```
from flask import Flask, render_template_string, request, redirect,
url_for, flash, session, send_file
from werkzeug.security import generate_password_hash,
check_password_hash
import sqlite3
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from io import BytesIO
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

# Iniciar la aplicación Flask
app = Flask(__name__)
app.secret_key = 'supersecretkey'  # Clave secreta para manejar
sesiones

# Cargar y limpiar los datos
data = pd.read_csv('C:/Users/NACHO-PC/Desktop/Proyecto/archive/
diabetes.csv')
data_cleaned = data.drop(['SkinThickness'], axis=1)  # Eliminar la
columna SkinThickness ya que no será utilizada

# Rellenar los valores 0 en la columna 'Insulin' con la mediana
median_insulin = data_cleaned['Insulin'].replace(0, pd.NA).median()
data_cleaned['Insulin'] = data_cleaned['Insulin'].replace(0,
median_insulin)

# Separar las características (X) y el objetivo (y)
X = data_cleaned.drop(['Outcome', 'DiabetesPedigreeFunction',
'Pregnancies'], axis=1)  # Eliminar columnas irrelevantes
y = data_cleaned['Outcome']

# Escalar los datos para que todas las características tengan el mismo
rango
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Usar SMOTE para manejar desbalance en los datos
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Dividir los datos en conjuntos de entrenamiento, validación y prueba
X_train, X_temp, y_train, y_temp = train_test_split(X_resampled,
y_resampled, test_size=0.30, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
```

```
test_size=0.70, random_state=42)

# Definir y realizar la búsqueda de hiperparámetros para Logistic
Regression
param_grid_lr = {'C': [0.01, 0.1, 1, 10], 'max_iter': [1000, 2000,
3000]}
grid_lr = GridSearchCV(LogisticRegression(), param_grid_lr, cv=5,
scoring='accuracy')
grid_lr.fit(X_train, y_train)
best_lr = grid_lr.best_estimator_  # Mejor modelo de Logistic
Regression

# Definir y realizar la búsqueda de hiperparámetros para Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, 30, None],
    'class_weight': ['balanced', 'balanced_subsample', None]
}
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42),
param_grid_rf, cv=5, scoring='accuracy')
grid_rf.fit(X_train, y_train)
best_rf = grid_rf.best_estimator_  # Mejor modelo de Random Forest

# Seleccionar el mejor modelo basado en la precisión en el conjunto de
validación
best_model = best_rf if accuracy_score(y_val, best_rf.predict(X_val))
> accuracy_score(y_val, best_lr.predict(X_val)) else best_lr

# Función para obtener una conexión a la base de datos SQLite
def get_db_connection():
    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row  # Esto hace que la consulta
retorne un diccionario
    return conn

# Función para crear las tablas necesarias en la base de datos
def create_tables():
    conn = get_db_connection()

    # Eliminar la tabla de predicciones si ya existe
    conn.execute('DROP TABLE IF EXISTS predictions')

    # Crear la tabla de usuarios con la columna 'keyword' para
recuperación de contraseña
    conn.execute('''CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        first_name TEXT NOT NULL,
        last_name TEXT NOT NULL,
        email TEXT NOT NULL UNIQUE,
        password TEXT NOT NULL,
```

```
        keyword TEXT NOT NULL  -- Nueva columna para la palabra clave
    )''')

    # Crear la tabla de predicciones para guardar las predicciones de
los usuarios
    conn.execute('''CREATE TABLE IF NOT EXISTS predictions (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        glucose INTEGER,
        blood_pressure INTEGER,
        insulin INTEGER,
        bmi REAL,
        age INTEGER,
        prediction TEXT,
        timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (user_id) REFERENCES users(id)
    )''')

    conn.commit()  # Confirmar los cambios en la base de datos
    conn.close()  # Cerrar la conexión

# Llamar a la función para crear las tablas al iniciar la aplicación
create_tables()

# Ruta de inicio, redirige al formulario de login
@app.route('/')
def index():
    return redirect(url_for('login'))

# Ruta para registrar un nuevo usuario
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        first_name = request.form['first_name']
        last_name = request.form['last_name']
        email = request.form['email']
        password = generate_password_hash(request.form['password'])  #
Encriptar la contraseña
        keyword = request.form['keyword']  # Guardar la palabra clave

        # Verificar si el correo ya está registrado
        conn = get_db_connection()
        existing_user = conn.execute('SELECT * FROM users WHERE email
= ?', (email,)).fetchone()

        if existing_user:
            flash('Este correo electrónico ya ha sido registrado.')
        else:
            # Insertar el nuevo usuario en la base de datos
            conn.execute('INSERT INTO users (first_name, last_name,
```

```
email, password, keyword) VALUES (?, ?, ?, ?, ?)',
                        (first_name, last_name, email, password,
keyword))
            conn.commit()
            flash('Registro exitoso. Ahora puedes iniciar sesión.')
            return redirect(url_for('login'))

        conn.close()  # Cerrar la conexión

    # Mostrar el formulario de registro
    return render_template_string('''
    <style>
        /* Estilos para el formulario */
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
        }
        .container {
            background-color: white;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            width: 300px;
        }
        h2 {
            text-align: center;
        }
        input {
            width: 100%;
            padding: 10px;
            margin: 5px 0;
            border: 1px solid #ddd;
            border-radius: 5px;
        }
        button {
            width: 100%;
            padding: 10px;
            background-color: #5cb85c;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
        button:hover {
            background-color: #4cae4c;
        }
```

```
        a {
            display: block;
            text-align: center;
            margin-top: 10px;
            color: #007bff;
            text-decoration: none;
        }
    </style>
    <div class="container">
        <h2>Registro</h2>
        {% with messages = get_flashed_messages() %}
          {% if messages %}
            <ul>
            {% for message in messages %}
              <li>{{ message }}</li>
            {% endfor %}
            </ul>
          {% endif %}
        {% endwith %}
        <form method="POST">
            <input type="text" name="first_name" placeholder="Nombre"
required>
            <input type="text" name="last_name" placeholder="Apellido"
required>
            <input type="email" name="email" placeholder="Email"
required>
            <input type="password" name="password"
placeholder="Contraseña" required>
            <input type="text" name="keyword" placeholder="Palabra
Clave" required>  <!-- Campo de palabra clave -->
            <button type="submit">Registrar</button>
        </form>
        <a href="/login">Ya tengo cuenta</a>
    </div>
    ''')

# Ruta para iniciar sesión
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        # Verificar el correo y contraseña
        conn = get_db_connection()
        user = conn.execute('SELECT * FROM users WHERE email = ?',
(email,)).fetchone()
        conn.close()

        if user and check_password_hash(user['password'], password):
```

```
            session['user_id'] = user['id']  # Guardar el id del
usuario en la sesión
            return redirect(url_for('predict_form'))
        else:
            flash('Correo electrónico o contraseña incorrectos.')

    # Mostrar el formulario de inicio de sesión
    return render_template_string('''
    <style>
        /* Estilos para el formulario de inicio de sesión */
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
        }
        .container {
            background-color: white;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            width: 300px;
        }
        h2 {
            text-align: center;
        }
        input {
            width: 100%;
            padding: 10px;
            margin: 5px 0;
            border: 1px solid #ddd;
            border-radius: 5px;
        }
        button {
            width: 100%;
            padding: 10px;
            background-color: #5cb85c;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
        button:hover {
            background-color: #4cae4c;
        }
        a {
            display: block;
            text-align: center;
```

```
            margin-top: 10px;
            color: #007bff;
            text-decoration: none;
        }
    </style>
    <div class="container">
        <h2>Iniciar Sesión</h2>
        {% with messages = get_flashed_messages() %}
          {% if messages %}
            <ul>
            {% for message in messages %}
              <li>{{ message }}</li>
            {% endfor %}
            </ul>
          {% endif %}
        {% endwith %}
        <form method="POST">
            <input type="email" name="email" placeholder="Email"
required>
            <input type="password" name="password"
placeholder="Contraseña" required>
            <button type="submit">Iniciar Sesión</button>
        </form>
        <a href="/register">Crear una cuenta</a>
        <a href="/forgot-password">Olvidé mi contraseña</a>
    </div>
    ''')
```