PROGRAMMING FOR BUSINESS PROJECT REPORT

Muhammad Ali ,Musab Sarmad

23L-5565,23L-5526

Project Name: Nikkah Swipe

Introduction: Nikkah Swipe is a modern matchmaking application designed to facilitate the process of finding compatible partners for individuals interested in marriage or long-term relationships within specific cultural or religious communities. The app provides a user-friendly interface for users to create profiles, browse potential matches, and communicate with each other.

Purpose: The primary purpose of Nikkah Swipe is to streamline the matchmaking process by leveraging modern technology while respecting cultural values and traditions. It aims to provide a convenient platform for individuals seeking meaningful connections within their community or culture.

Significance:

- Cultural Sensitivity: Nikkah Swipe acknowledges and respects cultural or religious preferences in partner selection, catering to users within specific communities.
- Modern Alternative: It offers a modern and convenient alternative to traditional matchmaking methods, incorporating features like profile swiping and messaging.
- Enhanced User Experience: By providing user-friendly interfaces and intuitive features, Nikkah Swipe enhances the overall experience of finding compatible partners.
- Community Building: The app fosters community building by connecting individuals with shared cultural backgrounds or values, potentially leading to long-lasting relationships and marriages.

Imports

1. ttkbootstrap:

- **import ttkbootstrap as ttk**: This import statement brings in the **ttkbootstrap** library, which is used to enhance the visual appearance of Tkinter widgets. It provides themes and styles to customize the look and feel of the GUI elements.

2. Standard Library Imports:

- **import random**: The **random** module is used for generating random numbers and making random selections, which might be utilized in various parts of the application, such as matchmaking or shuffling data.
- **import sys**: The **sys** module provides access to some variables used or maintained by the Python interpreter and functions that interact strongly with the interpreter. It may be used for system-related functionalities, such as exiting the program.

- **import os**: The **os** module provides functions for interacting with the operating system, such as file operations, directory manipulations, etc. It is used for tasks like listing directory contents and joining file paths.
- **import csv**: The **csv** module is used for reading and writing CSV (Comma Separated Values) files, which are commonly used for storing tabular data. It facilitates operations related to user data storage and retrieval.
- **import tkinter as tk**: The **tkinter** module provides the Tk GUI toolkit for Python. It allows the creation of graphical user interfaces and interaction with the user through various widgets like buttons, labels, entry fields, etc.

3. External Libraries:

- **import pyinputplus as pyinp**: The **pyinputplus** library provides input validation and various other features for handling user inputs. It ensures that the user inputs are valid and appropriate for the application's requirements.

4. Additional Libraries:

- **from PIL import Image, ImageTk**: The **PIL** (Python Imaging Library) module is used for image processing tasks. Specifically, **ImageTk** is used to convert images from PIL format to Tkinter format for display within Tkinter GUIs.

Function Used

Functions in the provided code serve as reusable blocks of code that perform specific tasks. They encapsulate logic, promote code reusability, and improve code organization.

1. Gui Class Constructor (__init__):

- Purpose: Initializes instance variables and sets up the initial state of the GUI.
- Tasks:
    - Sets default values for font, background colour, and other GUI-related attributes.
    - Initializes flags, counters, and lists used for managing program state and data.
    - Calls the **startingWindow** method to initiate the GUI.

2. User Interface Functions:

- **Login_gui** and **Signup_gui**: Display login and signup interfaces respectively.
- **startingWindow**: Initializes the starting window of the application with login and signup options.
- **Window**: Creates a new window with specified attributes using Tkinter.
- **Button**, **d_Button**, **s_Button**: Create different types of buttons (e.g., simple, positioned, styled) for user interaction.

- **create_Label**, **com_Label**, **user_Label**: Generate labels for displaying text information on the GUI.

3. User Authentication and Profile Management Functions:

- **login**, **login_auth**: Handle user login authentication by checking credentials against stored data.
- **emailcheck_ui**, **emailcheck**: Verify user email during signup to avoid duplication.
- **age_verify**: Prompt user to enter age and check if they are above 18 years old for registration.
- **user_new_form**: Collect user information during signup and store it in CSV files.
- **profile_makeing_gui**, **profile_makeing**: Gather additional profile details like city, gender, hobbies, etc., during profile creation.
- **hobbies_gui**, **hobbies**: Capture user hobbies as part of the profile creation process.

4. Matchmaking and Messaging Functions:

- **swipe**, **ime_counter**, **rightswipe**: Implement the swiping feature for browsing potential matches and selecting preferences.
- **mess**, **send**: Enable messaging functionality between matched users.
- **random_user**, **u**: Handle the selection of random users for matchmaking purposes.

5. Miscellaneous Functions:

- **send_text**: Handle sending and displaying text messages in the messaging window.

Variables used in code.

1. Instance Variables:

- **font:** Stores the font configuration for GUI elements, ensuring consistency in text appearance.
- **Bg**: Represents the background colour of GUI windows and elements, maintaining a cohesive visual theme.
- **entry_bg**: Defines the background colour for entry fields, enhancing readability and user experience.
- **flag**: Boolean flag variable used for controlling program flow and executing conditional logic.
- **_h**: List variable storing user hobbies during profile creation.
- **files:** List of filenames in the "wall" directory, containing images for display.

- **done, counter, d1, ui9, win1**: Integer variables maintaining program state and controlling GUI behaviour.
- **showuser_n, showuser, selected_user, finaluser:** Lists storing user data and selections during matchmaking and messaging processes.
- **r, t1**: Temporary storage lists for handling intermediate data in various functions.

2. GUI Element Variables:

- **window:** Tkinter window object representing the main application window or specific interface windows.
- **label, img_label, name_label, detail_label, detl_label:** Tkinter label objects for displaying text and images on the GUI.
- **entryemail_l, entrypassword_l, entryname_Name, entryname_phone, entryname_pass, entry_age, entryname_mail, entry_City, entry_gender, entry_hobby_1, entry_hobby_2, entry_hobby_3, userentry:** Tkinter entry objects for user input.
- **b, b1, utton:** Tkinter button objects for user interaction**.**

3. User Data Variables:

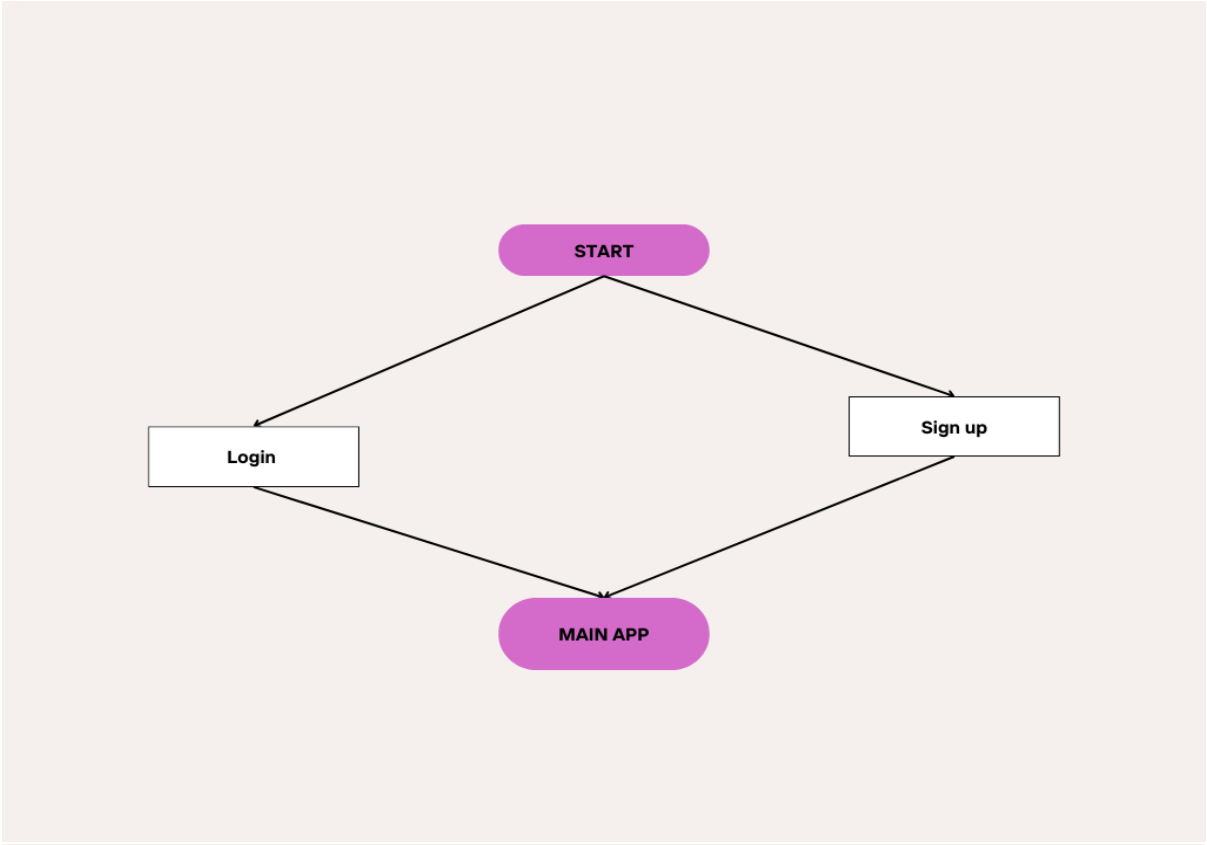
- **user_data:** List storing user information during signup, including name, phone, password, age, and email**.**
- **mail_user_input, user_password_input:** Strings holding user input for login authentication.
- **mail:** String containing user email for signup and login processes.
- **gender, city, hobby_1, hobby_2, hobby_3:** Strings storing additional user profile details.
- **finauser:** String representing a randomly selected user for matchmaking.

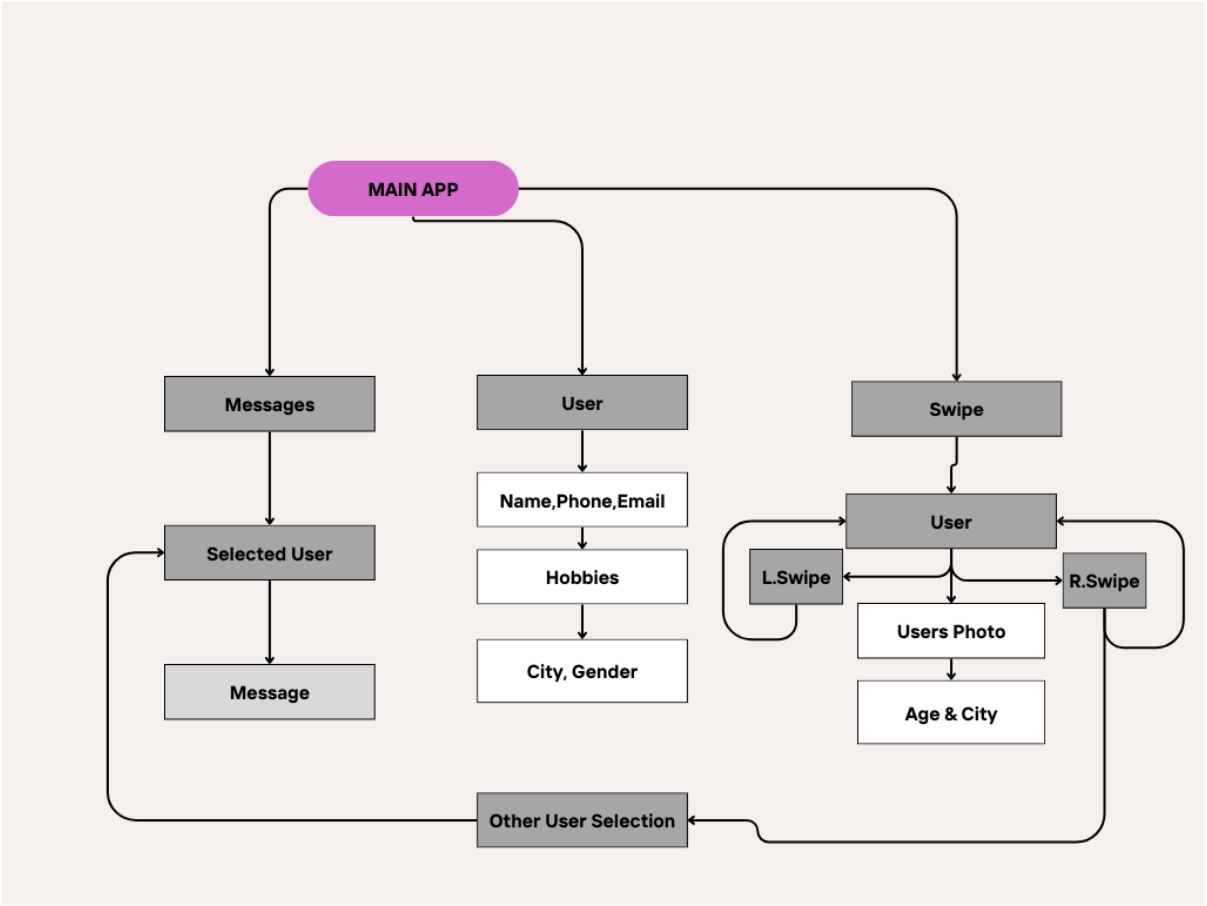
4. File Handling Variables:

- **csv_writer, csv_reader**: CSV writer and reader objects for handling user data stored in CSV files.
- **csv_data:** List storing data read from CSV files for processing.
- **file:** String representing the filename in the "wall" directory for displaying images.

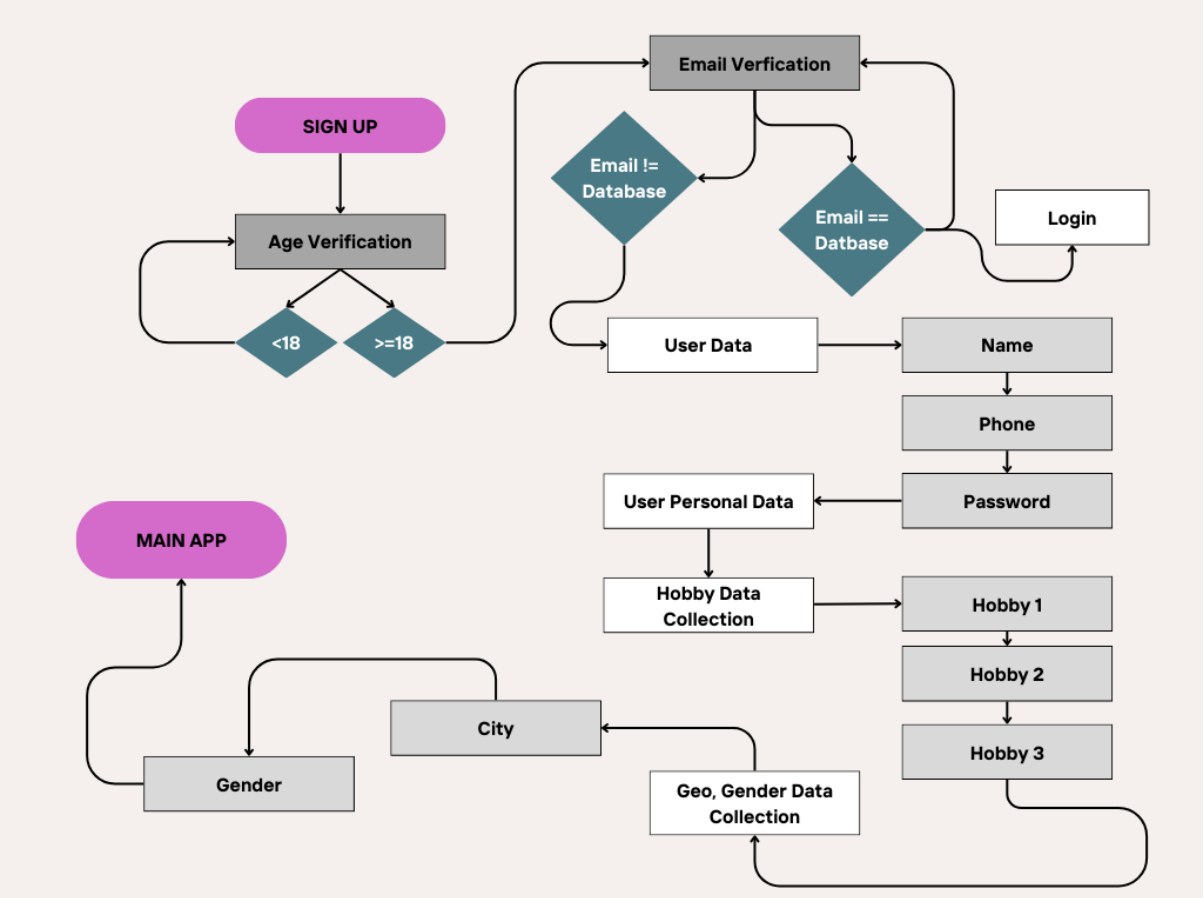
Muhammadali,Musab Saramd ,23l-5565,23l-5526
PFproject ,Nikkah Swipe app on python

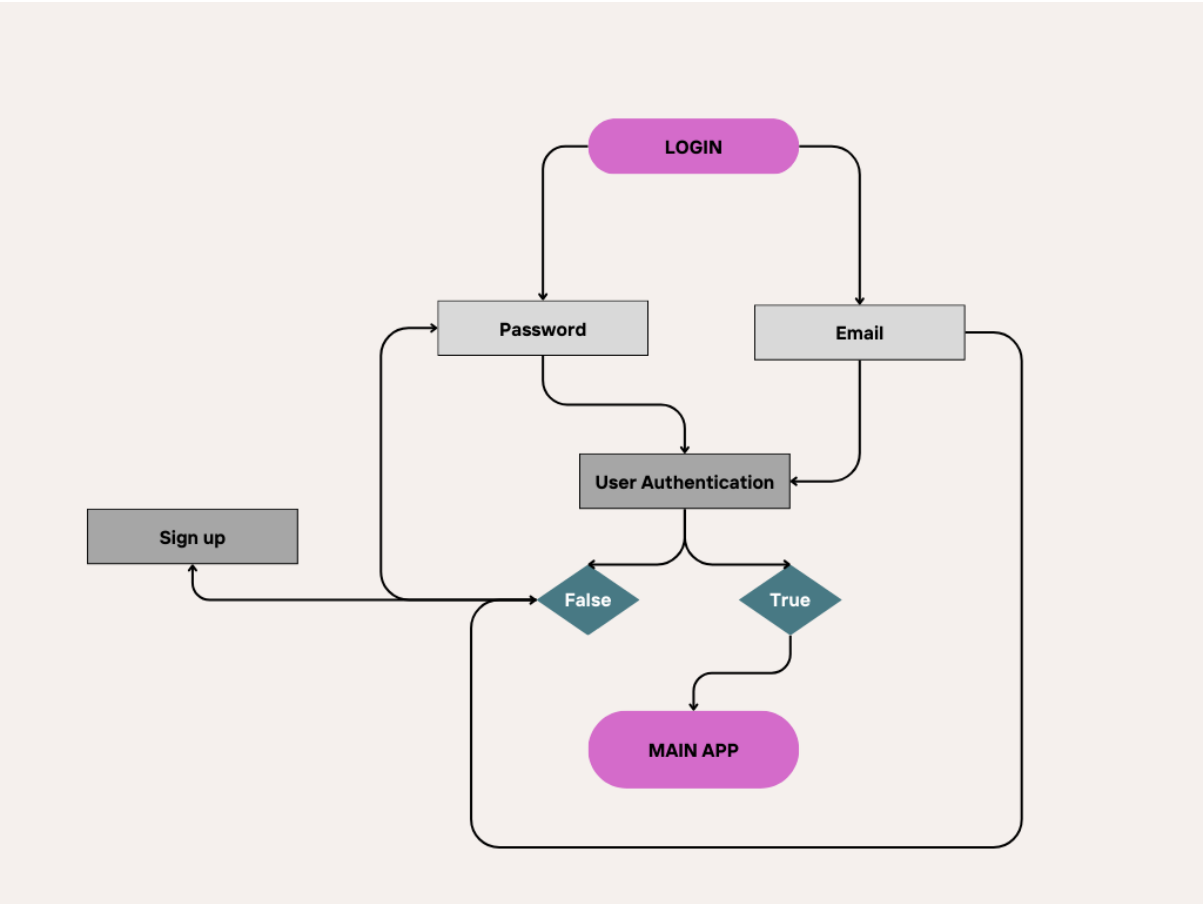
Flowcharts





Muhammadali,Musab Saramd ,23l-5565,23l-5526
PFproject ,Nikkah Swipe app on python





- Explain how your application handles data, including storage and processing.

1. User Data Storage:

- **CSV Files:** User data such as name, phone number, email, password, age, hobbies, gender, city, etc., are stored in CSV files (**Userdata.csv**, **Usersmail.csv**, **User_personal_data.csv**). Each CSV file represents a table with rows corresponding to individual users and columns representing different attributes.

2. Data Processing:

- **Input Handling:** The application utilizes input validation libraries like **pyinputplus** to ensure that user inputs are appropriate and meet specified criteria. This helps maintain data integrity and prevents errors during data entry.
- **Signup and Profile Creation:** When a user signs up, their information is collected through GUI input fields and stored in the appropriate CSV files using CSV writer objects. This includes validating email uniqueness, checking age eligibility, and storing profile details such as hobbies, gender, and city.
- **Login Authentication:** During the login process, the application reads user credentials from CSV files using CSV reader objects. It then compares the provided email and password against the stored data to authenticate the user. If the credentials match, the user is granted access to the application.
- **Matchmaking:** The application implements a swiping feature for matchmaking, where users can browse through potential matches. User selections are stored in memory temporarily and later written to a CSV file (**Selected.csv**) for further processing.
- **Messaging:** Matched users can communicate via messaging. The application manages message data by storing it in memory (list **t1**) and displaying it in the messaging window.

3. Error Handling:

- The application incorporates error handling mechanisms using Tkinter's messagebox for displaying warnings and errors to the user. For example, it notifies users if they are underage during signup, if login credentials are incorrect, or if a user already exists during signup.
- Error handling ensures that users receive appropriate feedback and guidance in case of invalid inputs or unexpected situations, enhancing the user experience.

4. Data Retrieval and Display:

- User data stored in CSV files is retrieved during runtime for display in various GUI elements such as labels, entry fields, and message boxes. This allows users to view their profiles, potential matches, and messages within the application interface.

Provide details on the user interface and user interactions.

. Main Window:

- **Title:** "Nikkah App"
- **Dimensions:** 400x600 pixels
- **Background Color:** Pink (**#FFC0CB**)
- **Components:**
    - Application title label ("Nikkah App")
    - Buttons for login and signup options
    - Label indicating "Back Button under construction" (under development)

2. Login Interface:

- **Window Transition:** Upon selecting the "Login" button from the main window, the interface transitions to the login screen.
- **Components:**
    - Entry fields for email and password
    - Login button
- **Functionality:**
    - Users enter their email and password to log in.
    - Authentication is performed by checking the entered credentials against stored user data.
    - If the credentials are correct, the main window is displayed. Otherwise, appropriate error messages are shown.

3. Signup Interface:

- **Window Transition:** Upon selecting the "Signup" button from the main window, the interface transitions to the signup screen.
- **Components:**
    - Entry fields for name, phone number, password, email, and age
    - Signup button
- **Functionality:**
    - Users enter their information to create a new account.
    - Input fields include validation for email uniqueness and age eligibility.

- Upon successful signup, users are prompted to enter additional profile details.

4. Additional Profile Details Interface:

- **Window Transition:** After successful signup, the interface transitions to collect additional profile details.
- **Components:**
  - Entry fields for gender, city, and hobbies (up to three)
  - Submit button
- **Functionality:**
  - Users provide additional details such as gender, city, and hobbies to complete their profile.
  - Input fields allow users to input their preferences and interests.

5. Matchmaking Interface:

- **Components:**
  - Swiping mechanism for browsing potential matches
  - Display of user images and profile details
  - Buttons for left and right swipes
- **Functionality:**
  - Users can browse through profiles by swiping left or right.
  - Selections are stored for further processing.

6. Messaging Interface:

- **Components:**
  - Display of matched users for messaging
  - Input field for sending messages
  - Button for sending messages
- **Functionality:**
  - Matched users can communicate via text messages.
  - Users can send and receive messages within the application.

7. Error Handling:

- Error messages are displayed using Tkinter's messagebox for scenarios such as incorrect login credentials, underage signup, duplicate emails, etc.

- Error handling ensures users receive appropriate feedback and guidance during interactions.

Discuss how you have implemented key programming concepts.

1. Object-Oriented Programming (OOP):

- **Class Structure:** The application is organized into a single class **GUi**, encapsulating the entire functionality of the matchmaking system. This promotes code organization, modularity, and reusability.
- **Encapsulation:** Data and functionality are encapsulated within class methods, ensuring data integrity and preventing unauthorized access.
- **Inheritance and Polymorphism:** While not explicitly shown in the provided code, OOP concepts like inheritance and polymorphism can be applied to extend functionality in future iterations, such as creating subclasses for different user types or interface variations.

2. File Handling:

- **CSV File Operations:** The application utilizes CSV files (**Userdata.csv**, **Usersmail.csv**, **User_personal_data.csv**, **Selected.csv**) for storing and managing user data. It employs CSV reader and writer objects to read from and write to these files, enabling persistent storage and retrieval of user information.

3. User Input Validation:

- **Input Handling:** The application incorporates input validation using the **pyinputplus** library to ensure that user inputs are of the correct format and meet specified criteria. This prevents invalid data entry and enhances data integrity.
- **Error Handling:** Error handling mechanisms are implemented using Tkinter's **messagebox** module to display appropriate warnings and error messages to users in case of invalid inputs or unexpected situations.

4. GUI Development:

- **Tkinter Library:** The user interface of the application is developed using the Tkinter library, which provides a set of tools for creating graphical user interfaces in Python.
- **GUI Components:** The code includes various GUI components such as windows, labels, buttons, entry fields, and message boxes to create an interactive and visually appealing interface for users.
- **Event Handling:** The application handles user interactions such as button clicks, text input, and window transitions through event-driven programming, where callback functions are triggered in response to user actions.

5. Control Structures:

- **Conditional Statements:** Conditional statements (**if**, **elif**, **else**) are used throughout the code to implement branching logic and make decisions based on user inputs or program state.
- **Loops:** Looping constructs (**for**, **while**) are employed for iterating over data structures, such as CSV rows or image files, and performing repetitive tasks like displaying multiple profiles or processing user selections.

6. Error Handling:

- **Exception Handling:** The application incorporates exception handling mechanisms to catch and handle errors that may occur during runtime, ensuring graceful error recovery and preventing program crashes.
- **User Feedback:** Error messages are displayed using Tkinter's messagebox module to provide clear and informative feedback to users in case of errors or invalid inputs

How Matchmaking works?

1. **Swiping Interface**:
   - Users are presented with profiles of other users one at a time.
   - Each profile typically includes details like name, age, city, and possibly hobbies or other personal information.
2. **Swiping Actions**:
   - Users can swipe left to indicate disinterest or swipe right to express interest in a particular profile.
   - Swiping actions are typically handled by methods like **swipe**, **rightswipe**, and **ime_counter**.
3. **Profile Selection**:
   - When a user swipes right on a profile, indicating interest, the selected profile is added to a list of potential matches.
   - This list may be stored temporarily or written to a CSV file like **Selected.csv**.
4. **Matching Algorithm**:
   - The matching algorithm in the provided code is relatively simple.
   - It randomly selects profiles from the list of potential matches, simulating the matching process.
   - The **random_user** method randomly selects users from the list of swiped profiles and writes them to a CSV file for further processing.
5. **Messaging**:
   - Once matches are made, users can access a messaging interface to communicate with their matches.
   - The **mess** method displays matched users and provides options for sending messages.
6. **Additional Features**:
   - The code may include additional features related to profile creation, user authentication, and profile viewing.

- These features contribute to the overall matchmaking experience and user engagement.

Project Summary: The project, named "Nikkah Swipe," is a GUI application designed to facilitate matchmaking within specific cultural or religious communities. It allows users to create profiles, browse potential matches, and communicate with each other through messaging. The application aims to provide a modern and convenient platform for individuals seeking meaningful connections and long-term relationships while respecting cultural values and traditions.

Outcomes:

1. **Basic Functionality:** The project demonstrates basic functionality such as user authentication, profile creation, swiping through profiles, and messaging.
2. **Cultural Sensitivity:** It acknowledges and respects cultural or religious preferences in partner selection, catering to users within specific communities.
3. **User Interface:** The application features a user-friendly interface with intuitive navigation and visually appealing design.
4. **Community Building:** By connecting individuals with shared cultural backgrounds or values, the app fosters community building and potentially leads to long-lasting relationships and marriages.

Potential Future Improvements:

1. **Advanced Matching Algorithms:** Implement more sophisticated algorithms for matching users based on compatibility metrics such as interests, values, and personality traits.
2. **Personalized Recommendations:** Provide personalized recommendations for potential matches based on user preferences and previous interactions.
3. **Enhanced Messaging Features:** Improve messaging functionalities with features like multimedia support, message filtering, and conversation management.
4. **User Feedback Integration:** Gather user feedback to identify areas for improvement and prioritize feature development based on user preferences and needs.
5. **Security Enhancements:** Implement robust security measures to protect user data and ensure privacy, such as encryption, secure authentication methods, and data access controls.
6. **Performance Optimization:** Optimize application performance to ensure smooth user experience, especially during peak usage times or when handling large amounts of data.

7. **Localization and Globalization:** Support multiple languages and cultural norms to make the application accessible and relevant to users worldwide.
8. **Community Engagement:** Foster community engagement through features like user forums, events, and support for community-driven initiatives.
9. **Accessibility:** Ensure accessibility for users with disabilities by adhering to accessibility standards and providing features like screen reader support and keyboard navigation.
10. **Continuous Maintenance and Support:** Regularly update and maintain the application to address bugs, security vulnerabilities, and evolving user needs, providing ongoing support and enhancements.