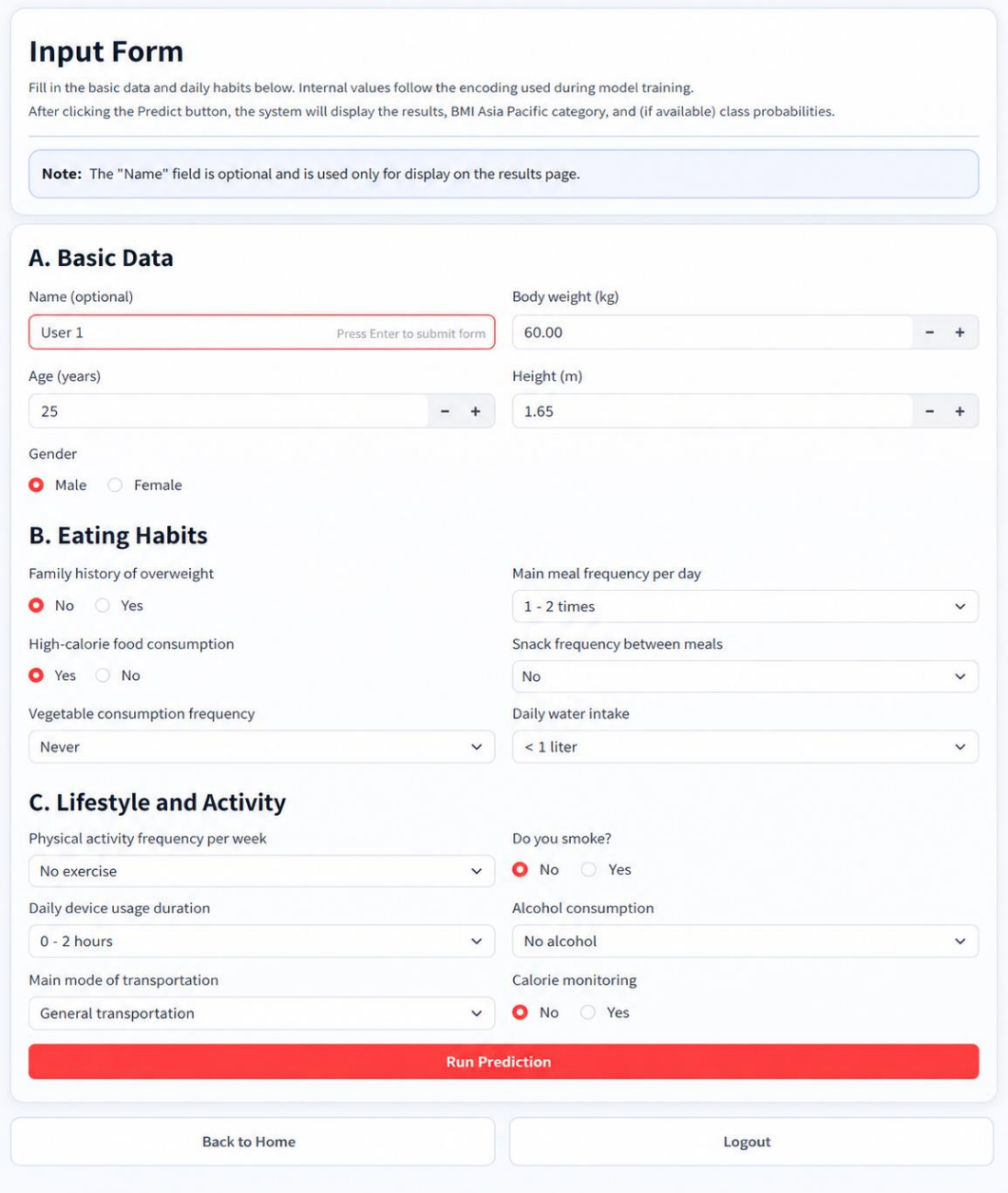



**Figure 4.129. Input Form Page Implementation Result**

After users click the Start Prediction Test button, the system displays the Input Form page. This page is used to enter individual characteristics and lifestyle habit data required by the model to predict obesity level. The input components on this page are divided into three main sections, namely Basic Data, Eating Habits, and Lifestyle and Activities. In the Basic Data section, users enter their name, age, gender, weight, and height. In the Eating Habits section, users enter family history of overweight, high calorie food consumption, vegetable consumption frequency,

daily main meal frequency, between meal food consumption frequency, and daily water intake. Meanwhile, in the Lifestyle and Activities section, users enter physical activity, device usage duration, mode of transportation, smoking habit, alcohol consumption, and calorie monitoring. The system also displays a note that the name field is optional and is only used as a greeting on the result page.

Based on Figure 4.129, all input components are displayed in a structured form layout, making it easier for users to enter data. Categorical variables are presented as selection options, while numerical variables such as age, weight, and height are presented as numerical input fields. At the bottom of the page, there is a Run Prediction button to process the data entered by the user, along with Back to Home and Logout buttons as additional navigation options. With this design, the data entry process becomes more systematic and user friendly. To demonstrate how the system works, users can enter their characteristics and habits according to their individual conditions, then click the Run Prediction button. After the button is clicked, the system forms a feature vector according to the model format and runs the model to produce an obesity category prediction. Therefore, this page functions not only as an input form, but also as a direct connector between user data and the model inference process.

### 4.3.5. Result Page Implementation Results

After the prediction process is executed, the system displays the Prediction Result page. In the latest version, this page presents a result summary at the top, including the user's name, main prediction label, Asia Pacific BMI category, indicative BMI value, model label, current body weight, and normal weight range based on the user's height. After that, the result information is divided into four main tabs, namely Interpretation, Model Probability, Input Summary, and Model Graph. Each tab is designed to present different information so that users can understand the prediction results from various aspects, ranging from classification results to factors that influence the model prediction. With this structure, the result page does not only display the final prediction class, but also provides additional context so that the results are easier to understand and help users interpret the displayed information.

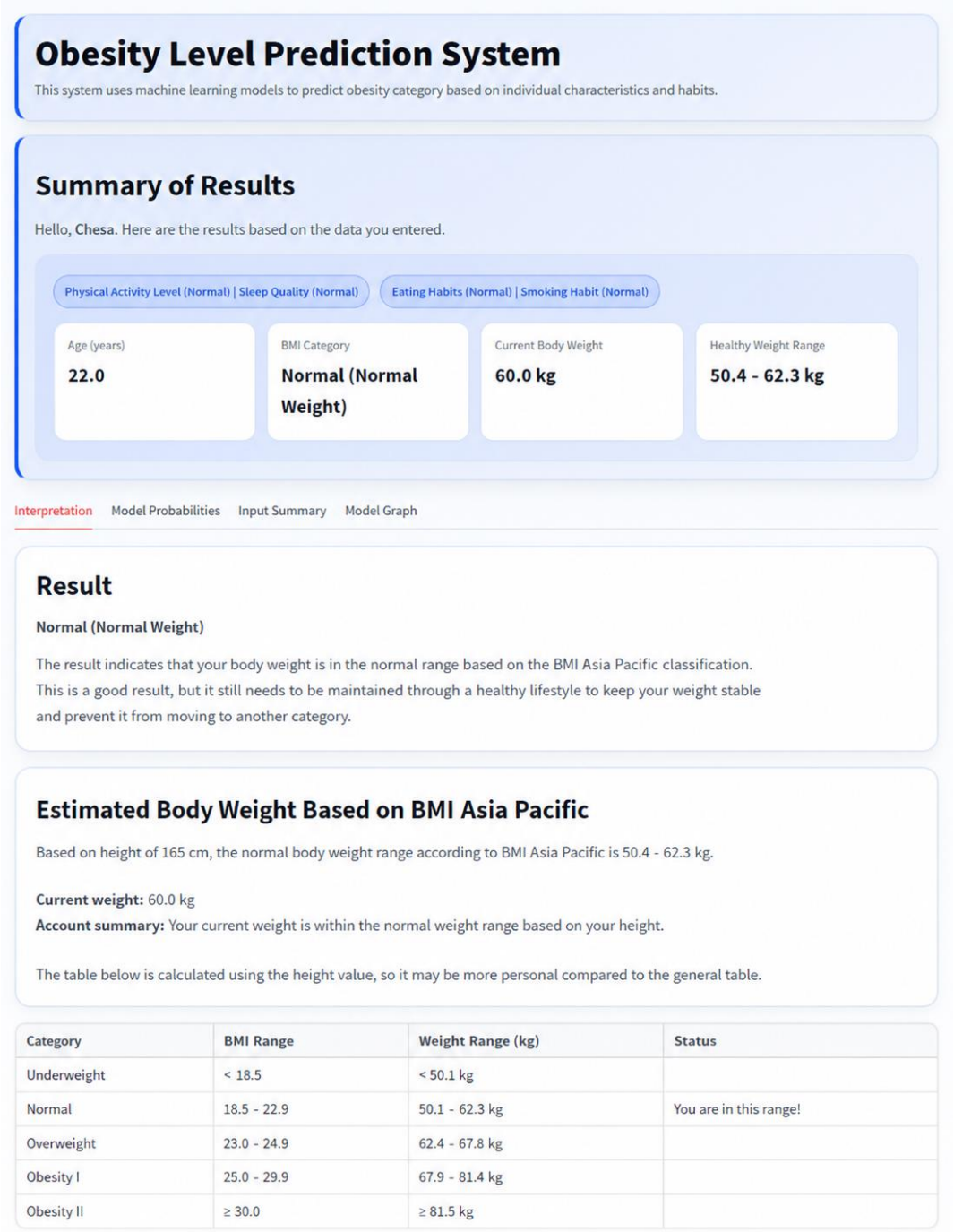



**Figure 4.130. Result Summary and Interpretation Tab**

Based on Figure 4.130, the top section of the page displays the Result Summary, which contains the Main Prediction/Interpretation, Asia Pacific BMI Category, indicative BMI, model label, current body weight, and normal weight range. In the example shown, the system produces a Normal prediction with the Asia Pacific BMI category of Normal, allowing users to immediately identify the main result. In the Interpretation tab, the system presents four sections, namely

Result, Weight Estimation Based on Asia Pacific BMI, Factors to Consider, and Suggestions and Closing Notes. The displayed information includes an explanation of the prediction category, estimated normal weight range based on height, factors that influence the result such as BMI, physical activity, vegetable consumption, and water intake, as well as general suggestions adjusted to the prediction result. The system also emphasizes that the result is informative and not a medical diagnosis. With this design, the prediction result becomes more concise, structured, and easier for users to understand.

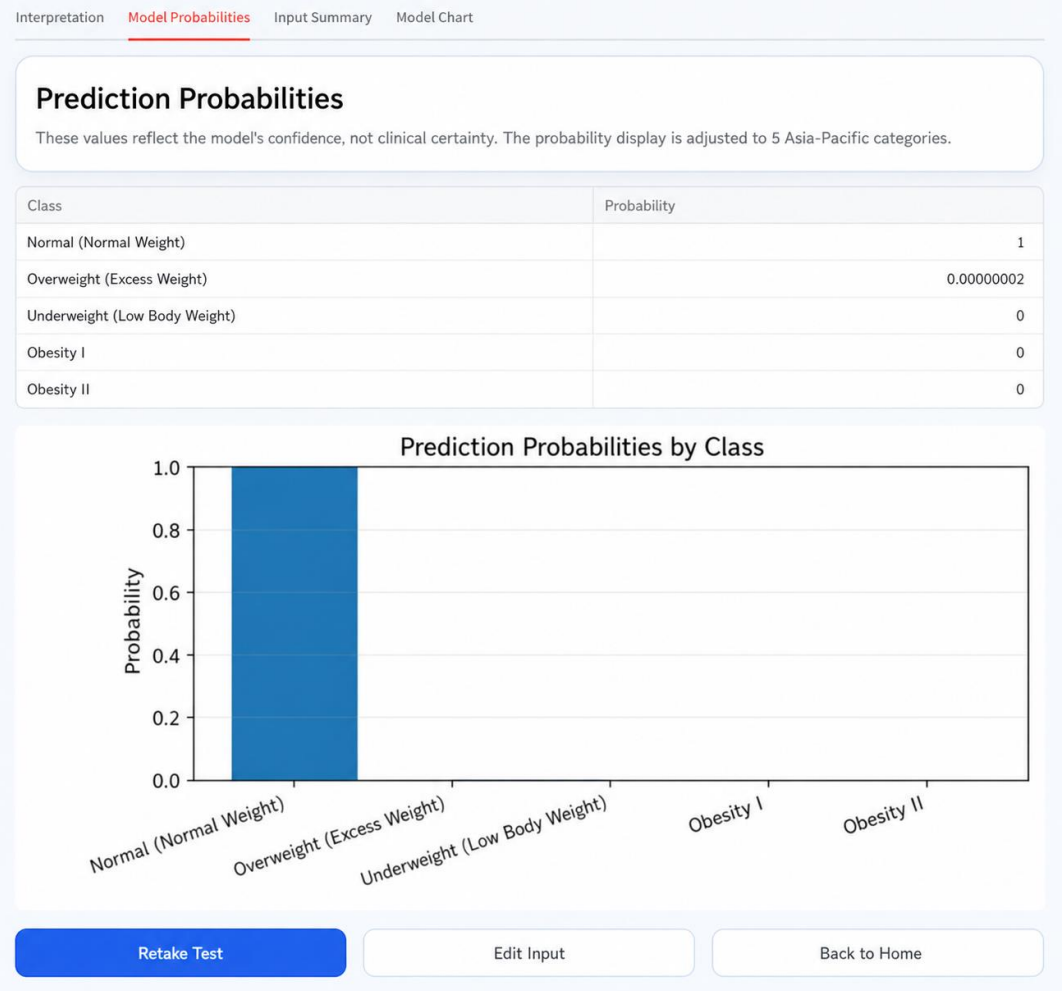



**Figure 4.131. Model Probability Tab Implementation Result**

Based on Figure 4.131, in the Model Probability tab, the system displays the prediction probability for each obesity class in the form of a table and a bar chart. The probability table presents five obesity categories according to the Asia Pacific BMI classification, namely Underweight, Normal, Overweight, Obesity I, and Obesity II. In the example shown, the highest probability is found in the Normal

class, while the probabilities of the other classes are very small or close to zero. The bar chart below the table visually clarifies the probability distribution. This display is useful to show that the model does not only produce one final label, but also provides confidence levels for all target classes.

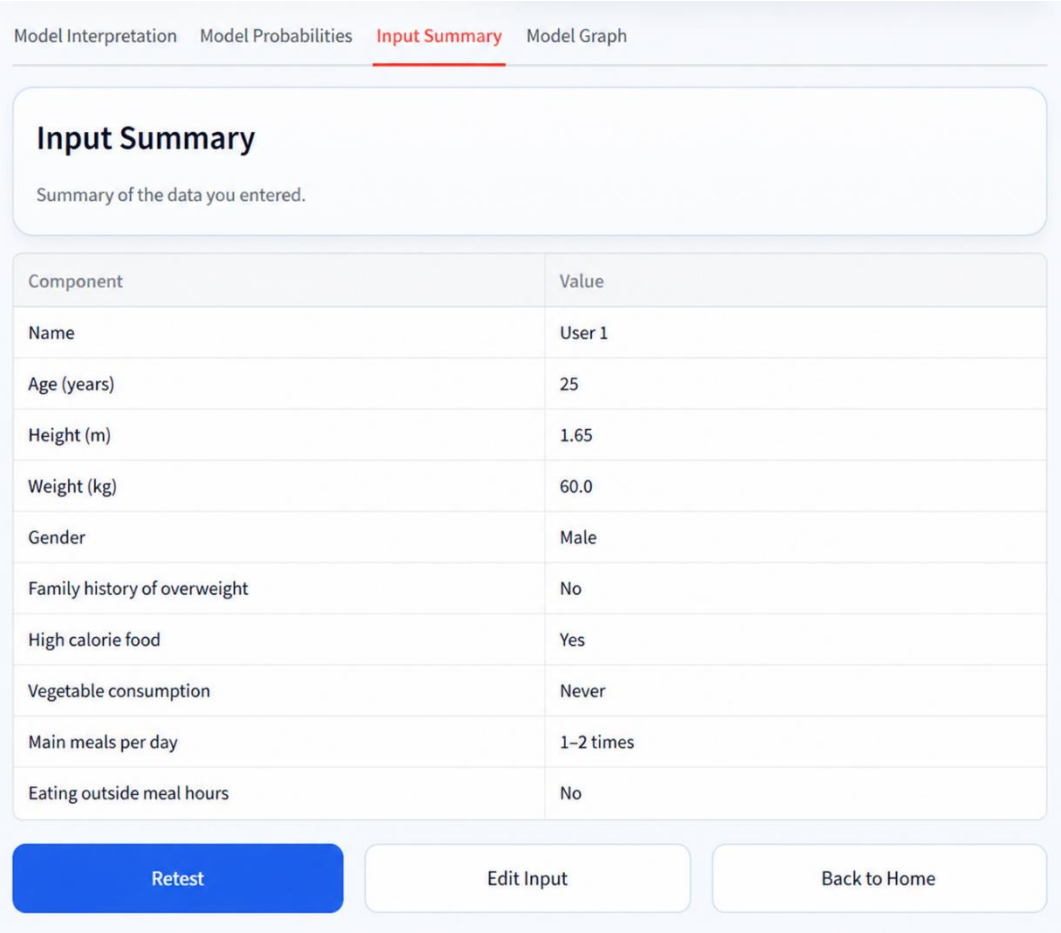



**Figure 4.132. Input Summary Tab Implementation Result**

Based on Figure 4.132, in the Input Summary tab, the system displays the data previously entered by the user before the prediction process was executed. The displayed information includes name, age, height, weight, gender, family history of overweight, high calorie food consumption, vegetable consumption, daily main meal frequency, and between meal eating frequency. This display serves as a verification feature so users can ensure that the obtained prediction result is based on the appropriate input data. Therefore, the result page does not only display the model output, but also maintains traceability between user input and the prediction result.

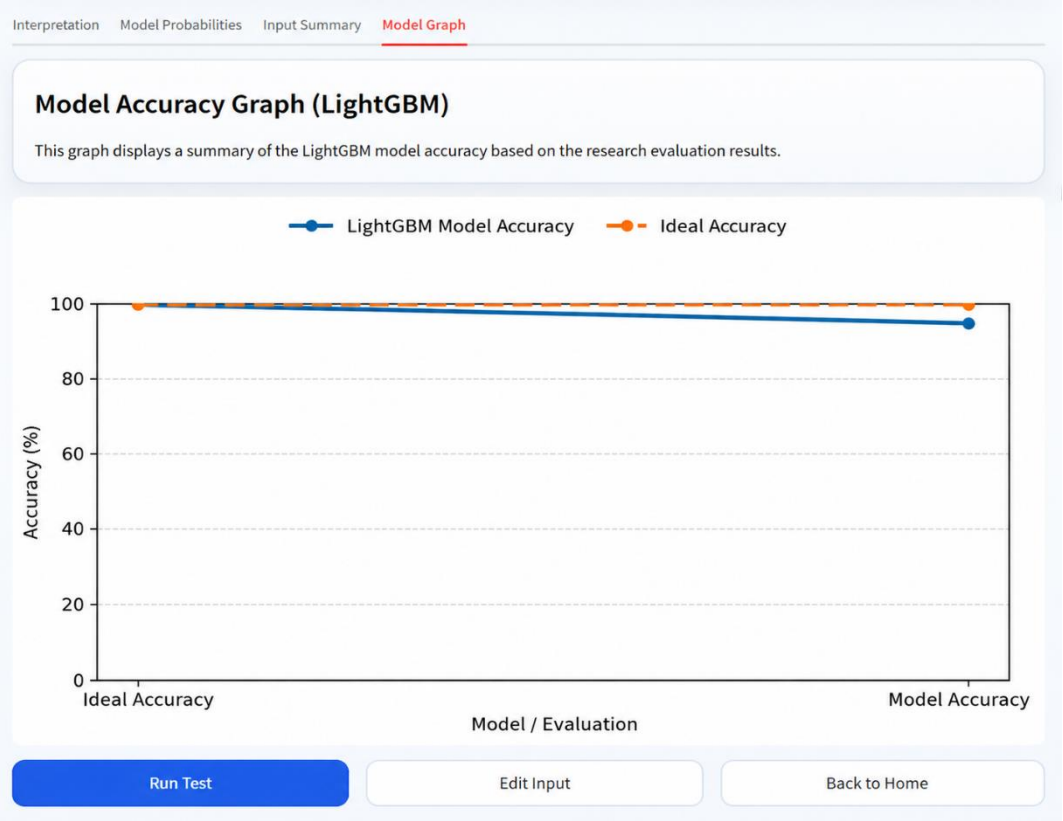



**Figure 4.133. Model Graph Tab Implementation Result**

Based on Figure 4.133, in the Model Graph tab, the system displays a line graph showing a summary of the LightGBM model accuracy based on the research evaluation results. This graph compares the ideal accuracy with the accuracy of the model used in the system. The vertical axis shows the accuracy value in percentage, while the horizontal axis shows the comparison between ideal accuracy and model accuracy. This visualization serves as a complementary feature to provide a brief overview of the quality of the model implemented in the application. With this graph, users do not only obtain prediction results, but also receive general information about the performance of the model used.

As an additional feature in the web implementation, the system also provides a special result display if the user entering the data is under 18 years old. In this condition, the system does not use the adult BMI category, but displays an initial screening based on the WHO BMI for age reference. This feature is only used as supporting information so that the interpretation for children and adolescents is not treated the same as the adult BMI category.

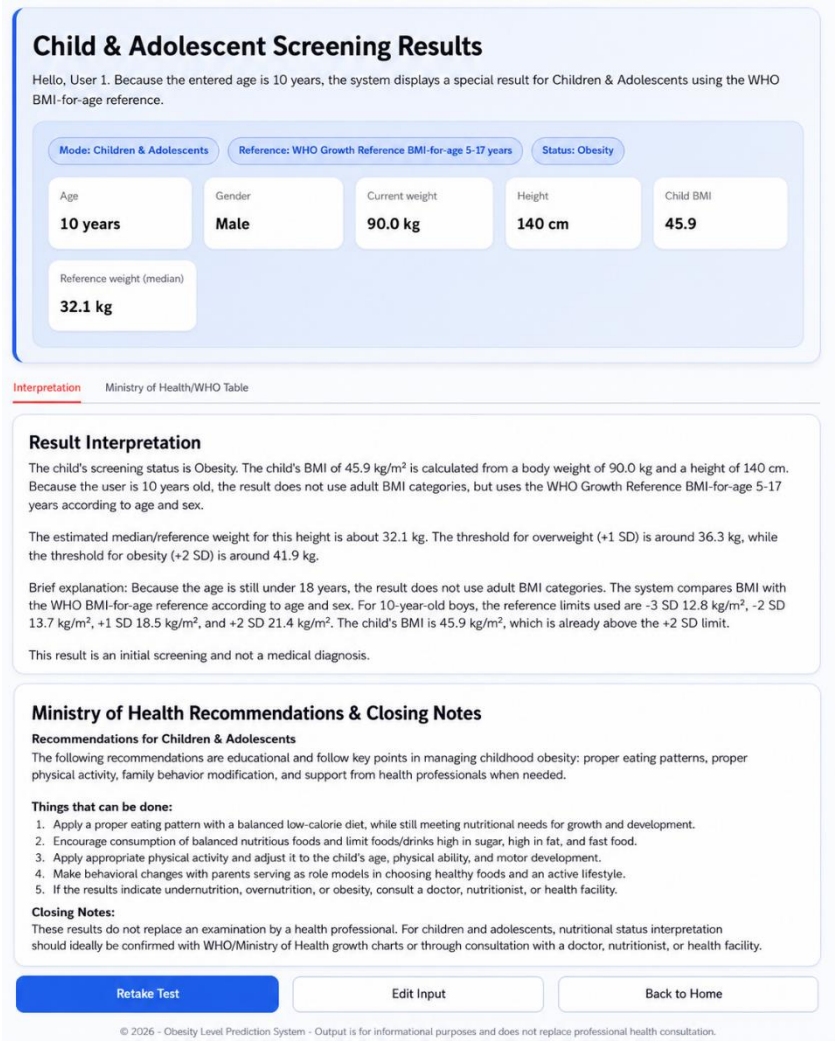



**Figure 4.134. Children and Adolescents Result Page**

Based on Figure 4.134, the children and adolescents result page displays a screening summary that includes screening status, the reference used, age, gender, weight, height, child BMI, and median reference weight. This display is used as initial information based on the user's age and gender, so the results for children and adolescents are not equated with the adult BMI classification. The page also provides a result interpretation section that explains why the WHO BMI-for-age reference is used for users under 18 years old. In addition, the system presents the estimated median reference weight and threshold information to help users understand the screening result more clearly. The recommendation section provides educational suggestions related to eating patterns, physical activity, and healthy lifestyle changes. Therefore, this page functions as both a screening output and an informative guide for children and adolescent users.

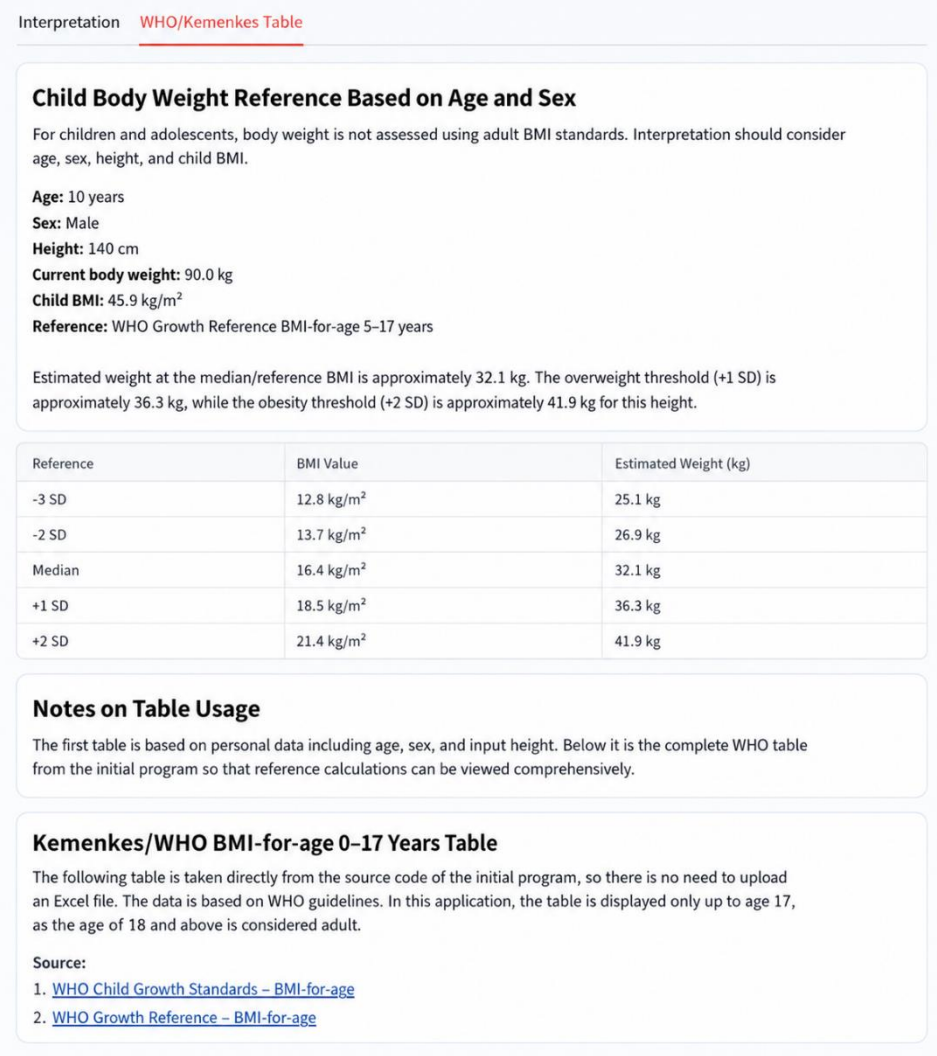



**Figure 4.135. Indonesian Ministry of Health/WHO Table Tab**

Based on Figure 4.135, the system displays the Indonesian Ministry of Health/WHO Table tab, which contains child weight references based on age, gender, and height. This tab also provides the WHO BMI-for-age reference table as supporting information for children and adolescents. Although this feature is not the main focus of the study, it helps complete the system interface and supports result interpretation based on appropriate growth references. At the bottom of the result page, there are three navigation buttons: Repeat Test, Edit Input, and Back to Home. These buttons help users repeat the test, revise the input data, or return to the main page more easily. Overall, the prediction result page has been successfully developed as an informative interface that displays classification results, interpretation, input summary, and additional screening information in one accessible page..

*This page is intentionally left blank.*