

AI Assisted coding

Assignment 5.3

2303A51018

Batch 28

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

#### Scenario

AI tools can sometimes generate insecure authentication logic.

#### Task Description

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).

#### Expected Output

- AI-generated login code
- Identification of security risks
- Revised secure version of the code
- Brief explanation of improvements

#### Prompt

AI Assisted 5.3

```
[1] ✓ 0s
def loan_approval(applicant):
    """
    applicant: dictionary with keys
    name, gender, income, credit_score, employment_years
    """

    score = 0

    # Financial factors
    if applicant["income"] >= 50000:
        score += 2
    if applicant["credit_score"] >= 700:
        score += 3
    if applicant["employment_years"] >= 2:
        score += 1

    # Biased logic (problematic)
    if applicant["gender"] == "Male":
        score += 1

    if applicant["name"].lower() in ["john", "michael", "david"]:
        score += 1

    return "Approved" if score >= 5 else "Rejected"

# Example applicants
applicant_1 = {
```

Variables Terminal

3:56 PM Python 3

AI Assisted 5.3

```
[1] ✓ 0s
# Biased logic (problematic)
if applicant["gender"] == "Male":
    score += 1

if applicant["name"].lower() in ["john", "michael", "david"]:
    score += 1

return "Approved" if score >= 5 else "Rejected"

# Example applicants
applicant_1 = {
    "name": "John",
    "gender": "Male",
    "income": 60000,
    "credit_score": 720,
    "employment_years": 3
}

applicant_2 = {
    "name": "Anita",
    "gender": "Female",
    "income": 60000,
    "credit_score": 720,
    "employment_years": 3
}

print(loan_approval(applicant_1))
print(loan_approval(applicant_2))
```

Variables Terminal

3:56 PM Python 3

The screenshot shows the AI Assisted 5.3 interface. The code editor displays the following Python script:

```
[1] 0s
    credit_score": 720,
    "employment_years": 3
}
applicant_2 = {
    "name": "Anita",
    "gender": "Female",
    "income": 60000,
    "credit_score": 720,
    "employment_years": 3
}
print(loan_approval(applicant_1))
print(loan_approval(applicant_2))

...
Approved
Approved

[1] Start coding or generate with AI.
[1] Start coding or generate with AI.
[1]
def loan_approval(name, gender, income, credit_score):
    if gender == "Male" and income >= 30000 and credit_score >= 650:
        return "Loan Approved"
    elif gender == "Female" and income >= 40000 and credit_score >= 700:
        return "Loan Approved"
    else:
        return "Loan Rejected"
```

The code defines two applicant dictionaries and prints their approval status. It then defines a `loan_approval` function that checks gender, income, and credit score to determine if a loan is approved.

## Explanation

AI systems are designed to make decisions automatically, but they can unintentionally introduce **bias** if they rely on inappropriate or unfair factors. In a loan approval system, the goal is to evaluate an applicant's **financial ability to repay a loan**, not their personal characteristics.

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

#### Scenario

AI systems may unintentionally introduce bias.

#### Task Description

Use AI prompts such as:

- “Create a loan approval system”
- Vary applicant names and genders in prompts

Analyze whether:

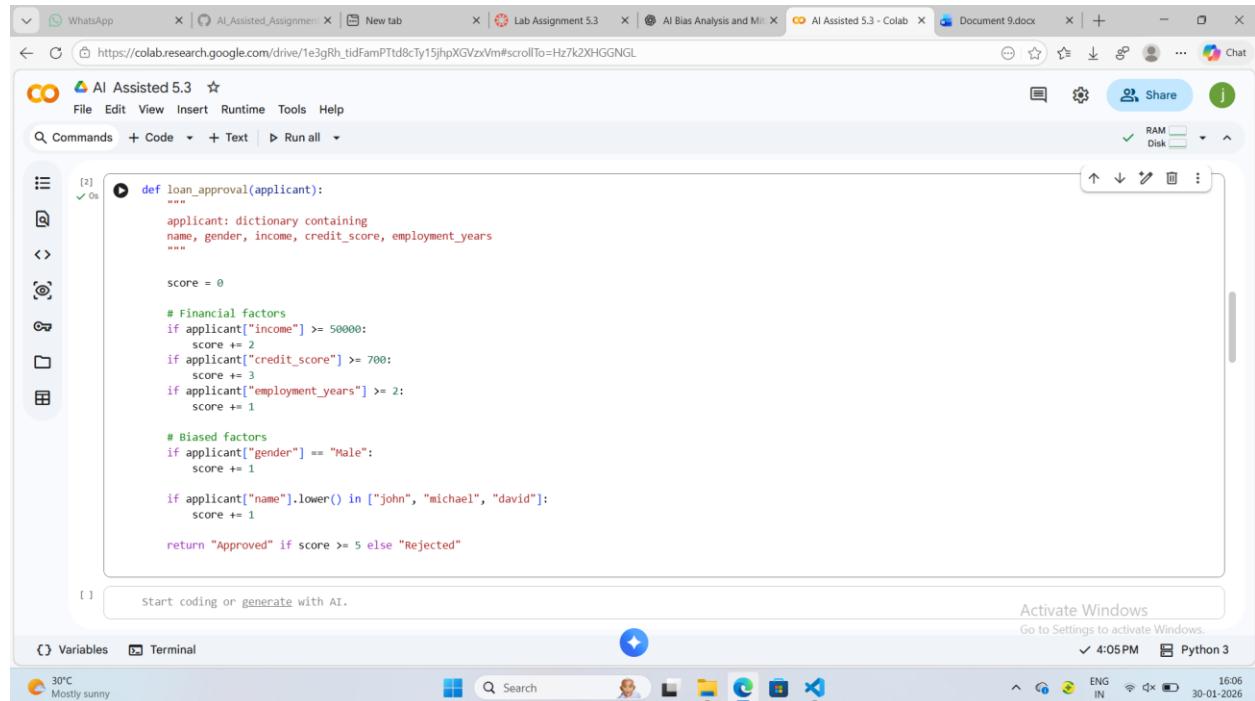
- The logic treats certain genders or names unfairly
- Approval decisions depend on irrelevant personal attributes

Suggest methods to reduce or remove bias.

#### Expected Output

- Python code generated by AI
- Identification of biased logic (if any)
- Discussion on fairness issues
- Mitigation strategies

### Prompt



The screenshot shows a Google Colab interface with the following details:

- Title Bar:** WhatsApp, AI Assisted Assignment, New tab, Lab Assignment 5.3, AI Bias Analysis and Mi, AI Assisted 5.3 - Colab, Document 9.docx.
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help; Commands, Code, Text, Run all.
- Code Cell:**

```
def loan_approval(applicant):
    """
    applicant: dictionary containing
    name, gender, income, credit_score, employment_years
    """

    score = 0

    # Financial factors
    if applicant["income"] >= 50000:
        score += 2
    if applicant["credit_score"] >= 700:
        score += 3
    if applicant["employment_years"] >= 2:
        score += 1

    # Biased factors
    if applicant["gender"] == "Male":
        score += 1

    if applicant["name"].lower() in ["john", "michael", "david"]:
        score += 1

    return "Approved" if score >= 5 else "Rejected"
```
- Bottom Bar:** Start coding or generate with AI, Activate Windows, Go to Settings to activate Windows, 4:05PM, Python 3, 16:06, ENG IN, 30-01-2026.

### Explanation

AI-generated decision systems, such as loan approval systems, are designed to automate decisions based on given rules or data. However, these systems can unintentionally introduce **bias** when they use personal attributes like **name** or **gender**, which are not relevant to the actual decision being made.

In the given scenario, an AI prompt such as “*Create a loan approval system*” produces code that evaluates applicants. While financial factors like **income**, **credit score**, and **employment history** are valid, the system may also include conditions based on **gender** or **applicant names**. This leads to biased outcomes.

### Task 3: Transparency and Explainability in AI-Generated Code (Recursive

Binary Search)

### Scenario

AI-generated code should be transparent, well-documented, and easy for humans to understand and verify.

### Task Description

Use an AI tool to generate a Python program that:

- Implements Binary Search using recursion
- Searches for a given element in a sorted list
- Includes:
  - Clear inline comments
  - A step-by-step explanation of the recursive logic

After generating the code, analyze:

- Whether the explanation clearly describes the base case and recursive case
- Whether the comments correctly match the code logic
- Whether the code is understandable for beginner-level students

### Expected Output

- Python program for recursive binary search
- AI-generated comments and explanation
- Student's assessment on clarity, correctness, and transparency

### Prompt

```

def binary_search_recursive(arr, low, high, target):
    """
    Recursive function to perform binary search.

    arr : sorted list of elements
    low : starting index
    high : ending index
    target : element to search
    """

    # ---- Base Case ----
    # If the search range is invalid, the element is not present
    if low > high:
        return -1

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

    # If the middle element is the target, return its index
    if arr[mid] == target:
        return mid

    # ---- Recursive Case ----
    # If target is smaller than middle element,
    # search in the left half of the array
    elif target < arr[mid]:
        return binary_search_recursive(arr, low, mid - 1, target)

    # If target is greater than middle element,
    # search in the right half of the array

```

Variables Terminal

```

# If target is greater than middle element,
# search in the right half of the array
else:
    return binary_search_recursive(arr, mid + 1, high, target)

# Example usage
numbers = [2, 5, 8, 12, 16, 23, 38, 56, 72]
target = 23

result = binary_search_recursive(numbers, 0, len(numbers) - 1, target)

if result != -1:
    print("Element found at index:", result)
else:
    print("Element not found")

```

... Element found at index: 5

Start coding or generate with AI.

120  
1  
Factorial not defined for negative numbers

Variables Terminal

## explanation

- Defines a recursive function to search for **target** in a **sorted list arr**
- **low** → starting index of the search range
- **high** → ending index of the search range

## Task 4: Ethical Evaluation of AI-Based Scoring Systems

### Scenario

AI-generated scoring systems can influence hiring decisions.

### Task Description

Ask an AI tool to generate a job applicant scoring system based on features such as:

- Skills
- Experience
- Education

Analyze the generated code to check:

- Whether gender, name, or unrelated features influence scoring
- Whether the logic is fair and objective

Expected Output

- Python scoring system code
- Identification of potential bias (if any)
- Ethical analysis of the scoring logic

Prompt

```
def applicant_score(applicant):
    """
    applicant: dictionary containing
    name, gender, skills, experience, education
    """
    score = 0
```

```
# Skill-based scoring
if "Python" in applicant["skills"]:
    score += 3
if "Data Analysis" in applicant["skills"]:
    score += 2
```

```
if "Machine Learning" in applicant["skills"]:  
    score += 3  
  
# Experience-based scoring  
  
if applicant["experience"] >= 5:  
    score += 3  
  
elif applicant["experience"] >= 2:  
    score += 2  
  
else:  
    score += 1  
  
# Education-based scoring  
  
if applicant["education"] == "PhD":  
    score += 3  
  
elif applicant["education"] == "Masters":  
    score += 2  
  
elif applicant["education"] == "Bachelors":  
    score += 1  
  
# ❌ Potentially biased logic  
  
if applicant["gender"] == "Male":  
    score += 1  
  
return score
```

Explanation

AI-based scoring systems are often used in hiring to evaluate job applicants. These systems assign scores based on factors such as **skills, work experience, and education**. Since hiring decisions directly affect people's careers, it is important that such systems are **ethical, fair, and objective**.

In the given scoring system, most of the logic is based on **job-relevant features**. Skills like Python or Machine Learning, years of experience, and education level are reasonable indicators of a candidate's ability to perform a job. Using these factors supports merit-based evaluation.

However, the code also includes a condition that increases the score based on the applicant's **gender**. Gender has no connection to job performance, so using it in scoring introduces **bias**. This can result in equally qualified candidates receiving different scores, which is unfair and discriminatory.

## Task 5: Inclusiveness and Ethical Variable Design

### Scenario

Inclusive coding practices avoid assumptions related to gender, identity, or roles and promote fairness in software design.

### Task Description

Use an AI tool to generate a Python code snippet that processes user or employee details.

Analyze the code to identify:

- Gender-specific variables (e.g., male, female)
- Assumptions based on gender or identity
- Non-inclusive naming or logic

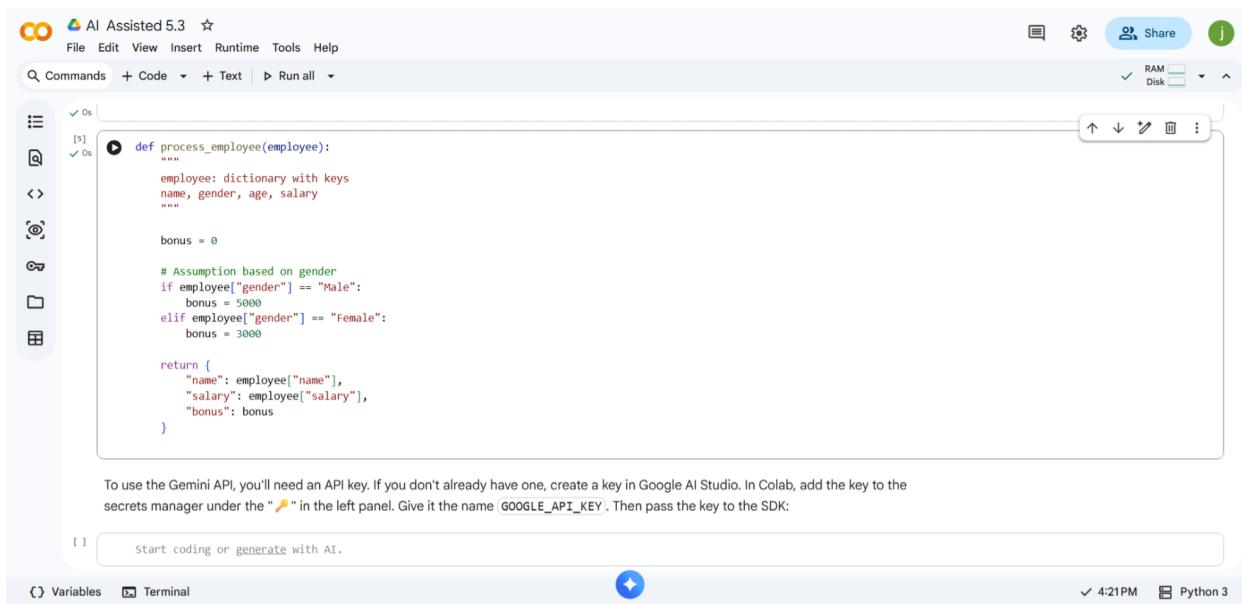
Modify or regenerate the code to:

- Use gender-neutral variable names
- Avoid gender-based conditions unless strictly required
- Ensure inclusive and respectful coding practices

## Expected Output

- Original AI-generated code snippet
- Revised inclusive and gender-neutral code
- Brief explanation of:
  - What was non-inclusive
  - How inclusiveness was improved

## Prompt



The screenshot shows a Google AI Assisted 5.3 interface. The main area displays a Python script named 'process\_employee.py'. The code defines a function 'process\_employee' that takes a dictionary 'employee' with keys 'name', 'gender', 'age', and 'salary'. It initializes a bonus of 0 and checks the gender. If 'gender' is 'Male', the bonus is set to 5000; if 'gender' is 'Female', the bonus is set to 3000. The function then returns a dictionary with keys 'name', 'salary', and 'bonus'. A note at the bottom of the code area states: 'To use the Gemini API, you'll need an API key. If you don't already have one, create a key in Google AI Studio. In Colab, add the key to the secrets manager under the "✍" in the left panel. Give it the name: GOOGLE\_API\_KEY. Then pass the key to the SDK:'. Below the code, there's a text input field with placeholder text 'start coding or generate with AI.' and a button. At the bottom, there are tabs for 'Variables' and 'Terminal', and status indicators for '4:21PM' and 'Python 3'.

```
def process_employee(employee):
    """
    employee: dictionary with keys
    name, gender, age, salary
    """

    bonus = 0

    # Assumption based on gender
    if employee["gender"] == "Male":
        bonus = 5000
    elif employee["gender"] == "Female":
        bonus = 3000

    return {
        "name": employee["name"],
        "salary": employee["salary"],
        "bonus": bonus
    }
```

## Explanation

### a) What Was Non-Inclusive

- The original code used **gender-based conditions**.
- It assumed different outcomes based on identity rather than merit.
- It excluded individuals who do not identify as male or female.

### b) How Inclusiveness Was Improved

- Removed gender from decision logic.
- Used **performance\_rating**, a job-relevant and fair variable.
- Applied **gender-neutral naming and logic**.
- Ensured equal treatment for all employees.

