Task 6: Real-Time Collaboration with Socket.IO

Objective: To enable two users to join a shared session where they can chat in real-time and see the results of data analyses performed by either user instantly.
Estimated Time: 5-7 hours

Step 1: Create the Collaboration Session Model

File to Modify: app/models.py
Action:
We need a way to track who is collaborating with whom. We'll create a new table in the database for this.

1. Create a new class CollaborationSession that inherits from db.Model.
2. Give it the following columns:
   - id: An integer, primary key.
   - session_uuid: A unique string that will be used in the URL. Using a UUID (Universally Unique Identifier) is more secure than using the simple integer id.
   - user1_id: An integer, a foreign key linking to the users.id table.
   - user2_id: Another integer, also a foreign key, which can be nullable (it will be empty until the second user joins).
3. After defining the model, generate and apply the database migration:
   ```
   flask db migrate -m "Add collaboration session model"
   flask db upgrade
   ```

Why: This model creates a persistent record of the collaborative sessions. Using a session_uuid for the public-facing URL prevents users from guessing session numbers and trying to access sessions that aren't theirs.

Step 2: Implement Session Management Routes

File to Modify: app/main/routes.py
Action:
You need routes for users to create, join, and view a session.

1. **Create a /create_session route.** This route should:
   - Be protected with @login_required.
   - Create a new CollaborationSession object.
   - Generate a unique UUID for the session_uuid field. (You can use Python's built-in uuid module).
   - Set user1_id to the current_user.id.
   - Add the new session to the database.
   - Redirect the user to the main session view, e.g., /session/<session_uuid>.
2. **Create the /session/<session_uuid> route.** This will be the main workspace. It should:
   - Be protected with @login_required.
   - Find the session in the database using the session_uuid.

  - Check if the current_user is either user1 or user2. If not, deny access (show a 403 Forbidden error).
  - If the current_user is the second person to arrive and user2_id is null, set user2_id to current_user.id and save the change.
  - Render a new template, session.html, passing the session object to it. This template will contain your dashboard (table, chart) and the new chat interface.

Why: These routes control the lifecycle of a collaboration. They ensure only authorized users can enter a session and handle the logic of a second user joining an existing session.

Step 3: Build the Backend Socket.IO Event Handlers

New File: app/main/events.py
Action:
This file will handle all real-time communication.

1. Import socketio from app.extensions and join_room, leave_room from flask_socketio.
2. Create a handler for the 'join' event. This function will be triggered by the client-side JavaScript. It should accept the session_uuid (which we'll call the "room" name) and use join_room(room_name) to add the user to a specific communication channel.
3. Create a handler for the 'send_message' event. This function will receive a message from a client. It should then broadcast that message to all clients in the same room using socketio.emit('receive_message', message_data, room=room_name).
4. **Modify your analysis logic.** When an analysis is performed (e.g., in your dashboard route), after you get the results, you will now also emit a Socket.IO event to the room. For example: socketio.emit('analysis_update', {'result': '...'}, room=session_uuid).

Why: Socket.IO rooms are the key to private communication. They ensure that messages and data from one session don't get accidentally sent to another. The event handlers are the "listeners" on the server that react to actions taken by users in their browsers.
Documentation:

- Flask-SocketIO Rooms

Step 4: Design the Frontend UI and Client-Side JavaScript

Files to Modify: app/templates/main/session.html, app/static/js/socket_handler.js
Action:

1. **Create session.html:**
   - This template will be a combination of your dashboard.html and a new chat interface.
   - Design a chat box UI: a message display area (<div>), a text input (<input type="text">), and a send button (<button>).
   - Include the Socket.IO client library and your socket_handler.js file in a <script> tag.
2. **Create socket_handler.js:**
   - Initialize the connection: const socket = io();.
   - When the page loads, emit the 'join' event to the server, sending the session_uuid so the server knows which room to put you in.
   - Add a click listener to your chat send button. When clicked, it should get the text

    from the input field and emit the 'send_message' event to the server with the message data.
    - Create a listener for the 'receive_message' event from the server (socket.on('receive_message', ...)). When a message comes in, this function should create a new element and append it to your chat display area.
    - Create a listener for the 'analysis_update' event. For now, it could just alert() the user that a new analysis has been run. Later, you could make it update the chart or table dynamically.

Why: The client-side JavaScript is the other half of the real-time system. It sends events to the server when the user acts (joins, sends a message) and listens for events from the server to update the UI without needing to reload the entire page.

Congratulations! Once you complete this final task, you will have a fully functional, multi-user, real-time data analysis application. This is a complex and impressive project that demonstrates a wide range of full-stack development skills.