Assignment-5.2

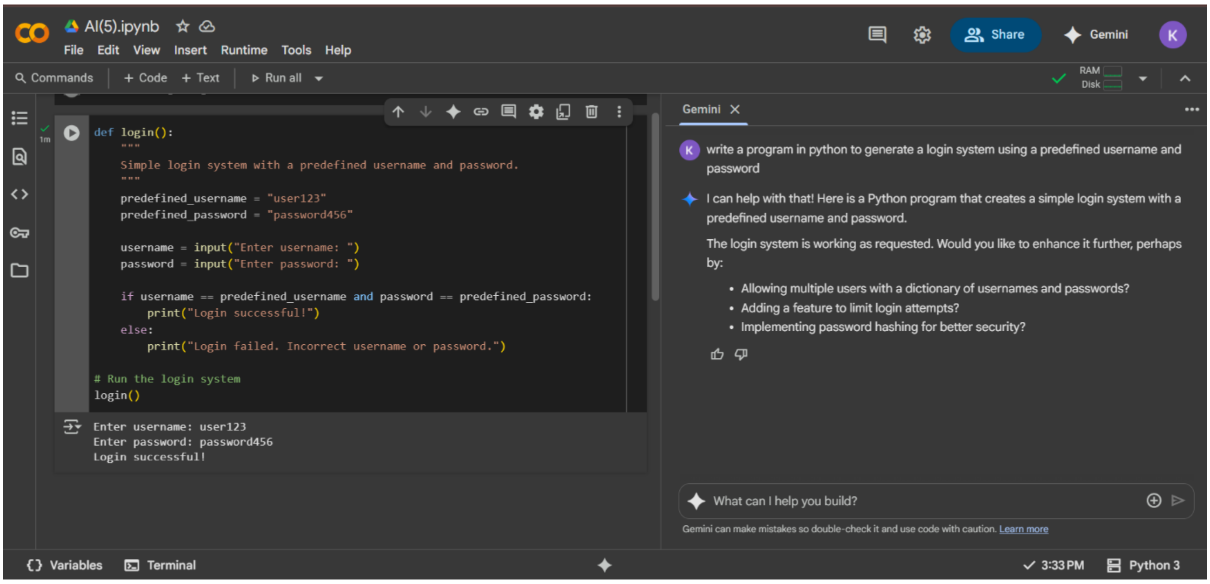
Name : K . Vinod

2403a52132

Task 1:

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

code:



```python
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!
```

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.

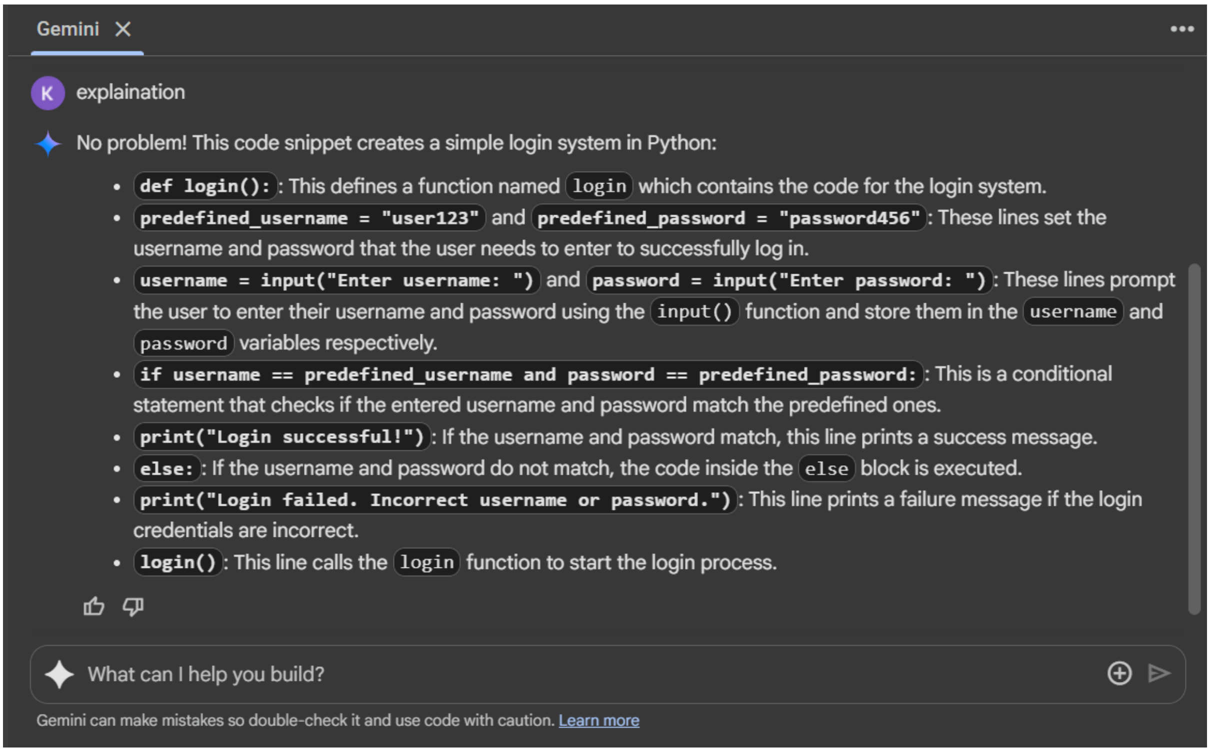
The login system is working as requested. Would you like to enhance it further, perhaps by:

- 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

Explanation:



explaination

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.

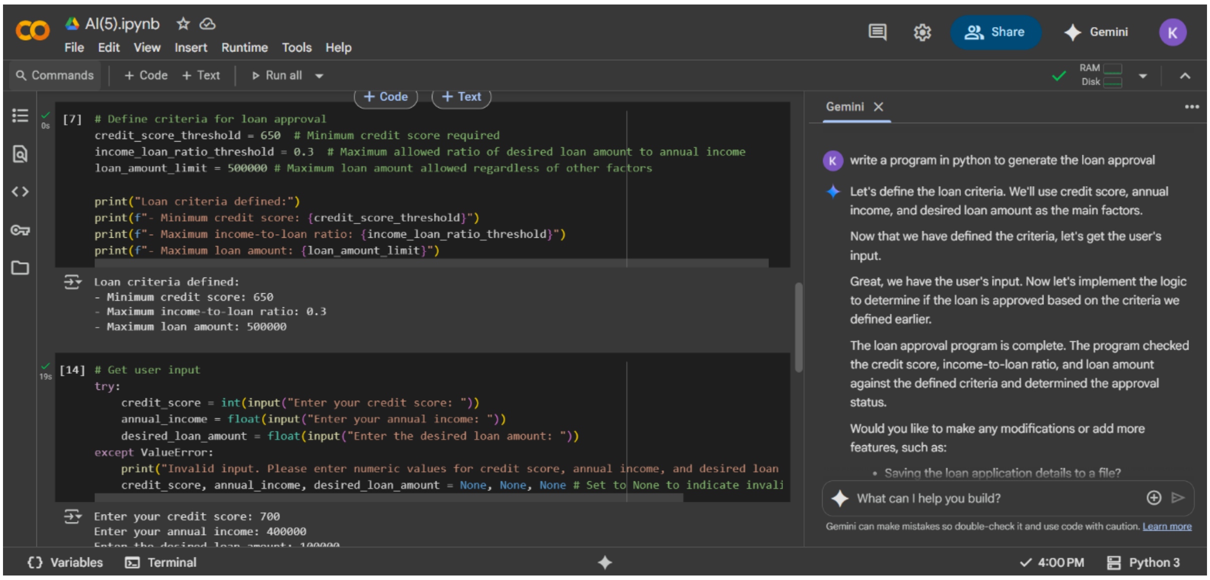
What can I help you build?

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:



```
[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
         credit_score, annual_income, desired_loan_amount = None, None, None # Set to None to indicate invali
```

```
Enter your credit score: 700
Enter your annual income: 400000
```

Gemini

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.

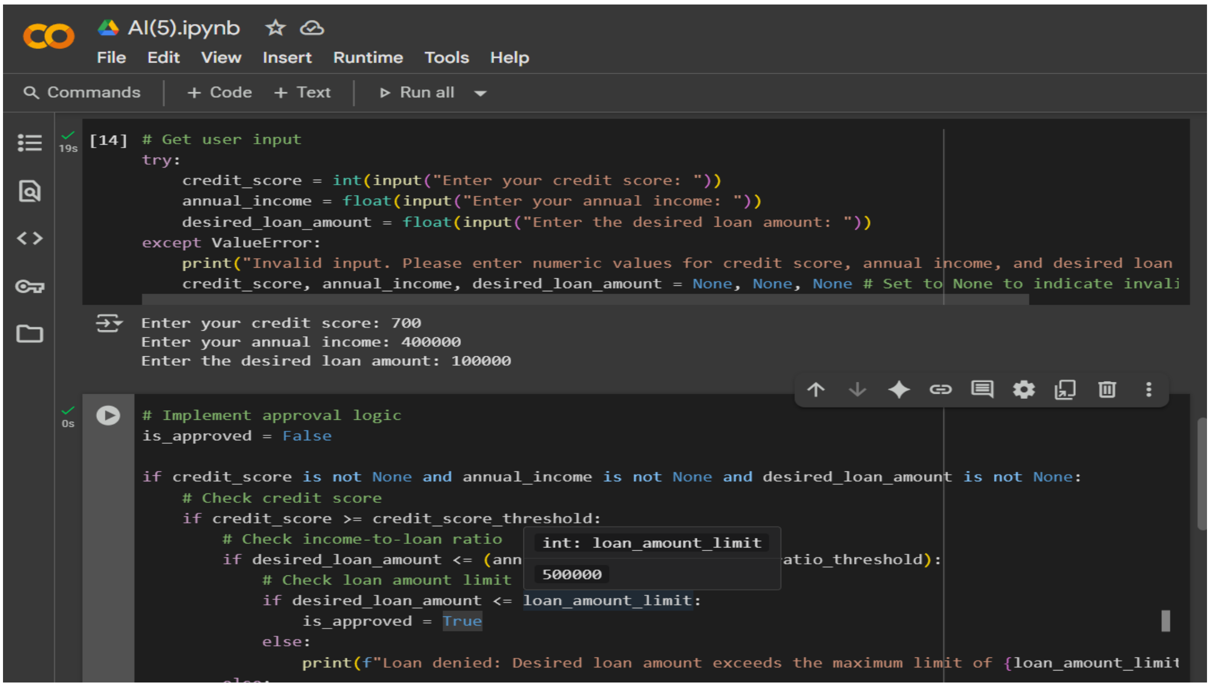
The loan approval program is complete. The program checked the credit score, income-to-loan ratio, and loan amount against the defined criteria and determined the approval status.

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

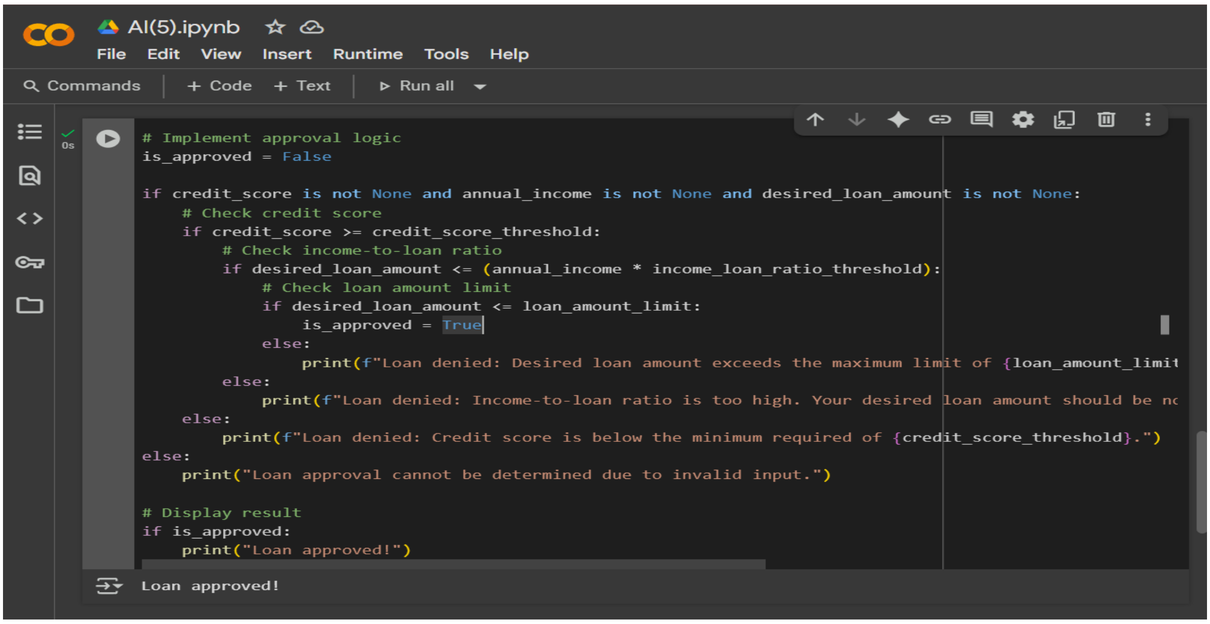


```
[14] # Get user input
     try:
         credit_score = int(input("Enter your credit score: "))
         annual_income = float(input("Enter your annual income: "))
         desired_loan_amount = float(input("Enter the desired loan amount: "))
     except ValueError:
         print("Invalid input.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

```
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 <= (ann                          atio_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
```

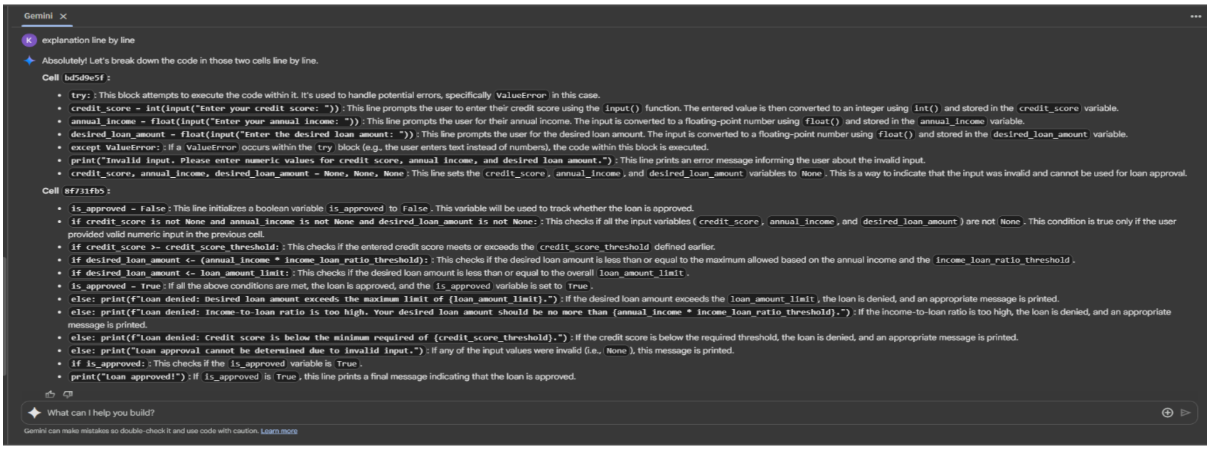
```
int: loan_amount_limit
500000
```



```
# 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 nc
    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:



explanation line by line

Absolutely! Let's break down the code in those two cells line by line.

Cell `bd5d9e5f`:

- `try:` : This block attempts to execute the code within it. It's used to handle potential errors, specifically `ValueError` in this case.
- `credit_score = int(input("Enter your credit score: "))` : This line prompts the user to enter their credit score using the `input()` function. The entered value is then converted to an integer using `int()` and stored in the `credit_score` variable.
- `annual_income = float(input("Enter your annual income: "))` : This line prompts the user for their annual income. The input is converted to a floating-point number using `float()` and stored in the `annual_income` variable.
- `desired_loan_amount = float(input("Enter the desired loan amount: "))` : This line prompts the user for the desired loan amount. The input is converted to a floating-point number using `float()` and stored in the `desired_loan_amount` variable.
- `except ValueError:` : If a `ValueError` occurs within the `try` block (e.g., the user enters text instead of numbers), the code within this block is executed.
- `print("Invalid input. Please enter numeric values for credit score, annual income, and desired loan amount.")` : This line prints an error message informing the user about the invalid input.
- `credit_score, annual_income, desired_loan_amount = None, None, None` : This line sets the `credit_score`, `annual_income`, and `desired_loan_amount` variables to `None`. This is a way to indicate that the input was invalid and cannot be used for loan approval.

Cell `8f731fb5`:

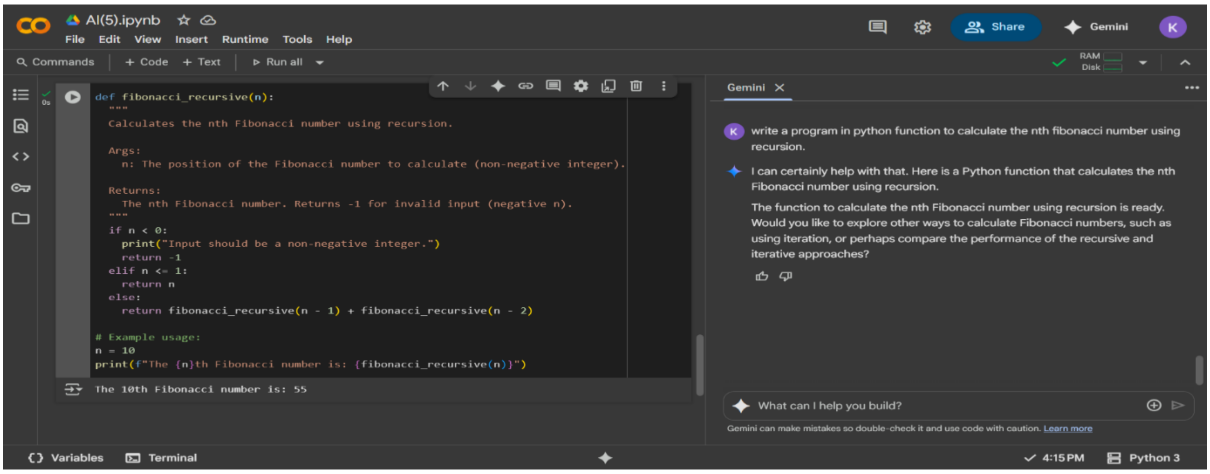
- `is_approved = False` : This line initializes a boolean variable `is_approved` to `False`. This variable will be used to track whether the loan is approved.
- `if credit_score is not None and annual_income is not None and desired_loan_amount is not None:` : This checks if all the input variables (`credit_score`, `annual_income`, and `desired_loan_amount`) are not `None`. This condition is true only if the user provided valid numeric input in the previous cell.
- `if credit_score >= credit_score_threshold:` : This checks if the entered credit score meets or exceeds the `credit_score_threshold` defined earlier.
- `if desired_loan_amount <= (annual_income * income_loan_ratio_threshold):` : This checks if the desired loan amount is less than or equal to the maximum allowed based on the annual income and the `income_loan_ratio_threshold`.
- `if desired_loan_amount <= loan_amount_limit:` : This checks if the desired loan amount is less than or equal to the overall `loan_amount_limit`.
- `is_approved = True` : If all the above conditions are met, the loan is approved, and the `is_approved` variable is set to `True`.
- `else: print(f"Loan denied: Desired loan amount exceeds the maximum limit of {loan_amount_limit}.")` : If the desired loan amount exceeds the `loan_amount_limit`, the loan is denied, and an appropriate message is printed.
- `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}.")` : If the income-to-loan ratio is too high, the loan is denied, and an appropriate message is printed.
- `else: print(f"Loan denied: Credit score is below the minimum required of {credit_score_threshold}.")` : If the credit score is below the required threshold, the loan is denied, and an appropriate message is printed.
- `else: print("Loan approval cannot be determined due to invalid input.")` : If any of the input values were invalid (i.e., `None`), this message is printed.
- `if is_approved:` : This checks if the `is_approved` variable is `True`.
- `print("Loan approved!")` : If `is_approved` is `True`, this line prints a final message indicating that the loan is approved.

Gemini can make mistakes so double-check it and use code with caution. Learn more

Task 3:

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

Code and output:



```
def fibonacci_recursive(n):
  """
  Calculates the nth Fibonacci number using recursion.

  Args:
    n: The position of the Fibonacci number to calculate (non-negative integer).

  Returns:
    The nth Fibonacci number. Returns -1 for invalid input (negative n).
  """
  if n < 0:
    print("Input should be a non-negative integer.")
    return -1
  elif n <= 1:
    return n
  else:
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

# Example usage:
n = 10
print(f"The {n}th Fibonacci number is: {fibonacci_recursive(n)}")
```

```
The 10th Fibonacci number is: 55
```

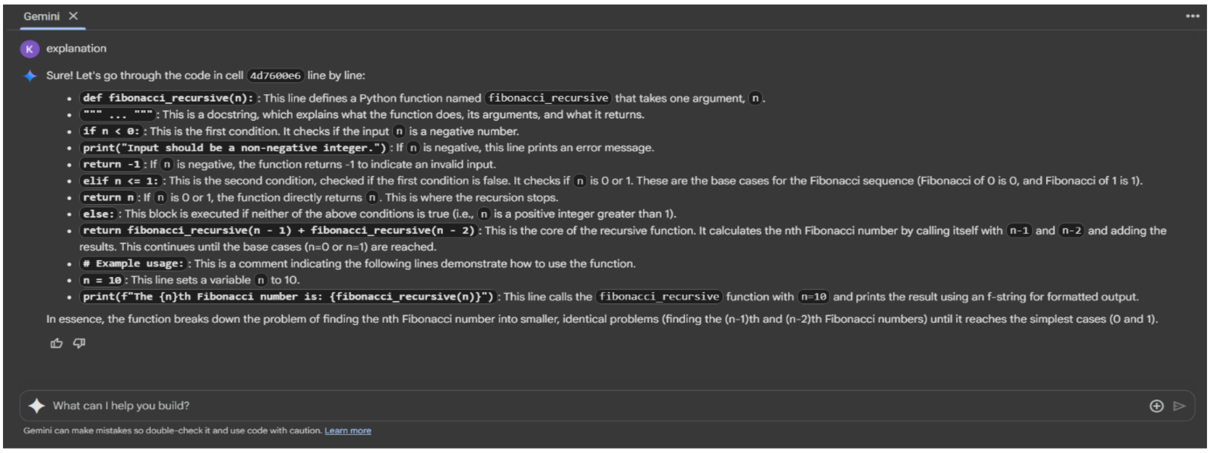
write a program in python function to calculate the nth fibonacci number using recursion.

I can certainly help with that. Here is a Python function that calculates the nth Fibonacci number using recursion.

The function to calculate the nth Fibonacci number using recursion is ready. Would you like to explore other ways to calculate Fibonacci numbers, such as using iteration, or perhaps compare the performance of the recursive and iterative approaches?

Gemini can make mistakes so double-check it and use code with caution. Learn more

Explanation:



explanation

Sure! Let's go through the code in cell `4d7600e6` line by line:

- `def fibonacci_recursive(n):` : This line defines a Python function named `fibonacci_recursive` that takes one argument, `n`.
- `""" ... """` : This is a docstring, which explains what the function does, its arguments, and what it returns.
- `if n < 0:` : This is the first condition. It checks if the input `n` is a negative number.
- `print("Input should be a non-negative integer.")` : If `n` is negative, this line prints an error message.
- `return -1` : If `n` is negative, the function returns -1 to indicate an invalid input.
- `elif n <= 1:` : This is the second condition, checked if the first condition is false. It checks if `n` is 0 or 1. These are the base cases for the Fibonacci sequence (Fibonacci of 0 is 0, and Fibonacci of 1 is 1).
- `return n` : If `n` is 0 or 1, the function directly returns `n`. This is where the recursion stops.
- `else:` : This block is executed if neither of the above conditions is true (i.e., `n` is a positive integer greater than 1).
- `return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)` : This is the core of the recursive function. It calculates the nth Fibonacci number by calling itself with `n-1` and `n-2` and adding the results. This continues until the base cases (n=0 or n=1) are reached.
- `# Example usage:` : This is a comment indicating the following lines demonstrate how to use the function.
- `n = 10` : This line sets a variable `n` to 10.
- `print(f"The {n}th Fibonacci number is: {fibonacci_recursive(n)}")` : This line calls the `fibonacci_recursive` function with `n=10` and prints the result using an f-string for formatted output.

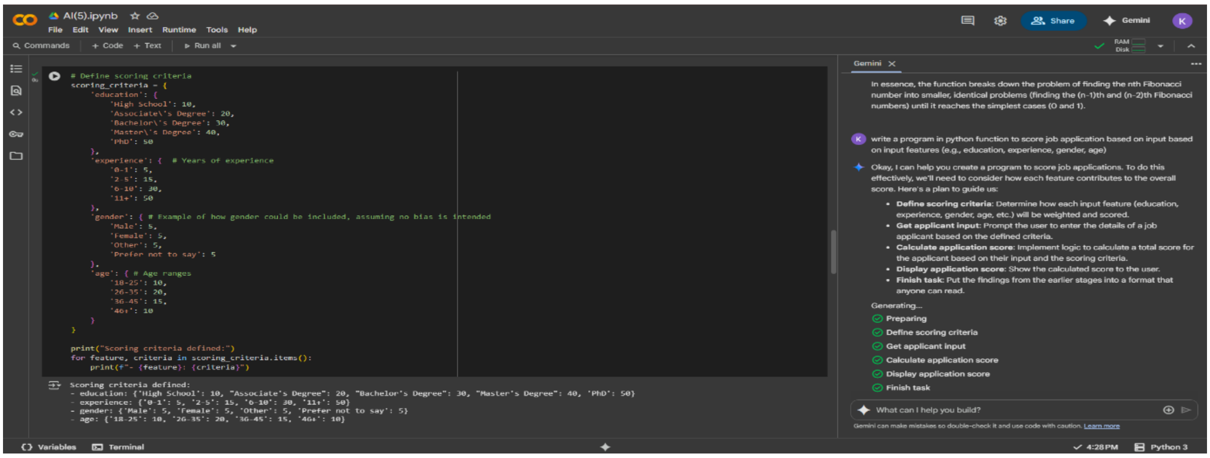
In essence, the function breaks down the problem of finding the nth Fibonacci number into smaller, identical problems (finding the (n-1)th and (n-2)th Fibonacci numbers) until it reaches the simplest cases (0 and 1).

Gemini can make mistakes so double-check it and use code with caution. Learn more

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': 30,
        '11+': 50
    },
    'gender': { # Example of how gender could be included, assuming no bias is 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': 30, '11+': 50}
- gender: {'Male': 5, 'Female': 5, 'Other': 5, 'Prefer not to say': 5}
- age: {'18-25': 10, '26-35': 20, '36-45': 15, '46+': 10}
```

Gemini ✕

In essence, the function breaks down the problem of finding the nth Fibonacci number into smaller, identical problems (finding the (n-1)th and (n-2)th Fibonacci numbers) until it reaches the simplest cases (0 and 1).

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

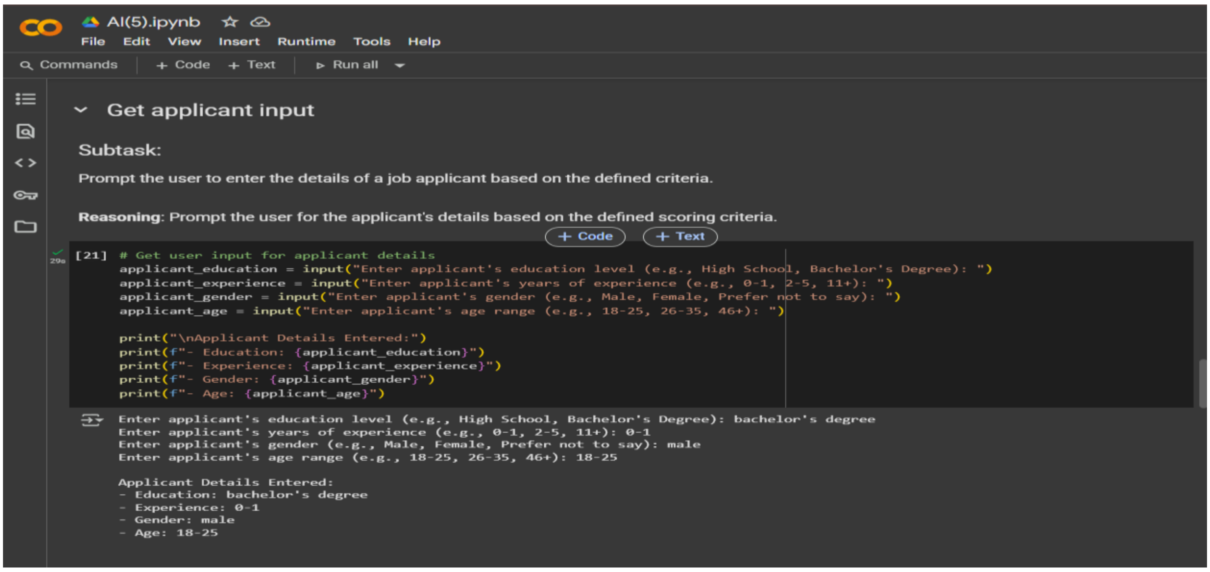
Okay, I can help you create a program to score job applications. To do this effectively, we'll need to consider how each feature contributes to the overall score. Here's a plan to guide us:

- **Define scoring criteria**: Determine how each input feature (education, experience, gender, age, etc.) will be weighted and scored.
- **Get applicant input**: Prompt the user to enter the details of a job applicant based on the defined criteria.
- **Calculate application score**: Implement logic to calculate a total score for the applicant based on their input and the scoring criteria.
- **Display application score**: Show the calculated score to the user.
- **Finish task**: Put the findings from the earlier stages into a format that anyone can read.

Generating...

✓ Preparing
✓ Define scoring criteria
✓ Get applicant input
✓ Calculate application score
✓ Display application score
✓ Finish task

Gemini can make mistakes so double-check it and use code with caution. Learn more



Get applicant input

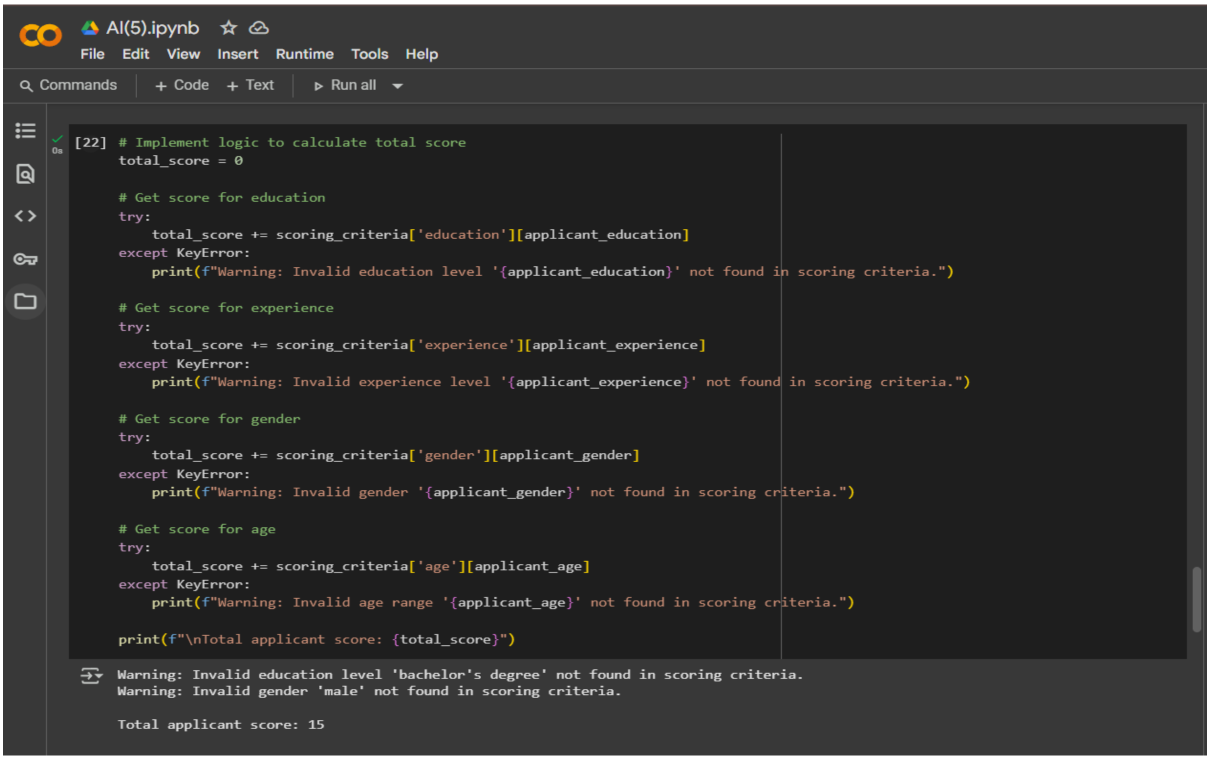
Subtask:

Prompt the user to enter the details of a job applicant based on the defined criteria.

Reasoning: Prompt the user for the applicant's details based on the defined scoring criteria.

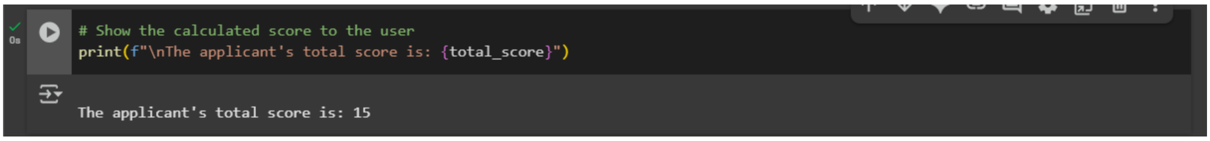
```
[21] # Get user input for applicant details
     applicant_education = input("Enter applicant's education level (e.g., High School, Bachelor's Degree): ")
     applicant_experience = input("Enter applicant's years of experience (e.g., 0-1, 2-5, 11+): ")
     applicant_gender = input("Enter applicant's 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
```



```
[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}")
```

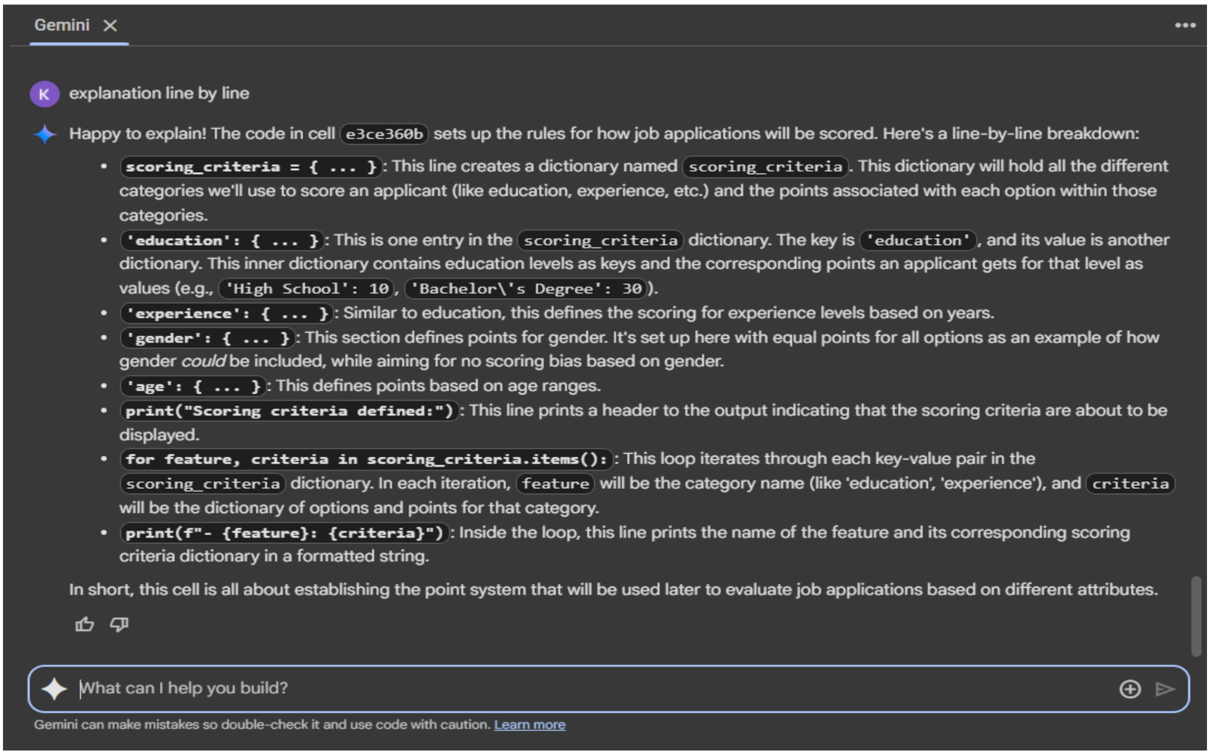
```
Warning: Invalid education level 'bachelor's degree' not found in scoring criteria.
Warning: Invalid gender 'male' not found in scoring criteria.

Total applicant score: 15
```



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

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

Explanation:



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.

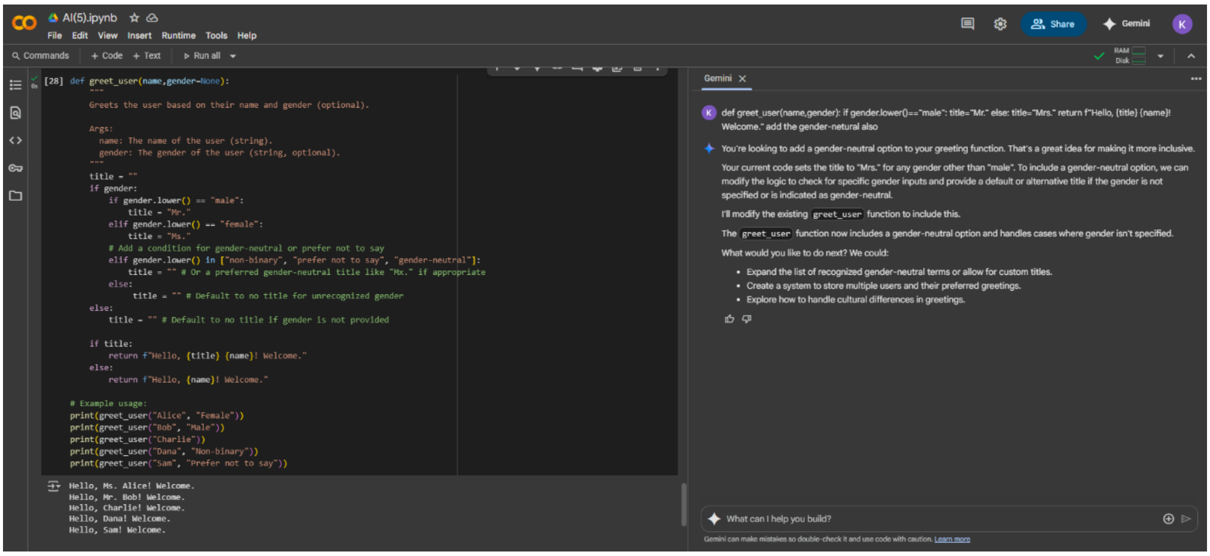
In short, this cell is all about establishing the point system that will be used later to evaluate job applications based on different attributes.

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:



```
[28] def greet_user(name,gender=None):
        """
        Greets 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 = "Ms."
            # 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.
```

K 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.

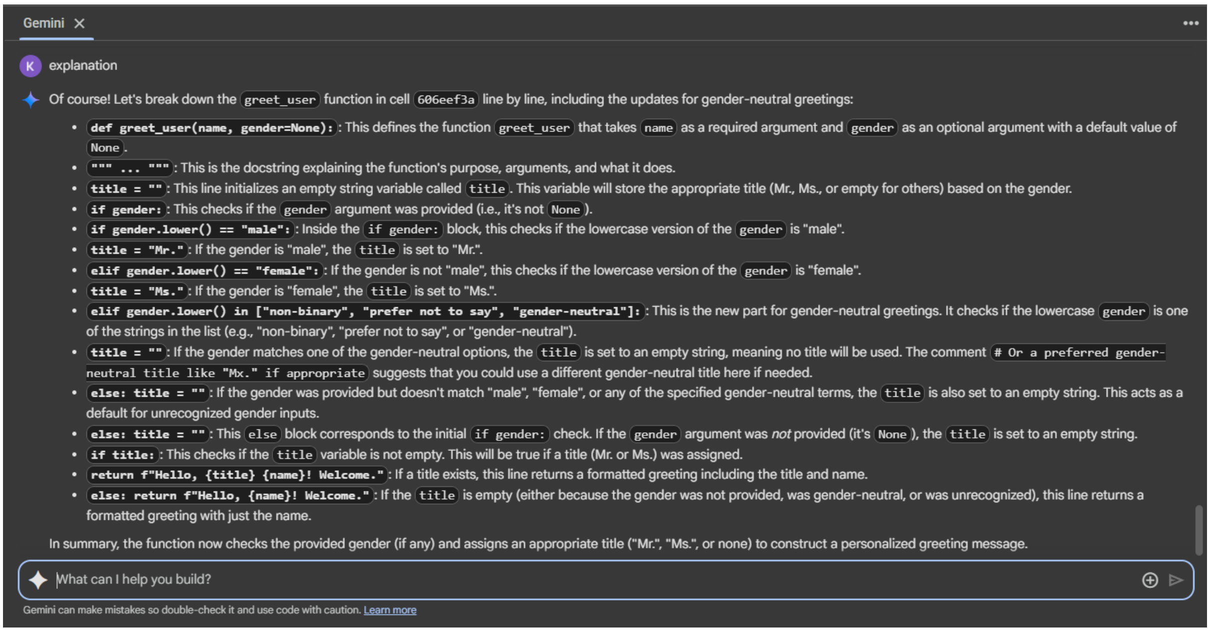
The `greet_user` function now includes a gender-neutral option and handles cases where gender isn't specified.

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:



K 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