SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**	
ProgramName:B. Tech	**Assignment Type: Lab**	**AcademicYear:**2025-2026
CourseCoordinatorName	Venkataramana Veeramsetty	
Instructor(s)Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	

CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week3 - Tuesday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	

AssignmentNumber:5.2(Present assignment number)/**24**(Total number of assignments)

	.Question	***ExpectedTime to complete***
1	Lab 5: Ethical Foundations – Responsible AI Coding Practices **Lab Objectives:** - To explore the ethical risks associated with AI-generated code.	Week3 - Wednesday

- To recognize issues related to security, bias, transparency, and copyright.
- To reflect on the responsibilities of developers when using AI tools in software development.
- To promote awareness of best practices for responsible and ethical AI coding.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Identify and avoid insecure coding patterns generated by AI tools.
- Detect and analyze potential bias or discriminatory logic in AI-generated outputs.
- Evaluate originality and licensing concerns in reused AI-generated code.
- Understand the importance of explainability and transparency in AI-assisted programming.
- Reflect on accountability and the human role in ethical AI coding practices..

Task Description#1 (Privacy and Data Security)

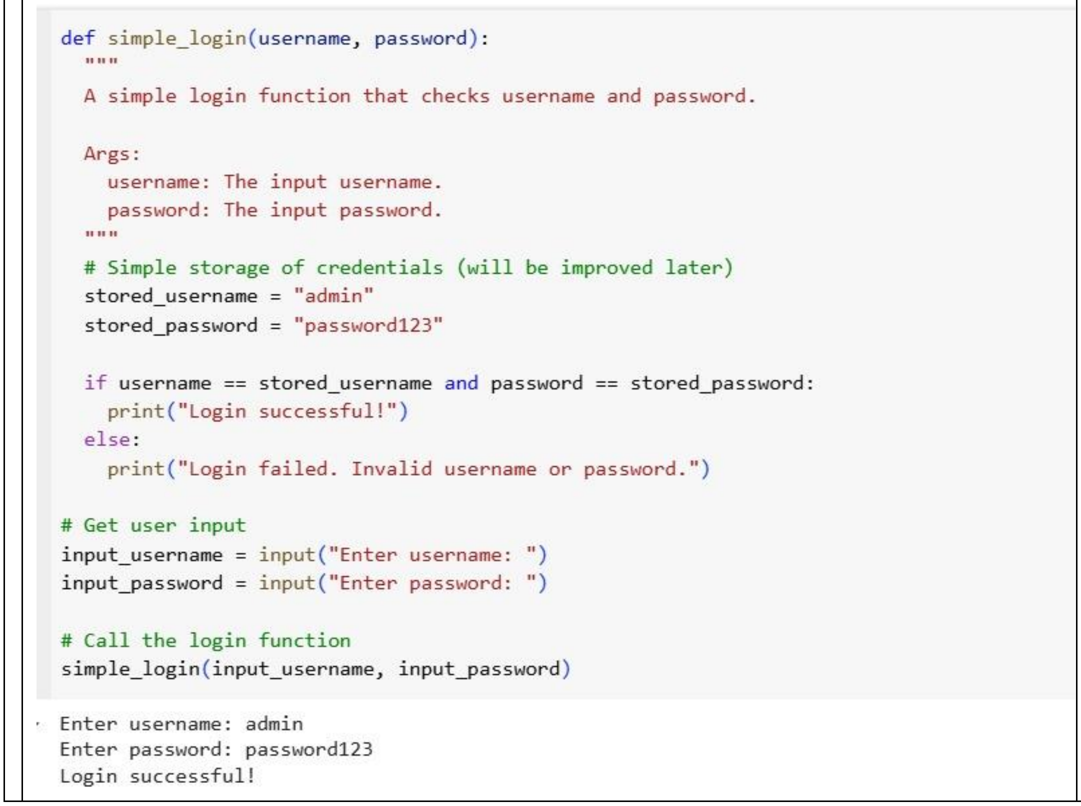
- Use an AI tool (e.g., Copilot, Gemini, Cursor) to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption.

Expected Output#1

- Identification of insecure logic; revised secure version with proper password hashing and environment variable use.

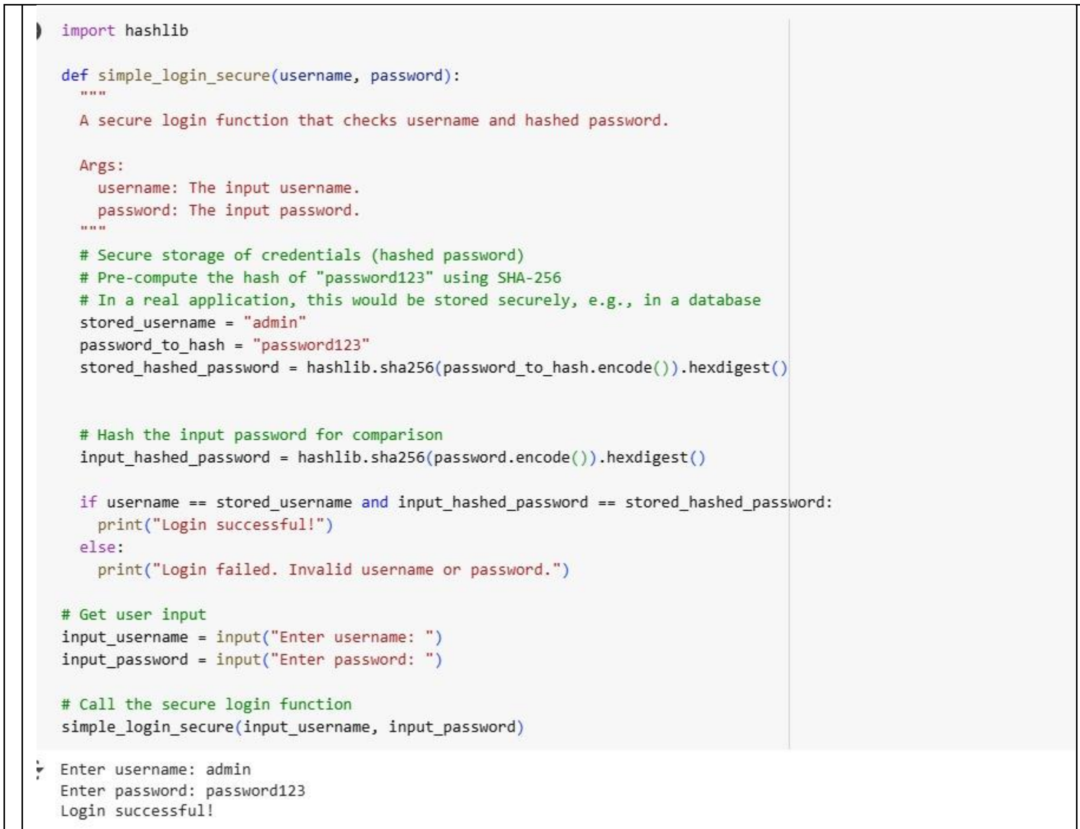
PROMPT :

code to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption.username and password will be given by the user.



```
def simple_login(username, password):
  """
  A simple login function that checks username and password.

  Args:
    username: The input username.
    password: The input password.
  """
  # Simple storage of credentials (will be improved later)
  stored_username = "admin"
  stored_password = "password123"

  if username == stored_username and password == stored_password:
    print("Login successful!")
  else:
    print("Login failed. Invalid username or password.")

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

# Call the login function
simple_login(input_username, input_password)
```

```
Enter username: admin
Enter password: password123
Login successful!
```



```python
import hashlib

def simple_login_secure(username, password):
  """
  A secure login function that checks username and hashed password.

  Args:
    username: The input username.
    password: The input password.
  """
  # Secure storage of credentials (hashed password)
  # Pre-compute the hash of "password123" using SHA-256
  # In a real application, this would be stored securely, e.g., in a database
  stored_username = "admin"
  password_to_hash = "password123"
  stored_hashed_password = hashlib.sha256(password_to_hash.encode()).hexdigest()


  # Hash the input password for comparison
  input_hashed_password = hashlib.sha256(password.encode()).hexdigest()

  if username == stored_username and input_hashed_password == stored_hashed_password:
    print("Login successful!")
  else:
    print("Login failed. Invalid username or password.")

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

# Call the secure login function
simple_login_secure(input_username, input_password)
```

```
Enter username: admin
Enter password: password123
Login successful!
```

OBSERVATION :

Initial code had hardcoded credentials and plain-text passwords.

Refined code uses password hashing (SHA-256) to improve security.

Hardcoded credentials still need to be addressed in the future.

Task Description#2 (Bias)

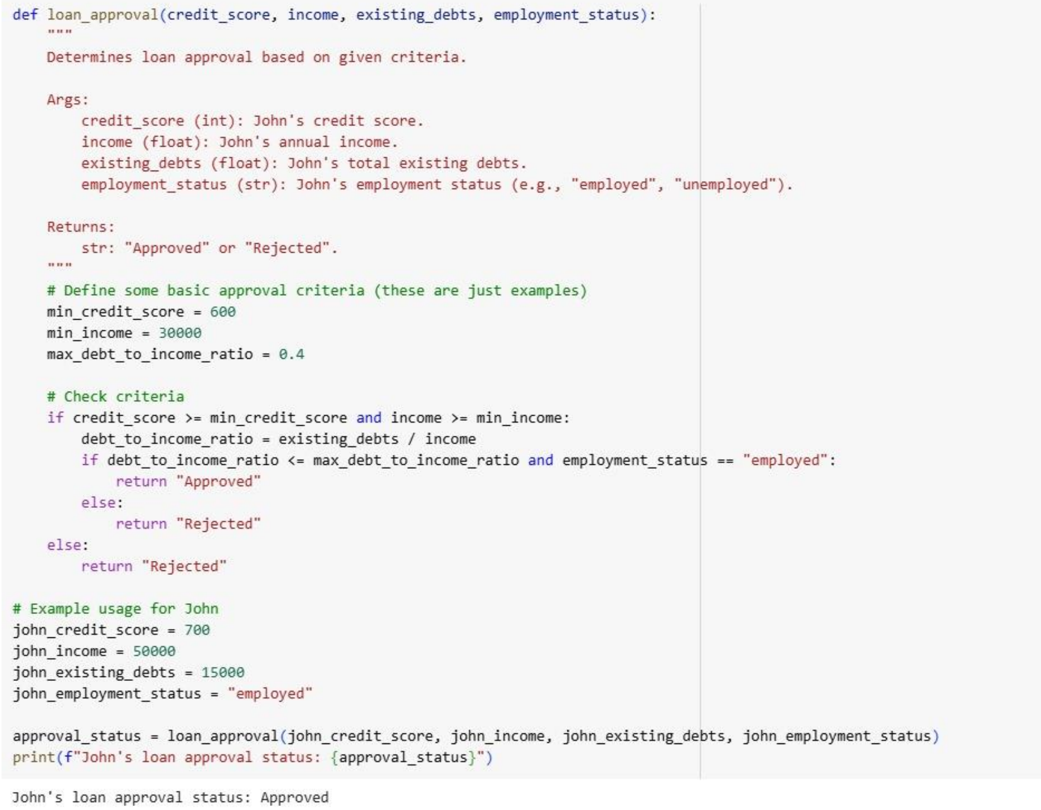
- Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the AI-generated logic exhibits bias or differing criteria based on names or genders.

Expected Output#2

- Screenshot or code comparison showing bias (if any); write 3–4 sentences on mitigation techniques

PROMPT :
WRITE A PYHTON CODE FOR LOAN APPROVAL FOR JOHN AND LOAN APPROVAL FOR PRIYA.

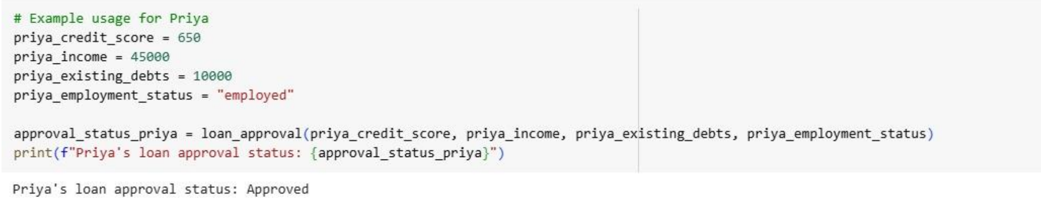
LOAN APPROVAL FOR JOHN:



```
def loan_approval(credit_score, income, existing_debts, employment_status):
    """
    Determines loan approval based on given criteria.

    Args:
        credit_score (int): John's credit score.
        income (float): John's annual income.
        existing_debts (float): John's total existing debts.
        employment_status (str): John's employment status (e.g., "employed", "unemployed").

    Returns:
        str: "Approved" or "Rejected".
    """
    # Define some basic approval criteria (these are just examples)
    min_credit_score = 600
    min_income = 30000
    max_debt_to_income_ratio = 0.4

    # Check criteria
    if credit_score >= min_credit_score and income >= min_income:
        debt_to_income_ratio = existing_debts / income
        if debt_to_income_ratio <= max_debt_to_income_ratio and employment_status == "employed":
            return "Approved"
        else:
            return "Rejected"
    else:
        return "Rejected"

# Example usage for John
john_credit_score = 700
john_income = 50000
john_existing_debts = 15000
john_employment_status = "employed"

approval_status = loan_approval(john_credit_score, john_income, john_existing_debts, john_employment_status)
print(f"John's loan approval status: {approval_status}")
```

```
John's loan approval status: Approved
```

LOAN APPROVAL FOR PRIYA:



```
# Example usage for Priya
priya_credit_score = 650
priya_income = 45000
priya_existing_debts = 10000
priya_employment_status = "employed"

approval_status_priya = loan_approval(priya_credit_score, priya_income, priya_existing_debts, priya_employment_status)
print(f"Priya's loan approval status: {approval_status_priya}")
```

```
Priya's loan approval status: Approved
```

OBSERVATION:

The code defines a function `loan_approval` that takes credit score, income, existing debts, and employment status as input and returns "Approved" or "Rejected" based on pre-defined criteria. The code then uses this function to determine and print the loan approval status for John and Priya with example values.

Task Description#3 (Transparency)

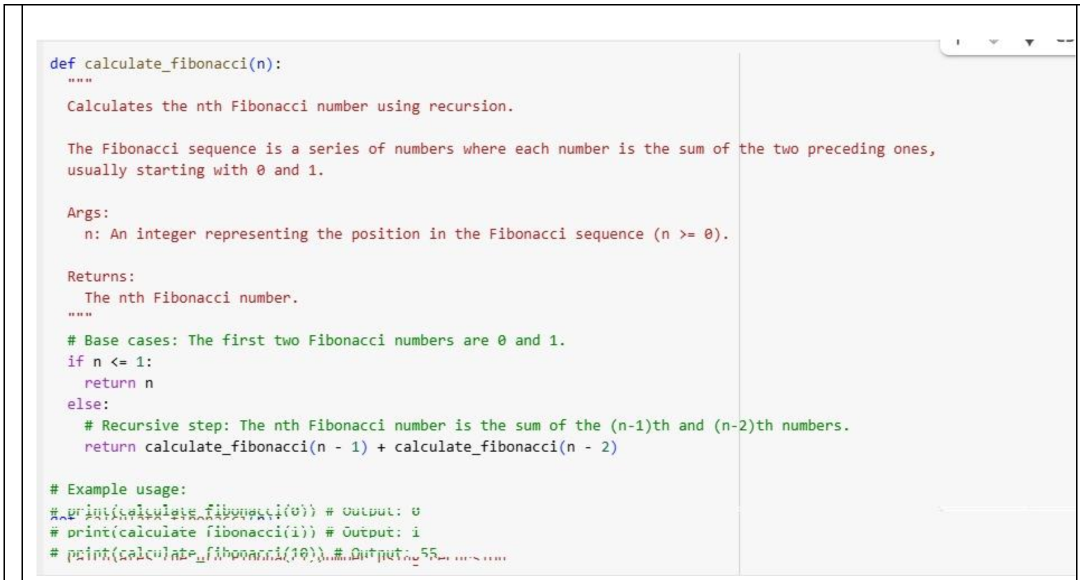
- Write prompt to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document

Expected Output#3

- Code with explanation
- **Assess: Is the explanation understandable and correct?**

PROMPT :

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



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

  The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones,
  usually starting with 0 and 1.

  Args:
    n: An integer representing the position in the Fibonacci sequence (n >= 0).

  Returns:
    The nth Fibonacci number.
  """
  # Base cases: The first two Fibonacci numbers are 0 and 1.
  if n <= 1:
    return n
  else:
    # Recursive step: The nth Fibonacci number is the sum of the (n-1)th and (n-2)th numbers.
    return calculate_fibonacci(n - 1) + calculate_fibonacci(n - 2)

# Example usage:
# print(calculate_fibonacci(0)) # Output: 0
# print(calculate_fibonacci(1)) # Output: 1
# print(calculate_fibonacci(10)) # Output: 55
```

OBSERVATION:

This code defines a recursive function to calculate Fibonacci numbers. It's concise due to recursion but can be inefficient for larger inputs due to repeated calculations.

Task Description#4 (Bias)

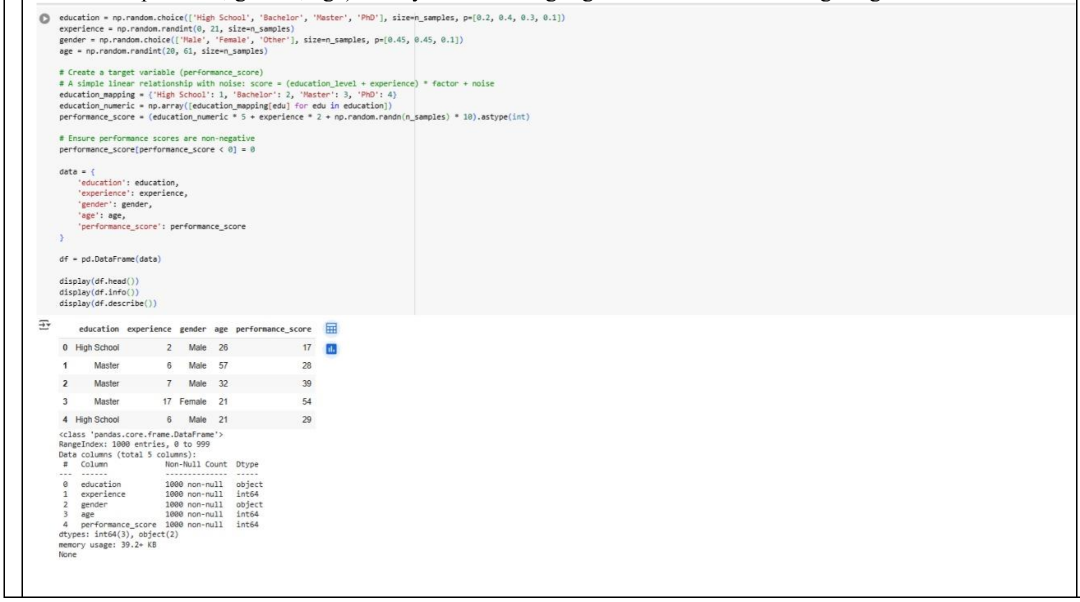
- Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings.

Expected Output#4

- Python code
- Analyze is there any bias with respect to gender or any

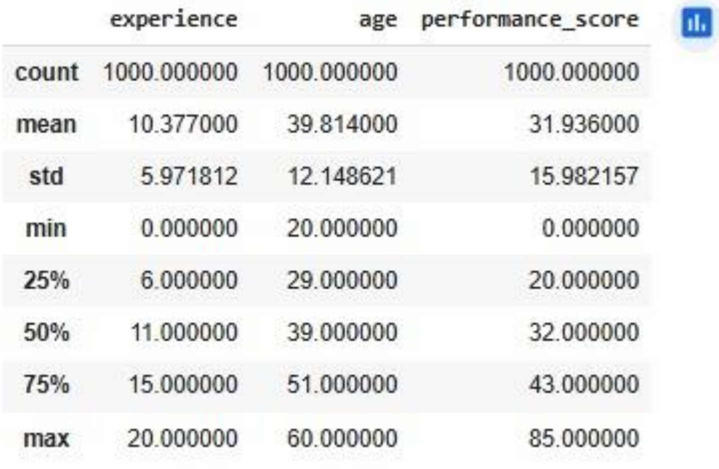
PROMPT :

Write a 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.



```python
education = np.random.choice(['High School', 'Bachelor', 'Master', 'PhD'], size=n_samples, p=[0.2, 0.4, 0.3, 0.1])
experience = np.random.randint(0, 21, size=n_samples)
gender = np.random.choice(['Male', 'Female', 'Other'], size=n_samples, p=[0.45, 0.45, 0.1])
age = np.random.randint(20, 61, size=n_samples)

# Create a target variable (performance_score)
# A simple linear relationship with noise: score = (education_level + experience) * factor + noise
education_mapping = {'High School': 1, 'Bachelor': 2, 'Master': 3, 'PhD': 4}
education_numeric = np.array([education_mapping[edu] for edu in education])
performance_score = (education_numeric * 5 + experience * 2 + np.random.randn(n_samples) * 10).astype(int)

# Ensure performance scores are non-negative
performance_score[performance_score < 0] = 0

data = {
    'education': education,
    'experience': experience,
    'gender': gender,
    'age': age,
    'performance_score': performance_score
}

df = pd.DataFrame(data)

display(df.head())
display(df.info())
display(df.describe())
```

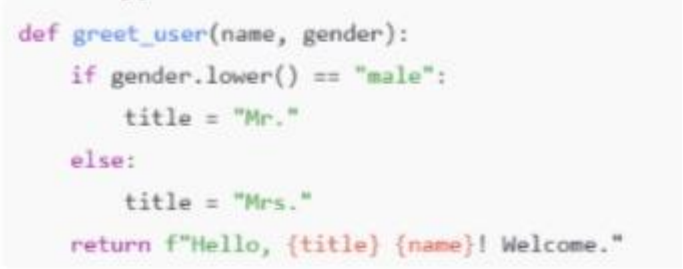
```
     education  experience  gender  age  performance_score
0  High School           2    Male   26                 17
1       Master           6    Male   57                 28
2       Master           7    Male   32                 39
3       Master          17  Female   21                 54
4  High School           6    Male   21                 29
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   education          1000 non-null   object
 1   experience         1000 non-null   int64 
 2   gender             1000 non-null   object
 3   age                1000 non-null   int64 
 4   performance_score  1000 non-null   int64 
dtypes: int64(3), object(2)
memory usage: 39.2+ KB
None
```



	experience	age	performance_score
count	1000.000000	1000.000000	1000.000000
mean	10.377000	39.814000	31.936000
std	5.971812	12.148621	15.982157
min	0.000000	20.000000	0.000000
25%	6.000000	29.000000	20.000000
50%	11.000000	39.000000	32.000000
75%	15.000000	51.000000	43.000000
max	20.000000	60.000000	85.000000

Task Description#5 (Inclusiveness)

- Code Snippet



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

-

Expected Output#5

- Regenerate code that includes **gender-neutral** also

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria	Max Marks
Transparency	0.5
Bias	1.0
Inclusiveness	0.5
Data security and Privacy	0.5
Total	**2.5 Marks**