LAMPIRAN

1. Berita Acara

PERKUMPULAN PENGELOLA PENDIDIKAN UMUM DAN TEKNOLOGI NASIONAL MALANG

INSTITUT TEKNOLOGI NASIONAL MALANG

FAKULTAS TEKNOLOGI INDUSTRI
FAKULTAS TEKNIK SIPIL DAN PERENCANAAN
PROGRAM PASCASARJANA MAGISTER TEKNIK

PT. BNI (PERSERO) MALANG
BANK NIAGA MALANG

Kampus I : Jl. Bendungan Sigura-gura No. 2 Telp. (0341) 551431 (Hunting), Fax. (0341) 553015 Malang 65145
Kampus II : Jl. Raya Karanglo, Km 2 Telp. (0341) 417636 Fax. (0341) 417634 Malang

BERITA ACARA UJIAN SKRIPSI

FAKULTAS TEKNOLOGI INDUSTRI

Nama	:	**Adi Julia Saputra**
Nim	:	**2118061**
Jurusan	:	**Teknik Informatika S-1**
Judul	:	**Penggunaan Metode Logistic Regression Untuk Analisis Sentimen Pembangunan Ibu Kota Nusantara Pada Media Sosial**

Dipertahankan Dihadapan Majelis Penguji Skripsi Jenjang Strata Satu(S-1) Pada

Hari	:	**Selasa**
Tanggal	:	**21 Januari 2025**
Nilai	:	**91 (A)**

Panitia Ujian Skripsi :
Ketua Majelis Penguji

Yosep Agus Pranoto, S.T., M.T.
NIP.P 1031000432

Anggota Penguji :

Dosen Penguji I

Yosep Agus Pranoto, S.T., M.T.
NIP.P 1031000432

Dosen Penguji II

Febriana Santi Wahyuni, S.Kom., M.Kom.
NIP.P 1031000425

2. Formulir Perbaikan Skripsi Dosen Penguji

PERKUMPULAN PENGELOLA PENDIDIKAN UMUM DAN TEKNOLOGI NASIONAL MALANG
INSTITUT TEKNOLOGI NASIONAL MALANG
FAKULTAS TEKNOLOGI INDUSTRI
FAKULTAS TEKNIK SIPIL DAN PERENCANAAN
PROGRAM PASCASARJANA MAGISTER TEKNIK
PT. BNI (PERSERO) MALANG
BANK NIAGA MALANG
Kampus I : Jl. Bendungan Sigura-gura No. 2 Telp. (0341) 551431 (Hunting), Fax. (0341) 553015 Malang 65145
Kampus II : Jl. Raya Karanglo, Km 2 Telp. (0341) 417636 Fax. (0341) 417634 Malang

FORMULIR PERBAIKAN SKRIPSI

Dalam pelaksanaan ujian skripsi jenjang Strata 1 Program Studi Teknik Informatika, maka perlu adanya perbaikan skripsi untuk mahasiswa :

NAMA : Adi Julia Saputra
NIM : 2118061
JURUSAN : Teknik Informatika S-1
JUDUL : **PENGGUNAAN METODE LOGISTIC REGRESSION UNTUK ANALISIS SENTIMEN PEMBANGUNAN IBU KOTA NUSANTARA PADA MEDIA SOSIAL**

No.	Penguji	Tanggal	Uraian	Paraf
1.	Penguji I	21 Januari 2025	1. Pahami implementasi metode di coding.	
2.	Penguji II	21 Januari 2025	1. Revisi program dengan penambahan fitur klasifikasi. 2. Revisi laporan.	

Anggota Penguji :

Dosen Penguji I

Yosep Agus Pranoto, S.T., M.T.
NIP.P 1031000432

Dosen Penguji II

Febriana Santi Wahyuni, S.Kom., M.Kom.
NIP.P 1031000425

Mengetahui :

Dosen Pembimbing I

Dr. Ir. Sentot Achmadi, M.Si.
NIP. 1039500281

Dosen Pembimbing II

Karina Auliasari, S.T., M.Eng.
NIP.P 1031000426

3. Tanda Tangan Bimbingan Dosen I

FORMULIR BIMBINGAN SKRIPSI

Nama : Adi Julia Saputra

Nim : 21180061

Masa Bimbingan : 21 Agustus 2024 s.d. 21 Februari 2025

Judul Skripsi : Penggunaan Metode Logistic Regression untuk Analisis Sentimen Pembangunan Ibu kota Nusantara Pada Media Sosial

No.	Tanggal	Uraian	Paraf Pembimbing
1.	3 - 10 - 2024	Konsultasi Aplikasi website	
2.	29 - 10 - 2024	Revisi hasil perbaikan proposal	
3.	30 - 10 - 2024	Proposal yang telah diperbaiki disetujui	
4.	19 - 11 - 2024	Konsultasi Laporan terkait revisi	
5.	22 - 11 - 2024	Konsultasi mengenai upload Jurnal	
6.	29 - 11 - 2024	Revisi laporan serta perbaikan	
7.	2 - 12 - 2024	Proposal yang telah diperbaiki disetujui	
8.	10 - 1 - 2025	Diskusi revisi sembas (laporan)	
9.	14 - 1 - 2025	Cek Revisi laporan	
10.	16 - 1 - 2025	ACC kompre	

Malang, 30 - 10 - 2024

Dosen Pembimbing

4. Tanda Tangan Bimbingan Dosen II

FORMULIR BIMBINGAN SKRIPSI

Nama : Adi Jaka Saputra

Nim : 211806 1

Masa Bimbingan : 21 Agustus 2024 s.d. 21 Februari 2025

Judul Skripsi : Penggunaan Metode Logistic Regression untuk Analisis Sentimen Pembangunan Ibu Kota Nusantara Pada Media Sosial

No.	Tanggal	Uraian	Paraf Pembimbing
1.	7 - 10 - 2024	Konsultasi hasil seminar judul	
2.	9 - 10 - 2024	Pengumpulan data opini	
3.	15 - 10 - 2024	Simulasi metode di Google Colab	
4.	21 - 10 - 2024	Database	
5.	29 - 10 - 2024	Web Dashboard	
6.	3 - 11 - 2024	Revisi hasil akurasi dg menambah capacity (gmlneuron & epoch)	
7.	15 - 11 - 2024	Revisi paper bag. hasil	
8.	20 - 11 - 2024	Submitt paper jurnal IJAI	
9.	26 - 11 - 2024	Revisi Laporan Bab III & IV	
10.	17 - 1 - 2025	Acc Laporan & Acc Maju Kompre	

Malang, 7 - 10 - 2024

Dosen Pembimbing

Kavina Auliasari

5. SK Dosen Pembimbing I

PERKUMPULAN PENGELOLA PENDIDIKAN UMUM DAN TEKNOLOGI NASIONAL MALANG

INSTITUT TEKNOLOGI NASIONAL MALANG

FAKULTAS TEKNOLOGI INDUSTRI
FAKULTAS TEKNIK SIPIL DAN PERENCANAAN
PROGRAM PASCASARJANA MAGISTER TEKNIK

PT. BNI (PERSERO) MALANG
BANK NIAGA MALANG

Kampus I : Jl. Bendungan Sigura-gura No. 2 Telp. (0341) 551431 (Hunting), Fax. (0341) 553015 Malang 65145
Kampus II : Jl. Raya Karanglo, Km 2 Telp. (0341) 417636 Fax. (0341) 417634 Malang

Malang, 04 Oktober 2024

Nomor : ITN-883/III.INF/TA/2024
Lampiran : ---
Perihal : *Pembimbing Utama Skripsi*

Kepada : **Yth. Bpk/Ibu Dr. Ir. Sentot Achmadi Msi.**
Dosen Program Studi Teknik Informatika S-1
Institut Teknologi Nasional
Malang

Dengan Hormat,
Sesuai dengan permohonan dan persetujuan dalam proposal skripsi untuk mahasiswa :

Nama : Adi Julia Saputra
Nim : 2118061
Prodi : Teknik Informatika S-1
Fakultas : Teknologi Industri

Maka dengan ini pembimbingan kami serahkan sepenuhnya kepada Saudara/i selama waktu 6 (enam) bulan, terhitung mulai tanggal :

21 Agustus 2024 s/d 21 Pebruari 2025

Sebagai satu syarat untuk menempuh Ujian Akhir Sarjana Teknik, Program Studi Teknik Informatika S-1.
Demikian agar maklum dan atas perhatian serta bantuannya kami sampaikan terima kasih.

Mengetahui
Program Studi Teknik Informatika S-1
Ketua,

Yosep Agus Pranoto, ST., MT.
NIP.P. 1031000432

Form S-4a

6. SK Dosen Pembimbing II

PERKUMPULAN PENGELOLA PENDIDIKAN UMUM DAN TEKNOLOGI NASIONAL MALANG

INSTITUT TEKNOLOGI NASIONAL MALANG

FAKULTAS TEKNOLOGI INDUSTRI
FAKULTAS TEKNIK SIPIL DAN PERENCANAAN
PROGRAM PASCASARJANA MAGISTER TEKNIK

PT. BNI (PERSERO) MALANG
BANK NIAGA MALANG

Kampus I : Jl. Bendungan Sigura-gura No. 2 Telp. (0341) 551431 (Hunting), Fax. (0341) 553015 Malang 65145
Kampus II : Jl. Raya Karanglo, Km 2 Telp. (0341) 417636 Fax. (0341) 417634 Malang

Malang, 04 Oktober 2024

Nomor : ITN-883/III.INF/TA/2024
Lampiran : ---
Perihal : *Pembimbing Pendamping Skripsi*

Kepada : **Yth. Bpk/Ibu Karina Auliasari ST., M.Eng.**
Dosen Pembina Program Studi Teknik Informatika S-1
Institut Teknologi Nasional
Malang

Dengan Hormat,
Sesuai dengan permohonan dan persetujuan dalam proposal skripsi untuk mahasiswa :

Nama : Adi Julia Saputra
Nim : 2118061
Prodi : Teknik Informatika S-1
Fakultas : Teknologi Industri

Maka dengan ini pembimbingan kami serahkan sepenuhnya kepada Saudara/i selama waktu 6 (enam) bulan, terhitung mulai tanggal :

21 Agustus 2024 s/d 21 Pebruari 2025

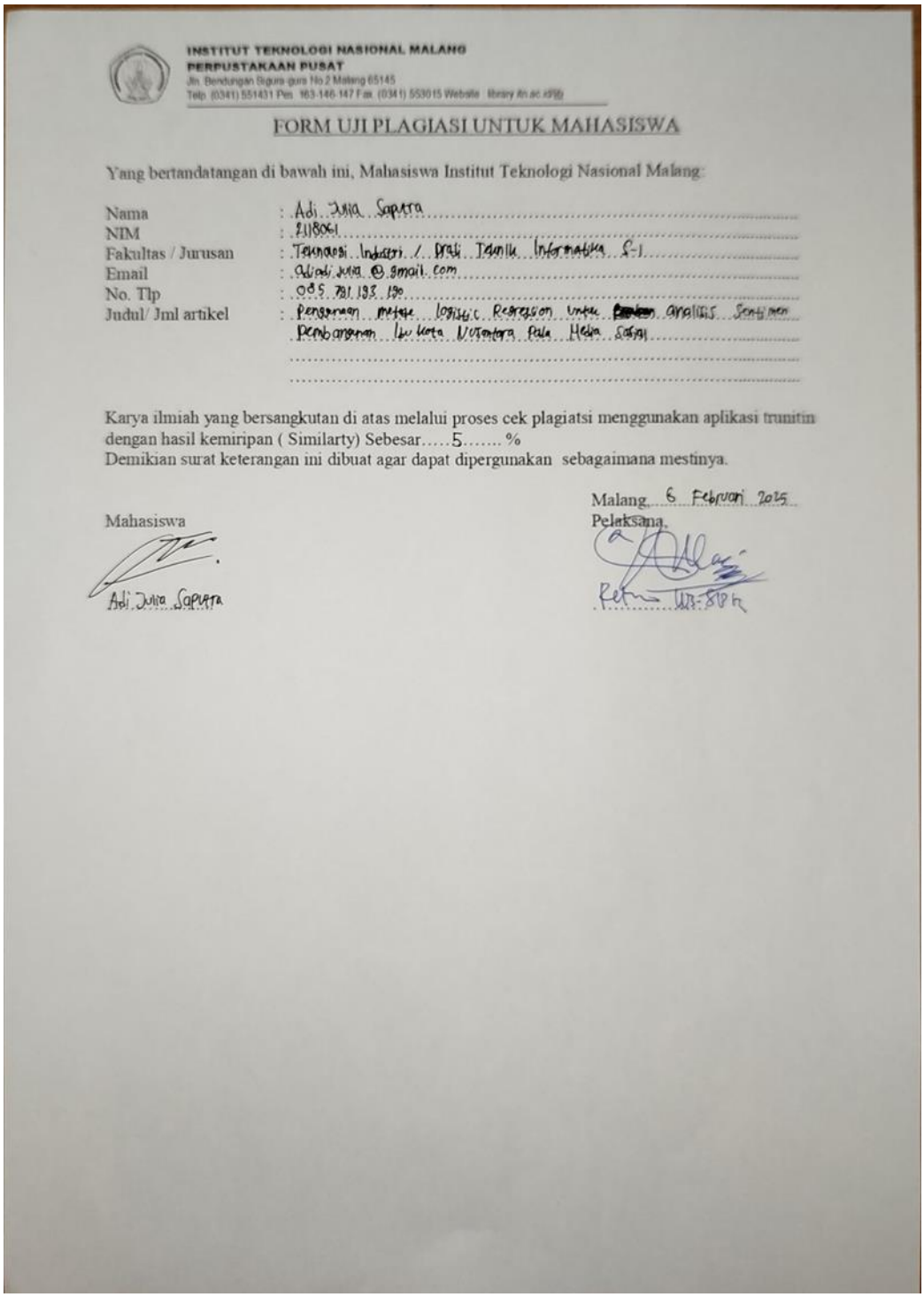
Sebagai satu syarat untuk menempuh Ujian Akhir Sarjana Teknik, Program Studi Teknik Informatika S-1.
Demikian agar maklum dan atas perhatian serta bantuannya kami sampaikan terima kasih.

Mengetahui
Program Studi Teknik Informatika S-1
K e t u a,

Yosep Agus Pranoto, ST., MT.
NIP.P. 1031000432

Form S-4a

7. Formulir Uji Plagiasi Perpustakaan ITN Kampus 2 Malang



INSTITUT TEKNOLOGI NASIONAL MALANG
PERPUSTAKAAN PUSAT
Jln. Bendungan Sigura-gura No.2 Malang 65145
Telp. (0341) 551431 Pes. 163-146-147 Fax. (0341) 553015 Website library.itn.ac.id

FORM UJI PLAGIASI UNTUK MAHASISWA

Yang bertandatangan di bawah ini, Mahasiswa Institut Teknologi Nasional Malang:

Nama : Adi Julia Saputra
NIM : 21180061
Fakultas / Jurusan : Teknologi Industri / Prodi Teknik Informatika S-1
Email : adiadi.julia@gmail.com
No. Tlp : 085 781 193 190
Judul/ Jml artikel : Pengaruh metode logistic Regression untuk [redacted] analisis Sentimen Pembangunan Ibu kota Nusantara Pada Media Sosial

Karya ilmiah yang bersangkutan di atas melalui proses cek plagiatsi menggunakan aplikasi trunitin dengan hasil kemiripan (Similarty) Sebesar.....5..... %
Demikian surat keterangan ini dibuat agar dapat dipergunakan sebagaimana mestinya.

Malang, 6 Februari 2025

Mahasiswa

Adi Julia Saputra

Pelaksana,

Retno Tri Sih H

8. Lembar Persentase Plagiasi Perpustakaan ITN Kampus 2 Malang

PENGGUNAAN METODE LOGISTIC REGRESSION UNTUK ANALISIS SENTIMEN PEMBANGUNAN IBU KOTA NUSANTARA PADA MEDIA SOSIAL

ORIGINALITY REPORT

5%	4%	3%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eprints.itn.ac.id Internet Source	4%
2	Andra Setiawan, Ryan Randy Suryono. "Analisis Sentimen Ibu Kota Nusantara menggunakan Algoritma Support Vector Machine dan Naïve Bayes", Edumatic: Jurnal Pendidikan Informatika, 2024 Publication	1%
3	Submitted to Universitas Airlangga Student Paper	1%
4	ojs.unud.ac.id Internet Source	1%

Exclude quotes	Off	Exclude matches	< 1%
Exclude bibliography	On		

LAMPIRAN SOURCE CODE

1. Source Code login.py

```
from flask import Blueprint, render_template, request, redirect,
url_for, session, flash, current_app
from werkzeug.security import check_password_hash
from db_config import connect_db  # Koneksi ke database

login_bp = Blueprint('login', __name__)

@login_bp.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email_username = request.form['email-username']
        password = request.form['password']

        db_connection = connect_db(current_app)
        cursor = db_connection.cursor(dictionary=True)

        try:
            query = "SELECT * FROM users WHERE email = %s OR
username = %s"
            cursor.execute(query,               (email_username,
email_username))
            user = cursor.fetchone()

            if  user  and  check_password_hash(user['password'],
password):
                session['user_id'] = user['id']
                session['role'] = user['role']
                session['username'] = user['username']
                flash('Login successful!', 'success')
                # Redirect berdasarkan role
                return redirect(url_for('dashboard.dashboard'))
            flash('Invalid email/username or password.', 'error')
        except Exception as e:
            flash(f"An error occurred: {str(e)}", 'error')
        finally:
            cursor.close()
            db_connection.close()
    return render_template('auth/login.html')
@login_bp.route('/logout')
def logout():
    session.clear()
    flash('You have been logged out.', 'info')
    return redirect(url_for('login.login'))
```

2. Source Code register.py

```
from flask import Blueprint, render_template, request, redirect,
url_for, flash, current_app
from werkzeug.security import generate_password_hash
from db_config import connect_db
```

```
import mysql.connector

register_bp = Blueprint('register', __name__)

@register_bp.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        # Ambil data dari form
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        terms = request.form.get('terms')

        # Validasi data input
        if not username or not email or not password:
            flash("All fields are required!", "error")
            return render_template('auth/register.html')

        if not terms:
            flash("You must agree to the terms and conditions.",
"error")
            return render_template('auth/register.html')

        # Hash password
        hashed_password = generate_password_hash(password)

        # Simpan ke database
        db_connection = connect_db(current_app)
        cursor = db_connection.cursor(dictionary=True)

        try:
            # Periksa apakah email atau username sudah digunakan
            check_query = "SELECT * FROM users WHERE email = %s
OR username = %s"
            cursor.execute(check_query, (email, username))
            existing_user = cursor.fetchone()

            if existing_user:
                flash("Email or username already exists. Please
choose another one.", "error")
                return render_template('auth/register.html')

            # Tambahkan user ke database
            insert_query = """
                INSERT INTO users (username, email, password,
role)
                VALUES (%s, %s, %s, 'user')
            """
            cursor.execute(insert_query,    (username,    email,
hashed_password))
            db_connection.commit()

            flash("Registration successful! Please log in.",
"success")
```

```
            return redirect(url_for('login.login'))

        except mysql.connector.Error as err:
            flash(f"An error occurred: {err}", "error")
        finally:
            cursor.close()
            db_connection.close()

    return render_template('auth/register.html')
```

3. Source Code dashboard.py

```
from flask import Blueprint, render_template, current_app,
session, redirect, url_for, flash, request, jsonify
from db_config import connect_db
import mysql.connector
import pickle
import re
import string
import pandas as pd
import nltk
from auth_utils import role_required
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

dashboard_bp = Blueprint('dashboard', __name__)

@dashboard_bp.route("/dashboard")
def dashboard():
    db_connection = connect_db(current_app)
    cursor = db_connection.cursor()

    try:
        # Menghitung total jumlah data di tabel data_sentiment
        cursor.execute("SELECT COUNT(*) FROM data_sentiment")
        total_data = cursor.fetchone()[0]

        # Menghitung jumlah kolom pada tabel data_sentiment
        cursor.execute("SHOW COLUMNS FROM data_sentiment")
        jumlah_kolom = len(cursor.fetchall())

        # Mengambil hasil akurasi dari tabel data_training_hasil
        cursor.execute("SELECT training_accuracy_percent FROM
data_training_hasil")
        training_accuracy = cursor.fetchone()
        training_accuracy_percent = training_accuracy[0] if
training_accuracy else "N/A"

        # Mengambil hasil akurasi dari tabel data_testing_hasil
        cursor.execute("SELECT testing_accuracy_percent FROM
data_testing_hasil")
        testing_accuracy = cursor.fetchone()
```

```
        testing_accuracy_percent = testing_accuracy[0] if
testing_accuracy else "N/A"

        # Mengambil data testing untuk prediksi
        cursor.execute("SELECT label FROM data_testing")
        rows = cursor.fetchall()

        if not rows:
            return jsonify({"error": "No data available for
testing."}), 400

        # Hitung jumlah kalimat positif, negatif, dan netral
        labels_list = [row[0] for row in rows]
        positive_count = labels_list.count("Positif")
        negative_count = labels_list.count("Negatif")
        neutral_count = labels_list.count("Netral")

        # Tentukan sentimen dominan
        dominant_sentiment = "Positif" if positive_count >
negative_count else "Negatif" if negative_count > positive_count
else "Positif" if positive_count > 0 else "Negatif"

    except mysql.connector.Error as err:
        total_data = "Error: " + str(err)
        jumlah_kolom = "Error: " + str(err)
        training_accuracy_percent = "Error"
        testing_accuracy_percent = "Error"
        dominant_sentiment = "Error"

    finally:
        cursor.close()
        db_connection.close()

    return render_template(
        "index.html",
        jumlah_data=total_data,
        jumlah_kolom=jumlah_kolom,
        training_accuracy_percent=training_accuracy_percent,
        testing_accuracy_percent=testing_accuracy_percent,
        dominant_sentiment=dominant_sentiment,
    )

# Inisialisasi blueprint dan file pickle model serta vectorizer
model_path = 'static/models/sentiment_model.pkl'
vectorizer_path = 'static/models/tfidf_vectorizer.pkl'

with open(vectorizer_path, 'rb') as f:
    vectorizer = pickle.load(f)

with open(model_path, 'rb') as f:
    model = pickle.load(f)

# Fungsi Preprocessing
def preprocess_text(text):
```

```
    text = text.lower()
    text = re.sub(r'\d+', '', text)  # Hapus angka
    text       =       text.translate(str.maketrans('',       '',
string.punctuation))  # Hapus tanda baca
    stop_words = set(stopwords.words('indonesian'))
    words = text.split()
    words = [word for word in words if word not in stop_words]
    return ' '.join(words)

@dashboard_bp.route('/klasifikasi_realtime',     methods=['GET',
'POST'])
def klasifikasi_realtime():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        if request.method == 'POST':
                # Mengambil teks dari form (atau body JSON jika
menggunakan AJAX)
                text   =   request.json.get('text')      #   Jika
menggunakan AJAX dan JSON

                if not text:
                    return jsonify({"error": "Tidak ada teks yang
diberikan"}), 400

                # Proses teks dan klasifikasi
                cleaned_text = preprocess_text(text)
                text_vectorized                                   =
vectorizer.transform([cleaned_text])
                prediction = model.predict(text_vectorized)[0]

                #    Mengembalikan    hasil    klasifikasi    dan
preprocessing dalam format JSON
                return jsonify({
                    "full_text": text,
                    "preprocessed_text": cleaned_text,
                    "sentiment": prediction
                })

        return render_template('dashboard.html')

    except mysql.connector.Error as err:
        return f"Error: {err}"
    finally:
        cursor.close()
        db.close()
@dashboard_bp.route('/add-record', methods=['GET', 'POST'])
@role_required('admin', 'super_admin')
def add_record():
    if request.method == 'POST':
        # Proses input data (misalnya simpan ke database)
        pass
    return render_template('add_record.html')
```

4. Source Code import_data.py

```
from flask import Blueprint, render_template, current_app,
request, redirect, render_template, flash, url_for
import pandas as pd
from db_config import connect_db
import mysql.connector
import chardet
from auth_utils import role_required

import_data_bp = Blueprint('import_data', __name__)

@import_data_bp.route('/import_data')
def import_data():
    db_connection = connect_db(current_app)
    cursor = db_connection.cursor()

    try:
        # Menghitung total jumlah data di tabel data_sentiment
        cursor.execute("SELECT COUNT(*) FROM data_sentiment")
        total_data = cursor.fetchone()[0]  # Mengambil hasil dari
query

        # Menghitung jumlah kolom pada tabel data_sentiment
        cursor.execute("SHOW COLUMNS FROM data_sentiment")
        jumlah_kolom  =  len(cursor.fetchall())    #  Menghitung
jumlah kolom

    except mysql.connector.Error as err:
        total_data = "Error: " + str(err)
        jumlah_kolom = "Error: " + str(err)

    finally:
        cursor.close()
        db_connection.close()

    return                          render_template('import.html',
jumlah_data=total_data, jumlah_kolom=jumlah_kolom)

@import_data_bp.route('/import_csv', methods=['POST'])
def import_csv():
    db = connect_db(current_app)
    cursor = db.cursor()

    if 'csv_file' not in request.files:
        flash("No file part", "error")
        return redirect(url_for('import_data.import_data'))

    file = request.files['csv_file']
    if file.filename == '':
        flash("No selected file", "error")
        return redirect(url_for('import_data.import_data'))

    if file:
```

```
        try:
            raw_data = file.read()
            result = chardet.detect(raw_data)
            detected_encoding = result['encoding']

            # Baca file CSV dengan encoding yang terdeteksi
            file.seek(0)
            data = pd.read_csv(
                file,
                sep=';',
                usecols=['full_text', 'username', 'label'],
                skip_blank_lines=True,
                on_bad_lines='skip',
                encoding=detected_encoding
            )

            # Menghapus spasi di sekitar nama kolom
            data.columns = data.columns.str.strip()

            # Memastikan kolom yang diperlukan ada
            required_columns = ['full_text', 'username', 'label']
            if not all(col in data.columns for col in
required_columns):
                flash("CSV file is missing required columns",
"error")
                return
redirect(url_for('import_data.import_data'))

            for _, row in data.iterrows():
                sql = "INSERT INTO data_sentiment (full_text,
username, label) VALUES (%s, %s, %s)"
                cursor.execute(sql,           (row['full_text'],
row['username'], row['label']))

            db.commit()
            cursor.close()
            db.close()

            flash("File imported successfully", "success")
            return redirect(url_for('import_data.import_data'))

        except ValueError as e:
            flash(f"ValueError: {e}", "error")
            return redirect(url_for('import_data.import_data'))
        except pd.errors.ParserError as e:
            flash(f"ParserError: {e}", "error")
            return redirect(url_for('import_data.import_data'))

    flash("File import failed", "error")
    return redirect(url_for('import_data.import_data'))
```

```
@import_data_bp.route('/show_data', methods=['GET'])
def show_data():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        # Menghitung jumlah username unik
        cursor.execute("SELECT  COUNT(DISTINCT  username)  FROM
data_sentiment")
        total_items = cursor.fetchone()[0]

        # Ambil semua data tanpa paginasi
        cursor.execute("SELECT  full_text,  username,  label  FROM
data_sentiment")
        data = cursor.fetchall()

        # Kolom yang akan ditampilkan
        columns = ['full_text', 'username', 'label']

        # Render template dengan data dari MySQL dan jumlah data
        return     render_template('import.html',     data=data,
columns=columns, jumlah_data=len(data))

    except mysql.connector.Error as err:
        flash("No selected file", "error")
        return f"Error: {err}"
    finally:
        cursor.close()
        db.close()

@import_data_bp.route('/reset_table_import', methods=['POST'])
def reset_table_import():
    db = connect_db(current_app)
    cursor = db.cursor()

    cursor.execute("TRUNCATE TABLE data_sentiment")

    db.commit()
    cursor.close()
    db.close()
    return render_template('import.html')
```

5. Source Code preprocessing.py

```
from   flask   import   Blueprint,   request,   render_template,
current_app, session, abort
import pandas as pd
import mysql.connector
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
from db_config import connect_db
```

```
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')

preprocessing_bp = Blueprint('preprocessing', __name__)

@preprocessing_bp.route('/hal_preprocessing',    methods=['GET',
'POST'])
def hal_preprocessing():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        if request.method == 'POST':
            user_role = session.get('user_role')
            if user_role not in ['admin', 'super_admin']:
                abort(403)
            preprocessing()


        cursor.execute("SELECT   username,   label,   full_text,
text_stemmed FROM data_preprocessing")
        rows = cursor.fetchall()

        # Membuat DataFrame dari data yang diambil untuk dikirim
ke template
        data = pd.DataFrame(rows, columns=['username', 'label',
'full_text','text_stemmed'])
        columns = data.columns.tolist()
        data_list = data.values.tolist()

        return        render_template('preprocessing_data.html',
columns=columns, data=data_list, jumlah_data=len(data_list))

    except mysql.connector.Error as err:
        return f"Error: {err}"
    finally:
        cursor.close()
        db.close()

@preprocessing_bp.route('/preprocessing')
def preprocessing():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        # Mengambil data dari tabel data_sentiment
        cursor.execute("SELECT username, full_text, label FROM
data_sentiment")
        rows = cursor.fetchall()

        # Membuat DataFrame dari data yang diambil
        data    =    pd.DataFrame(rows,    columns=['username',
'full_text', 'label'])
```

```
        # 1. Handling missing value
        data = data.dropna()

        # 2. Cek duplikasi data
        data = data.drop_duplicates()

        # 3. Casefolding
        data['full_text'] = data['full_text'].str.lower()

        # 4. Cleansing
        def clean_text(tweet):
            tweet = re.sub(r'http\S+|www\.\S+', '', tweet)   #
Menghapus URL
            tweet = re.sub(r'<.*?>', '', tweet)  # Menghapus HTML
tags
            tweet = re.sub(r'@\S+', '', tweet)    # Menghapus
mention
            tweet = re.sub(r'['
                        u'\U0001F600-\U0001F64F'
                        u'\U0001F300-\U0001F5FF'
                        u'\U0001F680-\U0001F6FF'
                        u'\U0001F1E0-\U0001F1FF'
                        ']+', '', tweet)  # Menghapus emoji
            tweet = re.sub(r'\d+', '', tweet)  # Menghapus angka
            tweet = re.sub(r'[^a-zA-Z\s]', '', tweet)  # Menghapus
simbol dan tanda baca, kecuali spasi
            return tweet

        data['text_cleaned']                                    =
data['full_text'].apply(clean_text)

        # 5. Normalisasi
        kamus_path = "static/kamus/kamus_kata_alay.xlsx"
        slang_word = pd.read_excel(kamus_path)

        # Membuat dictionary untuk kata slang
        slang_word_dict = {row[0]: row[1] for _, row in
slang_word.iterrows()}

        def normalize_text(text):
            """
            Mengganti kata-kata slang dalam teks dengan kata
standar sesuai kamus.
            """
            text = text.split()   # Tokenisasi sederhana dengan
spasi
            text = [slang_word_dict[term] if term in
slang_word_dict else term for term in text]
            return ' '.join(text)  # menggabungkan kembali menjadi
string

        data['text_normalized']                                 =
data['text_cleaned'].apply(normalize_text)
```

```
        # 6. Tokenizing
        nltk.download('punkt')
        data['text_tokenized']                                          =
data['text_normalized'].apply(word_tokenize)

        # 7. Stopwords removal
        nltk.download('stopwords')
        stop_words = set(stopwords.words('indonesian'))

        def remove_stopwords(text):
            return  [word  for  word  in  text  if  word  not  in
stop_words]

        data['text_filtered']                                           =
data['text_tokenized'].apply(remove_stopwords)

        # 8. Stemming
        # untuk mengubah kata ke bentuk dasar, e.g. (berlarian =
lari)
        factory = StemmerFactory()
        stemmer = factory.create_stemmer()

        def stem_text(text):
            return [stemmer.stem(word) for word in text]

        data['text_stemmed']                                            =
data['text_filtered'].apply(stem_text)

        # Menggabungkan kembali kata-kata hasil stemming menjadi
kalimat
        data['text_stemmed'] = data['text_stemmed'].apply(lambda
x: ' '.join(x))

        # 9. Hapus data sebelumnya di tabel data_preprocessing
        cursor.execute("DELETE FROM data_preprocessing")

        #    10.    Simpan    hasil    preprocessing    ke    tabel
data_preprocessing
        for _, row in data.iterrows():
            sql  =  "INSERT  INTO  data_preprocessing  (username,
label, full_text, text_stemmed) VALUES (%s, %s, %s, %s)"
            cursor.execute(sql,  (row['username'],  row['label'],
row['full_text'], row['text_stemmed']))

        db.commit()
        return render_template('preprocessing_data.html')

    except mysql.connector.Error as err:
        return f"Error: {err}"
    finally:
        cursor.close()
        db.close()
```

```
@preprocessing_bp.route('/reset_table_preprocessing',
methods=['POST'])
def reset_table_preprocessing():
    db = connect_db(current_app)
    cursor = db.cursor()

    cursor.execute("TRUNCATE TABLE data_preprocessing")

    db.commit()
    cursor.close()
    db.close()
    return render_template('preprocessing_data.html')
```

6. Source Code feature_extraction.py

```
from sklearn.feature_extraction.text import TfidfVectorizer
from flask import Blueprint, current_app, render_template,
request, url_for, redirect
import pandas as pd
import mysql.connector
import matplotlib
import pickle
from db_config import connect_db
from sklearn.model_selection import train_test_split
matplotlib.use('Agg')

feature_extraction_bp = Blueprint('feature_extraction', __name__)

@feature_extraction_bp.route('/hal_feature_extraction',
methods=['GET', 'POST'])
def hal_feature_extraction():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        if request.method == 'POST':
            feature_extraction()
            return
redirect(url_for('feature_extraction.hal_feature_extraction'))

        cursor.execute("SELECT username, label, text_stemmed,
text_extraction FROM data_extraction")
        rows = cursor.fetchall()
        data = pd.DataFrame(rows, columns=['username', 'label',
'text_stemmed', 'text_extraction'])

        # Menghapus kolom text_extraction sebelum mengirim data
ke template
        data = data.drop(columns=['text_extraction'])

        columns = data.columns.tolist()
        data_list = data.values.tolist()
```

```
        return        render_template('feature_extraction.html',
columns=columns, data=data_list, jumlah_data=len(data_list))

    except mysql.connector.Error as err:
        return f"Error: {err}"
    finally:
        cursor.close()
        db.close()

@feature_extraction_bp.route('/feature_extraction',
methods=['POST'])
def feature_extraction():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        # Mengambil data dari tabel data_preprocessing
        cursor.execute("SELECT   username,   label,   full_text,
text_stemmed FROM data_preprocessing")
        rows = cursor.fetchall()

        if not rows:
            return "No data available for TF-IDF extraction."

        # Memisahkan data ke dalam list
        username_list,        label_list,        full_text_list,
text_stemmed_list = zip(*rows)

        # Inisialisasi TF-IDF Vectorizer
        vectorizer = TfidfVectorizer()

        # Melakukan transformasi TF-IDF
        tfidf_matrix                                              =
vectorizer.fit_transform(text_stemmed_list)
        tfidf_features = vectorizer.get_feature_names_out()

        # Menyimpan hasil TF-IDF ke dalam tabel data_extraction
        insert_query = """
        INSERT INTO data_extraction (username, label, full_text,
text_stemmed, text_extraction)
        VALUES (%s, %s, %s, %s, %s)
        """
        for i, row in enumerate(tfidf_matrix.toarray()):
            #  Konversi  vektor  TF-IDF  menjadi  string  yang
dipisahkan koma
            text_extraction = ','.join(map(str, row))
            cursor.execute(insert_query, (
                username_list[i],
                label_list[i],
                full_text_list[i],
                text_stemmed_list[i],
                text_extraction
            ))
```

```
        db.commit()

        # Simpan vectorizer ke dalam file pickle
        vectorizer_path = 'static/models/tfidf_vectorizer.pkl'
        with open(vectorizer_path, 'wb') as f:
            pickle.dump(vectorizer, f)

        return "TF-IDF successfully extracted and stored in
data_extraction."

    except mysql.connector.Error as err:
        return f"Error: {err}"

    finally:
        cursor.close()
        db.close()

@feature_extraction_bp.route('/split_data', methods=['POST'])
def split_data_route():
    result = split_data_proses()
    return
redirect(url_for('feature_extraction.hal_feature_extraction',
message=result))

@feature_extraction_bp.route('/split_data_proses',
methods=['POST'])
def split_data_proses():
    db = connect_db(current_app)
    cursor = db.cursor()
    try:
        # Mengambil data dari tabel data_extraction
        cursor.execute("SELECT   username,   label,   full_text,
text_stemmed, text_extraction FROM data_extraction")
        rows = cursor.fetchall()

        if not rows:
            return "No data available for splitting."

        # Membuat DataFrame
        data = pd.DataFrame(rows, columns=['username', 'label',
'full_text', 'text_stemmed', 'text_extraction'])

        # Melakukan split data
        train_data,    test_data    =    train_test_split(data,
test_size=0.2, random_state=42)

        cursor.execute("DELETE FROM data_training")
        cursor.execute("DELETE FROM data_testing")

        # Menyimpan data training ke tabel data_training
        for _, row in train_data.iterrows():
            sql_train = """
                INSERT  INTO  data_training  (username,  label,
full_text, text_stemmed, text_extraction)
```

```
                VALUES (%s, %s, %s, %s, %s)
            """
            cursor.execute(sql_train,           (row['username'],
row['label'],       row['full_text'],       row['text_stemmed'],
row['text_extraction']))

        # Menyimpan data testing ke tabel data_testing
        for _, row in test_data.iterrows():
            sql_test = """
                INSERT   INTO   data_testing   (username,   label,
full_text, text_stemmed, text_extraction)
                VALUES (%s, %s, %s, %s, %s)
            """
            cursor.execute(sql_test,           (row['username'],
row['label'],       row['full_text'],       row['text_stemmed'],
row['text_extraction']))

        db.commit()
        return "Data successfully split into training and testing
sets."



    except mysql.connector.Error as err:
        return f"Error: {err}"
    finally:
        cursor.close()
        db.close()

@feature_extraction_bp.route('/reset_table_extraction',
methods=['POST'])
def reset_table_extraction():
    db = connect_db(current_app)
    cursor = db.cursor()

    cursor.execute("TRUNCATE TABLE data_extraction")

    db.commit()
    cursor.close()
    db.close()
    return render_template('feature_extraction.html')
```

7. Source Code traintest.py

```
from flask import Blueprint, jsonify, current_app, request,
url_for, redirect
import mysql.connector
import numpy as np
import json
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from      sklearn.metrics      import      classification_report,
confusion_matrix
from db_config import connect_db
import os
import pickle


traintest_bp = Blueprint('train_test', __name__)

def check_data_exists(cursor):
    cursor.execute("SELECT COUNT(*) FROM data_training_hasil")
    training_count = cursor.fetchone()[0]
    cursor.execute("SELECT COUNT(*) FROM data_testing_hasil")
    testing_count = cursor.fetchone()[0]
    return training_count > 0, testing_count > 0

@traintest_bp.route('/trainingtesting', methods=['POST'])
def trainingtesting():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        # Mengambil parameter redirect_page dari form agar kembali
ke form semulaa
        redirect_page = request.form.get('redirect_page')

        # Cek apakah data sudah ada di tabel
        training_exists,            testing_exists            =
check_data_exists(cursor)

        if training_exists and testing_exists:
            return redirect(url_for(redirect_page))

        # Ambil data dari tabel data_extraction
        cursor.execute("SELECT   label,   text_extraction   FROM
data_extraction")
        rows = cursor.fetchall()

        if not rows:
            return  jsonify({"error":  "No  data  available  for
training."}), 400

        text_labeled, text_extraction = zip(*rows)

        X = []
        y = []
        invalid_data_count = 0

        for    extraction,    label    in    zip(text_extraction,
text_labeled):
            if extraction and label:
                try:
                    extraction_values     =     list(map(float,
extraction.split(',')))
```

```
                    X.append(extraction_values)
                    y.append(label)
                except Exception as e:
                    print(f"Error processing data: {e} for
extraction: {extraction}")
                    invalid_data_count += 1
                    continue
            else:
                invalid_data_count += 1

        if len(X) != len(y):
            return jsonify({"error": "Mismatch in number of
samples between features and labels."}), 400

        if len(X) == 0:
            return jsonify({"error": "No valid data available for
training after cleaning invalid data."}), 400

        print(f"Valid data count: {len(X)}")
        print(f"Invalid data count: {invalid_data_count}")

        X = np.array(X)
        y = np.array(y)

        # Pisahkan data menjadi data training dan testing (80%
training, 20% testing)
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

        # Pelatihan model Logistic Regression
        logreg = LogisticRegression(penalty='l2',
solver='liblinear', random_state=0, class_weight='balanced')
        # solver='liblinear' untuk pengoptimalan yang digunakan
oleh Logistic Regression. random_state=0: Mengatur nilai acak
untuk hasil yang konsisten.

        # Cek apakah model sudah ada
        model_path = 'static/models/sentiment_model.pkl'
        vectorizer_path = 'static/models/tfidf_vectorizer.pkl'

        if os.path.exists(model_path) and
os.path.exists(vectorizer_path):
            return jsonify({"message": "Model sudah ada, tidak
perlu training ulang."}), 200

        logreg.fit(X_train, y_train)

        # Mengukur akurasi pada training set
        score_train = logreg.score(X_train, y_train)
        report_train = classification_report(y_train,
logreg.predict(X_train), output_dict=True)

        metrics_summary_train = {
```

```
            "precision":                    report_train['weighted
avg']['precision'],
            "recall": report_train['weighted avg']['recall'],
            "f1_score":    report_train['weighted    avg']['f1-
score'],
            "support": report_train['weighted avg']['support'],
        }

        # Menyimpan hasil ke tabel data_training_hasil
        model_name = 'Logistic Regression'
        classification_report_train_json                          =
json.dumps(report_train)
        training_accuracy_percent = score_train * 100

        cursor.execute("""
            INSERT   INTO   data_training_hasil   (`model_name`,
`training_accuracy`,                `training_accuracy_percent`,
`classification_report`, `precision_score`, `recall`, `f1_score`,
`support`)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """, (
            model_name,
            score_train,
            training_accuracy_percent,
            classification_report_train_json,
            metrics_summary_train["precision"],
            metrics_summary_train["recall"],
            metrics_summary_train["f1_score"],
            metrics_summary_train["support"],
        ))
        db.commit()

        # Evaluasi model pada data testing
        score_test = logreg.score(X_test, y_test)
        report_test       =       classification_report(y_test,
logreg.predict(X_test), output_dict=True)

        metrics_summary_test = {
            "precision":                    report_test['weighted
avg']['precision'],
            "recall": report_test['weighted avg']['recall'],
            "f1_score": report_test['weighted avg']['f1-score'],
            "support": report_test['weighted avg']['support'],
        }

        # Menyimpan hasil ke tabel data_testing_hasil
        classification_report_test_json                           =
json.dumps(report_test)
        testing_accuracy_percent = score_test * 100

        cursor.execute("""
            INSERT   INTO   data_testing_hasil   (`model_name`,
`testing_accuracy`,                `testing_accuracy_percent`,
```

```
`classification_report`, `precision_score`, `recall`, `f1_score`,
`support`)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """, (
            model_name,
            score_test,
            testing_accuracy_percent,
            classification_report_test_json,
            metrics_summary_test["precision"],
            metrics_summary_test["recall"],
            metrics_summary_test["f1_score"],
            metrics_summary_test["support"],
        ))
        db.commit()

        # Menghitung confusion matrix
        y_pred = logreg.predict(X_test)
        cm = confusion_matrix(y_test, y_pred)

        # Membuat heatmap untuk confusion matrix
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm,   annot=True,   fmt='d',   cmap='Blues',
xticklabels=np.unique(y), yticklabels=np.unique(y))
        plt.ylabel('Actual Labels')
        plt.xlabel('Predicted Labels')
        plt.title('Confusion Matrix')

        cm_image_path  =  os.path.join(current_app.static_folder,
'images', 'confusion_matrix.png')
        plt.savefig(cm_image_path)
        plt.close()

        with  open('static/models/sentiment_model.pkl',  'wb')  as
f:
            pickle.dump(logreg, f)
        return redirect(url_for(redirect_page))
    except mysql.connector.Error as err:
        return jsonify({"error": f"Database error: {err}"}), 500
    except Exception as e:
        return jsonify({"error": str(e)}), 500
    finally:
        cursor.close()
        db.close()
```

8. Source Code datatraining.py

```
from flask import (
    Blueprint,
    render_template,
    current_app,
    Blueprint,
)
import pandas as pd
import mysql.connector
```

```
from db_config import connect_db
import json

datatraining_bp = Blueprint("data_training", __name__)

# untuk menampilkan hasilnya di halaman utama
@datatraining_bp.route("/hal_data_training",     methods=["GET",
"POST"])
def hal_data_training():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        # Jika metode adalah GET, ambil data untuk ditampilkan
        # Data Sentimen
        cursor.execute("SELECT username, label, text_stemmed FROM
data_training")
        rows_sentimen = cursor.fetchall()

        # Hasil Training
        cursor.execute(
            "SELECT        model_name,        training_accuracy,
training_accuracy_percent,  precision_score,  recall,  f1_score,
support FROM data_training_hasil"
        )
        rows_training = cursor.fetchall()

        # Membuat DataFrame dari data yang diambil
        data_sentimen = pd.DataFrame(
            rows_sentimen,     columns=["username",     "label",
"text_stemmed"]
        )
        training_results = pd.DataFrame(
            rows_training,
            columns=[
                "model_name",
                "training_accuracy",
                "training_accuracy_percent",
                "precision_score",
                "recall",
                "f1_score",
                "support",
            ],
        )

        classification_report_table = []   # Default jika tidak
ada data
        classification_report_columns  =  ["Class",  "Precision",
"Recall", "F1-Score", "Support"]

        # Jika ada hasil testing di database, ambil dan masukkan
ke classification_report_table
        cursor.execute("SELECT    classification_report    FROM
data_training_hasil WHERE model_name = 'Logistic Regression'")
```

```
        classification_report_json = cursor.fetchone()

        classification_report_table = []

        if classification_report_json:
            try:
                report_test                                    =
json.loads(classification_report_json[0])   # Parsing JSON dari
database

                for label, metrics in report_test.items():
                    if isinstance(metrics, dict) and label not in
["accuracy", "macro avg", "weighted avg"]:  # Proses hanya data
label
                        classification_report_table.append(
                            [
                                label,
                                round(metrics.get("precision",
0), 2),  # Ganti precision_score dengan precision
                                round(metrics.get("recall",  0),
2),  # Gunakan get untuk keamanan
                                round(metrics.get("f1-score",
0), 2),  # Gunakan get untuk keamanan
                                metrics.get("support",  0),    #
Gunakan get untuk keamanan
                            ]
                        )

                # Tambahkan rata-rata weighted
                if "weighted avg" in report_test:
                    classification_report_table.append(
                        [
                            "Weighted Avg",
                            round(report_test["weighted
avg"]["precision"], 2),
                            round(report_test["weighted
avg"]["recall"], 2),
                            round(report_test["weighted
avg"]["f1-score"], 2),
                            report_test["weighted
avg"]["support"],
                        ]
                    )
            except json.JSONDecodeError:
                print("Error  decoding  JSON  from  classification
report.")
            except Exception as e:
                print(f"An error occurred: {e}")
        else:
            print("No  classification  report  found  for  the
specified model.")

        return render_template(
            "data_training.html",
```

```
            data_sentimen=data_sentimen.values.tolist(),
            data_training=training_results.values.tolist(),
            jumlah_data_sentimen=len(data_sentimen),
            jumlah_data_training=len(training_results),

classification_report_table=classification_report_table,

classification_report_columns=classification_report_columns,
        )
    except mysql.connector.Error as err:
        return f"Error: {err}"
    finally:
        cursor.close()
        db.close()


@datatraining_bp.route("/reset_table_training",
methods=["POST"])
def reset_table_training():
    db = connect_db(current_app)
    cursor = db.cursor()

    # Hapus semua data dari tabel
    cursor.execute(
        "TRUNCATE TABLE data_training_hasil"
    )   # Ini akan menghapus semua data dan mengatur ulang
AUTO_INCREMENT
    cursor.execute("TRUNCATE TABLE data_training")
    db.commit()
    cursor.close()
    db.close()
    return render_template("data_training.html")
```

9. Source Code datatesting.py

```
from flask import Blueprint, render_template, current_app
import pandas as pd
import mysql.connector
from db_config import connect_db
import json

datatesting_bp = Blueprint("data_testing", __name__)


@datatesting_bp.route("/hal_data_testing",       methods=["GET",
"POST"])
def hal_data_testing():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        # Data Sentimen
        cursor.execute(
```

```
            "SELECT    username,    label,    text_stemmed    FROM
data_testing"
        )
        rows_sentimen = cursor.fetchall()  # Pastikan hasil query
pertama dikonsumsi

        # Hasil Testing
        cursor.execute(
            "SELECT        model_name,        testing_accuracy,
testing_accuracy_percent,  precision_score,  recall,  f1_score,
support FROM data_testing_hasil"
        )
        rows_testing = cursor.fetchall()  # Pastikan hasil query
pertama dikonsumsi
        # Ensure you handle the rows properly before using them
in a DataFrame

        # Membuat DataFrame dari data yang diambil untuk dikirim
ke template
        data_sentimen = pd.DataFrame(
            rows_sentimen,
            columns=["username", "label", "text_stemmed"],
        )
        data_testing = pd.DataFrame(
            rows_testing,
            columns=[
                "model_name",
                "testing_accuracy",
                "testing_accuracy_percent",
                "precision_score",
                "recall",
                "f1_score",
                "support",
            ],
        )

        classification_report_table = []   # Default jika tidak
ada data
        classification_report_columns = ["Class", "Precision",
"Recall", "F1-Score", "Support"]

        # Jika ada hasil testing di database, ambil dan masukkan
ke classification_report_table
        cursor.execute("SELECT    classification_report    FROM
data_testing_hasil WHERE model_name = 'Logistic Regression'")
        classification_report_json = cursor.fetchone()

        classification_report_table = []

        if classification_report_json:
            try:
                report_test                                    =
json.loads(classification_report_json[0])   # Parsing JSON dari
database
```

```
                for label, metrics in report_test.items():
                    if isinstance(metrics, dict) and label not in
["accuracy", "macro avg", "weighted avg"]:  # Proses hanya data
label
                        classification_report_table.append(
                            [
                                label,
                                round(metrics.get("precision",
0), 2),  # Ganti precision_score dengan precision
                                round(metrics.get("recall",  0),
2),  # Gunakan get untuk keamanan
                                round(metrics.get("f1-score",
0), 2),  # Gunakan get untuk keamanan
                                metrics.get("support",  0),    #
Gunakan get untuk keamanan
                            ]
                        )

                # Tambahkan rata-rata weighted
                if "weighted avg" in report_test:
                    classification_report_table.append(
                        [
                            "Weighted Avg",
                            round(report_test["weighted
avg"]["precision"], 2),
                            round(report_test["weighted
avg"]["recall"], 2),
                            round(report_test["weighted
avg"]["f1-score"], 2),
                            report_test["weighted
avg"]["support"],
                        ]
                    )
            except json.JSONDecodeError:
                print("Error  decoding  JSON  from  classification
report.")
            except Exception as e:
                print(f"An error occurred: {e}")


        # Kirim ke template
        return render_template(
            "data_testing.html",
            data_sentimen=data_sentimen.values.tolist(),
            data_testing=data_testing.values.tolist(),
            jumlah_data_sentimen=len(data_sentimen),
            jumlah_data_testing=len(data_testing),

classification_report_table=classification_report_table,

classification_report_columns=classification_report_columns,
        )
```

```
    except mysql.connector.Error as err:
        return f"Error: {err}"
    finally:
        cursor.close()
        db.close()


@datatesting_bp.route("/reset_table_testing", methods=["POST"])
def reset_table_testing():
    db = connect_db(current_app)
    cursor = db.cursor()

    # Hapus semua data dari tabel
    cursor.execute(
        "TRUNCATE TABLE data_testing_hasil"
    )    # Ini akan menghapus semua data dan mengatur ulang
AUTO_INCREMENT
    cursor.execute(
        "TRUNCATE TABLE data_testing"
    )
    db.commit()
    cursor.close()
    db.close()
    return render_template("data_testing.html")
```

10. Source Code testresult.py

```
from flask import Blueprint, render_template, request, jsonify,
current_app
from wordcloud import WordCloud, STOPWORDS
import mysql.connector
from db_config import connect_db
import os

testresult_bp = Blueprint("test_result", __name__)


@testresult_bp.route("/hal_test_result",        methods=["POST",
"GET"])
def hal_test_result():
    if request.method == "POST":
        return testresult()

    return render_template("test_result.html")


@testresult_bp.route("/testresult", methods=["POST", "GET"])
def testresult():
    db = connect_db(current_app)
    cursor = db.cursor()

    try:
        # Fetch model and testing data
        cursor.execute(
```

```
            "SELECT        model_name,        training_accuracy,
training_accuracy_percent,  precision_score,  recall,  f1_score,
support FROM data_training_hasil"
        )
        data_training = cursor.fetchall()

        cursor.execute(
            """
            SELECT         model_name,         testing_accuracy,
testing_accuracy_percent,  precision_score,  recall,  f1_score,
support
            FROM data_testing_hasil
        """
        )
        data_testing = cursor.fetchall()

        # Ambil data testing untuk prediksi
        cursor.execute("SELECT       label,        text_stemmed,
text_extraction FROM data_testing")
        rows = cursor.fetchall()

        if not rows:
            return  jsonify({"error":  "No  data  available  for
testing."}), 400

        labels_list,  text_stemmed_list,  text_extraction_list  =
zip(*rows)

        # Hitung jumlah kalimat positif, negatif, dan netral
        positive_count = labels_list.count("Positif")
        negative_count = labels_list.count("Negatif")
        neutral_count = labels_list.count("Netral")

        summary_data = [
            {"Label": "Positif", "Count": positive_count},
            {"Label": "Negatif", "Count": negative_count},
            {"Label": "Netral", "Count": neutral_count},
        ]

        # Mengonversi text_extraction menjadi array
        X = []
        for extraction in text_extraction_list:
            try:
                extraction_values                                =
extraction.strip().split(",")
                X.append([float(x) for x in extraction_values])
            except ValueError as e:
                return    jsonify({"error":    f"Error    parsing
text_extraction: {e}"}), 400

        # Pisahkan teks berdasarkan label (positif dan negatif)
        positive_texts = [
            text_stemmed_list[i]
            for i in range(len(labels_list))
```

```
            if labels_list[i] == "Positif"
        ]
        negative_texts = [
            text_stemmed_list[i]
            for i in range(len(labels_list))
            if labels_list[i] == "Negatif"
        ]

        # Gabungkan teks menjadi satu string
        positive_text    =    "    ".join(positive_texts)    if
positive_texts else ""
        negative_text    =    "    ".join(negative_texts)    if
negative_texts else ""

        # Jika teks positif atau negatif kosong, kirim error
        if not positive_text and not negative_text:
            return (
                jsonify(
                    {
                        "error": "Tidak ada teks positif atau
negatif yang tersedia untuk membuat WordCloud."
                    }
                ),
                400,
            )

        # Stopwords untuk menghapus kata umum
        stopwords = STOPWORDS

        # Tentukan path untuk menyimpan gambar WordCloud
        positive_wc_path = os.path.join(
            current_app.static_folder,                "images",
"positive_wordcloud.png"
        )
        negative_wc_path = os.path.join(
            current_app.static_folder,                "images",
"negative_wordcloud.png"
        )

        # Membuat WordCloud untuk teks positif
        wc_positive = WordCloud(
            background_color="white",
            stopwords=stopwords,
            height=600,
            width=800,
            max_words=100,
            colormap="viridis",
        ).generate(positive_text)

        # Simpan WordCloud positif ke file
        wc_positive.to_file(positive_wc_path)

        # Membuat WordCloud untuk teks negatif
        wc_negative = WordCloud(
```

```
            background_color="white",
            stopwords=stopwords,
            height=600,
            width=800,
            max_words=100,
            colormap="plasma",
        ).generate(negative_text)

        # Simpan WordCloud negatif ke file
        wc_negative.to_file(negative_wc_path)

        #  Render  hasil  di  halaman  test_result.html  dengan
WordCloud dan gambar confusion matrix
        return render_template(
            "test_result.html",
            data_training=data_training,
            data_testing=data_testing,
            summary_data=summary_data,
            confusion_matrix_img="images/confusion_matrix.png",
# Kirim path gambar confusion matrix ke template

positive_wordcloud_img="images/positive_wordcloud.png",  # Kirim
image WordCloud positif ke template

negative_wordcloud_img="images/negative_wordcloud.png",  # Kirim
image WordCloud negatif ke template
        )

    except mysql.connector.Error as err:
        return jsonify({"error": f"Database error: {err}"}), 500

    except Exception as e:
        return jsonify({"error": str(e)}), 500

    finally:
        cursor.close()
        db.close()
```