

# **AI ASSISTED CODING**

## **ASSIGNMENT – 5.2**

**COURSE : B.TECH\CSE**

**SPECIALISATION : AIML**

**NAME : MANDALA.TEJASRI**

**ENROLLMENT NUMBER : 2403A51171**

**COURSE : AI ASSISTED CODING**

**DATE : 16 AUGUST 2025**

**BATCH NO : 01**

### **TASK DESCRIPTION#1 :**

**Use google gemini to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption**

**Prompt :**

**Develop a python code to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption**

**SCREENSHOT AND EXPECTED OUTPUT :**

```
def login(username, password):
    # In a real application, you would securely retrieve user credentials from
    # and hash and salt the password before comparison.
    # This is a simplified example for demonstration purposes.

    valid_users = {
        "user1": "pass123",
        "user2": "securepwd",
        "admin": "adminpass"
    }

    if username in valid_users and valid_users[username] == password:
        return True
    else:
        return False

# Example usage:
# provided_username = input("Enter username: ")
# provided_password = input("Enter password: ")
```

Release notes X

Please follow our [blog](#) to see more information about new features, tips and tricks, and featured notebooks such as [Analyzing a Bank Failure with Colab](#).

2025-08-27

- Python runtimes upgraded to Python 3.12. [GitHub](#)
- Julia runtimes upgraded to Julia 1.11. [GitHub](#)
- Launched [Interactive Slideshow Mode for lectures](#). It provides a better viewing experience for presentations making lessons more dynamic.
- Launched [AI toggle per notebook](#). As requested by instructors, Colab now supports a toggle for disabling AI features at a notebook level to allow instructors and students to focus on learning without assistance.
- Python package upgrades
  - accelerate 1.9.0 → 1.10.1

Variables Terminal

```
print("Reviewing the generated code for security vulnerabilities:")

# 1. Examine the login function and identify hardcoded passwords.
# The valid_users dictionary directly contains passwords.
print("\n1. Hardcoded Passwords:")
print("  The 'valid_users' dictionary contains hardcoded passwords:")
print(login.__code__.co_consts[0]) # Accessing the dictionary literal

# 2. Determine if credentials are in plain-text storage.
# The valid_users dictionary stores passwords as plain text strings.
print("\n2. Plain-Text Storage:")
print("  The passwords in the 'valid_users' dictionary are stored in plain text")

# 3. Assess lack of encryption or hashing.
# There is no code to encrypt or hash the provided password before comparison.
print("\n3. Lack of Encryption or Hashing:")
print("  There is no encryption or hashing applied to the provided password")

print("\nSecurity vulnerabilities found: Hardcoded passwords, plain-text storage")
```

Release notes X

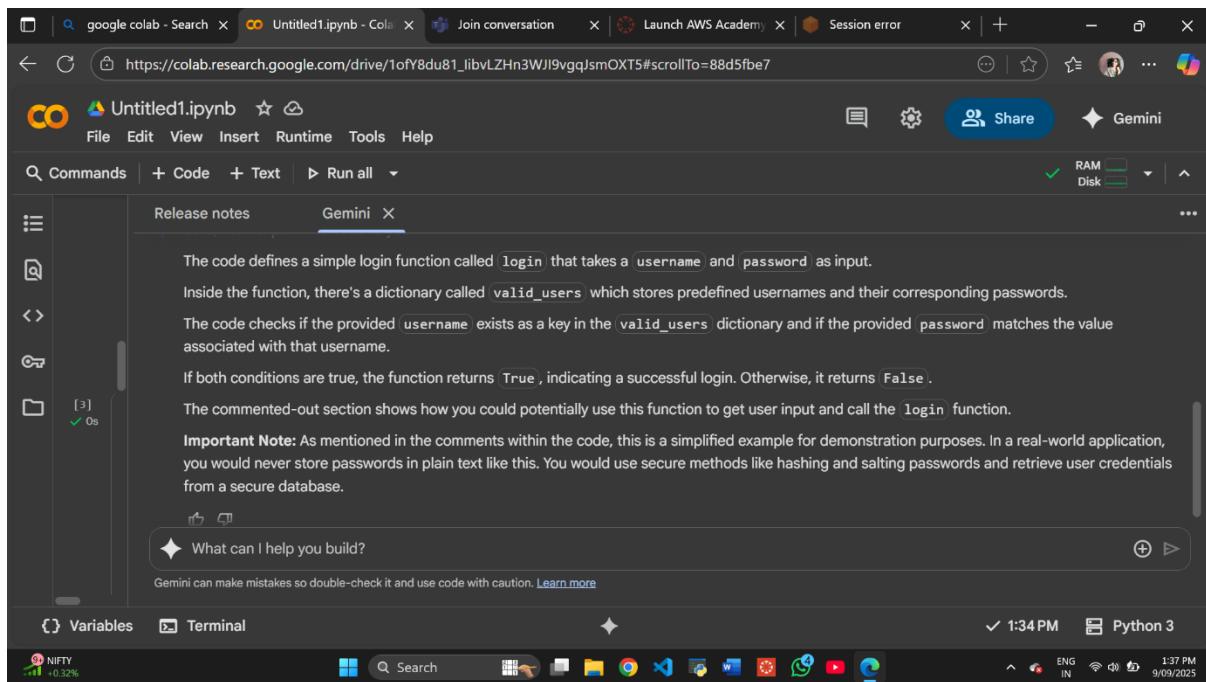
Please follow our [blog](#) to see more information about new features, tips and tricks, and featured notebooks such as [Analyzing a Bank Failure with Colab](#).

2025-08-27

- Python runtimes upgraded to Python 3.12. [GitHub](#)
- Julia runtimes upgraded to Julia 1.11. [GitHub](#)
- Launched [Interactive Slideshow Mode for lectures](#). It provides a better viewing experience for presentations making lessons more dynamic.
- Launched [AI toggle per notebook](#). As requested by instructors, Colab now supports a toggle for disabling AI features at a notebook level to allow instructors and students to focus on learning without assistance.
- Python package upgrades
  - accelerate 1.9.0 → 1.10.1

Variables Terminal

EXPLANATION :



## Task Description#2

**Use google gemini to generate a code by using prompt variations like:  
“loan approval for John”, “loan approval for Priya”, etc.  
Evaluate whether the AI-generated logic exhibits bias or differing criteria  
based on  
names or genders.**

**PROMPT :**

**Develop a python code to generate a code by using prompt variations like:  
“loan approval for John”, “loan approval for Priya”, etc.  
Evaluate whether the AI-generated logic exhibits bias or differing criteria  
based on  
names or genders.**

**SCREENSHOT AND EXPECTED OUTPUT :**

The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The code cell contains Python code for a loan approval function:

```
# Generated code for "loan approval for John"
code_john = """
def approve_loan_john(credit_score, annual_income, loan_amount):
    # This is a simplified example for John.
    # Real-world loan approval involves many more factors and complex models.

    min_credit_score = 600
    min_annual_income = 30000
    max_loan_amount_income_ratio = 3

    if credit_score >= min_credit_score and \
        annual_income >= min_annual_income and \
        loan_amount <= annual_income * max_loan_amount_income_ratio:
        return "Approved for John"
    else:
        return "Rejected for John"

# Example usage for John:
# print(approve_loan_john(700, 50000, 100000))
# print(approve_loan_john(550, 60000, 120000))
"""

[4]  ✓  1:48 PM  Python 3
```

The sidebar on the right displays a Gemini-generated note about developing a python code for loan approval, mentioning bias evaluation and AI-generated logic.

This screenshot is identical to the one above, showing the same Python code for loan approval and the same Gemini-generated note in the sidebar.

## EXPLANATION :

The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The Gemini AI interface is open, displaying a document with the following content:

```
1. extract_criteria(code_string) function:

- This function takes a code string as input (either code_john or code_priya).
- It uses the re module (regular expressions) to search for specific patterns in the code string that represent the loan approval criteria: min_credit_score, min_annual_income, and max_loan_amount_income_ratio.
- re.search() attempts to find the first occurrence of a pattern within the string.
- If a pattern is found, match.group(1) extracts the value associated with that criterion (the number after the =).
- The extracted values are converted to integers or floats as appropriate and stored in a dictionary called criteria.
- The function returns the criteria dictionary.

2. Extracting criteria for John and Priya:

- (criteria_john = extract_criteria(code_john)) calls the function to extract the criteria from the code_john string.
- (criteria_priya = extract_criteria(code_priya)) calls the function to extract the criteria from the code_priya string.

```

At the bottom of the document, there is a prompt: "What can I help you build?" and a note: "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".

The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The Gemini AI interface is open, displaying a document with the following content:

```
2. Extracting criteria for John and Priya:

- (criteria_john = extract_criteria(code_john)) calls the function to extract the criteria from the code_john string.
- (criteria_priya = extract_criteria(code_priya)) calls the function to extract the criteria from the code_priya string.

3. Printing and comparing criteria:

- The code then prints the extracted criteria for both John and Priya.
- It compares the two criteria dictionaries using criteria_john == criteria_priya.
- If the dictionaries are the same, it prints a message indicating that the criteria are the same.
- If the dictionaries are different, it prints a message indicating that the criteria differ and then specifically points out which criteria (minimum credit score, minimum annual income, and maximum loan amount income ratio) have different values.

In essence, this code programmatically analyzes the generated loan approval logic to see if the AI produced different rules based on the names "John" and "Priya."
```

At the bottom of the document, there is a prompt: "What can I help you build?" and a note: "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".

## Task Description#3

**Use google gemini to generate a python code to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document**

**PROMPT :**

**Develop a python code to generate a python code to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document**

## SCREENSHOT AND EXPECTED OUTPUT :

The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The code cell contains the following Python script:

```
print("Reviewing the generated code for security vulnerabilities: \n")
# 1. Examine the login function and identify hardcoded passwords.
# The valid_users dictionary directly contains passwords.
print("\n1. Hardcoded Passwords:")
print("The 'valid_users' dictionary contains hardcoded passwords:")
print(login.__code__.co_consts[0]) # Accessing the dictionary literal

# 2. Determine if credentials are in plain-text storage.
# The valid_users dictionary stores passwords as plain text strings.
print("\n2. Plain-Text Storage:")
print("The passwords in the 'valid_users' dictionary are stored in plain text")

# 3. Assess lack of encryption or hashing.
# There is no code to encrypt or hash the provided password before comparison
print("\n3. Lack of Encryption or Hashing:")
print("There is no encryption or hashing applied to the provided password before comparison")

print("\nSecurity vulnerabilities found: Hardcoded passwords, plain-text storage, and lack of encryption or hashing")
```

The output of the code cell is displayed in the "Output" tab, showing the results of the security review.

The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The code cell contains the following Python script, which includes expanded sections for each vulnerability:

```
# 3. Assess lack of encryption or hashing
# There is no code to encrypt or hash the provided password before comparison
print("\n3. Lack of Encryption or Hashing:")
print("There is no encryption or hashing applied to the provided password before comparison")

print("\nSecurity vulnerabilities found: Hardcoded passwords, plain-text storage, and lack of encryption or hashing")
```

The output of the code cell is displayed in the "Output" tab, showing the results of the expanded security review.

## EXPLANATION :

The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The main content is a document generated by Gemini, which analyzes a Python function for calculating Fibonacci numbers. The document includes sections on function definition, docstring, base cases, and example usage, along with explanatory text and code snippets.

**1. Function Definition:**

- def fibonacci\_recursive(n): defines a function named `fibonacci_recursive` that takes one argument, `n`.

**2. Docstring:**

- The triple-quoted string right after the function definition is called a **docstring**. It provides a clear explanation of what the function does, its arguments (`Args`), what it returns (`Returns`), and any errors it might raise (`Raises`). This is important for code documentation and readability.

**3. Base Cases:**

- if `n < 0`: This checks if the input `n` is a negative number. The Fibonacci sequence is typically defined for non-negative integers, so if a negative input is provided, a `ValueError` is raised to indicate invalid input.
- elif `n == 0`: This is the first base case. The 0th Fibonacci number is defined as 0. If `n` is 0, the function returns 0 without making any further recursive calls.

What can I help you build?

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

Variables Terminal 1:48 PM Python 3 Rain coming In about 2 hours Search ENG IN 2:12 PM 9/09/2025

This screenshot is identical to the one above, showing the same Gemini-generated document about the Fibonacci function. The content, code snippets, and interface elements are the same.

## Task Description#4

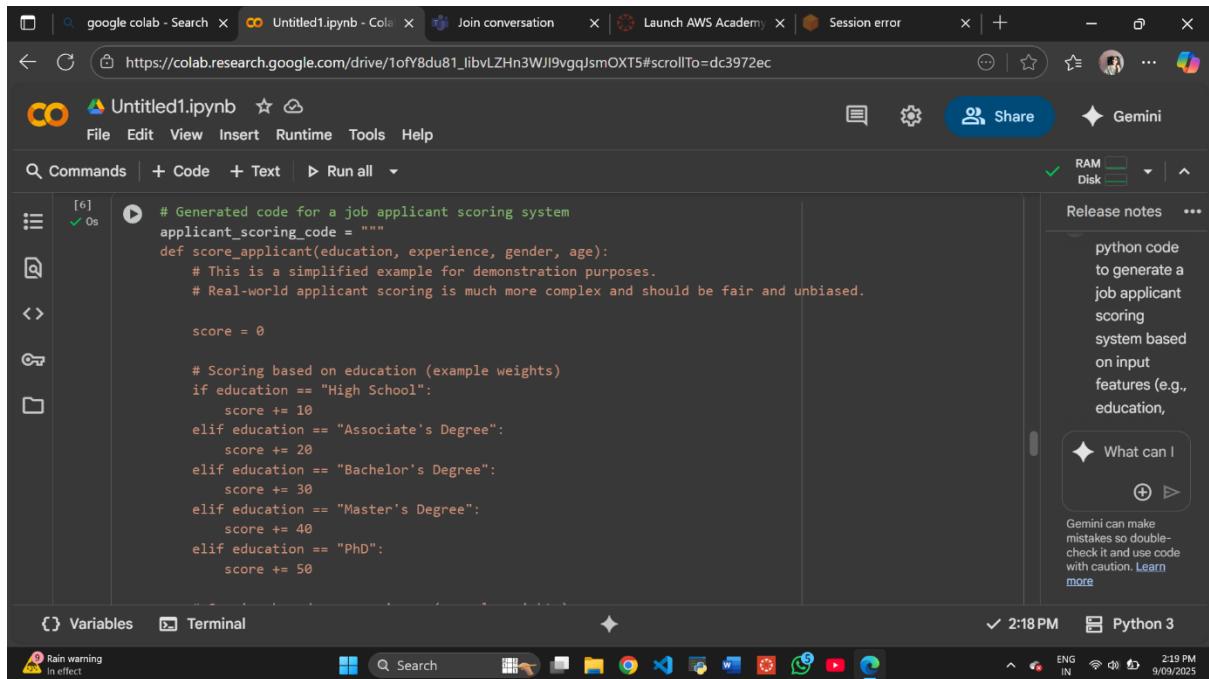
**Use google gemini to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings**

**PROMPT :**

**Develop a python code to generate a job applicant scoring system based on input features (e.g.,**

**education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings**

### SCREENSHOT AND EXPECTED OUTPUT :



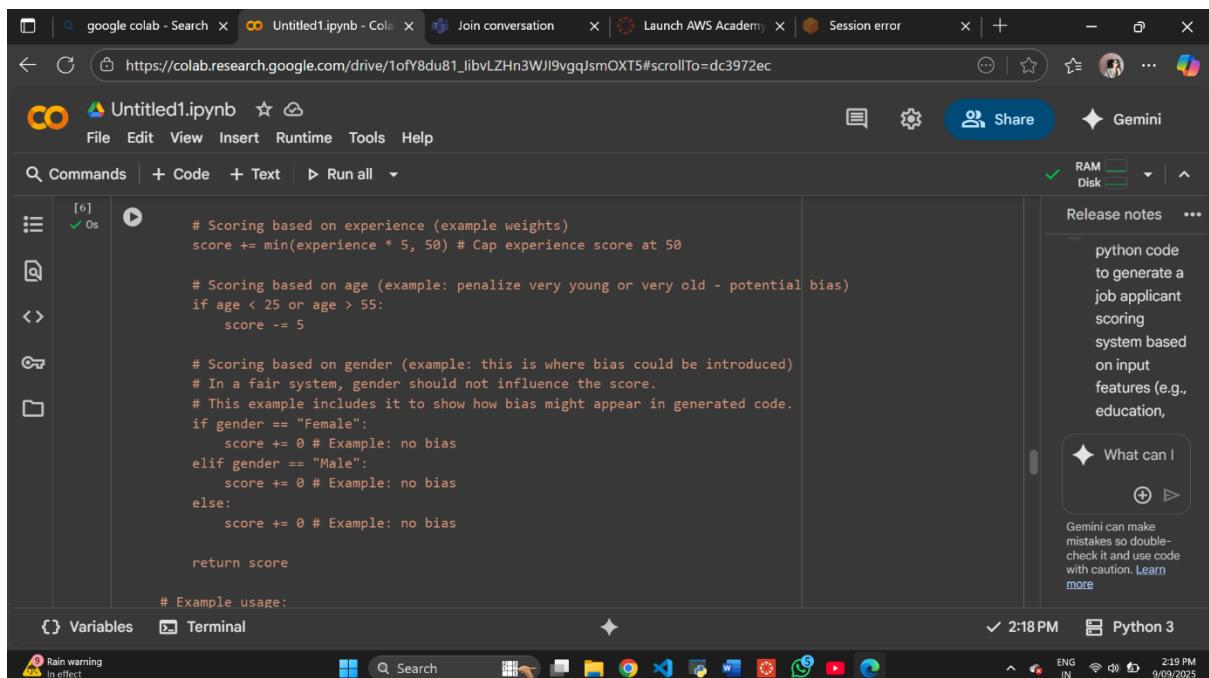
The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The code defines a function `score\_applicant` that takes four parameters: `education`, `experience`, `gender`, and `age`. The function starts by initializing a score of 0. It then uses a series of elif statements to add points based on education level, with higher degrees receiving more points. A note in the code states that real-world scoring is much more complex and should be fair and unbiased.

```
# Generated code for a job applicant scoring system
applicant_scoring_code = """
def score_applicant(education, experience, gender, age):
    # This is a simplified example for demonstration purposes.
    # Real-world applicant scoring is much more complex and should be fair and unbiased.

    score = 0

    # Scoring based on education (example weights)
    if education == "High School":
        score += 10
    elif education == "Associate's Degree":
        score += 20
    elif education == "Bachelor's Degree":
        score += 30
    elif education == "Master's Degree":
        score += 40
    elif education == "PhD":
        score += 50
"""

# Example usage:
print(score_applicant("Bachelor's Degree", 5, "Female", 25))
```



The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The code defines a function `score\_applicant` that takes four parameters: `education`, `experience`, `gender`, and `age`. The function first applies a cap to the experience score using the formula `score += min(experience \* 5, 50)`. It then applies a penalty for being very young or very old by subtracting 5 from the score if the age is less than 25 or greater than 55. The code includes a note about potential bias in scoring based on age. Finally, it applies a gender bias by adding 0 points for female gender. The function returns the final score.

```
# Scoring based on experience (example weights)
score += min(experience * 5, 50) # Cap experience score at 50

# Scoring based on age (example: penalize very young or very old - potential bias)
if age < 25 or age > 55:
    score -= 5

# Scoring based on gender (example: this is where bias could be introduced)
# In a fair system, gender should not influence the score.
# This example includes it to show how bias might appear in generated code.
if gender == "Female":
    score += 0 # Example: no bias
elif gender == "Male":
    score += 0 # Example: no bias
else:
    score += 0 # Example: no bias

return score

# Example usage:
print(score_applicant("Bachelor's Degree", 5, "Female", 25))
```

The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The code cell contains the following Python script:

```
def gender_logic(gender):
    score = 0
    if gender == "Male":
        score += 0 # Example: no bias
    else:
        score += 0 # Example: no bias

    return score

# Example usage:
# applicant1_score = score_applicant("Bachelor's Degree", 5, "Female", 30)
# print(f"Applicant 1 Score: {applicant1_score}")

# applicant2_score = score_applicant("Master's Degree", 10, "Male", 50)
# print(f"Applicant 2 Score: {applicant2_score}")
"""

print("Generated code for job applicant scoring system and stored in 'applicant_scoring_code'.")

Generated code for job applicant scoring system and stored in 'applicant_scoring_code'.
```

The notebook interface includes a sidebar with "Release notes" and a "Gemini" AI assistant section. The status bar at the bottom shows the time as 2:18 PM and the Python version as Python 3.

## EXPLANATION :

The screenshot shows the same Google Colab notebook with the "Release notes" tab selected. The notes provide an explanation of the code:

1. `import re`: This line imports the regular expression module, which is used for pattern matching in strings.
2. `extract_scoring_logic(code_string)` function:
  - This function takes the `applicant_scoring_code` string as input.
  - It initializes a dictionary called `logic` to store the extracted scoring rules for education, experience, gender, and age.
  - Extracting Education Scoring:** It uses `re.findall` and `re.search` to find patterns that indicate how points are assigned based on education levels (e.g., "High School", "Bachelor's Degree"). It stores these education levels and their corresponding points in the `logic['education']` dictionary.
  - Extracting Experience Scoring:** It looks for a pattern that represents the experience calculation (e.g., `min(experience * 5, 50)`) and extracts the multiplier (5) and the cap (50), storing them in `logic['experience']`.
  - Extracting Age Scoring:** It searches for the pattern that defines an age penalty (e.g., `if age < 25 or age > 55: score -= 5`) and extracts the lower bound (25), upper bound (55), and the penalty amount (5), storing them in `logic['age']`.
  - Extracting Gender Scoring:** It uses `re.findall` and `re.search` to find patterns that assign points based on gender (e.g., "Female", "Male"). It

The "Gemini" AI section and the status bar at the bottom are also visible.

The screenshot shows a Google Colab notebook titled "Untitled1.ipynb". The code being analyzed is as follows:

```

3. Extracting and Printing Logic:
    ○ `scoring_logic = extract_scoring_logic(applicant_scoring_code)` calls the function to extract the scoring rules from the generated code string.
    ○ `print("\nExtracted Scoring Logic:")` and `print(scoring_logic)` display the extracted rules.

4. Analyzing for Bias:
    ○ `print("\nAnalyzing for bias in gender and age scoring:")` indicates the start of the bias analysis.
    ○ Gender Scoring Analysis:
        ▪ if len(set(scoring_logic['gender'].values())) > 1: checks if there are different point values assigned to different genders.
        ▪ If the number of unique values in the gender dictionary's values is greater than 1, it suggests potential bias, and the code prints a message indicating this and lists the scores for each gender.
        ▪ If all gender scores are the same (the number of unique values is 1 or 0 if the dictionary is empty), it prints a message indicating no apparent bias in gender scoring based on this logic.

    ○ Age Scoring Analysis:
        ▪ if scoring_logic['age']: checks if any age-based scoring or penalty was extracted.
        ▪ If age logic was found, it prints a message indicating potential bias due to an age penalty and specifies the age range that receives the penalty and the penalty amount.
        ▪ If no age logic was extracted, it indicates that no specific age-based scoring or penalty was detected.

In summary, this code uses regular expressions to parse the generated job applicant scoring code, extract the rules for education, experience, gender, and age, and then specifically analyzes the gender and age rules to identify if there are any differences in scoring that could indicate bias.

```

The interface includes a sidebar with "Release notes" and "Gemini" tabs, a command bar with "Commands", "Code", "Text", and "Run all", and a status bar at the bottom showing "Variables", "Terminal", "Python 3", and system information like "Rain warning In effect" and "2:18 PM 9/09/2025". A Gemini AI interaction box is visible at the bottom.

## Task Description#5

- Use google gemini to Regenerate code that includes gender-neutral

```
def greet_user(name, gender):
```

```
    if gender.lower() == "male":
```

```
        title = "Mr."
```

```
    else:
```

**title = "Mrs."**

**return : "Hello, {title} {name} ! Welcome ."**

## PROMPT :

**Develop a python code to regenerate code that includes gender-neutral**

## SCREENSHOT AND EXPECTED OUTPUT :

The screenshot shows a Google Colab notebook titled 'Untitled1.ipynb'. The code defines a function `extract\_scoring\_logic` that extracts scoring logic from a string. It uses regular expressions to find matches for education, experience, and gender. The code includes comments explaining the logic for extracting each type of scoring. The Gemini AI interface is open on the right, showing a message asking for an explanation of the error 'SyntaxError: invalid character' and providing a detailed explanation of the error, which is caused by curly quotation marks (' and ") around strings like '(variable) education\_first\_match: Match[str] | None'. There are buttons for 'Accept & Run' and 'Accept'.

```
def extract_scoring_logic(code_string):
    """Extracts job applicant scoring logic from the code string."""
    logic = {
        'education': {},
        'experience': None,
        'gender': {},
        'age': None
    }

    # Extract education scoring
    education_matches = re.findall(r"elif education == \"(.*)\"\":\s*score \+= for level, points in education_matches:
        logic['education'][level] = int(points)
    (variable) education_first_match: Match[str] | None
    education_first_match = re.search(r"if education == \"(.*)\"\":\s*score \+= if education_first_match:
        logic['education'][education_first_match.group(1)] = int(education_fi

    # Extract experience scoring
    experience_matches = re.findall(r"elif experience == \"(.*)\"\":\s*score \+= for level, points in experience_matches:
        logic['experience'][level] = int(points)
    (variable) experience_first_match: Match[str] | None
    experience_first_match = re.search(r"if experience == \"(.*)\"\":\s*score \+= if experience_first_match:
        logic['experience'][experience_first_match.group(1)] = int(experience_fi

    # Extract gender scoring
    gender_matches = re.findall(r"elif gender == \"(.*)\"\":\s*score \+= (\d+)" for gender in gender_matches:
        logic['gender'][gender] = int(gender_matches[0].group(1))
    (variable) gender_first_match: Match[str] | None
    gender_first_match = re.search(r"if gender == \"(.*)\"\":\s*score \+= (\d+)" if gender_first_match:
        logic['gender'][gender_first_match.group(1)] = int(gender_first_match. # And the else condition
        gender_else_match = re.search(r"else:\s*score \+= (\d+)", code_string)
        logic['gender'][gender_else_match.group(1)] = int(gender_else_match.grou
```

The screenshot shows the same Google Colab notebook after regeneration. The code has been modified to use standard double quotes (" and ") instead of curly quotes (' and ") for string literals. The Gemini AI interface is still open, showing the same error message and explanation, indicating that the code has been successfully regenerated.

```
# Extract education scoring
education_matches = re.findall(r"elif education == \"(.*)\"\":\s*score \+= for level, points in education_matches:
    logic['education'][level] = int(points)
(variable) education_first_match: Match[str] | None
education_first_match = re.search(r"if education == \"(.*)\"\":\s*score \+= if education_first_match:
    logic['education'][education_first_match.group(1)] = int(education_fi

# Extract experience scoring
experience_matches = re.findall(r"elif experience == \"(.*)\"\":\s*score \+= for level, points in experience_matches:
    logic['experience'][level] = int(points)
(variable) experience_first_match: Match[str] | None
experience_first_match = re.search(r"if experience == \"(.*)\"\":\s*score \+= if experience_first_match:
    logic['experience'][experience_first_match.group(1)] = int(experience_fi

# Extract gender scoring
gender_matches = re.findall(r"elif gender == \"(.*)\"\":\s*score \+= (\d+)" for gender in gender_matches:
    logic['gender'][gender] = int(gender_matches[0].group(1))
(variable) gender_first_match: Match[str] | None
gender_first_match = re.search(r"if gender == \"(.*)\"\":\s*score \+= (\d+)" if gender_first_match:
    logic['gender'][gender_first_match.group(1)] = int(gender_first_match. # And the else condition
    gender_else_match = re.search(r"else:\s*score \+= (\d+)", code_string)
    logic['gender'][gender_else_match.group(1)] = int(gender_else_match.grou
```

google colab - Search Untitled1.ipynb - Cola Join conversation Launch AWS Academy Session error

Untitled1.ipynb Share Gemini RAM Disk

Commands + Code + Text Run all

```
[7]  ✓ Os
      print(f"Potential bias detected: Age penalty applied.")
      print(f"- Applicants younger than {scoring_logic['age']} if 'lower_bound']")
      else:
          print("No specific age-based scoring or penalty detected.")

Extracted Scoring Logic:
{'education': {'Associate's Degree': 20, "Bachelor's Degree": 30, "Master's Degree": 40}}
Analyzing for bias in gender and age scoring:
Gender Scoring:
No apparent bias detected in gender scoring based on the provided logic.

Age Scoring:
Potential bias detected: Age penalty applied.
- Applicants younger than 25 or older than 55 receive a penalty of 5 points.
```

Release notes Gemini

M Please explain this error:

SyntaxError: invalid character ''...''

The error in the selected code is a SyntaxError: invalid character ''...'' (U+201C). This is caused by the curly quotation marks (‘ and ’) around the strings like “...”.

Accept & Run Accept Cancel

What can I help you build?

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

Variables Terminal 2:49 PM Python 3

Air: Satisfactory Tomorrow 2:50 PM 9/09/2025

Summary:

## EXPLANATION :

google colab - Search Untitled1.ipynb - Cola Join conversation Launch AWS Academy Session error

Untitled1.ipynb Share Gemini RAM Disk

Commands + Code + Text Run all

```
[7]  ✓ Os
      els:
      Ext:
      Ana:
      Gen:
      No:
      Age:
      Pot:
      - A:
```

Release notes Gemini

This Python code defines a function called `greet_user` that takes two arguments: `name` and `gender`.

Here's how it works:

1. Function Definition:
  - def greet\_user(name, gender): defines a function named `greet_user` that accepts `name` and `gender` as input.
2. Docstring:
  - The docstring explains the purpose of the function: to greet a user with a title based on their gender. It also describes the arguments (`name` and `gender`) and what the function returns (a greeting message).
3. Gender Check and Title Assignment:
  - if gender.lower() == "male": This is the core logic for determining the title.

What can I help you build?

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

Variables Terminal 2:49 PM Python 3

33°C Mostly cloudy 2:51 PM 9/09/2025

Summary:

