AI ASSISTED CODING

Name: P.Swaran Raj **Roll.No:** 2303A52247

Batch: 37 **Assignment-** 5.3

Task Description 1: Privacy and Data Security in AI-Generated Code

Use an AI tool to generate a simple login system in Python.

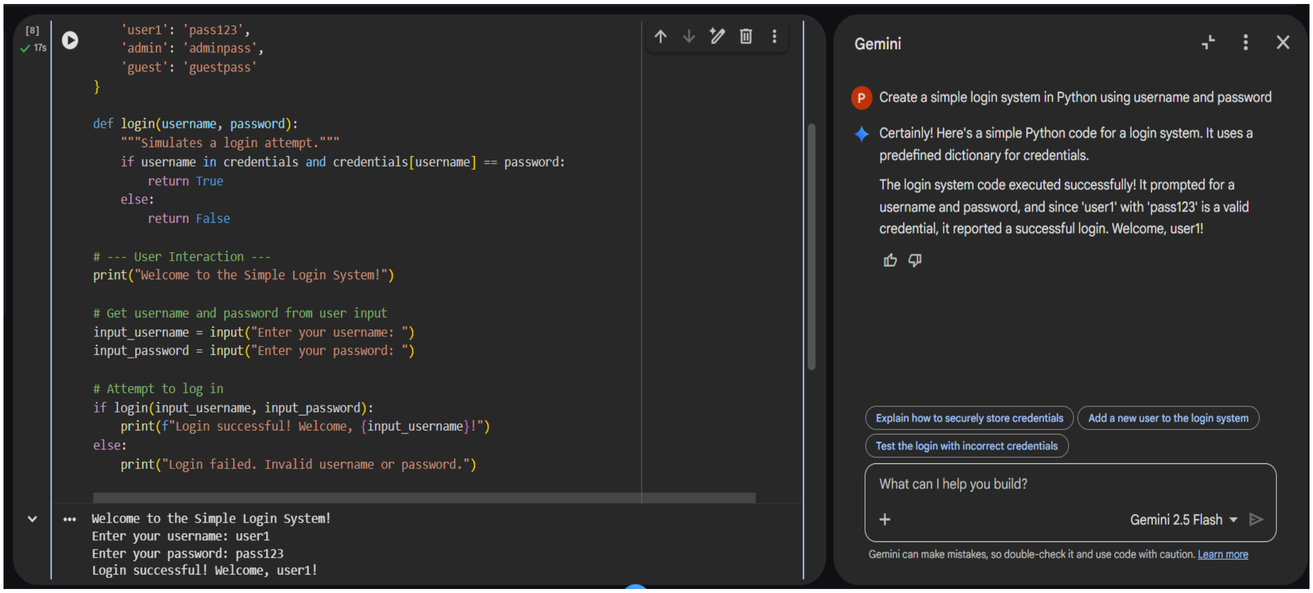
Analyze the generated code to check:

- Whether credentials are hardcoded
- Whether passwords are stored or compared in plain text
- Whether insecure logic is used

Then, revise the code to improve security (e.g., avoid hardcoding, use input validation).

Prompt used:

"#Create a simple login system in Python using username and password."



```python
    'user1': 'pass123',
    'admin': 'adminpass',
    'guest': 'guestpass'
}

def login(username, password):
    """Simulates a login attempt."""
    if username in credentials and credentials[username] == password:
        return True
    else:
        return False

# --- User Interaction ---
print("Welcome to the Simple Login System!")

# Get username and password from user input
input_username = input("Enter your username: ")
input_password = input("Enter your password: ")

# Attempt to log in
if login(input_username, input_password):
    print(f"Login successful! Welcome, {input_username}!")
else:
    print("Login failed. Invalid username or password.")
```

```
Welcome to the Simple Login System!
Enter your username: user1
Enter your password: pass123
Login successful! Welcome, user1!
```

Create a simple login system in Python using username and password

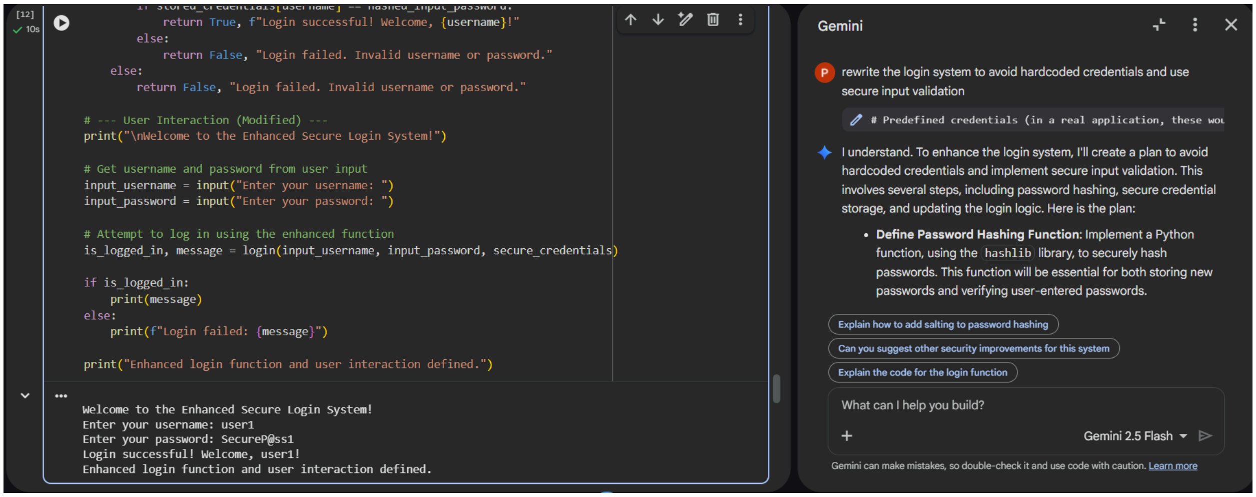
Certainly! Here's a simple Python code for a login system. It uses a predefined dictionary for credentials.

The login system code executed successfully! It prompted for a username and password, and since 'user1' with 'pass123' is a valid credential, it reported a successful login. Welcome, user1!

[8]
17s
Gemini
Explain how to securely store credentials
Add a new user to the login system
Test the login with incorrect credentials
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more

Prompt used:

## “#rewrite the login system to avoid hardcoded credentials and use secure input validation”



```
            return True, f"Login successful! Welcome, {username}!"
        else:
            return False, "Login failed. Invalid username or password."
    else:
        return False, "Login failed. Invalid username or password."

# --- User Interaction (Modified) ---
print("\nWelcome to the Enhanced Secure Login System!")

# Get username and password from user input
input_username = input("Enter your username: ")
input_password = input("Enter your password: ")

# Attempt to log in using the enhanced function
is_logged_in, message = login(input_username, input_password, secure_credentials)

if is_logged_in:
    print(message)
else:
    print(f"Login failed: {message}")

print("Enhanced login function and user interaction defined.")
```

```
Welcome to the Enhanced Secure Login System!
Enter your username: user1
Enter your password: SecureP@ss1
Login successful! Welcome, user1!
Enhanced login function and user interaction defined.
```

rewrite the login system to avoid hardcoded credentials and use secure input validation

`# Predefined credentials (in a real application, these wou`

I understand. To enhance the login system, I'll create a plan to avoid hardcoded credentials and implement secure input validation. This involves several steps, including password hashing, secure credential storage, and updating the login logic. Here is the plan:

- **Define Password Hashing Function**: Implement a Python function, using the `hashlib` library, to securely hash passwords. This function will be essential for both storing new passwords and verifying user-entered passwords.

Explain how to add salting to password hashing

Can you suggest other security improvements for this system

Explain the code for the login function

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

Code explanation:

The AI initially generated a login system with hardcoded credentials and plain-text password comparison. Such practices are insecure as credentials can be easily exposed or misused.

Revised Code Explanation:

The revised version removes hardcoded credentials and uses user input with basic validation. This improves security by avoiding plain-text storage and reducing exposure of sensitive data.

Output:

Users are prompted to enter username and password, and login is validated securely.

Observation:

AI-generated code should be reviewed to ensure privacy and security. Human intervention is necessary to make AI-assisted code responsible and safe.

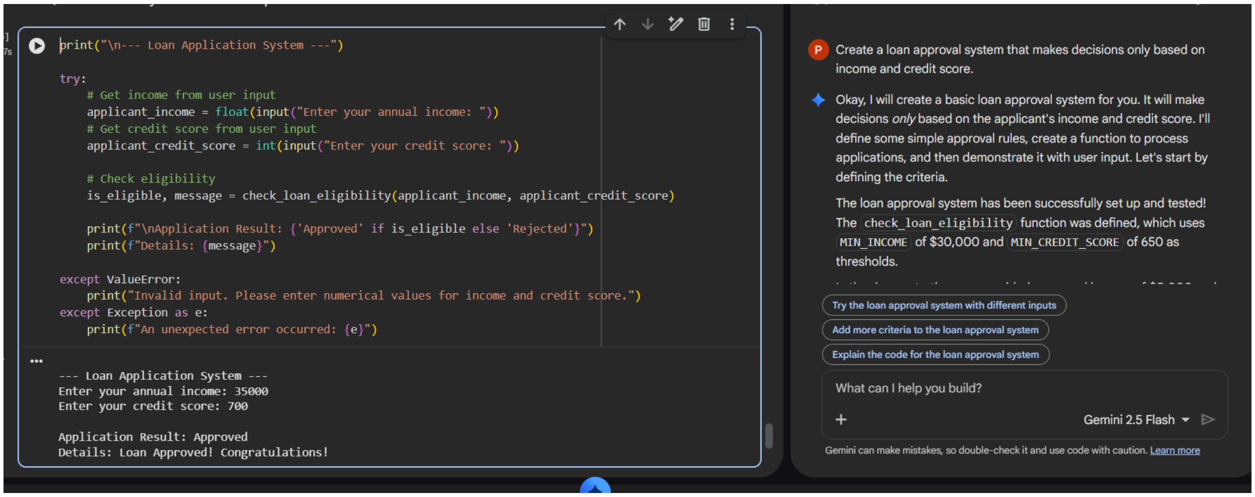
Task Description 2: Bias Detection in AI-Generated Decision Systems

Use AI prompts such as:

- “Create a loan approval system”
- Vary applicant names and genders in prompts Analyze whether:
- The logic treats certain genders or names unfairly
- Approval decisions depend on irrelevant personal attributes Suggest methods to reduce or remove bias.

Prompt used:

“#Create a loan approval system that makes decisions only based on income and credit score.”



```python
print("\n--- Loan Application System ---")

try:
    # Get income from user input
    applicant_income = float(input("Enter your annual income: "))
    # Get credit score from user input
    applicant_credit_score = int(input("Enter your credit score: "))

    # Check eligibility
    is_eligible, message = check_loan_eligibility(applicant_income, applicant_credit_score)

    print(f"\nApplication Result: {'Approved' if is_eligible else 'Rejected'}")
    print(f"Details: {message}")

except ValueError:
    print("Invalid input. Please enter numerical values for income and credit score.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

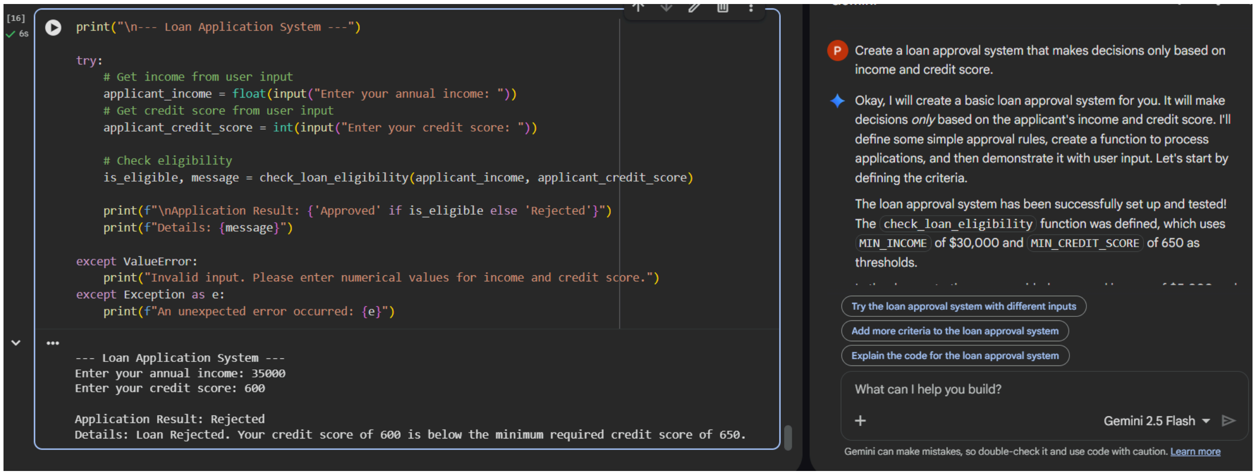
```
--- Loan Application System ---
Enter your annual income: 35000
Enter your credit score: 700

Application Result: Approved
Details: Loan Approved! Congratulations!
```

Create a loan approval system that makes decisions only based on income and credit score.

Okay, I will create a basic loan approval system for you. It will make decisions *only* based on the applicant's income and credit score. I'll define some simple approval rules, create a function to process applications, and then demonstrate it with user input. Let's start by defining the criteria.

The loan approval system has been successfully set up and tested! The `check_loan_eligibility` function was defined, which uses `MIN_INCOME` of $30,000 and `MIN_CREDIT_SCORE` of 650 as thresholds.

Try the loan approval system with different inputs
Add more criteria to the loan approval system
Explain the code for the loan approval system
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more



```python
print("\n--- Loan Application System ---")

try:
    # Get income from user input
    applicant_income = float(input("Enter your annual income: "))
    # Get credit score from user input
    applicant_credit_score = int(input("Enter your credit score: "))

    # Check eligibility
    is_eligible, message = check_loan_eligibility(applicant_income, applicant_credit_score)

    print(f"\nApplication Result: {'Approved' if is_eligible else 'Rejected'}")
    print(f"Details: {message}")

except ValueError:
    print("Invalid input. Please enter numerical values for income and credit score.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

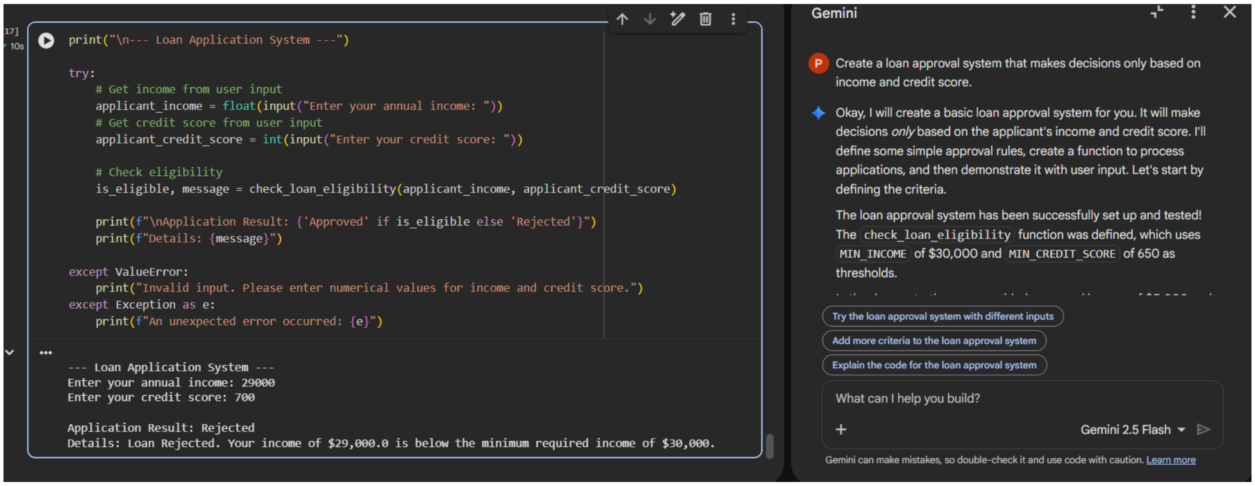
```
--- Loan Application System ---
Enter your annual income: 35000
Enter your credit score: 600

Application Result: Rejected
Details: Loan Rejected. Your credit score of 600 is below the minimum required credit score of 650.
```

Create a loan approval system that makes decisions only based on income and credit score.

Okay, I will create a basic loan approval system for you. It will make decisions *only* based on the applicant's income and credit score. I'll define some simple approval rules, create a function to process applications, and then demonstrate it with user input. Let's start by defining the criteria.

The loan approval system has been successfully set up and tested! The `check_loan_eligibility` function was defined, which uses `MIN_INCOME` of $30,000 and `MIN_CREDIT_SCORE` of 650 as thresholds.

Try the loan approval system with different inputs
Add more criteria to the loan approval system
Explain the code for the loan approval system
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more



```
print("\n--- Loan Application System ---")

try:
    # Get income from user input
    applicant_income = float(input("Enter your annual income: "))
    # Get credit score from user input
    applicant_credit_score = int(input("Enter your credit score: "))

    # Check eligibility
    is_eligible, message = check_loan_eligibility(applicant_income, applicant_credit_score)

    print(f"\nApplication Result: {'Approved' if is_eligible else 'Rejected'}")
    print(f"Details: {message}")

except ValueError:
    print("Invalid input. Please enter numerical values for income and credit score.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
--- Loan Application System ---
Enter your annual income: 29000
Enter your credit score: 700

Application Result: Rejected
Details: Loan Rejected. Your income of $29,000.0 is below the minimum required income of $30,000.
```

Gemini

Create a loan approval system that makes decisions only based on income and credit score.

Okay, I will create a basic loan approval system for you. It will make decisions *only* based on the applicant's income and credit score. I'll define some simple approval rules, create a function to process applications, and then demonstrate it with user input. Let's start by defining the criteria.

The loan approval system has been successfully set up and tested! The `check_loan_eligibility` function was defined, which uses `MIN_INCOME` of $30,000 and `MIN_CREDIT_SCORE` of 650 as thresholds.

Try the loan approval system with different inputs
Add more criteria to the loan approval system
Explain the code for the loan approval system
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more

Code explanation:

The program evaluates loan eligibility using objective financial factors.
It avoids personal attributes, ensuring fairness and unbiased decision-making.

Output:

Loan approval or rejection is displayed based on the given financial inputs.

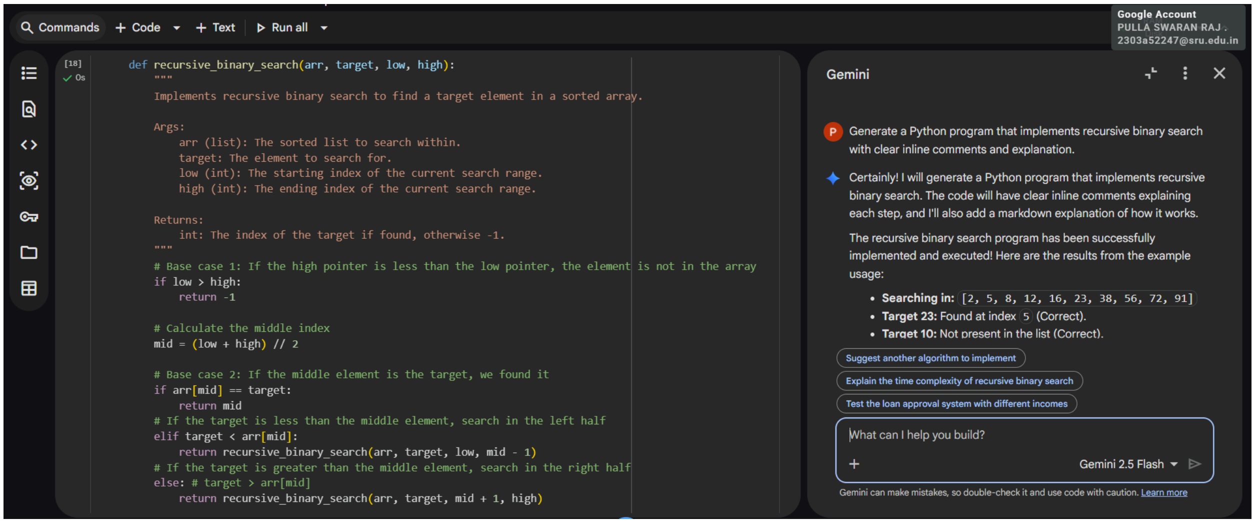
Observation:

Using only relevant financial data reduces bias in decision systems.
Clear prompts help AI generate fair and responsible code.

Task 3: Transparency and Explainability in AI-Generated Code (Recursive Binary Search)

Prompt used:

“#Generate a Python program that implements recursive binary search with clear inline comments and explanation.”



```python
def recursive_binary_search(arr, target, low, high):
    """
    Implements recursive binary search to find a target element in a sorted array.

    Args:
        arr (list): The sorted list to search within.
        target: The element to search for.
        low (int): The starting index of the current search range.
        high (int): The ending index of the current search range.

    Returns:
        int: The index of the target if found, otherwise -1.
    """
    # Base case 1: If the high pointer is less than the low pointer, the element is not in the array
    if low > high:
        return -1

    # Calculate the middle index
    mid = (low + high) // 2

    # Base case 2: If the middle element is the target, we found it
    if arr[mid] == target:
        return mid
    # If the target is less than the middle element, search in the left half
    elif target < arr[mid]:
        return recursive_binary_search(arr, target, low, mid - 1)
    # If the target is greater than the middle element, search in the right half
    else: # target > arr[mid]
        return recursive_binary_search(arr, target, mid + 1, high)
```

Generate a Python program that implements recursive binary search with clear inline comments and explanation.

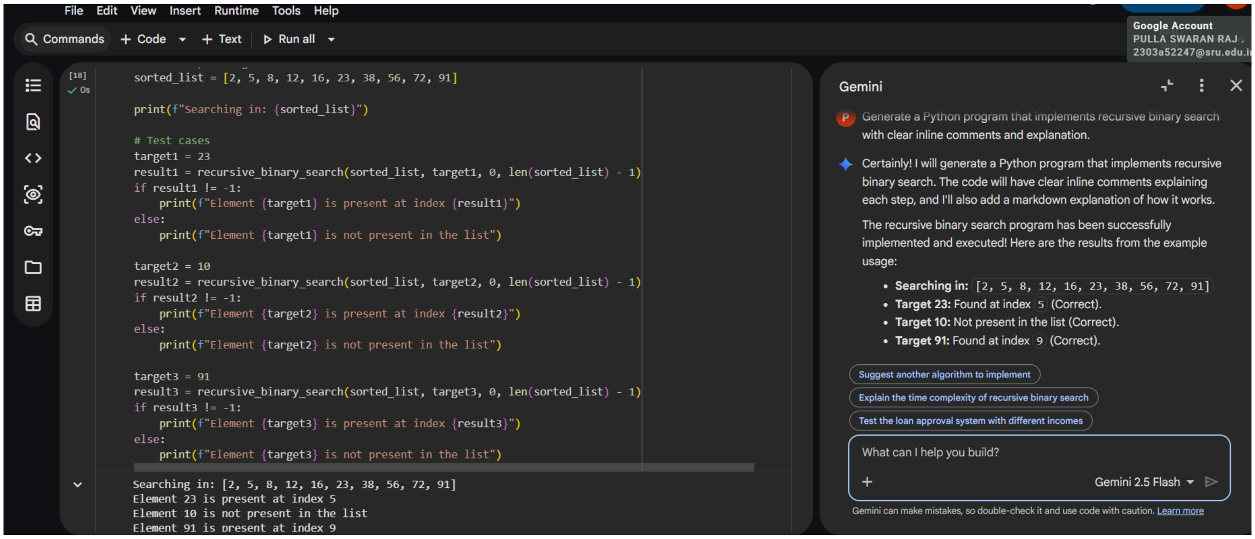
Certainly! I will generate a Python program that implements recursive binary search. The code will have clear inline comments explaining each step, and I'll also add a markdown explanation of how it works.

The recursive binary search program has been successfully implemented and executed! Here are the results from the example usage:

- **Searching in:** `[2, 5, 8, 12, 16, 23, 38, 56, 72, 91]`
- **Target 23:** Found at index `5` (Correct).
- **Target 10:** Not present in the list (Correct).

Suggest another algorithm to implement
Explain the time complexity of recursive binary search
Test the loan approval system with different incomes
What can I help you build?
Gemini 2.5 Flash

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



```python
sorted_list = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]

print(f"Searching in: {sorted_list}")

# Test cases
target1 = 23
result1 = recursive_binary_search(sorted_list, target1, 0, len(sorted_list) - 1)
if result1 != -1:
    print(f"Element {target1} is present at index {result1}")
else:
    print(f"Element {target1} is not present in the list")

target2 = 10
result2 = recursive_binary_search(sorted_list, target2, 0, len(sorted_list) - 1)
if result2 != -1:
    print(f"Element {target2} is present at index {result2}")
else:
    print(f"Element {target2} is not present in the list")

target3 = 91
result3 = recursive_binary_search(sorted_list, target3, 0, len(sorted_list) - 1)
if result3 != -1:
    print(f"Element {target3} is present at index {result3}")
else:
    print(f"Element {target3} is not present in the list")
```

```
Searching in: [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
Element 23 is present at index 5
Element 10 is not present in the list
Element 91 is present at index 9
```

Generate a Python program that implements recursive binary search with clear inline comments and explanation.

Certainly! I will generate a Python program that implements recursive binary search. The code will have clear inline comments explaining each step, and I'll also add a markdown explanation of how it works.

The recursive binary search program has been successfully implemented and executed! Here are the results from the example usage:

- **Searching in:** `[2, 5, 8, 12, 16, 23, 38, 56, 72, 91]`
- **Target 23:** Found at index `5` (Correct).
- **Target 10:** Not present in the list (Correct).
- **Target 91:** Found at index `9` (Correct).

Suggest another algorithm to implement
Explain the time complexity of recursive binary search
Test the loan approval system with different incomes
What can I help you build?
Gemini 2.5 Flash

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

Code explanation:

The program uses recursion to divide the sorted list into smaller halves. It clearly defines the base case and recursive case, making the logic easy to understand.

Output:

The program correctly displays whether the given element is found in the list.

Observation:

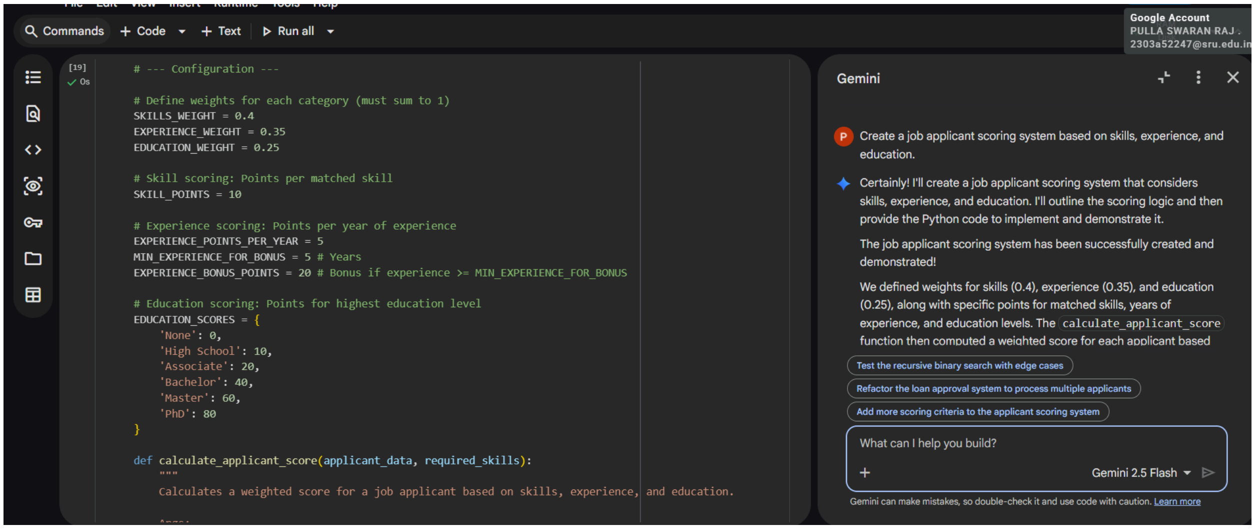
Clear comments and explanations improve transparency and trust in

AI-generated code. Such code is easier for beginners to understand and verify.

Task 4: Ethical Evaluation of AI-Based Scoring Systems

Prompt used:

“#Create a job applicant scoring system based on skills, experience, and education.”



```python
# --- Configuration ---

# Define weights for each category (must sum to 1)
SKILLS_WEIGHT = 0.4
EXPERIENCE_WEIGHT = 0.35
EDUCATION_WEIGHT = 0.25

# Skill scoring: Points per matched skill
SKILL_POINTS = 10

# Experience scoring: Points per year of experience
EXPERIENCE_POINTS_PER_YEAR = 5
MIN_EXPERIENCE_FOR_BONUS = 5 # Years
EXPERIENCE_BONUS_POINTS = 20 # Bonus if experience >= MIN_EXPERIENCE_FOR_BONUS

# Education scoring: Points for highest education level
EDUCATION_SCORES = {
    'None': 0,
    'High School': 10,
    'Associate': 20,
    'Bachelor': 40,
    'Master': 60,
    'PhD': 80
}

def calculate_applicant_score(applicant_data, required_skills):
    """
    Calculates a weighted score for a job applicant based on skills, experience, and education.
```

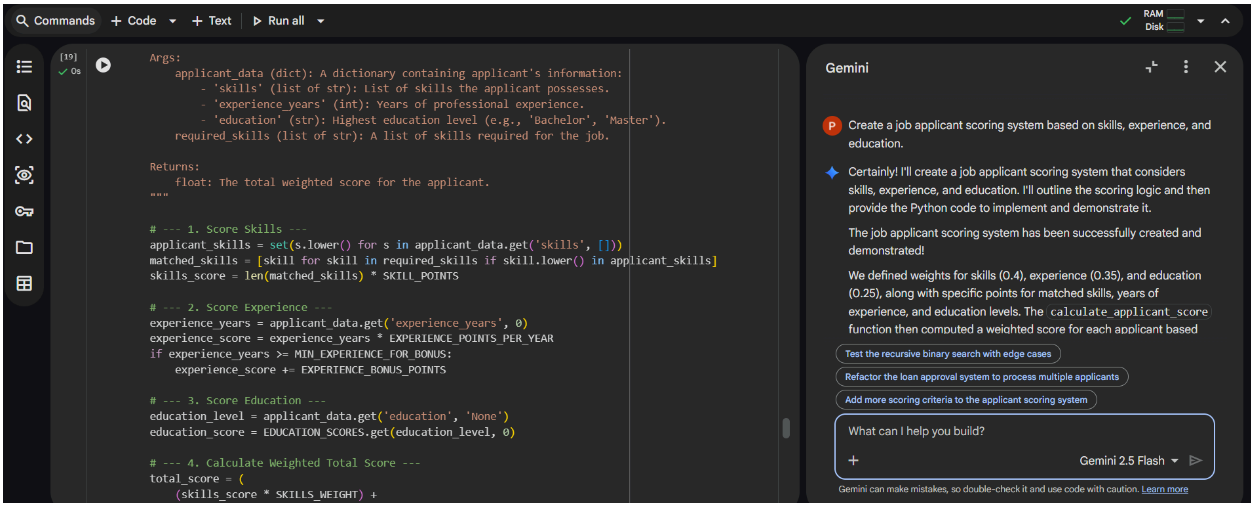
Create a job applicant scoring system based on skills, experience, and education.

Certainly! I'll create a job applicant scoring system that considers skills, experience, and education. I'll outline the scoring logic and then provide the Python code to implement and demonstrate it.

The job applicant scoring system has been successfully created and demonstrated!

We defined weights for skills (0.4), experience (0.35), and education (0.25), along with specific points for matched skills, years of experience, and education levels. The `calculate_applicant_score` function then computed a weighted score for each applicant based

Test the recursive binary search with edge cases
Refactor the loan approval system to process multiple applicants
Add more scoring criteria to the applicant scoring system
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more


Commands
+ Code
+ Text
Run all
RAM
Disk
[19]
0s
Gemini

```python
    Args:
        applicant_data (dict): A dictionary containing applicant's information:
            - 'skills' (list of str): List of skills the applicant possesses.
            - 'experience_years' (int): Years of professional experience.
            - 'education' (str): Highest education level (e.g., 'Bachelor', 'Master').
        required_skills (list of str): A list of skills required for the job.

    Returns:
        float: The total weighted score for the applicant.
    """

    # --- 1. Score Skills ---
    applicant_skills = set(s.lower() for s in applicant_data.get('skills', []))
    matched_skills = [skill for skill in required_skills if skill.lower() in applicant_skills]
    skills_score = len(matched_skills) * SKILL_POINTS

    # --- 2. Score Experience ---
    experience_years = applicant_data.get('experience_years', 0)
    experience_score = experience_years * EXPERIENCE_POINTS_PER_YEAR
    if experience_years >= MIN_EXPERIENCE_FOR_BONUS:
        experience_score += EXPERIENCE_BONUS_POINTS

    # --- 3. Score Education ---
    education_level = applicant_data.get('education', 'None')
    education_score = EDUCATION_SCORES.get(education_level, 0)

    # --- 4. Calculate Weighted Total Score ---
    total_score = (
        (skills_score * SKILLS_WEIGHT) +
```

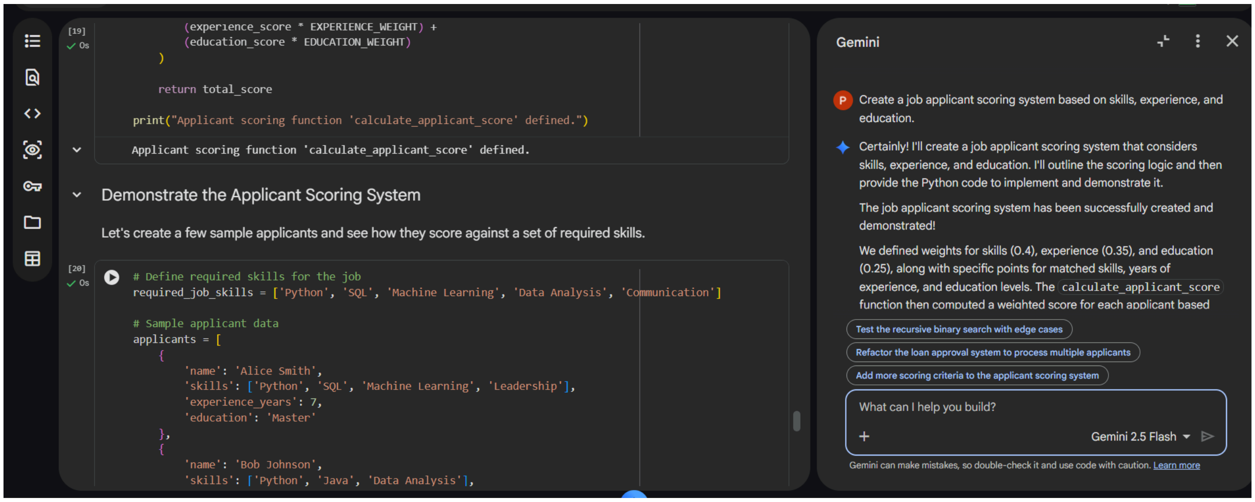
Create a job applicant scoring system based on skills, experience, and education.

Certainly! I'll create a job applicant scoring system that considers skills, experience, and education. I'll outline the scoring logic and then provide the Python code to implement and demonstrate it.

The job applicant scoring system has been successfully created and demonstrated!

We defined weights for skills (0.4), experience (0.35), and education (0.25), along with specific points for matched skills, years of experience, and education levels. The `calculate_applicant_score` function then computed a weighted score for each applicant based

Test the recursive binary search with edge cases
Refactor the loan approval system to process multiple applicants
Add more scoring criteria to the applicant scoring system
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more



```
            (experience_score * EXPERIENCE_WEIGHT) +
            (education_score * EDUCATION_WEIGHT)
        )

        return total_score

    print("Applicant scoring function 'calculate_applicant_score' defined.")
```

```
Applicant scoring function 'calculate_applicant_score' defined.
```

Demonstrate the Applicant Scoring System

Let's create a few sample applicants and see how they score against a set of required skills.

```
# Define required skills for the job
required_job_skills = ['Python', 'SQL', 'Machine Learning', 'Data Analysis', 'Communication']

# Sample applicant data
applicants = [
    {
        'name': 'Alice Smith',
        'skills': ['Python', 'SQL', 'Machine Learning', 'Leadership'],
        'experience_years': 7,
        'education': 'Master'
    },
    {
        'name': 'Bob Johnson',
        'skills': ['Python', 'Java', 'Data Analysis'],
```

Gemini

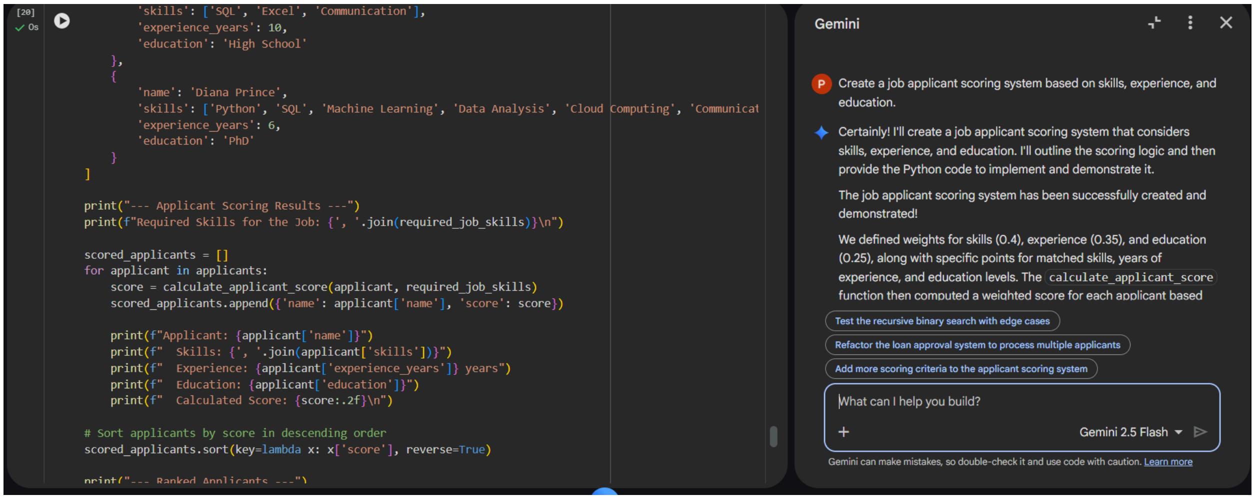
Create a job applicant scoring system based on skills, experience, and education.

Certainly! I'll create a job applicant scoring system that considers skills, experience, and education. I'll outline the scoring logic and then provide the Python code to implement and demonstrate it.

The job applicant scoring system has been successfully created and demonstrated!

We defined weights for skills (0.4), experience (0.35), and education (0.25), along with specific points for matched skills, years of experience, and education levels. The `calculate_applicant_score` function then computed a weighted score for each applicant based

Test the recursive binary search with edge cases
Refactor the loan approval system to process multiple applicants
Add more scoring criteria to the applicant scoring system
What can I help you build?
+
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more



```
        'skills': ['SQL', 'Excel', 'Communication'],
        'experience_years': 10,
        'education': 'High School'
    },
    {
        'name': 'Diana Prince',
        'skills': ['Python', 'SQL', 'Machine Learning', 'Data Analysis', 'Cloud Computing', 'Communicat
        'experience_years': 6,
        'education': 'PhD'
    }
]

print("--- Applicant Scoring Results ---")
print(f"Required Skills for the Job: {', '.join(required_job_skills)}\n")

scored_applicants = []
for applicant in applicants:
    score = calculate_applicant_score(applicant, required_job_skills)
    scored_applicants.append({'name': applicant['name'], 'score': score})

    print(f"Applicant: {applicant['name']}")
    print(f"  Skills: {', '.join(applicant['skills'])}")
    print(f"  Experience: {applicant['experience_years']} years")
    print(f"  Education: {applicant['education']}")
    print(f"  Calculated Score: {score:.2f}\n")

# Sort applicants by score in descending order
scored_applicants.sort(key=lambda x: x['score'], reverse=True)

print("--- Ranked Applicants ---")
```

Gemini

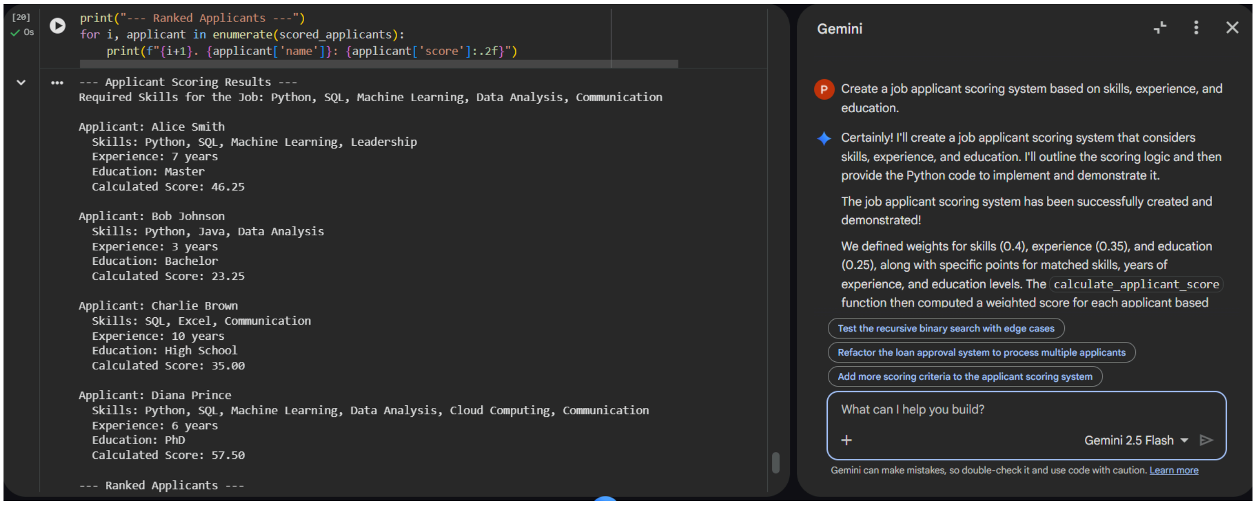
Create a job applicant scoring system based on skills, experience, and education.

Certainly! I'll create a job applicant scoring system that considers skills, experience, and education. I'll outline the scoring logic and then provide the Python code to implement and demonstrate it.

The job applicant scoring system has been successfully created and demonstrated!

We defined weights for skills (0.4), experience (0.35), and education (0.25), along with specific points for matched skills, years of experience, and education levels. The `calculate_applicant_score` function then computed a weighted score for each applicant based

Test the recursive binary search with edge cases
Refactor the loan approval system to process multiple applicants
Add more scoring criteria to the applicant scoring system
What can I help you build?
+
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more



```
print("--- Ranked Applicants ---")
for i, applicant in enumerate(scored_applicants):
    print(f"{i+1}. {applicant['name']}: {applicant['score']:.2f}")
```

```
--- Applicant Scoring Results ---
Required Skills for the Job: Python, SQL, Machine Learning, Data Analysis, Communication

Applicant: Alice Smith
  Skills: Python, SQL, Machine Learning, Leadership
  Experience: 7 years
  Education: Master
  Calculated Score: 46.25

Applicant: Bob Johnson
  Skills: Python, Java, Data Analysis
  Experience: 3 years
  Education: Bachelor
  Calculated Score: 23.25

Applicant: Charlie Brown
  Skills: SQL, Excel, Communication
  Experience: 10 years
  Education: High School
  Calculated Score: 35.00

Applicant: Diana Prince
  Skills: Python, SQL, Machine Learning, Data Analysis, Cloud Computing, Communication
  Experience: 6 years
  Education: PhD
  Calculated Score: 57.50

--- Ranked Applicants ---
```

Gemini

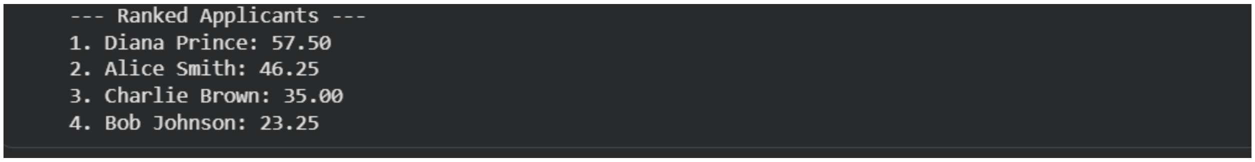
Create a job applicant scoring system based on skills, experience, and education.

Certainly! I'll create a job applicant scoring system that considers skills, experience, and education. I'll outline the scoring logic and then provide the Python code to implement and demonstrate it.

The job applicant scoring system has been successfully created and demonstrated!

We defined weights for skills (0.4), experience (0.35), and education (0.25), along with specific points for matched skills, years of experience, and education levels. The `calculate_applicant_score` function then computed a weighted score for each applicant based

Test the recursive binary search with edge cases
Refactor the loan approval system to process multiple applicants
Add more scoring criteria to the applicant scoring system
What can I help you build?
+
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more



```
--- Ranked Applicants ---
1. Diana Prince: 57.50
2. Alice Smith: 46.25
3. Charlie Brown: 35.00
4. Bob Johnson: 23.25
```

Code explanation:

The program assigns scores to applicants using job-relevant attributes such as skills, experience, and education.

It avoids using personal details like name or gender, ensuring objective and ethical scoring.

Output:

Applicants receive a numerical score based on professional qualifications.

Observation:

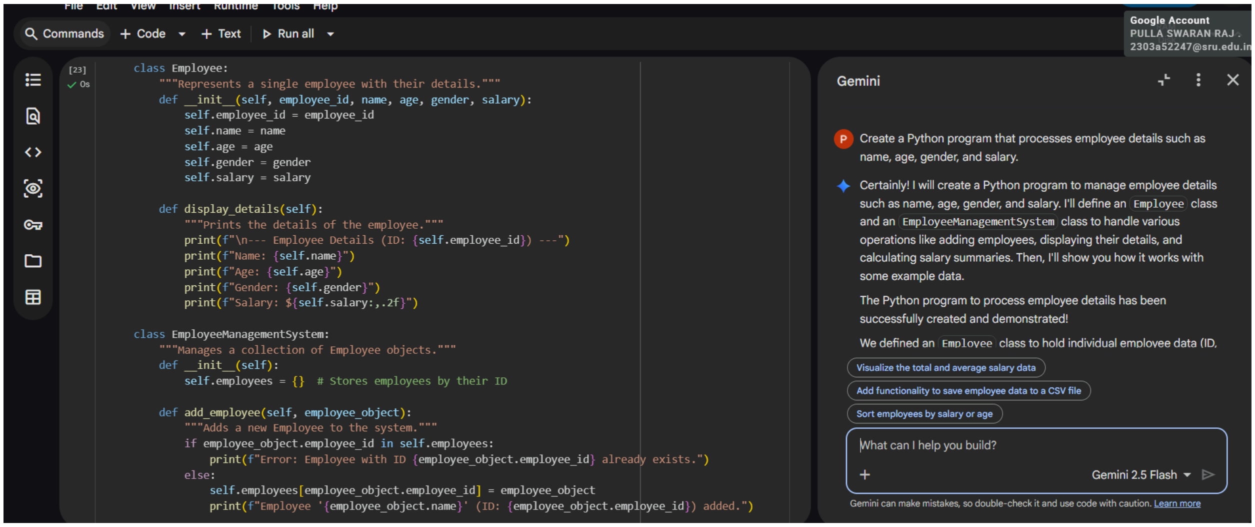
Using only relevant factors helps prevent bias in scoring systems.

Ethical evaluation is necessary to ensure fairness in AI-assisted hiring decisions.

Task 5: Inclusiveness and Ethical Variable Design

Prompt used:

"#Create a Python program that processes employee details such as name, age, gender, and salary."


File
Edit
View
Insert
Runtime
Tools
Help
Commands
+ Code
+ Text
Run all
[23]
0s
Google Account
PULLA SWARAN RAJ .
2303a52247@sru.edu.in
Gemini
Visualize the total and average salary data
Add functionality to save employee data to a CSV file
Sort employees by salary or age
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more

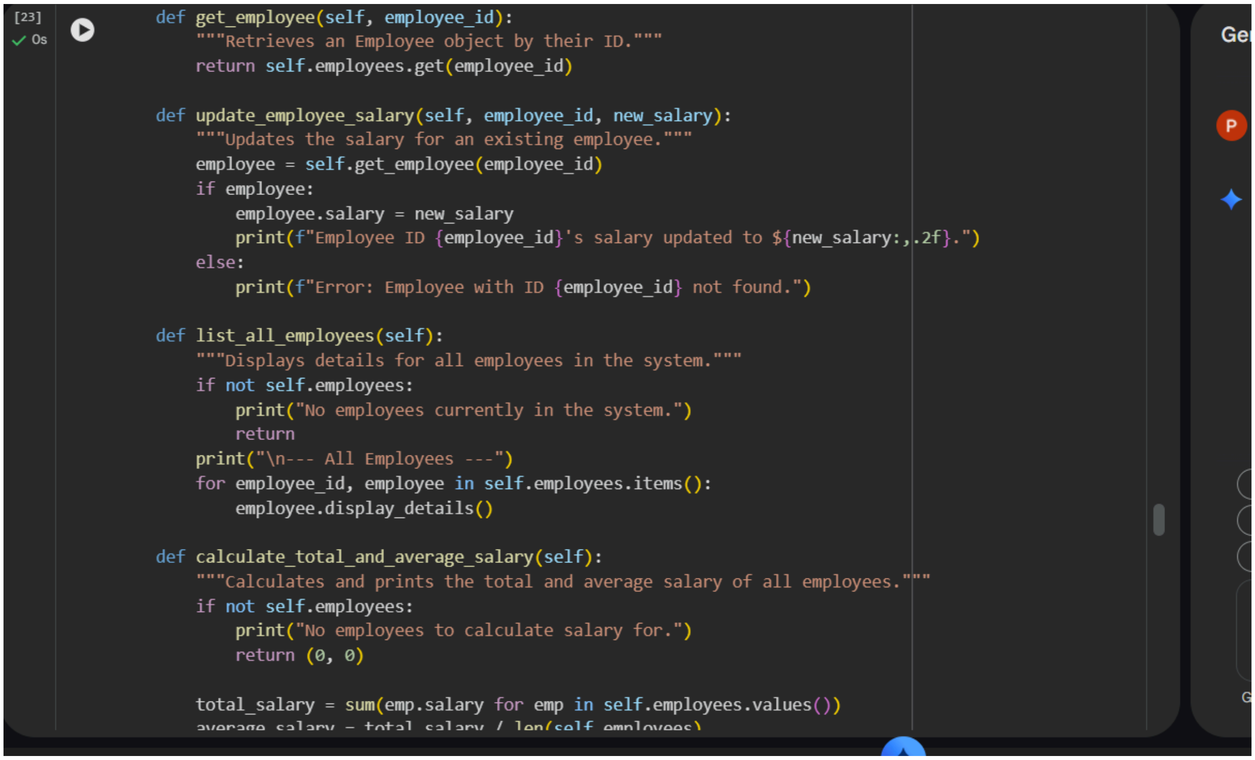
```python
class Employee:
    """Represents a single employee with their details."""
    def __init__(self, employee_id, name, age, gender, salary):
        self.employee_id = employee_id
        self.name = name
        self.age = age
        self.gender = gender
        self.salary = salary

    def display_details(self):
        """Prints the details of the employee."""
        print(f"\n--- Employee Details (ID: {self.employee_id}) ---")
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Gender: {self.gender}")
        print(f"Salary: ${self.salary:,.2f}")

class EmployeeManagementSystem:
    """Manages a collection of Employee objects."""
    def __init__(self):
        self.employees = {}  # Stores employees by their ID

    def add_employee(self, employee_object):
        """Adds a new Employee to the system."""
        if employee_object.employee_id in self.employees:
            print(f"Error: Employee with ID {employee_object.employee_id} already exists.")
        else:
            self.employees[employee_object.employee_id] = employee_object
            print(f"Employee '{employee_object.name}' (ID: {employee_object.employee_id}) added.")
```

P Create a Python program that processes employee details such as name, age, gender, and salary.

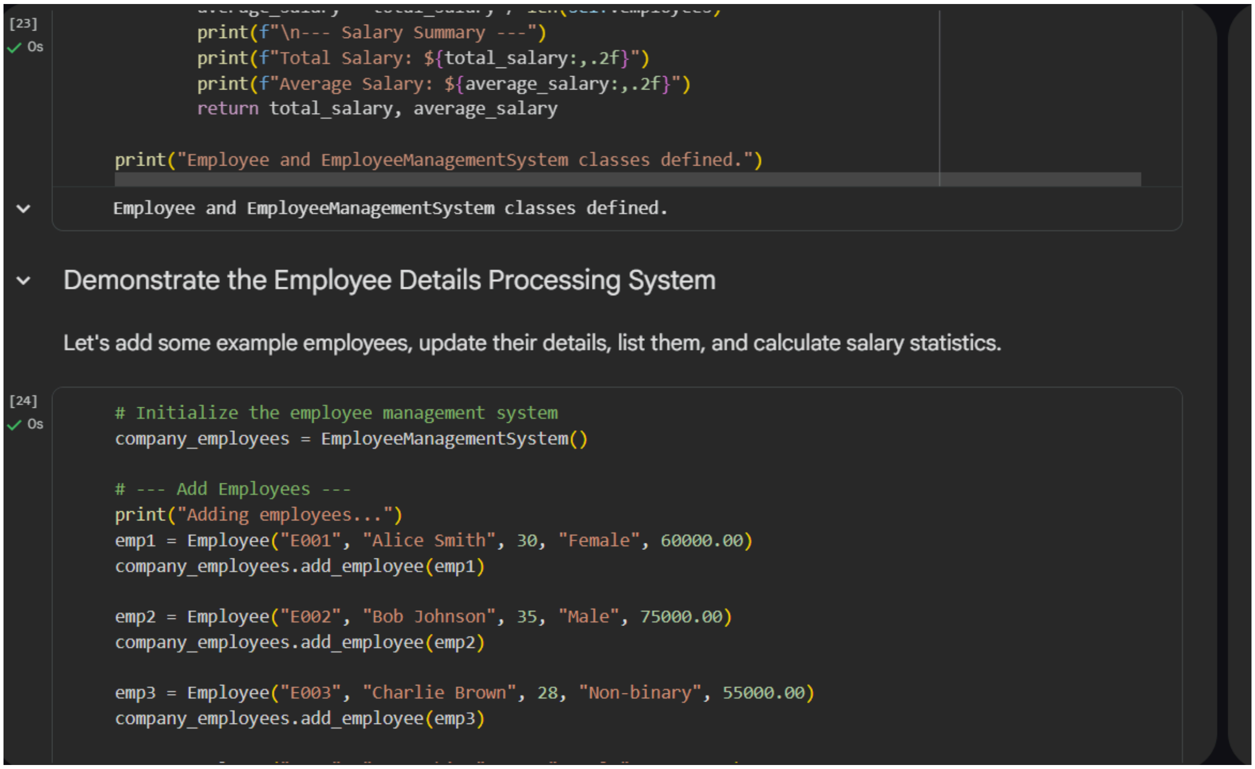
Certainly! I will create a Python program to manage employee details such as name, age, gender, and salary. I'll define an `Employee` class and an `EmployeeManagementSystem` class to handle various operations like adding employees, displaying their details, and calculating salary summaries. Then, I'll show you how it works with some example data.

The Python program to process employee details has been successfully created and demonstrated!

We defined an `Employee` class to hold individual employee data (ID,



```
    def get_employee(self, employee_id):
        """Retrieves an Employee object by their ID."""
        return self.employees.get(employee_id)

    def update_employee_salary(self, employee_id, new_salary):
        """Updates the salary for an existing employee."""
        employee = self.get_employee(employee_id)
        if employee:
            employee.salary = new_salary
            print(f"Employee ID {employee_id}'s salary updated to ${new_salary:,.2f}.")
        else:
            print(f"Error: Employee with ID {employee_id} not found.")

    def list_all_employees(self):
        """Displays details for all employees in the system."""
        if not self.employees:
            print("No employees currently in the system.")
            return
        print("\n--- All Employees ---")
        for employee_id, employee in self.employees.items():
            employee.display_details()

    def calculate_total_and_average_salary(self):
        """Calculates and prints the total and average salary of all employees."""
        if not self.employees:
            print("No employees to calculate salary for.")
            return (0, 0)

        total_salary = sum(emp.salary for emp in self.employees.values())
        average_salary = total_salary / len(self.employees)
```



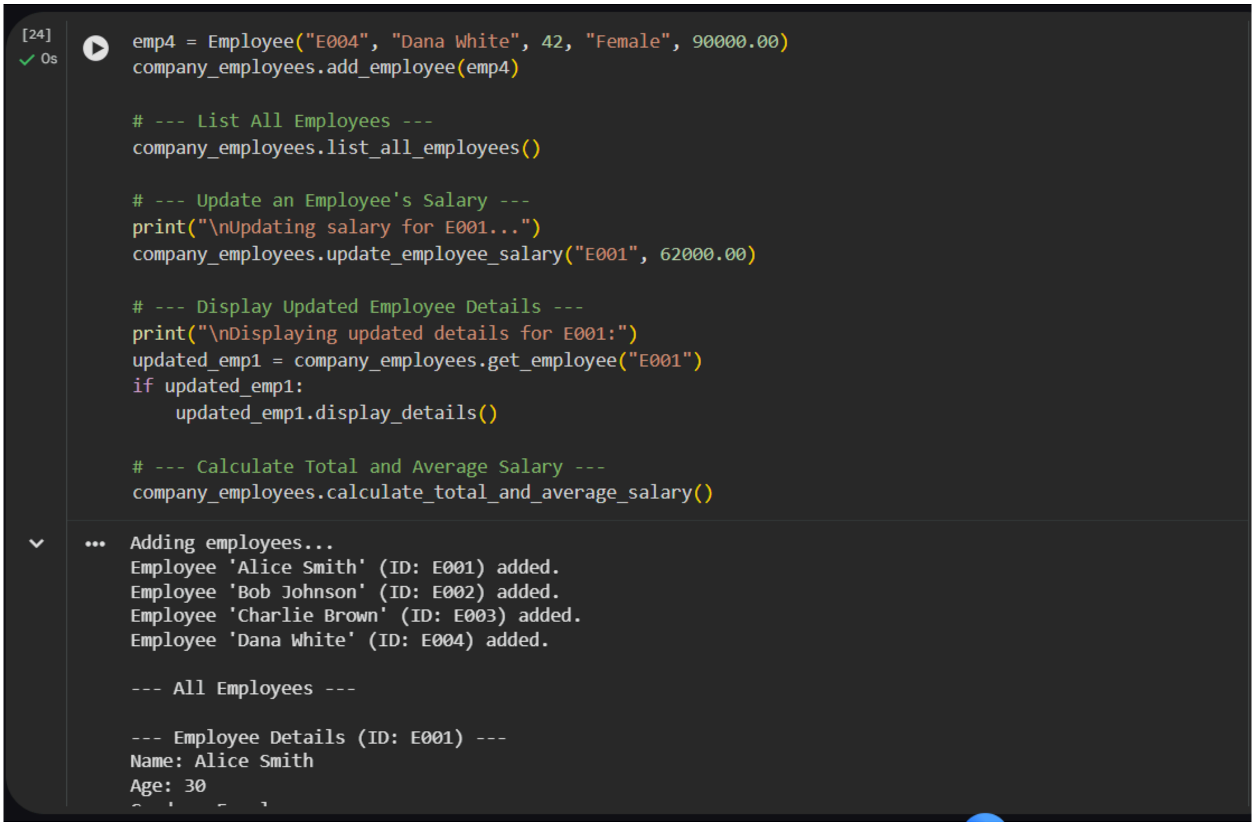
```
        print(f"\n--- Salary Summary ---")
        print(f"Total Salary: ${total_salary:,.2f}")
        print(f"Average Salary: ${average_salary:,.2f}")
        return total_salary, average_salary

print("Employee and EmployeeManagementSystem classes defined.")
```

```
Employee and EmployeeManagementSystem classes defined.
```

Demonstrate the Employee Details Processing System

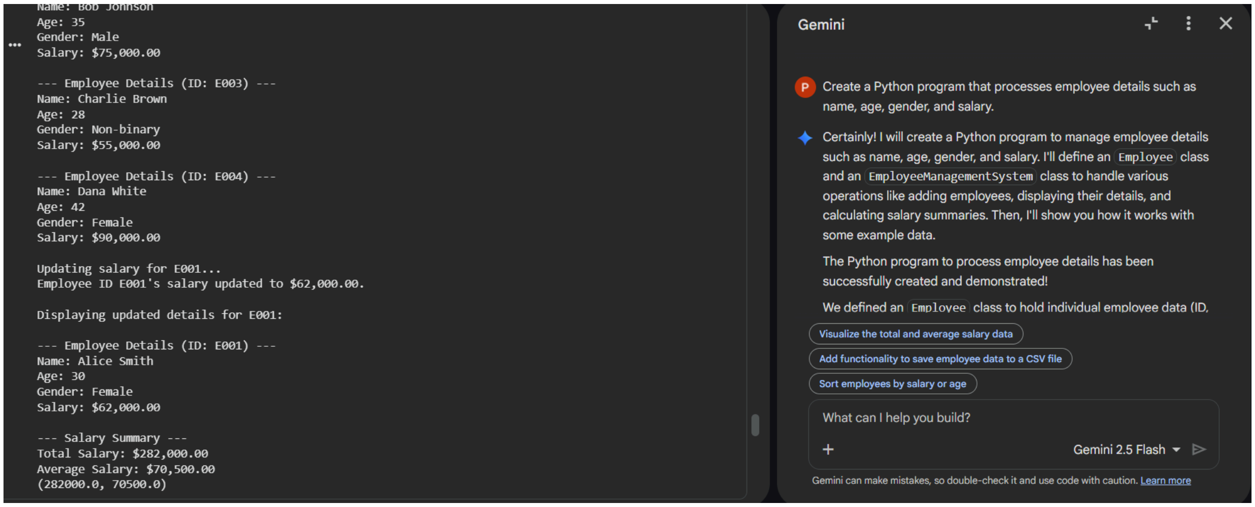
Let's add some example employees, update their details, list them, and calculate salary statistics.

```
# Initialize the employee management system
company_employees = EmployeeManagementSystem()

# --- Add Employees ---
print("Adding employees...")
emp1 = Employee("E001", "Alice Smith", 30, "Female", 60000.00)
company_employees.add_employee(emp1)

emp2 = Employee("E002", "Bob Johnson", 35, "Male", 75000.00)
company_employees.add_employee(emp2)

emp3 = Employee("E003", "Charlie Brown", 28, "Non-binary", 55000.00)
company_employees.add_employee(emp3)
```



```python
emp4 = Employee("E004", "Dana White", 42, "Female", 90000.00)
company_employees.add_employee(emp4)

# --- List All Employees ---
company_employees.list_all_employees()

# --- Update an Employee's Salary ---
print("\nUpdating salary for E001...")
company_employees.update_employee_salary("E001", 62000.00)

# --- Display Updated Employee Details ---
print("\nDisplaying updated details for E001:")
updated_emp1 = company_employees.get_employee("E001")
if updated_emp1:
    updated_emp1.display_details()

# --- Calculate Total and Average Salary ---
company_employees.calculate_total_and_average_salary()
```

```
Adding employees...
Employee 'Alice Smith' (ID: E001) added.
Employee 'Bob Johnson' (ID: E002) added.
Employee 'Charlie Brown' (ID: E003) added.
Employee 'Dana White' (ID: E004) added.

--- All Employees ---

--- Employee Details (ID: E001) ---
Name: Alice Smith
Age: 30
```



```
Name: Bob Johnson
Age: 35
Gender: Male
Salary: $75,000.00

--- Employee Details (ID: E003) ---
Name: Charlie Brown
Age: 28
Gender: Non-binary
Salary: $55,000.00

--- Employee Details (ID: E004) ---
Name: Dana White
Age: 42
Gender: Female
Salary: $90,000.00

Updating salary for E001...
Employee ID E001's salary updated to $62,000.00.

Displaying updated details for E001:

--- Employee Details (ID: E001) ---
Name: Alice Smith
Age: 30
Gender: Female
Salary: $62,000.00

--- Salary Summary ---
Total Salary: $282,000.00
Average Salary: $70,500.00
(282000.0, 70500.0)
```

P Create a Python program that processes employee details such as name, age, gender, and salary.

Certainly! I will create a Python program to manage employee details such as name, age, gender, and salary. I'll define an `Employee` class and an `EmployeeManagementSystem` class to handle various operations like adding employees, displaying their details, and calculating salary summaries. Then, I'll show you how it works with some example data.

The Python program to process employee details has been successfully created and demonstrated!

We defined an `Employee` class to hold individual employee data (ID,

Visualize the total and average salary data
Add functionality to save employee data to a CSV file
Sort employees by salary or age
What can I help you build?
+
Gemini 2.5 Flash

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

Code Explanation:

The program collects employee details using input statements and displays them in an organized manner.

It processes basic personal and salary information as provided by the user.

Output:

Employee details such as name, age, gender, and salary are displayed correctly based on user input.

Observation:

Although the code works correctly, including gender-specific variables can lead to assumptions.

AI-generated code should be reviewed to ensure inclusiveness and ethical design.

Conclusion:

This lab highlights the importance of reviewing AI-generated code to ensure privacy, fairness, transparency, and inclusiveness in responsible software development.