

# Assignment-5.2

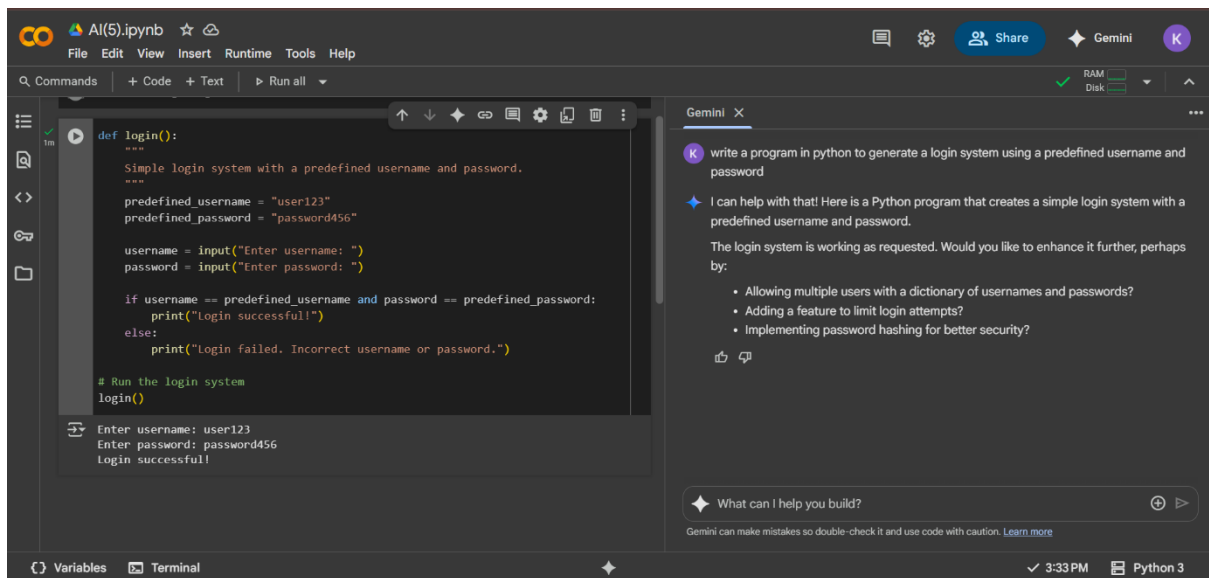
Name: N. Bhanuchander

Roll No: 2403A52085

Task 1:

write a program in python to generate a login system using a predefined username and password

code:



The screenshot shows a Jupyter Notebook environment with a dark theme. The main code cell contains a Python function `login()` that implements a simple login system. It defines predefined username and password, takes user input, and checks for a match. Below the code, the output shows the program being run with the predefined credentials, resulting in a successful login message. On the right side, a Gemini chat interface is open, displaying a conversation about the login program and suggestions for enhancements.

```
def login():  
    """  
    Simple login system with a predefined username and password.  
    """  
    predefined_username = "user123"  
    predefined_password = "password456"  
  
    username = input("Enter username: ")  
    password = input("Enter password: ")  
  
    if username == predefined_username and password == predefined_password:  
        print("Login successful!")  
    else:  
        print("Login failed. Incorrect username or password.")  
  
    # Run the login system  
    login()  
  
Enter username: user123  
Enter password: password456  
Login successful!
```

Gemini X

K write a program in python to generate a login system using a predefined username and password

I can help with that! Here is a Python program that creates a simple login system with a predefined username and password.

The login system is working as requested. Would you like to enhance it further, perhaps by:

- Allowing multiple users with a dictionary of usernames and passwords?
- Adding a feature to limit login attempts?
- Implementing password hashing for better security?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal 3:33 PM Python 3

Explanation:

Gemini X

K explanation

✦ No problem! This code snippet creates a simple login system in Python:

- `def login():` : This defines a function named `login` which contains the code for the login system.
- `predefined_username = "user123"` and `predefined_password = "password456"` : These lines set the username and password that the user needs to enter to successfully log in.
- `username = input("Enter username: ")` and `password = input("Enter password: ")` : These lines prompt the user to enter their username and password using the `input()` function and store them in the `username` and `password` variables respectively.
- `if username == predefined_username and password == predefined_password:` : This is a conditional statement that checks if the entered username and password match the predefined ones.
- `print("Login successful!")` : If the username and password match, this line prints a success message.
- `else:` : If the username and password do not match, the code inside the `else` block is executed.
- `print("Login failed. Incorrect username or password.")` : This line prints a failure message if the login credentials are incorrect.
- `login()` : This line calls the `login` function to start the login process.

👍 🗨

✦ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

## Task 2:

write a program in python to generate the loan approval

code and output:

Al(5).ipynb

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all

```
[7] # Define criteria for loan approval
credit_score_threshold = 650 # Minimum credit score required
income_loan_ratio_threshold = 0.3 # Maximum allowed ratio of desired loan amount to annual income
loan_amount_limit = 500000 # Maximum loan amount allowed regardless of other factors

print("Loan criteria defined:")
print(f"- Minimum credit score: {credit_score_threshold}")
print(f"- Maximum income-to-loan ratio: {income_loan_ratio_threshold}")
print(f"- Maximum loan amount: {loan_amount_limit}")
```

Loan criteria defined:  
- Minimum credit score: 650  
- Maximum income-to-loan ratio: 0.3  
- Maximum loan amount: 500000

```
[14] # Get user input
try:
    credit_score = int(input("Enter your credit score: "))
    annual_income = float(input("Enter your annual income: "))
    desired_loan_amount = float(input("Enter the desired loan amount: "))
except ValueError:
    print("Invalid input. Please enter numeric values for credit score, annual income, and desired loan amount.")
    credit_score, annual_income, desired_loan_amount = None, None, None # Set to None to indicate invalid input
```

Enter your credit score: 700  
Enter your annual income: 400000  
Enter the desired loan amount: 100000

Variables Terminal

4:00 PM Python 3

Gemini X

K write a program in python to generate the loan approval

✦ Let's define the loan criteria. We'll use credit score, annual income, and desired loan amount as the main factors.

Now that we have defined the criteria, let's get the user's input.

Great, we have the user's input. Now let's implement the logic to determine if the loan is approved based on the criteria we defined earlier.

The loan approval program is complete. The program checked the credit score, income-to-loan ratio, and loan amount against the defined criteria and determined the approval status.

Would you like to make any modifications or add more features, such as:

- Saving the loan application details to a file?

✦ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

```
AI(5).ipynb
File Edit View Insert Runtime Tools Help

[14] # Get user input
try:
    credit_score = int(input("Enter your credit score: "))
    annual_income = float(input("Enter your annual income: "))
    desired_loan_amount = float(input("Enter the desired loan amount: "))
except ValueError:
    print("Invalid input. Please enter numeric values for credit score, annual income, and desired loan amount.")
    credit_score, annual_income, desired_loan_amount = None, None, None # Set to None to indicate invalid input

Enter your credit score: 700
Enter your annual income: 400000
Enter the desired loan amount: 100000

# Implement approval logic
is_approved = False

if credit_score is not None and annual_income is not None and desired_loan_amount is not None:
    # Check credit score
    if credit_score >= credit_score_threshold:
        # Check income-to-loan ratio
        if desired_loan_amount <= (annual_income * income_loan_ratio_threshold):
            # Check loan amount limit
            if desired_loan_amount <= loan_amount_limit:
                is_approved = True
            else:
                print(f"Loan denied: Desired loan amount exceeds the maximum limit of {loan_amount_limit}."
                    )
        else:
            print(f"Loan denied: Income-to-loan ratio is too high. Your desired loan amount should be no more than {(annual_income * income_loan_ratio_threshold)}."
                )
    else:
        print(f"Loan denied: Credit score is below the minimum required of {credit_score_threshold}."
            )
else:
    print("Loan approval cannot be determined due to invalid input.")

# Display result
if is_approved:
    print("Loan approved!")
```

```
AI(5).ipynb
File Edit View Insert Runtime Tools Help

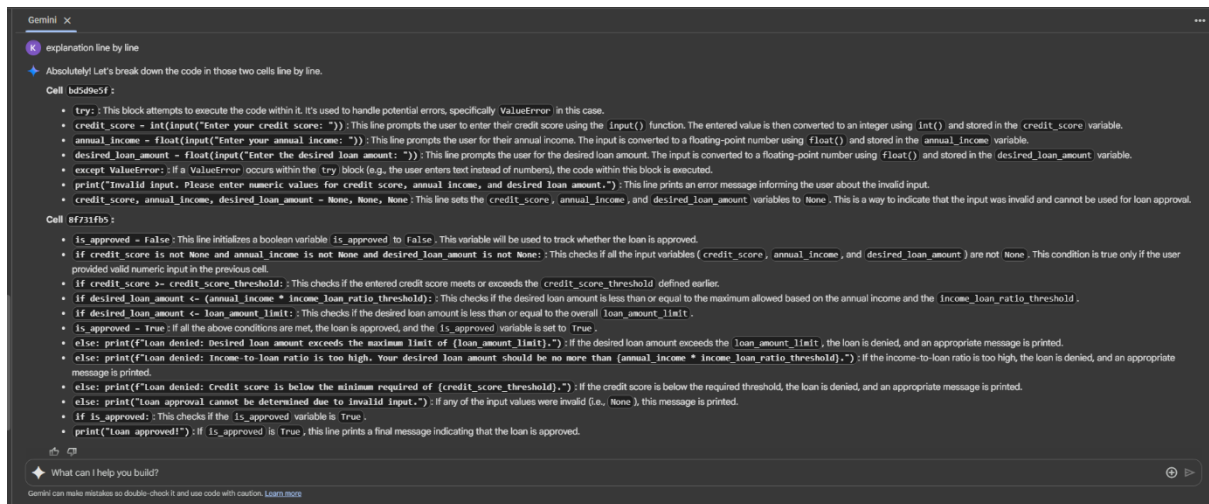
# Implement approval logic
is_approved = False

if credit_score is not None and annual_income is not None and desired_loan_amount is not None:
    # Check credit score
    if credit_score >= credit_score_threshold:
        # Check income-to-loan ratio
        if desired_loan_amount <= (annual_income * income_loan_ratio_threshold):
            # Check loan amount limit
            if desired_loan_amount <= loan_amount_limit:
                is_approved = True
            else:
                print(f"Loan denied: Desired loan amount exceeds the maximum limit of {loan_amount_limit}."
                    )
        else:
            print(f"Loan denied: Income-to-loan ratio is too high. Your desired loan amount should be no more than {(annual_income * income_loan_ratio_threshold)}."
                )
    else:
        print(f"Loan denied: Credit score is below the minimum required of {credit_score_threshold}."
            )
else:
    print("Loan approval cannot be determined due to invalid input.")

# Display result
if is_approved:
    print("Loan approved!")

Loan approved!
```

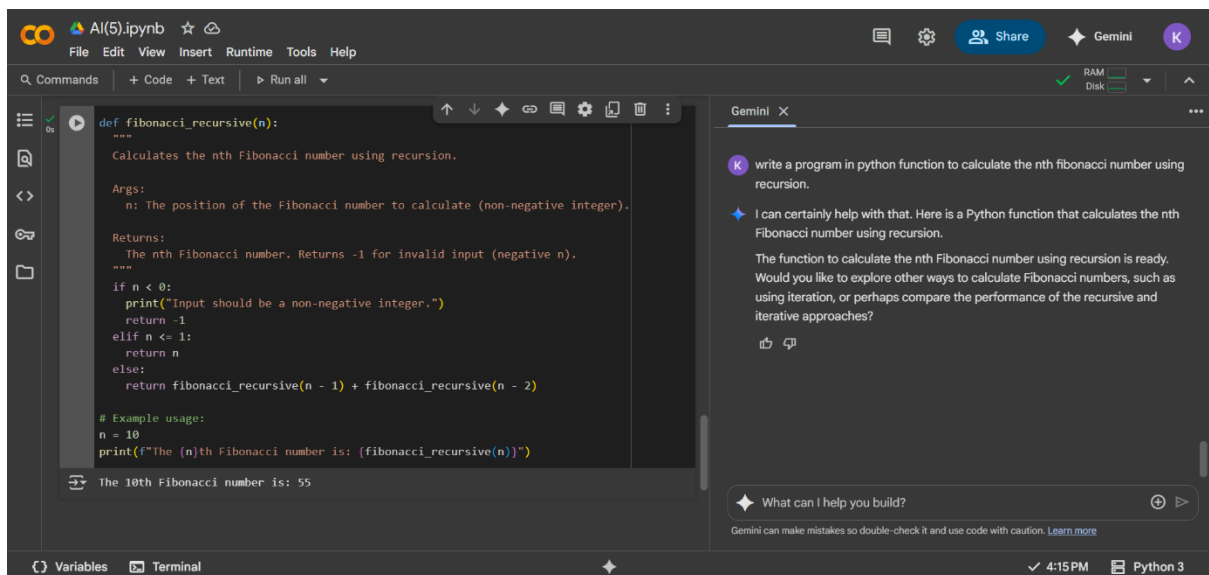
Explanation:



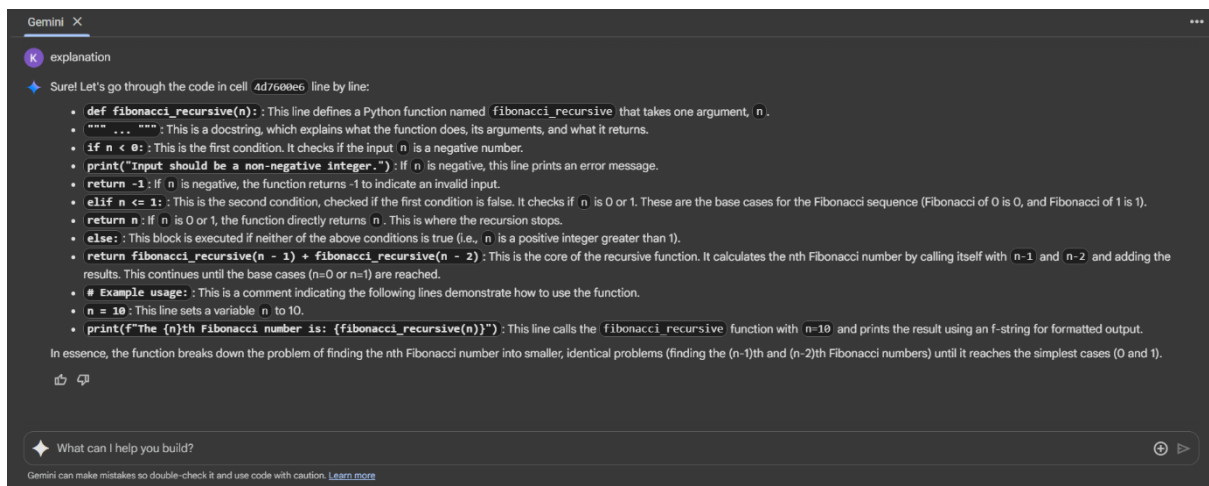
### Task 3:

write a program in python function to calculate the nth fibonacci number using recursion.

Code and output:



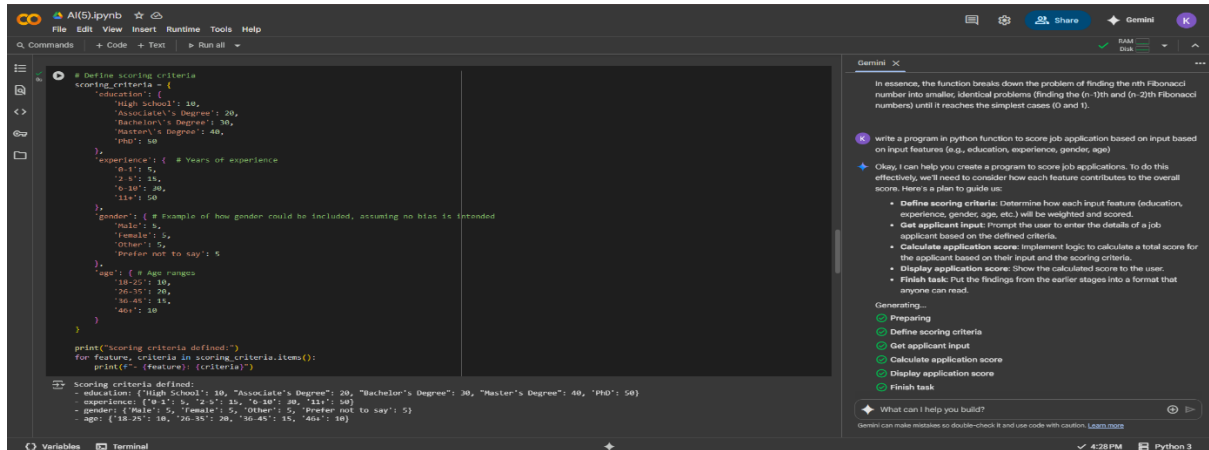
Explanation:



## Task 4:

write a program in python function to score job application based on input based on input features(e.g., education, experience, gender, age)

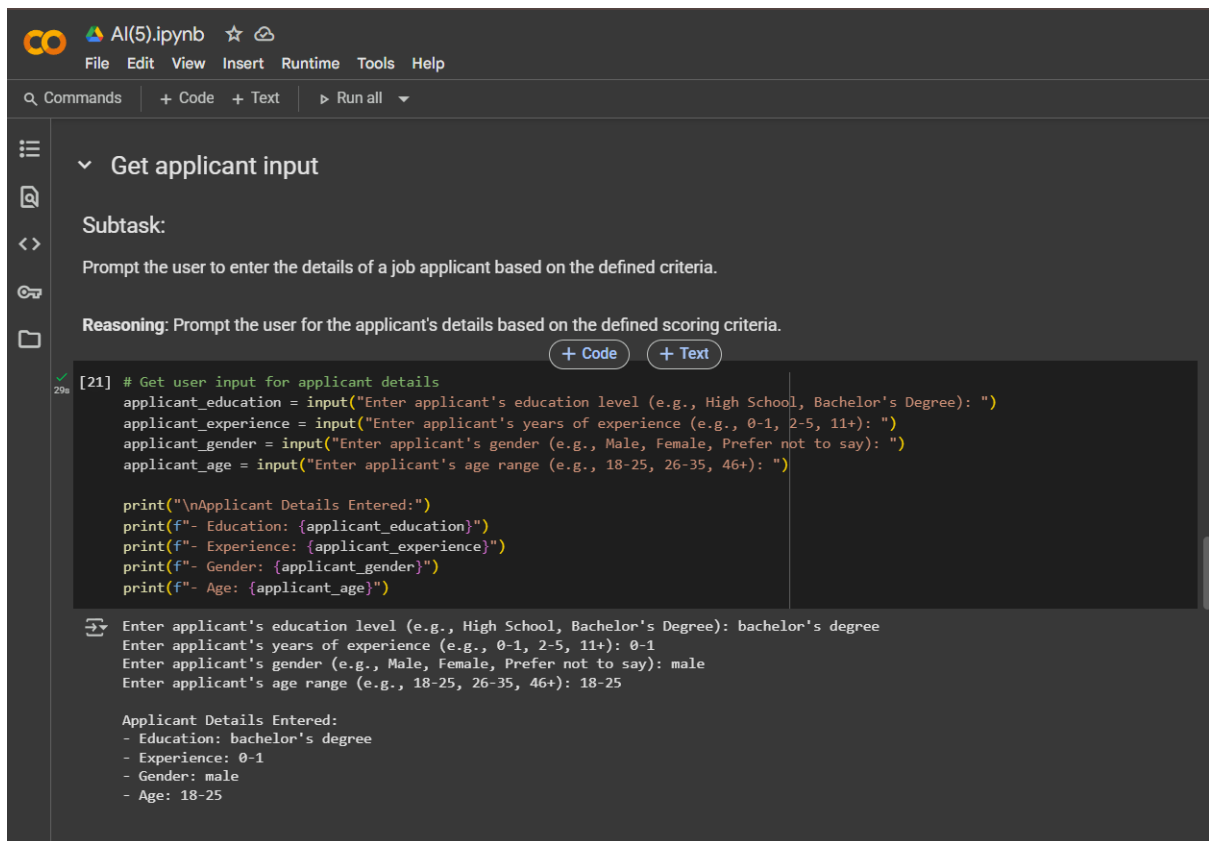
code and output:



```
# Define scoring criteria
scoring_criteria = {
    'education': {
        'High School': 10,
        'Associate's Degree': 20,
        'Bachelor's Degree': 30,
        'Master's Degree': 40,
        'PhD': 50
    },
    'experience': { # Years of experience
        '0-1': 5,
        '2-5': 15,
        '6-10': 20,
        '11+': 30
    },
    'gender': { # Example of how gender could be included, assuming no bias to intended
        'Male': 5,
        'Female': 5,
        'Other': 5,
        'Prefer not to say': 5
    },
    'age': { # Age ranges
        '18-25': 10,
        '26-35': 20,
        '36-45': 15,
        '46+': 10
    }
}

print("Scoring criteria defined:")
for feature, criteria in scoring_criteria.items():
    print(f"- {feature}: {criteria}")
```

Scoring criteria defined:  
- education: {'High School': 10, 'Associate's Degree': 20, 'Bachelor's Degree': 30, 'Master's Degree': 40, 'PhD': 50}  
- experience: {'0-1': 5, '2-5': 15, '6-10': 20, '11+': 30}  
- gender: {'Male': 5, 'Female': 5, 'Other': 5, 'Prefer not to say': 5}  
- age: {'18-25': 10, '26-35': 20, '36-45': 15, '46+': 10}



```
[21] # Get user input for applicant details
applicant_education = input("Enter applicant's education level (e.g., High School, Bachelor's Degree): ")
applicant_experience = input("Enter applicant's years of experience (e.g., 0-1, 2-5, 11+): ")
applicant_gender = input("Enter applicant's gender (e.g., Male, Female, Prefer not to say): ")
applicant_age = input("Enter applicant's age range (e.g., 18-25, 26-35, 46+): ")

print("\nApplicant Details Entered:")
print(f"- Education: {applicant_education}")
print(f"- Experience: {applicant_experience}")
print(f"- Gender: {applicant_gender}")
print(f"- Age: {applicant_age}")
```

Enter applicant's education level (e.g., High School, Bachelor's Degree): bachelor's degree  
Enter applicant's years of experience (e.g., 0-1, 2-5, 11+): 0-1  
Enter applicant's gender (e.g., Male, Female, Prefer not to say): male  
Enter applicant's age range (e.g., 18-25, 26-35, 46+): 18-25

Applicant Details Entered:  
- Education: bachelor's degree  
- Experience: 0-1  
- Gender: male  
- Age: 18-25

The screenshot displays a Jupyter Notebook environment with a dark theme. The top bar includes the Jupyter logo, the filename 'AI(5).ipynb', and standard menu items: File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all' buttons. The left sidebar contains icons for file management and navigation. The main area shows a code cell with the following Python code:

```
[22] # Implement logic to calculate total score
total_score = 0

# Get score for education
try:
    total_score += scoring_criteria['education'][applicant_education]
except KeyError:
    print(f"Warning: Invalid education level '{applicant_education}' not found in scoring criteria.")

# Get score for experience
try:
    total_score += scoring_criteria['experience'][applicant_experience]
except KeyError:
    print(f"Warning: Invalid experience level '{applicant_experience}' not found in scoring criteria.")

# Get score for gender
try:
    total_score += scoring_criteria['gender'][applicant_gender]
except KeyError:
    print(f"Warning: Invalid gender '{applicant_gender}' not found in scoring criteria.")

# Get score for age
try:
    total_score += scoring_criteria['age'][applicant_age]
except KeyError:
    print(f"Warning: Invalid age range '{applicant_age}' not found in scoring criteria.")

print(f"\nTotal applicant score: {total_score}")
```

Below the code cell, the output is displayed:

```
Warning: Invalid education level 'bachelor's degree' not found in scoring criteria.
Warning: Invalid gender 'male' not found in scoring criteria.

Total applicant score: 15
```

The next cell contains the following code:

```
# Show the calculated score to the user
print(f"\nThe applicant's total score is: {total_score}")
```

The output of this cell is:

```
The applicant's total score is: 15
```

Explanation:

Gemini X

K explanation line by line

Happy to explain! The code in cell `e3ce360b` sets up the rules for how job applications will be scored. Here's a line-by-line breakdown:

- `scoring_criteria = { ... }`: This line creates a dictionary named `scoring_criteria`. This dictionary will hold all the different categories we'll use to score an applicant (like education, experience, etc.) and the points associated with each option within those categories.
- `'education': { ... }`: This is one entry in the `scoring_criteria` dictionary. The key is `'education'`, and its value is another dictionary. This inner dictionary contains education levels as keys and the corresponding points an applicant gets for that level as values (e.g., `'High School': 10`, `'Bachelor's Degree': 30`).
- `'experience': { ... }`: Similar to education, this defines the scoring for experience levels based on years.
- `'gender': { ... }`: This section defines points for gender. It's set up here with equal points for all options as an example of how gender *could* be included, while aiming for no scoring bias based on gender.
- `'age': { ... }`: This defines points based on age ranges.
- `print("Scoring criteria defined:")`: This line prints a header to the output indicating that the scoring criteria are about to be displayed.
- `for feature, criteria in scoring_criteria.items():`: This loop iterates through each key-value pair in the `scoring_criteria` dictionary. In each iteration, `feature` will be the category name (like 'education', 'experience'), and `criteria` will be the dictionary of options and points for that category.
- `print(f"- {feature}: {criteria}")`: Inside the loop, this line prints the name of the feature and its corresponding scoring criteria dictionary in a formatted string.

In short, this cell is all about establishing the point system that will be used later to evaluate job applications based on different attributes.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Task 5:

`def greet_user(name,gender): if gender.lower()=="male": title="Mr." else: title="Mrs." return f"Hello, {title} {name}! Welcome."` add the gender-netural also

code and output:

AI(5).ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[28] def greet_user(name,gender=None):
    """
    Greet the user based on their name and gender (optional).

    Args:
        name: The name of the user (string).
        gender: The gender of the user (string, optional).
    """
    title = ""
    if gender:
        if gender.lower() == "male":
            title = "Mr."
        elif gender.lower() == "female":
            title = "Mrs."
        # Add a condition for gender-neutral or prefer not to say
        elif gender.lower() in ["non-binary", "prefer not to say", "gender-neutral"]:
            title = "" # Or a preferred gender-neutral title like "Mx." if appropriate
        else:
            title = "" # Default to no title for unrecognized gender
    else:
        title = "" # Default to no title if gender is not provided

    if title:
        return f"Hello, {title} {name}! Welcome."
    else:
        return f"Hello, {name}! Welcome."

# Example usage:
print(greet_user("Alice", "female"))
print(greet_user("Bob", "male"))
print(greet_user("Charlie"))
print(greet_user("Dana", "Non-binary"))
print(greet_user("Sam", "prefer not to say"))
```

Hello, Ms. Alice! Welcome.  
Hello, Mr. Bob! Welcome.  
Hello, Charlie! Welcome.  
Hello, Dana! Welcome.  
Hello, Sam! Welcome.

Gemini X

def greet\_user(name,gender): if gender.lower()=="male": title="Mr." else: title="Mrs." return f"Hello, {title} {name}! Welcome." add the gender-netural also

You're looking to add a gender-neutral option to your greeting function. That's a great idea for making it more inclusive. Your current code sets the title to "Mrs." for any gender other than "male". To include a gender-neutral option, we can modify the logic to check for specific gender inputs and provide a default or alternative title if the gender is not specified or is indicated as gender-neutral.

I'll modify the existing `greet_user` function to include this.

The `greet_user` function now includes a gender-neutral option and handles cases where gender isn't specified.

What would you like to do next? We could:

- Expand the list of recognized gender-neutral terms or allow for custom titles.
- Create a system to store multiple users and their preferred greetings.
- Explore how to handle cultural differences in greetings.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Explanation:

 explanation

✦ Of course! Let's break down the `greet_user` function in cell `606eef3a` line by line, including the updates for gender-neutral greetings:

- `def greet_user(name, gender=None):` : This defines the function `greet_user` that takes `name` as a required argument and `gender` as an optional argument with a default value of `None`.
- `""" ... """` : This is the docstring explaining the function's purpose, arguments, and what it does.
- `title = ""` : This line initializes an empty string variable called `title`. This variable will store the appropriate title (Mr., Ms., or empty for others) based on the gender.
- `if gender:` : This checks if the `gender` argument was provided (i.e., it's not `None`).
- `if gender.lower() == "male":` : Inside the `if gender:` block, this checks if the lowercase version of the `gender` is "male".
- `title = "Mr."` : If the gender is "male", the `title` is set to "Mr".
- `elif gender.lower() == "female":` : If the gender is not "male", this checks if the lowercase version of the `gender` is "female".
- `title = "Ms."` : If the gender is "female", the `title` is set to "Ms".
- `elif gender.lower() in ["non-binary", "prefer not to say", "gender-neutral"]:` : This is the new part for gender-neutral greetings. It checks if the lowercase `gender` is one of the strings in the list (e.g., "non-binary", "prefer not to say", or "gender-neutral").
- `title = ""` : If the gender matches one of the gender-neutral options, the `title` is set to an empty string, meaning no title will be used. The comment `# Or a preferred gender-neutral title like "Mx." if appropriate` suggests that you could use a different gender-neutral title here if needed.
- `else: title = ""` : If the gender was provided but doesn't match "male", "female", or any of the specified gender-neutral terms, the `title` is also set to an empty string. This acts as a default for unrecognized gender inputs.
- `else: title = ""` : This `else` block corresponds to the initial `if gender:` check. If the `gender` argument was *not* provided (it's `None`), the `title` is set to an empty string.
- `if title:` : This checks if the `title` variable is not empty. This will be true if a title (Mr. or Ms.) was assigned.
- `return f"Hello, {title} {name}! Welcome."` : If a title exists, this line returns a formatted greeting including the title and name.
- `else: return f"Hello, {name}! Welcome."` : If the `title` is empty (either because the gender was not provided, was gender-neutral, or was unrecognized), this line returns a formatted greeting with just the name.

In summary, the function now checks the provided gender (if any) and assigns an appropriate title ("Mr.", "Ms.", or none) to construct a personalized greeting message.

✦ What can I help you build?



Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)