

DAFTAR GAMBAR

Gambar 2. 1 Arduino Mega 2560.....	12
Gambar 2. 2 Sensor TCS3200.....	14
Gambar 2. 3 Rangkaian Sensor TCS3200.....	14
Gambar 2. 4 Photodiode.....	15
Gambar 2. 5 Sensor TGS2600.....	17
Gambar 2. 6 Sensor Turbidity	17
Gambar 2. 7 system logika fuzzy	19
Gambar 2. 8 Fungsi Keanggotaan Gaussian	20
Gambar 2. 9 Fungsi Keanggotaan Generalized Bell	20
Gambar 2. 10 LCD 16X4	23
Gambar 2. 11 Minyak Goreng Sawit	24
Gambar 3. 1 Flowchart Kerangka Penelitian	25
Gambar 3. 2 Blok Diagram	27
Gambar 3. 3 Diagram Alir.....	28
Gambar 3. 4 Perancangan Sensor TCS3200	30
Gambar 3. 5 Perancangan TGS2600	31
Gambar 3. 6 Perancangan Turbidity	31
Gambar 3. 7 Perancangan LCD 16x4 i2c.....	32
Gambar 3. 8 Perancangan Keseluruhan Sistem.....	32
Gambar 3. 9 Desain Alat	35
Gambar 3. 10 Diagram Blok Sistem Fuzzy.....	35
Gambar 3. 11 Tampilan GUI Python 1	38
Gambar 4. 1 Pengujian sensor warna dengan kertas merah	40
Gambar 4. 2 pengujian sensor warna dengan kertas hijau	40
Gambar 4. 3 pengujian sensor warna dengan kertas Biru	41
Gambar 4. 4 pengujian sensor warna dengan kertas Putih	41
Gambar 4. 5 pengujian sensor warna dengan kertas Kuning	42
Gambar 4. 6 pengujian sensor turbidity dengan minyak jernih.....	43
Gambar 4. 7 pengujian sensor turbidity dengan minyak sedang.....	44
Gambar 4. 8 pengujian sensor turbidity dengan minyak keruh.....	44
Gambar 4. 9 pengujian sensor gas dengan minyak jernih	45
Gambar 4. 10 pengujian sensor gas dengan minyak jelantah.....	46
Gambar 4. 11 Tampilan dan Struktur GUI python.....	48
Gambar 4. 12 Fitur Import Data.....	49
Gambar 4. 13 Pie Chart	53
Gambar 4. 14 Grafik Distribusi Sensor	54

Gambar 4. 15 Scatter Plot.....	55
--------------------------------	----

DAFTAR TABEL

Tabel 2. 1 Implementasi Metode Klasifikasi Support Vector Machine (SVM) Terhadap Pemakaian Minyak Goreng	5
Tabel 2. 2 Rancang Bangun Sistem Klasifikasi Kualitas Minyak Goreng dengan Parameter Kecerahan dan Warna menggunakan Metode Random Forest	6
Tabel 2. 3 Rancang Bangun Sistem Klasifikasi Kualitas Minyak Goreng berdasarkan Warna dan Kejernihan menggunakan Metode Naïve Bayes berbasis Arduino Uno...	7
Tabel 2. 4 Implementasi Logika Fuzzy pada Alat Pendeteksi Kualitas Minyak Goreng Berdasarkan pH dan Tingkat Kejernihan	9
Tabel 2. 5 Implementasi Multilayer Perceptron untuk Mendeteksi Kelayakan Minyak Goreng	10
Tabel 2. 6 Implementasi Logika Fuzzy Pada Alat Pendeteksi Kualitas Minyak Goreng Berdasarkan pH dan Tingkat Kejernihan	11
Tabel 2. 7 Prototype Alat Pendeteksi Kualitas Minyak Goreng Kelapa Sawit Menggunakan Sensor TCS3200 dan Metode Fuzzy Logic.....	12
Tabel 2. 8 Datasheet Arduino Mega 2560.....	13
Tabel 2. 9 Photodiode	16
Tabel 2. 10 Datasheet TGS2600	17
Tabel 2. 11 Datasheet Turbidity	18
Tabel 3. 1 Tabel Port Arduino Mega 2560 untuk Sistem Kualitas Minyak Goreng .	33
Tabel 3. 2 Tabel Aturan (rules) Algoritma Fuzzy Gaussian Bell Kualitas Minyak Goreng.....	36
Tabel 4. 1 Hasil Perhitungan Akurasi Sistem	57
Tabel 4. 2 Distribusi Kategori Kualitas.....	58
Tabel 4. 3 Statistik Skor Fuzzy Gaussian Bell	58
Tabel 4. 4 Rata-rata Per Jenis Minyak	58

LAMPIRAN

Lampiran 1 Program Arduino

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,20,4);
```

```
#define s0 4
```

```
#define s1 5
```

```
#define s2 6
```

```
#define s3 7
```

```
#define out 8
```

```
#define START_BUTTON 3
```

```
int redFrequency = 0;
```

```
int greenFrequency = 0;
```

```
int blueFrequency = 0;
```

```
bool blinkingFlag;
```

```
uint32_t prevMillisData=0, prevMillisBlinking=0;
```

```
int turbValue, odorValue;
```

```
int isButttonPressed[2] = {1,1};
```

```
int buttonState[2];
```

```
int lastButtonState[2]={0,0};
```

```
unsigned long lastDebounceTime[2]={0,0};
```

```

// TAMBAHAN: Mode kirim data

bool autoSendMode = true;

unsigned long sendInterval = 1000;


void setup()
{
    lcd.init();
    Serial.begin(9600);
    lcd.backlight();
    pinMode(s0, OUTPUT);
    pinMode(s1, OUTPUT);
    pinMode(s2, OUTPUT);
    pinMode(s3, OUTPUT);
    pinMode(out, INPUT);
    pinMode(START_BUTTON, INPUT_PULLUP);
    pinMode(A0, INPUT);
    pinMode(A1, INPUT);
    digitalWrite(s0, HIGH);
    digitalWrite(s1, LOW);


    // Tampilan awal
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Sistem Siap!");
    lcd.setCursor(0,1);

```

```

    lcd.print("Mode: Otomatis");

    delay(2000);

    lcd.clear();
}

void loop()
{
    if(blinkingFlag){
        GetColors();

        turbValue = analogRead(A0);
        odorValue = analogRead(A1);
    }

    displayColor();
    displayOdor();
    displayTurbidity();
    cursorBlinking(1000);
    readButton(START_BUTTON,10,0);

    // Mode manual: kirim data pas tombol ditekan
    if(isButttonPressed[0] == true && !autoSendMode){
        sendDataToSerial();
        isButttonPressed[0]=false;
    }
}

```

```

// Mode otomatis: kirim data setiap interval tertentu

if(autoSendMode){
    if(millis() - prevMillisData >= sendInterval){
        sendDataToSerial();
        prevMillisData = millis();
    }
}

// BONUS: Tekan dan tahan tombol 3 detik untuk toggle mode
checkLongPress();
}

void checkLongPress(){
    static unsigned long pressStart = 0;
    static bool wasPressed = false;

    if(digitalRead(START_BUTTON) == LOW && !wasPressed){
        pressStart = millis();
        wasPressed = true;
    }

    if(digitalRead(START_BUTTON) == LOW && wasPressed){
        if(millis() - pressStart > 3000){
            autoSendMode = !autoSendMode;
            lcd.clear();
        }
    }
}

```

```

    lcd.setCursor(0,0);
    if(autoSendMode){
        lcd.print("Mode: OTOMATIS");
        lcd.setCursor(0,1);
        lcd.print("Kirim setiap ");
        lcd.print(sendInterval/1000);
        lcd.print("s");
    } else {
        lcd.print("Mode: MANUAL");
        lcd.setCursor(0,1);
        lcd.print("Tekan tombol");
    }
    delay(2000);
    lcd.clear();
    wasPressed = false;
}

}

if(digitalRead(START_BUTTON) == HIGH){
    wasPressed = false;
}

}

void readButton(uint8_t buttonPin, uint8_t debounceDelay, bool buttonIndex){
    int reading = digitalRead(buttonPin);

```

```

if (reading != lastButtonState[buttonIndex]) {
    lastDebounceTime[buttonIndex] = millis();
}

if ((millis() - lastDebounceTime[buttonIndex]) > debounceDelay) {
    if (reading != buttonState[buttonIndex]) {
        buttonState[buttonIndex] = reading;
        if (buttonState[buttonIndex] == HIGH) {
            isButtonPressed[buttonIndex] = true;
        }
    }
}

lastButtonState[buttonIndex] = reading;
}

```

```

void cursorBlinking(uint16_t time){
    if(millis() - prevMillisBlinking >= time){
        blinkingFlag=!blinkingFlag;
        (blinkingFlag)? printLCD(19,0, "*") : printLCD(19,0, " ");
        prevMillisBlinking=millis();
    }
}

```

```

void sendDataToSerial(){
    // Kirim data dengan format: Red Green Blue Turb Bau (dipisah spasi)
    Serial.print(redFrequency);

```

```

Serial.print(" ");
Serial.print(greenFrequency);
Serial.print(" ");
Serial.print(blueFrequency);
Serial.print(" ");
Serial.print(turbValue);
Serial.print(" ");
Serial.println(odorValue);
}

```

```

void displayColor(){
    printLCD(0,0,"COLOR");
    printLCD(0,1,"R=");
    printLCD(2,1,String(redFrequency));
    if(redFrequency>0 && redFrequency <10)printLCD(3,1," ");
    else if(redFrequency>10 && redFrequency <100)printLCD(4,1," ");
    else if(redFrequency>100 && redFrequency <1000)printLCD(5,1," ");

    printLCD(0,2,"G=");
    printLCD(2,2,String(greenFrequency));
    if(greenFrequency>0 && greenFrequency <10)printLCD(3,2," ");
    else if(greenFrequency>10 && greenFrequency <100)printLCD(4,2," ");
    else if(greenFrequency>100 && greenFrequency <1000)printLCD(5,2," ");

    printLCD(0,3,"B=");

```

```

printLCD(2,3,String(blueFrequency));
if(blueFrequency>0 && blueFrequency <10)printLCD(3,3," ");
else if(blueFrequency>10 && blueFrequency <100)printLCD(4,3," ");
else if(blueFrequency>100 && blueFrequency <1000)printLCD(5,3," ");
}

```

```

void displayTurbidity(){
    printLCD(10,2,"TRB=");
    printLCD(14,2,String(turbValue));
    if(turbValue>0 && turbValue <10)printLCD(15,2," ");
    else if(turbValue>10 && turbValue <100)printLCD(16,2," ");
    else if(turbValue>100 && turbValue <1000)printLCD(17,2," ");
}

```

```

void displayOdor(){
    printLCD(10,1,"BAU=");
    printLCD(14,1,String(odorValue));
    if(odorValue>0 && odorValue <10)printLCD(15,1," ");
    else if(odorValue>10 && odorValue <100)printLCD(16,1," ");
    else if(odorValue>100 && odorValue <1000)printLCD(17,1," ");
}

```

```

void GetColors()
{
    digitalWrite(s2,LOW);
}

```

```
digitalWrite(s3,LOW);  
redFrequency = pulseIn(out, LOW);
```

```
digitalWrite(s2,HIGH);  
digitalWrite(s3,HIGH);  
greenFrequency = pulseIn(out, LOW);
```

```
digitalWrite(s2,LOW);  
digitalWrite(s3,HIGH);  
blueFrequency = pulseIn(out, LOW);  
}
```

```
static void printLCD(uint8_t column, uint8_t line, String text){  
    lcd.setCursor(column,line);  
    lcd.print(text);  
}
```

Lampiran 2 Program GUI python

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import pandas as pd
import seaborn as sns
from datetime import datetime
import random
from collections import Counter

class OilQualityClassifier:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Sistem Klasifikasi Kualitas Minyak Goreng - Fuzzy Gaussian Bell")
        self.root.geometry("1400x900")
        self.root.configure(bg='#f0f0f0')

        # Data
        self.data = None
        self.analysis_results = []

        # Jenis minyak dan kondisi
```

```

self.oil_types = ['minyak jernih', 'minyak sedang', 'minyak keruh']
self.conditions = ['baru', 'goreng 1x', 'goreng 2x', 'sedang', 'keruh']

self.create_widgets()

plt.style.use('seaborn-v0_8')

def gaussian_bell(self, x, center, sigma):
    """Fungsi keanggotaan Gaussian Bell"""
    return np.exp(-0.5 * ((x - center) / sigma) ** 2)

def fuzzify_gas_sensor(self, gas_value):
    """
    Fuzzifikasi Sensor Gas (Bau)

    Berdasarkan data:
    - Minyak Baru/Baik: 177-181
    - Minyak Sedang: 181-296
    - Minyak Keruh/Buruk: 190-276
    """
    fresh = self.gaussian_bell(gas_value, 179, 15)    # Center: 179, sigma: 15
    medium = self.gaussian_bell(gas_value, 235, 45)   # Center: 235, sigma: 45
    stale = self.gaussian_bell(gas_value, 250, 35)    # Center: 250, sigma: 35

    return {'fresh': fresh, 'medium': medium, 'stale': stale}

def fuzzify_color_sensor(self, r_value, g_value, b_value):

```

"""

Fuzzifikasi Sensor Warna RGB

Berdasarkan data:

- Minyak Baru (Clear): R:137-142, G:151-156, B:139-147
- Minyak Sedang (Yellow): R:162-174, G:192-207, B:181-193
- Minyak Keruh (Brown): R:195-221, G:234-265, B:204-232

"""

Clear (Minyak Baru)

clear_r = self.gaussian_bell(r_value, 140, 8)

clear_g = self.gaussian_bell(g_value, 153, 8)

clear_b = self.gaussian_bell(b_value, 143, 8)

clear = (clear_r + clear_g + clear_b) / 3

Yellow (Minyak Sedang)

yellow_r = self.gaussian_bell(r_value, 168, 12)

yellow_g = self.gaussian_bell(g_value, 200, 12)

yellow_b = self.gaussian_bell(b_value, 187, 12)

yellow = (yellow_r + yellow_g + yellow_b) / 3

Brown (Minyak Keruh)

brown_r = self.gaussian_bell(r_value, 208, 15)

brown_g = self.gaussian_bell(g_value, 245, 15)

brown_b = self.gaussian_bell(b_value, 218, 15)

brown = (brown_r + brown_g + brown_b) / 3

```

    return {'clear': clear, 'yellow': yellow, 'brown': brown}

def fuzzify_turbidity_sensor(self, turbidity_value):
    """
    Fuzzifikasi Sensor Kekeruhan (Turbidity)
    Berdasarkan data:
    - Minyak Baru (Clear): 158-181
    - Minyak Sedang (Cloudy): 370-392
    - Minyak Keruh (Very Cloudy): 476-506
    """
    15 clear = self.gaussian_bell(turbidity_value, 170, 15)    # Center: 170, sigma:
    20 cloudy = self.gaussian_bell(turbidity_value, 381, 20)    # Center: 381, sigma:
    very_cloudy = self.gaussian_bell(turbidity_value, 491, 20) # Center: 491,
    sigma: 20

    return {'clear': clear, 'cloudy': cloudy, 'very_cloudy': very_cloudy}

def fuzzy_rules(self, gas_fuzzy, color_fuzzy, turbidity_fuzzy):
    """
    Rule Base Fuzzy berdasarkan kombinasi sensor
    """
    # RULE UNTUK KUALITAS BAIK (Minyak Baru/Jernih)
    good_rules = [
        min(gas_fuzzy['fresh'], color_fuzzy['clear'], turbidity_fuzzy['clear']),

```

```

        min(gas_fuzzy['fresh'], color_fuzzy['clear'], turbidity_fuzzy['cloudy']) * 0.7,
        min(gas_fuzzy['medium'], color_fuzzy['clear'], turbidity_fuzzy['clear']) * 0.6,
    ]
    good_quality = max(good_rules)

# RULE UNTUK KUALITAS SEDANG
medium_rules = [
    min(gas_fuzzy['medium'], color_fuzzy['yellow'], turbidity_fuzzy['cloudy']),
    min(gas_fuzzy['medium'], color_fuzzy['yellow'], turbidity_fuzzy['clear']) *
0.8,
    min(gas_fuzzy['fresh'], color_fuzzy['yellow'], turbidity_fuzzy['cloudy']) *
0.7,
    min(gas_fuzzy['stale'], color_fuzzy['clear'], turbidity_fuzzy['cloudy']) * 0.5,
]
medium_quality = max(medium_rules)

# RULE UNTUK KUALITAS BURUK (Minyak Keruh)
bad_rules = [
    min(gas_fuzzy['stale'], color_fuzzy['brown'], turbidity_fuzzy['very_cloudy']),
    min(gas_fuzzy['medium'], color_fuzzy['brown'],
turbidity_fuzzy['very_cloudy']) * 0.9,
    min(gas_fuzzy['stale'], color_fuzzy['yellow'], turbidity_fuzzy['very_cloudy'])
* 0.8,
    min(gas_fuzzy['medium'], color_fuzzy['brown'], turbidity_fuzzy['cloudy']) *
0.7,
]
bad_quality = max(bad_rules)

```

```

    return {'Baik': good_quality, 'Sedang': medium_quality, 'Buruk': bad_quality}

def defuzzify(self, quality_scores):
    """
    Defuzzifikasi menggunakan metode Weighted Average
    """
    weights = {'Baik': 85, 'Sedang': 50, 'Buruk': 15}

    numerator = sum(score * weights[quality] for quality, score in
quality_scores.items())
    denominator = sum(quality_scores.values())

    if denominator == 0:
        return 50, 'Tidak Diketahui'

    final_score = numerator / denominator

    # Threshold klasifikasi
    if final_score >= 70:
        classification = 'Baik'
    elif final_score >= 40:
        classification = 'Sedang'
    else:
        classification = 'Buruk'

```

```

return final_score, classification

def create_widgets(self):
    main_frame = ttk.Frame(self.root, padding="10")
    main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

    title_label = ttk.Label(main_frame,
                             text="Sistem Klasifikasi Kualitas Minyak
Goreng\nMenggunakan Sensor Gas, Warna RGB, dan Kekeruhan\nDengan
Algoritma Fuzzy Gaussian Bell",
                             font=('Arial', 14, 'bold'),
                             justify='center')
    title_label.grid(row=0, column=0, columnspan=3, pady=(0, 20))

    self.notebook = ttk.Notebook(main_frame)
    self.notebook.grid(row=1, column=0, columnspan=3, sticky=(tk.W, tk.E, tk.N,
tk.S), pady=(0, 10))

    self.input_frame = ttk.Frame(self.notebook, padding="10")
    self.notebook.add(self.input_frame, text="Data & Analisis")

    self.histogram_frame = ttk.Frame(self.notebook, padding="10")
    self.notebook.add(self.histogram_frame, text="Histogram")

    self.scatter_frame = ttk.Frame(self.notebook, padding="10")

```

```

self.notebook.add(self.scatter_frame, text="Scatter Plot")

self.pie_frame = ttk.Frame(self.notebook, padding="10")
self.notebook.add(self.pie_frame, text="Pie Chart")

self.results_frame = ttk.Frame(self.notebook, padding="10")
self.notebook.add(self.results_frame, text="Hasil & Export")

self.create_input_tab()
self.create_histogram_tab()
self.create_scatter_tab()
self.create_pie_tab()
self.create_results_tab()

self.root.columnconfigure(0, weight=1)
self.root.rowconfigure(0, weight=1)
main_frame.columnconfigure(0, weight=1)
main_frame.rowconfigure(1, weight=1)

def create_input_tab(self):
    import_frame = ttk.LabelFrame(self.input_frame, text="Import Data",
padding="10")

    import_frame.grid(row=0, column=0, columnspan=2, sticky=(tk.W, tk.E),
pady=(0, 10))

```

```
info_text = """"Format File yang Diperlukan (CSV/Excel):
```

```
Kolom: No, Timestamp, Jenis_Minyak, Kondisi, Red, Green, Blue, Turb, Bau
```

- Jenis_Minyak: minyak jernih, minyak sedang, minyak keruh
- Kondisi: baru, sedang, keruh
- Red, Green, Blue: 0-255
- Turb: 0-1023 (tinggi=keruh)
- Bau: 0-1023 (rendah=baik)"""

```
info_label = ttk.Label(import_frame, text=info_text, font=('Arial', 9),  
foreground='blue')
```

```
info_label.grid(row=0, column=0, columnspan=3, sticky=tk.W, pady=(0, 10))
```

```
ttk.Button(import_frame, text="Import Data",  
command=self.import_data_file).grid(row=1, column=0, padx=5, pady=5)
```

```
ttk.Button(import_frame, text="Generate Sample",  
command=self.generate_sample_data).grid(row=1, column=1, padx=5, pady=5)
```

```
ttk.Button(import_frame, text="Analisis Data",  
command=self.analyze_data).grid(row=1, column=2, padx=5, pady=5)
```

```
self.status_label = ttk.Label(import_frame, text="Belum ada data yang dimuat",  
foreground='gray')
```

```
self.status_label.grid(row=2, column=0, columnspan=3, pady=(10, 0))
```

```
preview_frame = ttk.LabelFrame(self.input_frame, text="Preview Data",  
padding="10")
```

```
preview_frame.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S),  
padx=(0, 5))
```

```

self.data_tree = ttk.Treeview(preview_frame, height=12)

self.data_tree.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))


data_scroll = ttk.Scrollbar(preview_frame, orient=tk.VERTICAL,
command=self.data_tree.yview)

data_scroll.grid(row=0, column=1, sticky=(tk.N, tk.S))

self.data_tree.configure(yscrollcommand=data_scroll.set)


manual_frame = ttk.LabelFrame(self.input_frame, text="Input Manual",
padding="10")

manual_frame.grid(row=1, column=1, sticky=(tk.W, tk.E, tk.N, tk.S), padx=(5,
0))


ttk.Label(manual_frame, text="Jenis Minyak:").grid(row=0, column=0,
sticky=tk.W, pady=2)

self.oil_type_var = tk.StringVar(value="minyak jernih")

ttk.Combobox(manual_frame, textvariable=self.oil_type_var,
values=self.oil_types, state="readonly", width=15).grid(row=0, column=1,
sticky=tk.W, padx=(5, 0), pady=2)


ttk.Label(manual_frame, text="Kondisi:").grid(row=1, column=0, sticky=tk.W,
pady=2)

self.condition_var = tk.StringVar(value="baru")

ttk.Combobox(manual_frame, textvariable=self.condition_var,
values=self.conditions, state="readonly", width=15).grid(row=1, column=1,
sticky=tk.W, padx=(5, 0), pady=2)


ttk.Label(manual_frame, text="Red (0-255):").grid(row=2, column=0,
sticky=tk.W, pady=2)

```

```

self.r_var = tk.StringVar(value="140")

ttk.Entry(manual_frame, textvariable=self.r_var, width=18).grid(row=2,
column=1, sticky=tk.W, padx=(5, 0), pady=2)


ttk.Label(manual_frame, text="Green (0-255):").grid(row=3, column=0,
sticky=tk.W, pady=2)

self.g_var = tk.StringVar(value="153")

ttk.Entry(manual_frame, textvariable=self.g_var, width=18).grid(row=3,
column=1, sticky=tk.W, padx=(5, 0), pady=2)


ttk.Label(manual_frame, text="Blue (0-255):").grid(row=4, column=0,
sticky=tk.W, pady=2)

self.b_var = tk.StringVar(value="144")

ttk.Entry(manual_frame, textvariable=self.b_var, width=18).grid(row=4,
column=1, sticky=tk.W, padx=(5, 0), pady=2)


ttk.Label(manual_frame, text="Turb (0-1023):").grid(row=5, column=0,
sticky=tk.W, pady=2)

self.turbidity_var = tk.StringVar(value="168")

ttk.Entry(manual_frame, textvariable=self.turbidity_var,
width=18).grid(row=5, column=1, sticky=tk.W, padx=(5, 0), pady=2)


ttk.Label(manual_frame, text="Bau (0-1023):").grid(row=6, column=0,
sticky=tk.W, pady=2)

self.gas_var = tk.StringVar(value="179")

ttk.Entry(manual_frame, textvariable=self.gas_var, width=18).grid(row=6,
column=1, sticky=tk.W, padx=(5, 0), pady=2)

```

```
        ttk.Button(manual_frame, text="Analisis Manual",
command=self.analyze_manual_input).grid(row=7, column=0, columnspan=2,
pady=(10, 0))
```

```
        result_frame = ttk.LabelFrame(manual_frame, text="Hasil Analisis",
padding="5")
```

```
        result_frame.grid(row=8, column=0, columnspan=2, sticky=(tk.W, tk.E),
pady=(10, 0))
```

```
        self.manual_result = tk.Text(result_frame, height=10, width=25,
font=('Courier', 8))
```

```
        self.manual_result.grid(row=0, column=0, sticky=(tk.W, tk.E))
```

```
        self.input_frame.columnconfigure(0, weight=1)
```

```
        self.input_frame.columnconfigure(1, weight=1)
```

```
        self.input_frame.rowconfigure(1, weight=1)
```

```
        preview_frame.columnconfigure(0, weight=1)
```

```
        preview_frame.rowconfigure(0, weight=1)
```

```
def create_histogram_tab(self):
```

```
    title_label = ttk.Label(self.histogram_frame, text="Distribusi Data Sensor",
font=('Arial', 12, 'bold'))
```

```
    title_label.grid(row=0, column=0, pady=(0, 10))
```

```
    self.hist_fig, axes = plt.subplots(2, 3, figsize=(14, 8))
```

```
    self.hist_axes = axes.flatten()
```

```
    self.hist_canvas = FigureCanvasTkAgg(self.hist_fig, self.histogram_frame)
```

```
self.hist_canvas.get_tk_widget().grid(row=1, column=0, sticky=(tk.W, tk.E,  
tk.N, tk.S))
```

```
ttk.Button(self.histogram_frame, text="Refresh Histogram",  
command=self.update_histogram).grid(row=2, column=0, pady=(10, 0))
```

```
self.histogram_frame.columnconfigure(0, weight=1)
```

```
self.histogram_frame.rowconfigure(1, weight=1)
```

```
def create_scatter_tab(self):
```

```
    title_label = ttk.Label(self.scatter_frame, text="Scatter Plot Analysis",  
font=('Arial', 12, 'bold'))
```

```
    title_label.grid(row=0, column=0, pady=(0, 10))
```

```
self.scatter_fig, axes = plt.subplots(2, 2, figsize=(12, 8))
```

```
self.scatter_ax1, self.scatter_ax2, self.scatter_ax3, self.scatter_ax4 =  
axes.flatten()
```

```
self.scatter_canvas = FigureCanvasTkAgg(self.scatter_fig, self.scatter_frame)
```

```
self.scatter_canvas.get_tk_widget().grid(row=1, column=0, sticky=(tk.W, tk.E,  
tk.N, tk.S))
```

```
ttk.Button(self.scatter_frame, text="Refresh Scatter",  
command=self.update_scatter_plot).grid(row=2, column=0, pady=(10, 0))
```

```
self.scatter_frame.columnconfigure(0, weight=1)
```

```
self.scatter_frame.rowconfigure(1, weight=1)
```

```

def create_pie_tab(self):

    title_label = ttk.Label(self.pie_frame, text="Distribusi Klasifikasi",
font=('Arial', 12, 'bold'))

    title_label.grid(row=0, column=0, pady=(0, 10))


    self.pie_fig, axes = plt.subplots(2, 2, figsize=(12, 8))

    self.pie_ax1, self.pie_ax2, self.pie_ax3, self.pie_ax4 = axes.flatten()

    self.pie_canvas = FigureCanvasTkAgg(self.pie_fig, self.pie_frame)

    self.pie_canvas.get_tk_widget().grid(row=1, column=0, sticky=(tk.W, tk.E,
tk.N, tk.S))


    ttk.Button(self.pie_frame, text="Refresh Pie Chart",
command=self.update_pie_chart).grid(row=2, column=0, pady=(10, 0))


    self.pie_frame.columnconfigure(0, weight=1)

    self.pie_frame.rowconfigure(1, weight=1)


def create_results_tab(self):

    title_label = ttk.Label(self.results_frame, text="Hasil Analisis dan Export",
font=('Arial', 12, 'bold'))

    title_label.grid(row=0, column=0, columnspan=2, pady=(0, 10))


    results_tree_frame = ttk.LabelFrame(self.results_frame, text="Hasil
Klasifikasi", padding="10")

    results_tree_frame.grid(row=1, column=0, columnspan=2, sticky=(tk.W, tk.E,
tk.N, tk.S), pady=(0, 10))

```

```

self.results_tree = ttk.Treeview(results_tree_frame, height=15)

self.results_tree.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))


results_scroll = ttk.Scrollbar(results_tree_frame, orient=tk.VERTICAL,
command=self.results_tree.yview)

results_scroll.grid(row=0, column=1, sticky=(tk.N, tk.S))

self.results_tree.configure(yscrollcommand=results_scroll.set)


stats_frame = ttk.LabelFrame(self.results_frame, text="Statistik",
padding="10")

stats_frame.grid(row=2, column=0, sticky=(tk.W, tk.E), padx=(0, 5))


self.stats_text = tk.Text(stats_frame, height=8, width=40)

self.stats_text.grid(row=0, column=0, sticky=(tk.W, tk.E))


export_frame = ttk.LabelFrame(self.results_frame, text="Export Data",
padding="10")

export_frame.grid(row=2, column=1, sticky=(tk.W, tk.E), padx=(5, 0))


ttk.Button(export_frame, text="Export Hasil",
command=self.export_results_csv).grid(row=0, column=0, pady=5, sticky=tk.W)

ttk.Button(export_frame, text="Export Grafik",
command=self.export_all_graphs).grid(row=1, column=0, pady=5, sticky=tk.W)

ttk.Button(export_frame, text="Export Laporan",
command=self.export_report).grid(row=2, column=0, pady=5, sticky=tk.W)


self.results_frame.columnconfigure(0, weight=1)

```

```

self.results_frame.columnconfigure(1, weight=1)
self.results_frame.rowconfigure(1, weight=1)
results_tree_frame.columnconfigure(0, weight=1)
results_tree_frame.rowconfigure(0, weight=1)

def import_data_file(self):
    try:
        filename = filedialog.askopenfilename(
            title="Pilih File Data",
            filetypes=[("All supported", "*.csv;*.xls;*.xlsx"), ("CSV files", "*.csv"),
("Excel files", "*.xls;*.xlsx")]
        )

        if filename:
            file_extension = filename.lower().split('.')[-1]

            if file_extension == 'csv':
                self.data = pd.read_csv(filename)
            elif file_extension in ['xls', 'xlsx']:
                self.data = pd.read_excel(filename, sheet_name=0)
            else:
                messagebox.showerror("Error", "Format file tidak didukung.")
                return

        # Clean column names

```

```

self.data.columns = self.data.columns.str.strip()

required_cols = ['Jenis_Minyak', 'Kondisi', 'Red', 'Green', 'Blue', 'Turb',
'Bau']

missing_cols = [col for col in required_cols if col not in self.data.columns]

if missing_cols:
    messagebox.showerror("Error", f"Kolom hilang: {'',
'.join(missing_cols)}")
    return

self.data = self.data.dropna()

self.update_data_preview()

self.status_label.config(text=f"Data dimuat: {len(self.data)} records",
foreground='green')

except Exception as e:
    messagebox.showerror("Error", f"Error: {str(e)}")

def generate_sample_data(self):
    try:
        sample_data = []
        no = 1

        # Data berdasarkan pengujian aktual
        oil_configs = {

```

```

        'minyak jernih': {
            'baru': {'r': (137, 142), 'g': (151, 156), 'b': (139, 147), 't': (158, 181),
'bau': (177, 181)}

        },

        'minyak sedang': {
            'sedang': {'r': (162, 174), 'g': (192, 207), 'b': (181, 193), 't': (370, 392),
'bau': (181, 296)}

        },

        'minyak keruh': {
            'keruh': {'r': (195, 221), 'g': (234, 265), 'b': (204, 232), 't': (476, 506),
'bau': (190, 276)}

        }
    }
}

```

```

for oil_type, conditions in oil_configs.items():
    for condition, ranges in conditions.items():
        for _ in range(20): # 20 samples per combination
            timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            sample_data.append({
                'No': no,
                'Timestamp': timestamp,
                'Jenis_Minyak': oil_type,
                'Kondisi': condition,
                'Red': random.randint(*ranges['r']),
                'Green': random.randint(*ranges['g']),
                'Blue': random.randint(*ranges['b']),
            })

```

```

        'Turb': random.randint(*ranges['t']),
        'Bau': random.randint(*ranges['bau'])
    })
    no += 1

self.data = pd.DataFrame(sample_data)
self.data.to_excel('sample_oil_data.xlsx', index=False)
self.data.to_csv('sample_oil_data.csv', index=False)

self.update_data_preview()

self.status_label.config(text=f"Sample generated: {len(self.data)} records",
foreground='green')

messagebox.showinfo("Sukses", f"Sample data: {len(self.data)}
records\nsaved as .xlsx and .csv")

except Exception as e:

    messagebox.showerror("Error", f"Error: {str(e)}")

def update_data_preview(self):
    if self.data is None:
        return

    columns = list(self.data.columns)
    self.data_tree['columns'] = columns
    self.data_tree['show'] = 'headings'

```

```

for col in columns:

    self.data_tree.heading(col, text=col)

    if col == 'No':

        self.data_tree.column(col, width=40)

    elif col == 'Timestamp':

        self.data_tree.column(col, width=130)

    else:

        self.data_tree.column(col, width=90)


for item in self.data_tree.get_children():

    self.data_tree.delete(item)


for idx, row in self.data.head(100).iterrows():

    self.data_tree.insert("", 'end', values=list(row))


def analyze_manual_input(self):

    try:

        oil_type = self.oil_type_var.get()
        condition = self.condition_var.get()
        red = float(self.r_var.get())
        green = float(self.g_var.get())
        blue = float(self.b_var.get())
        turb = float(self.turbidity_var.get())
        bau = float(self.gas_var.get())

```

```

if not (0 <= bau <= 1023):
    messagebox.showerror("Error", "Bau: 0-1023")
    return

if not (0 <= red <= 255 and 0 <= green <= 255 and 0 <= blue <= 255):
    messagebox.showerror("Error", "RGB: 0-255")
    return

if not (0 <= turb <= 1023):
    messagebox.showerror("Error", "Turb: 0-1023")
    return

gas_fuzzy = self.fuzzify_gas_sensor(bau)
color_fuzzy = self.fuzzify_color_sensor(red, green, blue)
turbidity_fuzzy = self.fuzzify_turbidity_sensor(turb)

quality_scores = self.fuzzy_rules(gas_fuzzy, color_fuzzy, turbidity_fuzzy)
final_score, classification = self.defuzzify(quality_scores)

self.manual_result.delete(1.0, tk.END)
result_text = f"HASIL ANALISIS\n{' '*25}\n\n"
result_text += f"Jenis: {oil_type}\n"
result_text += f"Kondisi: {condition}\n\n"
result_text += f"SENSOR:\n"
result_text += f"Red: {red}\n"
result_text += f"Green: {green}\n"

```

```

result_text += f"Blue: {blue}\n"
result_text += f"Turb: {turb}\n"
result_text += f"Bau: {bau}\n\n"
result_text += f"FUZZY SCORES:\n"
result_text += f"Bau-Fresh: {gas_fuzzy['fresh']:.3f}\n"
result_text += f"Bau-Med: {gas_fuzzy['medium']:.3f}\n"
result_text += f"Bau-Stale: {gas_fuzzy['stale']:.3f}\n\n"
result_text += f"Col-Clear: {color_fuzzy['clear']:.3f}\n"
result_text += f"Col-Yellow: {color_fuzzy['yellow']:.3f}\n"
result_text += f"Col-Brown: {color_fuzzy['brown']:.3f}\n\n"
result_text += f"Turb-Clear: {turbidity_fuzzy['clear']:.3f}\n"
result_text += f"Turb-Cloudy: {turbidity_fuzzy['cloudy']:.3f}\n"
result_text += f"Turb-VCloudy: {turbidity_fuzzy['very_cloudy']:.3f}\n\n"
result_text += f"KUALITAS:\n"
result_text += f"Baik: {quality_scores['Baik']:.3f}\n"
result_text += f"Sedang: {quality_scores['Sedang']:.3f}\n"
result_text += f"Buruk: {quality_scores['Buruk']:.3f}\n\n"
result_text += f"HASIL:\n"
result_text += f"Skor: {final_score:.1f}/100\n"
result_text += f"Klasifikasi: {classification}\n"

self.manual_result.insert(tk.END, result_text)

```

```

except ValueError:

```

```

    messagebox.showerror("Error", "Input nilai numerik valid")

```

```

except Exception as e:
    messagebox.showerror("Error", f"Error: {str(e)}")

def analyze_data(self):
    if self.data is None:
        messagebox.showwarning("Peringatan", "Muat data terlebih dahulu")
        return

    try:
        self.analysis_results = []

        for idx, row in self.data.iterrows():
            oil_type = row['Jenis_Minyak']
            condition = row['Kondisi']
            red = float(row['Red'])
            green = float(row['Green'])
            blue = float(row['Blue'])
            turb = float(row['Turb'])
            bau = float(row['Bau'])

            gas_fuzzy = self.fuzzify_gas_sensor(bau)
            color_fuzzy = self.fuzzify_color_sensor(red, green, blue)
            turbidity_fuzzy = self.fuzzify_turbidity_sensor(turb)

            quality_scores = self.fuzzy_rules(gas_fuzzy, color_fuzzy, turbidity_fuzzy)

```

```

final_score, classification = self.defuzzify(quality_scores)

result = {
    'No': row.get('No', idx + 1),
    'Timestamp': row.get('Timestamp', datetime.now().strftime('%Y-%m-%d %H:%M:%S')),
    'Jenis_Minyak': oil_type,
    'Kondisi': condition,
    'Red': red,
    'Green': green,
    'Blue': blue,
    'Turb': turb,
    'Bau': bau,
    'Score_Baik': quality_scores['Baik'],
    'Score_Sedang': quality_scores['Sedang'],
    'Score_Buruk': quality_scores['Buruk'],
    'Final_Score': final_score,
    'Klasifikasi': classification
}

self.analysis_results.append(result)

self.update_results_display()
self.update_histogram()
self.update_scatter_plot()
self.update_pie_chart()

```

```

        self.update_statistics()

        messagebox.showinfo("Sukses", f"Analisis selesai!
{len(self.analysis_results)} data diproses.")

    except Exception as e:
        messagebox.showerror("Error", f"Error: {str(e)}")

def update_results_display(self):
    if not self.analysis_results:
        return

    columns = ['No', 'Timestamp', 'Jenis_Minyak', 'Kondisi', 'Red', 'Green', 'Blue',
               'Turb', 'Bau', 'Final_Score', 'Klasifikasi']
    self.results_tree['columns'] = columns
    self.results_tree['show'] = 'headings'

    for col in columns:
        self.results_tree.heading(col, text=col.replace('_', ' '))
        if col == 'No':
            self.results_tree.column(col, width=40)
        elif col == 'Timestamp':
            self.results_tree.column(col, width=130)
        elif col == 'Klasifikasi':
            self.results_tree.column(col, width=100)

```

```

elif col in ['Final_Score', 'Jenis_Minyak']:
    self.results_tree.column(col, width=70)
elif col == 'Kondisi':
    self.results_tree.column(col, width=100)
else:
    self.results_tree.column(col, width=60)

for item in self.results_tree.get_children():
    self.results_tree.delete(item)

for result in self.analysis_results:
    values = []
    for col in columns:
        if col == 'Final_Score':
            values.append(f"{result[col]:.1f}")
        elif col in ['Red', 'Green', 'Blue', 'Turb', 'Bau']:
            values.append(f"{result[col]:.0f}")
        else:
            values.append(result[col])
    self.results_tree.insert("", 'end', values=values)

def update_histogram(self):
    if not self.analysis_results:
        return

```

```

try:
    for ax in self.hist_axes:
        ax.clear()

        bau_data = [r['Bau'] for r in self.analysis_results]
        r_data = [r['Red'] for r in self.analysis_results]
        g_data = [r['Green'] for r in self.analysis_results]
        b_data = [r['Blue'] for r in self.analysis_results]
        turb_data = [r['Turb'] for r in self.analysis_results]
        score_data = [r['Final_Score'] for r in self.analysis_results]

        self.hist_axes[0].hist(bau_data, bins=20, alpha=0.7, color='purple',
                                edgecolor='black')

        self.hist_axes[0].set_title('Sensor Bau', fontweight='bold')
        self.hist_axes[0].set_xlabel('Nilai')
        self.hist_axes[0].set_ylabel('Frekuensi')
        self.hist_axes[0].grid(True, alpha=0.3)

        self.hist_axes[1].hist(r_data, bins=20, alpha=0.7, color='red',
                                edgecolor='black')

        self.hist_axes[1].set_title('Sensor Red', fontweight='bold')
        self.hist_axes[1].set_xlabel('Nilai Red')
        self.hist_axes[1].grid(True, alpha=0.3)

        self.hist_axes[2].hist(g_data, bins=20, alpha=0.7, color='green',
                                edgecolor='black')

```

```

self.hist_axes[2].set_title('Sensor Green', fontweight='bold')

self.hist_axes[2].set_xlabel('Nilai Green')

self.hist_axes[2].grid(True, alpha=0.3)


self.hist_axes[3].hist(b_data, bins=20, alpha=0.7, color='blue',
edgecolor='black')

self.hist_axes[3].set_title('Sensor Blue', fontweight='bold')

self.hist_axes[3].set_xlabel('Nilai Blue')

self.hist_axes[3].grid(True, alpha=0.3)


self.hist_axes[4].hist(turb_data, bins=20, alpha=0.7, color='cyan',
edgecolor='black')

self.hist_axes[4].set_title('Sensor Turb', fontweight='bold')

self.hist_axes[4].set_xlabel('Nilai')

self.hist_axes[4].grid(True, alpha=0.3)


self.hist_axes[5].hist(score_data, bins=15, alpha=0.7, color='orange',
edgecolor='black')

self.hist_axes[5].set_title('Skor Kualitas', fontweight='bold')

self.hist_axes[5].set_xlabel('Skor')

self.hist_axes[5].grid(True, alpha=0.3)


plt.tight_layout()

self.hist_canvas.draw()


except Exception as e:

```

```

print(f"Error histogram: {str(e)}")

def update_scatter_plot(self):
    if not self.analysis_results:
        return

    try:
        for ax in [self.scatter_ax1, self.scatter_ax2, self.scatter_ax3,
self.scatter_ax4]:
            ax.clear()

            bau_data = [r['Bau'] for r in self.analysis_results]
            r_data = [r['Red'] for r in self.analysis_results]
            g_data = [r['Green'] for r in self.analysis_results]
            b_data = [r['Blue'] for r in self.analysis_results]
            turb_data = [r['Turb'] for r in self.analysis_results]
            score_data = [r['Final_Score'] for r in self.analysis_results]
            classifications = [r['Klasifikasi'] for r in self.analysis_results]

            color_map = {'Baik': 'green', 'Sedang': 'orange', 'Buruk': 'red'}
            colors = [color_map[c] for c in classifications]

            self.scatter_ax1.scatter(bau_data, score_data, c=colors, alpha=0.7, s=60)
            self.scatter_ax1.set_title('Bau vs Skor', fontweight='bold')
            self.scatter_ax1.set_xlabel('Sensor Bau')

```

```

self.scatter_ax1.set_ylabel('Skor')
self.scatter_ax1.grid(True, alpha=0.3)

self.scatter_ax2.scatter(turb_data, score_data, c=colors, alpha=0.7, s=60)
self.scatter_ax2.set_title('Turb vs Skor', fontweight='bold')
self.scatter_ax2.set_xlabel('Turb')
self.scatter_ax2.set_ylabel('Skor')
self.scatter_ax2.grid(True, alpha=0.3)

rgb_avg = [(r_data[i] + g_data[i] + b_data[i])/3 for i in range(len(r_data))]
self.scatter_ax3.scatter(rgb_avg, score_data, c=colors, alpha=0.7, s=60)
self.scatter_ax3.set_title('RGB Avg vs Skor', fontweight='bold')
self.scatter_ax3.set_xlabel('RGB Average')
self.scatter_ax3.set_ylabel('Skor')
self.scatter_ax3.grid(True, alpha=0.3)

self.scatter_ax4.scatter(bau_data, turb_data, c=colors, alpha=0.7, s=60)
self.scatter_ax4.set_title('Bau vs Turb', fontweight='bold')
self.scatter_ax4.set_xlabel('Bau')
self.scatter_ax4.set_ylabel('Turb')
self.scatter_ax4.grid(True, alpha=0.3)

from matplotlib.patches import Patch
legend_elements = [Patch(facecolor='green', label='Baik'),
                    Patch(facecolor='orange', label='Sedang'),

```

```

        Patch(facecolor='red', label='Buruk')]

self.scatter_ax4.legend(handles=legend_elements, loc='upper right')

plt.tight_layout()
self.scatter_canvas.draw()

except Exception as e:
    print(f"Error scatter: {str(e)}")

def update_pie_chart(self):
    if not self.analysis_results:
        return

    try:
        for ax in [self.pie_ax1, self.pie_ax2, self.pie_ax3, self.pie_ax4]:
            ax.clear()

        classifications = [r['Klasifikasi'] for r in self.analysis_results]
        class_counts = Counter(classifications)

        colors_class = ['green', 'orange', 'red']

        self.pie_ax1.pie(class_counts.values(), labels=class_counts.keys(),
                        autopct='%1.1f%%', colors=colors_class, startangle=90)
        self.pie_ax1.set_title('Klasifikasi Keseluruhan', fontweight='bold')

```

```
oil_types = [r['Jenis_Minyak'] for r in self.analysis_results]
```

```
oil_counts = Counter(oil_types)
```

```
colors_oil = ['lightblue', 'lightgreen', 'lightsalmon']
```

```
self.pie_ax2.pie(oil_counts.values(), labels=oil_counts.keys(),  
                autopct='%1.1f%%', colors=colors_oil, startangle=90)
```

```
self.pie_ax2.set_title('Per Jenis Minyak', fontweight='bold')
```

```
conditions = [r['Kondisi'] for r in self.analysis_results]
```

```
cond_counts = Counter(conditions)
```

```
colors_cond = ['lightgreen', 'yellow', 'lightcoral']
```

```
self.pie_ax3.pie(cond_counts.values(), labels=cond_counts.keys(),  
                autopct='%1.1f%%', colors=colors_cond, startangle=90)
```

```
self.pie_ax3.set_title('Per Kondisi', fontweight='bold')
```

```
# Distribusi per jenis minyak pertama yang ada
```

```
first_oil = list(oil_counts.keys())[0] if oil_counts else None
```

```
if first_oil:
```

```
    oil_data = [r for r in self.analysis_results if r['Jenis_Minyak'] == first_oil]
```

```
    if oil_data:
```

```
        oil_class = [r['Klasifikasi'] for r in oil_data]
```

```
        oil_class_counts = Counter(oil_class)
```

```

        self.pie_ax4.pie(oil_class_counts.values(),
labels=oil_class_counts.keys(),

                        autopct='%1.1f%%', colors=colors_class, startangle=90)

        self.pie_ax4.set_title(f'Klasifikasi {first_oil}', fontweight='bold')
    else:

        self.pie_ax4.text(0.5, 0.5, 'Tidak ada data',

                        ha='center', va='center', transform=self.pie_ax4.transAxes)

        self.pie_ax4.set_title('Klasifikasi Detail', fontweight='bold')

plt.tight_layout()

self.pie_canvas.draw()

except Exception as e:

    print(f"Error pie: {str(e)}")

def update_statistics(self):

    if not self.analysis_results:

        return

    try:

        self.stats_text.delete(1.0, tk.END)

        total_data = len(self.analysis_results)

        classifications = [r['Klasifikasi'] for r in self.analysis_results]

        class_counts = Counter(classifications)

```

```

scores = [r['Final_Score'] for r in self.analysis_results]

stats_text = "STATISTIK ANALISIS\n"
stats_text += "=" * 30 + "\n\n"
stats_text += f"Total Data: {total_data}\n\n"

stats_text += "DISTRIBUSI KLASIFIKASI:\n"
for classification, count in class_counts.items():
    percentage = (count / total_data) * 100
    stats_text += f"{classification}: {count} ({percentage:.1f}%) \n"

stats_text += f"\nSTATISTIK SKOR:\n"
stats_text += f"Rata-rata: {np.mean(scores):.2f} \n"
stats_text += f"Std Dev: {np.std(scores):.2f} \n"
stats_text += f"Min: {np.min(scores):.2f} \n"
stats_text += f"Max: {np.max(scores):.2f} \n"

stats_text += f"\nPER JENIS MINYAK:\n"
oil_types = list(set([r['Jenis_Minyak'] for r in self.analysis_results]))
for oil_type in oil_types:
    oil_data = [r for r in self.analysis_results if r['Jenis_Minyak'] == oil_type]
    if oil_data:
        oil_scores = [r['Final_Score'] for r in oil_data]

```

```

        stats_text += f"{oil_type}: {np.mean(oil_scores):.1f}
({len(oil_data)})\n"

```

```

self.stats_text.insert(tk.END, stats_text)

```

```

except Exception as e:

```

```

    print(f"Error stats: {str(e)}")

```

```

def export_results_csv(self):

```

```

    if not self.analysis_results:

```

```

        messagebox.showwarning("Peringatan", "Belum ada hasil")

```

```

        return

```

```

    try:

```

```

        filename = filedialog.asksaveasfilename(

```

```

            defaultextension=".xlsx",

```

```

            filetypes=[("Excel files", "*.xlsx"), ("CSV files", "*.csv")],

```

```

            title="Export Hasil"

```

```

        )

```

```

    if filename:

```

```

        df = pd.DataFrame(self.analysis_results)

```

```

        column_order = ['No', 'Timestamp', 'Jenis_Minyak', 'Kondisi', 'Red',
'Green',

```

```

        'Blue', 'Turb', 'Bau', 'Score_Baik', 'Score_Sedang',
        'Score_Buruk', 'Final_Score', 'Klasifikasi']

df = df[column_order]

if filename.lower().endswith('.csv'):
    df.to_csv(filename, index=False)
else:
    df.to_excel(filename, index=False, engine='openpyxl')

messagebox.showinfo("Sukses", f"Berhasil export: {filename}")

except Exception as e:
    messagebox.showerror("Error", f"Error: {str(e)}")

def export_all_graphs(self):
    try:
        folder = filedialog.askdirectory(title="Pilih Folder")

        if folder:
            self.hist_fig.savefig(f"{folder}/histogram.png", dpi=300,
bbox_inches='tight')

            self.scatter_fig.savefig(f"{folder}/scatter.png", dpi=300,
bbox_inches='tight')

            self.pie_fig.savefig(f"{folder}/pie_chart.png", dpi=300,
bbox_inches='tight')

```

```

        messagebox.showinfo("Sukses", f"Grafik disimpan di: {folder}")

except Exception as e:
    messagebox.showerror("Error", f"Error: {str(e)}")

def export_report(self):
    if not self.analysis_results:
        messagebox.showwarning("Peringatan", "Belum ada data")
        return

    try:
        filename = filedialog.asksaveasfilename(
            defaultextension=".txt",
            filetypes=[("Text files", "*.txt")],
            title="Export Laporan"
        )

        if filename:
            with open(filename, 'w', encoding='utf-8') as f:
                f.write("LAPORAN KLASIFIKASI KUALITAS MINYAK\n")
                f.write("GORENG\n")
                f.write("=" * 80 + "\n\n")
                f.write(f"Tanggal: {datetime.now().strftime('%d/%m/%Y\n%H:%M:%S')}\n")
                f.write(f"Total Data: {len(self.analysis_results)}\n\n")

```

```

classifications = [r['Klasifikasi'] for r in self.analysis_results]
class_counts = Counter(classifications)

f.write("DISTRIBUSI KLASIFIKASI:\n")
f.write("-" * 40 + "\n")
for classification, count in class_counts.items():
    percentage = (count / len(self.analysis_results)) * 100
    f.write(f"{classification}: {count} ({percentage:.1f}%) \n")

f.write("\nPER JENIS MINYAK:\n")
f.write("-" * 40 + "\n")
oil_types = list(set([r['Jenis_Minyak'] for r in self.analysis_results]))
for oil_type in oil_types:
    oil_data = [r for r in self.analysis_results if r['Jenis_Minyak'] ==
oil_type]

    if oil_data:
        oil_class = [r['Klasifikasi'] for r in oil_data]
        oil_counts = Counter(oil_class)
        f.write(f"\n{oil_type}:\n")
        for classification, count in oil_counts.items():
            perc = (count / len(oil_data)) * 100
            f.write(f" {classification}: {count} ({perc:.1f}%) \n")

f.write("\nDETAIL (50 data pertama):\n")

```

```

        f.write("-" * 100 + "\n")

        f.write(f'{"No":<4} {"Minyak":<15} {"Kondisi":<12} {"Red":<5}
{"Green":<5} {"Blue":<5} {"Turb":<5} {"Bau":<5} {"Skor":<6} {"Kelas":<8}\n')

        f.write("-" * 100 + "\n")

    for result in self.analysis_results[:50]:

        f.write(f'{"result[\"No\"]":<4} {"result[\"Jenis_Minyak\"]":<15}
{result[\"Kondisi\"]":<12} "

                f'{"result[\"Red\"]":<5.0f} {"result[\"Green\"]":<5.0f} "

                f'{"result[\"Blue\"]":<5.0f} {"result[\"Turb\"]":<5.0f} "

                f'{"result[\"Bau\"]":<5.0f} {"result[\"Final_Score\"]":<6.1f} "

                f'{"result[\"Klasifikasi\"]":<8}\n')

    messagebox.showinfo("Sukses", f"Laporan disimpan: {filename}")

except Exception as e:

    messagebox.showerror("Error", f"Error: {str(e)}")

def run(self):

    self.root.mainloop()

if __name__ == "__main__":

    try:

        app = OilQualityClassifier()

        app.run()

```

```
except Exception as e:  
    print(f"Error: {str(e)}")
```

Lampiran 3 Hasil Klasifikasi Minyak Goreng

No	Jenis_Min_yak	Kon disi	R	G	B	T ur b	B a u	Score _Baik	Score_ Sedang	Score_ Buruk	Final_ Score	Klasi fikasi
1	F	baru	143	148	150	177	184	0,812207	0,032765	1,79E-23	83,64284	Baik
2	F	goreng 1x	141	145	160	187	184	0,526122	0,042477	2,59E-21	82,38533	Baik
3	F	goreng 2x	144	145	160	194	187	0,278037	0,057395	7,27E-20	79,01126	Baik
4	F	goreng 3x	167	155	153	357	252	5,04E-06	0,338494	0,005587	49,43223	Sedang
5	F	goreng 4x	193	194	172	511	284	1,60E-11	6,69E-10	0,387865	15	Buruk
6	M	baru	141	145	162	174	189	0,673679	0,051895	3,83E-24	82,49672	Baik
7	M	goreng 1x	136	135	160	192	198	0,341108	0,029154	2,84E-20	82,24411	Baik
8	M	goreng 2x	137	149	161	169	196	0,526122	0,035015	2,80E-25	82,81603	Baik
9	M	goreng 3x	166	166	153	390	248	1,78E-05	0,346607	0,00465	49,53849	Sedang
10	M	goreng 4x	168	153	153	409	213	0,053634	0,33951	0,006685	54,10985	Sedang
11	K	baru	144	145	150	176	173	0,83218	0,038434	1,07E-23	83,4549	Baik
12	K	goreng 1x	139	147	160	171	171	0,617212	0,035612	8,03E-25	83,09074	Baik
13	K	goreng 2x	136	155	160	155	193	0,606531	0,029069	1,31E-28	83,39928	Baik

14	K	gor eng 3x	1 7 3	1 5 4	1 6 6	39 6	2 2 3	0,009 477	0,3779 22	0,0159 1	49,44 173	Seda ng
15	K	gor eng 4x	1 7 4	1 5 9	1 6 5	35 6	2 4 8	1,78E -05	0,3572 29	0,0183 32	48,29 33	Seda ng
16	CUR AH	bar u	1 4 5	1 4 6	1 6 2	18 2	1 8 1	0,521 369	0,0729 41	2,22E- 22	80,70 439	Baik
17	CUR AH	gor eng 1x	1 4 0	1 5 1	1 6 0	17 1	1 8 4	0,691 271	0,0388 07	8,03E- 25	83,13 957	Baik
18	CUR AH	gor eng 2x	1 4 3	1 5 3	1 5 6	19 8	1 8 6	0,175 131	0,0400 48	4,62E- 19	78,48 602	Baik
19	CUR AH	gor eng 3x	2 0 6	1 8 1	1 9 3	51 8	2 9 3	2,01E -13	6,47E- 11	0,4020 21	15	Buru k
20	CUR AH	gor eng 4x	1 9 3	1 9 9	1 7 0	45 3	3 2 4	3,58E -21	0,0015 34	0,1273 06	15,41 667	Buru k