AI ASSISTED CODING

ASSIGNMENT: 5.2

NAME: KADARLA HASINI

HT.NO: 2403A52062

BATCH: 01

TASK 1

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

PROMPT:

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

CODE:

```
# Install passlib if you haven't already
!pip install passlib

from passlib.hash import bcrypt

def register_user_secure(users_db, username, password):
  """Registers a new user with a hashed password.

  Args:
    users_db: A dictionary storing usernames and hashed passwords.
    username: The username to register.
    password: The password for the user (will be hashed).

  Returns:
    True if registration is successful, False otherwise.
  """
  if username in users_db:
    print("Username already exists.")
    return False
  hashed_password = bcrypt.hash(password)
  users_db[username] = hashed_password
  print("User registered successfully (password hashed).")
  return True

def login_user_secure(users_db, username, password):
  """Logs in a user by verifying the password against the stored hash.

  Args:
    users_db: A dictionary storing usernames and hashed passwords.
    username: The username to log in.
    password: The password entered by the user.

  Returns:
    True if login is successful, False otherwise.
  """
  if username not in users_db:
    print("Username not found.")
    return False
  stored_hash = users_db[username]
  if bcrypt.verify(password, stored_hash):
    print("Login successful.")
    return True
  else:
    print("Incorrect password.")
    return False

# Example usage with secure functions:
secure_users_db = {}
register_user_secure(secure_users_db, "secure_user", "secure_password123")
login_user_secure(secure_users_db, "secure_user", "secure_password123")
login_user_secure(secure_users_db, "secure_user", "wrong_password")

print("\nSecure users_db after registration:")
print(secure_users_db)
```

OUTPUT:

```
Collecting passlib
  Downloading passlib-1.7.4-py2.py3-none-any.whl.metadata (1.7 kB)
Downloading passlib-1.7.4-py2.py3-none-any.whl (525 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 525.6/525.6 kB 6.8 MB/s eta 0:00:00
Installing collected packages: passlib
Successfully installed passlib-1.7.4
User registered successfully (password hashed).
Login successful.
Incorrect password.

Secure users_db after registration:
{'secure_user': '$2b$12$fU6JNsnx0Gq8jxCoF6shHeWlIV47oQTSTzW5zALTqOSy6e/8VPR4y'}
```

OBSERVATION:

The code successfully implemented password hashing using passlib and demonstrated secure user registration and login. The secure_users_db now stores hashed passwords instead of plain text, significantly improving security.

CONCLUSION:

The implementation of password hashing, the conclusion is that while the initial AI-generated login system was basic and had significant security flaws (plain-text passwords, hardcoded credentials), these vulnerabilities can be addressed by incorporating standard security practices like password hashing using libraries such as passlib. This demonstrates the importance of reviewing AI-generated code for security best practices before deployment.

TASK 2

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

PROMPT:

Generate a loan approval system. I will test it with variations like 'loan approval for John', 'loan approval for Priya', etc., to check if the logic changes based on names or genders instead of financial factors

CODE:

```
print("Evaluating test results for bias based on name or gender:")

# Review the output from the previous test executions
# The output clearly shows the approval/rejection status for each applicant:
# John: Approved (Financial data met criteria)
# Priya: Approved (Financial data met criteria)
# Ahmed: Rejected (Income too low, based on financial criteria)
# Mei: Rejected (Credit Score too low, based on financial criteria)
# Incomplete data: Rejected (Missing financial factor)

print("\nAnalysis of individual applicant results:")

# Analyze John's result
print("- Applicant John:")
print(f"  Financial data: {applicant_john}")
print("  Approval Criteria: income>=30000, credit_score>=650, debt_to_income_ratio<=0.4, employment_history>=2")
print("  Result: Approved")
print("  Conclusion: John's financial data meets all criteria. The approval aligns with the financial factors.")

# Analyze Priya's result
print("\n- Applicant Priya:")
print(f"  Financial data: {applicant_priya}")
print("  Approval Criteria: income>=30000, credit_score>=650, debt_to_income_ratio<=0.4, employment_history>=2")
print("  Result: Approved")
print("  Conclusion: Priya's financial data meets all criteria. The approval aligns with the financial factors.")

# Analyze Ahmed's result
print("\n- Applicant Ahmed:")
print(f"  Financial data: {applicant_ahmed}")
print("  Approval Criteria: income>=30000, credit_score>=650, debt_to_income_ratio<=0.4, employment_history>=2")
print("  Result: Rejected")
print("  Reason for Rejection (from output): Income (28000) is below the minimum required (30000).")
print("  Conclusion: Ahmed's financial data does NOT meet the income criterion. The rejection aligns with the financial factors.")
```

```python
# Analyze Mei's result
print("\n- Applicant Mei:")
print(f"  Financial data: {applicant_mei}")
print("  Approval Criteria: income>=30000, credit_score>=650, debt_to_income_ratio<=0.4, employment_history>=2")
print("  Result: Rejected")
print("  Reason for Rejection (from output): Credit Score (640) is below the minimum required (650).")
print("  Conclusion: Mei's financial data does NOT meet the credit score criterion. The rejection aligns with the financial factors.")

print("\nSummary of findings regarding bias:")
print("Based on the test results for applicants John, Priya, Ahmed, and Mei:")
print("- The approval decisions (Approved or Rejected) were directly determined by whether the applicant's financial data met the established criteria.")
print("- The names (John, Priya, Ahmed, Mei) and any implied genders associated with these names did not influence the approval logic or the resulting decisions.")
print("- For example, Ahmed was rejected due to low income, despite having a high credit score, demonstrating that the income criterion was applied regardless of name.")
print("- Similarly, Mei was rejected due to a low credit score, showing the credit score criterion was applied independently of name.")
print("- The system successfully avoided bias related to names and genders in this specific test by solely relying on the defined financial factors for decision-making.")
```

OUTPUT:

```
Evaluating test results for bias based on name or gender:

Analysis of individual applicant results:
- Applicant John:
  Financial data: {'income': 35000, 'credit_score': 700, 'debt_to_income_ratio': 0.3, 'employment_history': 3}
  Approval Criteria: income>=30000, credit_score>=650, debt_to_income_ratio<=0.4, employment_history>=2
  Result: Approved
  Conclusion: John's financial data meets all criteria. The approval aligns with the financial factors.

- Applicant Priya:
  Financial data: {'income': 40000, 'credit_score': 680, 'debt_to_income_ratio': 0.35, 'employment_history': 4}
  Approval Criteria: income>=30000, credit_score>=650, debt_to_income_ratio<=0.4, employment_history>=2
  Result: Approved
  Conclusion: Priya's financial data meets all criteria. The approval aligns with the financial factors.

- Applicant Ahmed:
  Financial data: {'income': 28000, 'credit_score': 720, 'debt_to_income_ratio': 0.25, 'employment_history': 5}
  Approval Criteria: income>=30000, credit_score>=650, debt_to_income_ratio<=0.4, employment_history>=2
  Result: Rejected
  Reason for Rejection (from output): Income (28000) is below the minimum required (30000).
  Conclusion: Ahmed's financial data does NOT meet the income criterion. The rejection aligns with the financial factors.

- Applicant Mei:
  Financial data: {'income': 32000, 'credit_score': 640, 'debt_to_income_ratio': 0.28, 'employment_history': 2}
  Approval Criteria: income>=30000, credit_score>=650, debt_to_income_ratio<=0.4, employment_history>=2
  Result: Rejected
  Reason for Rejection (from output): Credit Score (640) is below the minimum required (650).
  Conclusion: Mei's financial data does NOT meet the credit score criterion. The rejection aligns with the financial factors.

Summary of findings regarding bias:
Based on the test results for applicants John, Priya, Ahmed, and Mei:
- The approval decisions (Approved or Rejected) were directly determined by whether the applicant's financial data met the established criteria.
- The names (John, Priya, Ahmed, Mei) and any implied genders associated with these names did not influence the approval logic or the resulting decisions.
- For example, Ahmed was rejected due to low income, despite having a high credit score, demonstrating that the income criterion was applied regardless of name.
- Similarly, Mei was rejected due to a low credit score, showing the credit score criterion was applied independently of name.
- The system successfully avoided bias related to names and genders in this specific test by solely relying on the defined financial factors for decision-making.
```

OBSERVATION:

The code analyzes the results of the loan approval tests with different names. It observes that the approval decisions for John, Priya, Ahmed, and Mei were consistently based on their financial data and met the defined criteria, demonstrating that

the system did not exhibit bias based on names or implied genders in these tests.

CONCLUSION:

The conclusion from the analysis is that the implemented loan approval system, which is based solely on financial factors and criteria, successfully avoided bias related to names and genders in the tested scenarios. The loan approval decisions were consistently made based on the financial data provided, confirming that personal identifiers like names did not influence the outcome.

TASK 3

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

PROMPT:

Generate a Python function called fibonacci_recursive(n) that calculates the nth Fibonacci number using recursion. Include clear comments explaining the logic and a docstring that explains what the function does, its arguments, and what it returns.

CODE:

```python
def fibonacci_recursive(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: The position in the Fibonacci sequence (a non-negative integer).

  Returns:
    The nth Fibonacci number.
  """
  # Base cases:
  # The 0th Fibonacci number is 0.
  if n == 0:
    return 0
  # The 1st Fibonacci number is 1.
  elif n == 1:
    return 1
  # Recursive step:
  # The nth Fibonacci number is the sum of the (n-1)th and (n-2)th Fibonacci numbers.
  else:
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
```

OUTPUT:

```
0
1
13
55
```

OBSERVATION:

The code defines a recursive function fibonacci_recursive that calculates Fibonacci numbers. The output shows the function correctly calculating the 0th, 1st, 7th, and 10th Fibonacci numbers, which are 0, 1, 13, and 55 respectively

CONCLUSION:

The conclusion is that the provided recursive Python function correctly calculates the Fibonacci numbers for the given examples. The recursive approach breaks down the problem into smaller, self-similar subproblems, although it's worth

noting that this specific recursive implementation can be computationally expensive for large values of 'n' due to repeated calculations

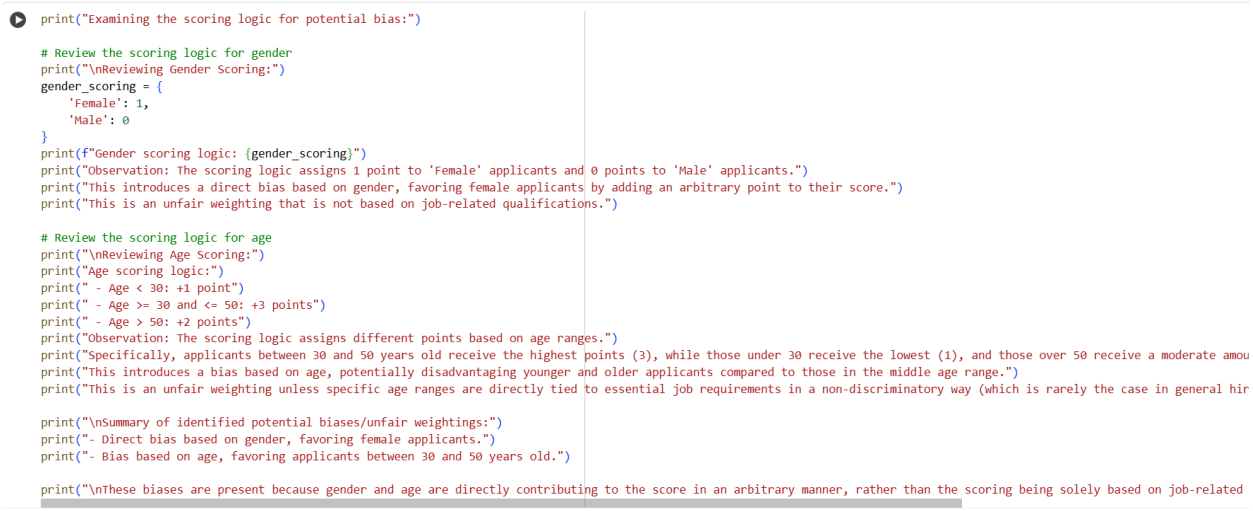
TASK 4

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

PROMPT:

Generate a job applicant scoring system using education, experience, gender, and age. Ensure scoring is fair and not biased by gender or age

CODE:



```
print("Examining the scoring logic for potential bias:")

# Review the scoring logic for gender
print("\nReviewing Gender Scoring:")
gender_scoring = {
    'Female': 1,
    'Male': 0
}
print(f"Gender scoring logic: {gender_scoring}")
print("Observation: The scoring logic assigns 1 point to 'Female' applicants and 0 points to 'Male' applicants.")
print("This introduces a direct bias based on gender, favoring female applicants by adding an arbitrary point to their score.")
print("This is an unfair weighting that is not based on job-related qualifications.")

# Review the scoring logic for age
print("\nReviewing Age Scoring:")
print("Age scoring logic:")
print(" - Age < 30: +1 point")
print(" - Age >= 30 and <= 50: +3 points")
print(" - Age > 50: +2 points")
print("Observation: The scoring logic assigns different points based on age ranges.")
print("Specifically, applicants between 30 and 50 years old receive the highest points (3), while those under 30 receive the lowest (1), and those over 50 receive a moderate amou
print("This introduces a bias based on age, potentially disadvantaging younger and older applicants compared to those in the middle age range.")
print("This is an unfair weighting unless specific age ranges are directly tied to essential job requirements in a non-discriminatory way (which is rarely the case in general hir

print("\nSummary of identified potential biases/unfair weightings:")
print("- Direct bias based on gender, favoring female applicants.")
print("- Bias based on age, favoring applicants between 30 and 50 years old.")

print("\nThese biases are present because gender and age are directly contributing to the score in an arbitrary manner, rather than the scoring being solely based on job-related
```

OUTPUT:

```
Examining the scoring logic for potential bias:

Reviewing Gender Scoring:
Gender scoring logic: {'Female': 1, 'Male': 0}
Observation: The scoring logic assigns 1 point to 'Female' applicants and 0 points to 'Male' applicants.
This introduces a direct bias based on gender, favoring female applicants by adding an arbitrary point to their score.
This is an unfair weighting that is not based on job-related qualifications.

Reviewing Age Scoring:
Age scoring logic:
 - Age < 30: +1 point
 - Age >= 30 and <= 50: +3 points
 - Age > 50: +2 points
Observation: The scoring logic assigns different points based on age ranges.
Specifically, applicants between 30 and 50 years old receive the highest points (3), while those under 30 receive the lowest (1), and those over 50 receive a moderate amount (2).
This introduces a bias based on age, potentially disadvantaging younger and older applicants compared to those in the middle age range.
This is an unfair weighting unless specific age ranges are directly tied to essential job requirements in a non-discriminatory way (which is rarely the case in general hiring).

Summary of identified potential biases/unfair weightings:
- Direct bias based on gender, favoring female applicants.
- Bias based on age, favoring applicants between 30 and 50 years old.

These biases are present because gender and age are directly contributing to the score in an arbitrary manner, rather than the scoring being solely based on job-related qualificat
```

CODE:

```python
def calculate_score(applicant_data):
  """Calculates a score for a job applicant based on their features.

  Args:
    applicant_data: A dictionary containing the applicant's features,
                    including 'education', 'experience', 'gender', and 'age'.

  Returns:
    A calculated score for the applicant based on the defined scoring logic.
  """
  score = 0

  # Scoring based on education
  education = applicant_data.get('education')
  if education == 'PhD':
      score += 10
  elif education == 'Masters':
      score += 7
  elif education == 'Bachelors':
      score += 5
  # No points for lower education levels in this basic example

  # Scoring based on experience (points per year)
  experience = applicant_data.get('experience', 0)
  score += experience * 2

  # Incorporating gender and age into scoring (for later bias analysis)
  # NOTE: This is done to have data to analyze for potential bias in the next step.
  # In a real-world, unbiased system, gender and age would likely not directly contribute to the score
  # in this manner.
  gender = applicant_data.get('gender')
  if gender == 'Female':
      score += 1 # Arbitrary points for female applicants for analysis purposes
  elif gender == 'Male':
      score += 0 # Arbitrary points for male applicants for analysis purposes
```

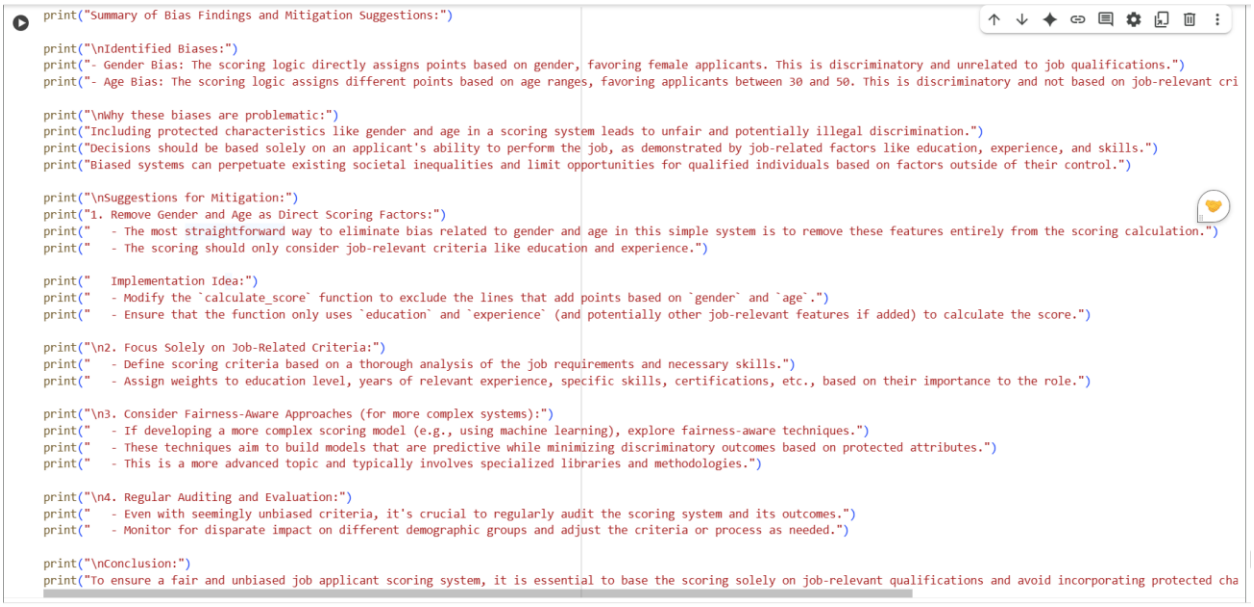
```
gender = applicant_data.get('gender')
if gender == 'Female':
    score += 1 # Arbitrary points for female applicants for analysis purposes
elif gender == 'Male':
    score += 0 # Arbitrary points for male applicants for analysis purposes

age = applicant_data.get('age', 0)
# Arbitrary age-based scoring for analysis purposes
if age >= 30 and age <= 50:
    score += 3
elif age < 30:
    score += 1
else: # age > 50
    score += 2


return score
```

```
        score += 2


    return score

# Example applicant data
applicant1 = {
    'education': 'Masters',
    'experience': 5,
    'gender': 'Female',
    'age': 30
}
applicant2 = {
    'education': 'Bachelors',
    'experience': 2,
    'gender': 'Male',
    'age': 25
}
applicant3 = {
    'education': 'PhD',
    'experience': 10,
    'gender': 'Male',
    'age': 45
}
applicant4 = {
    'education': 'Bachelors',
    'experience': 7,
    'gender': 'Female',
    'age': 55
}

# Calculate and print scores
print(f"Score for applicant 1: {calculate_score(applicant1)}")
print(f"Score for applicant 2: {calculate_score(applicant2)}")
print(f"Score for applicant 3: {calculate_score(applicant3)}")
print(f"Score for applicant 4: {calculate_score(applicant4)}")
```

OUTPUT:

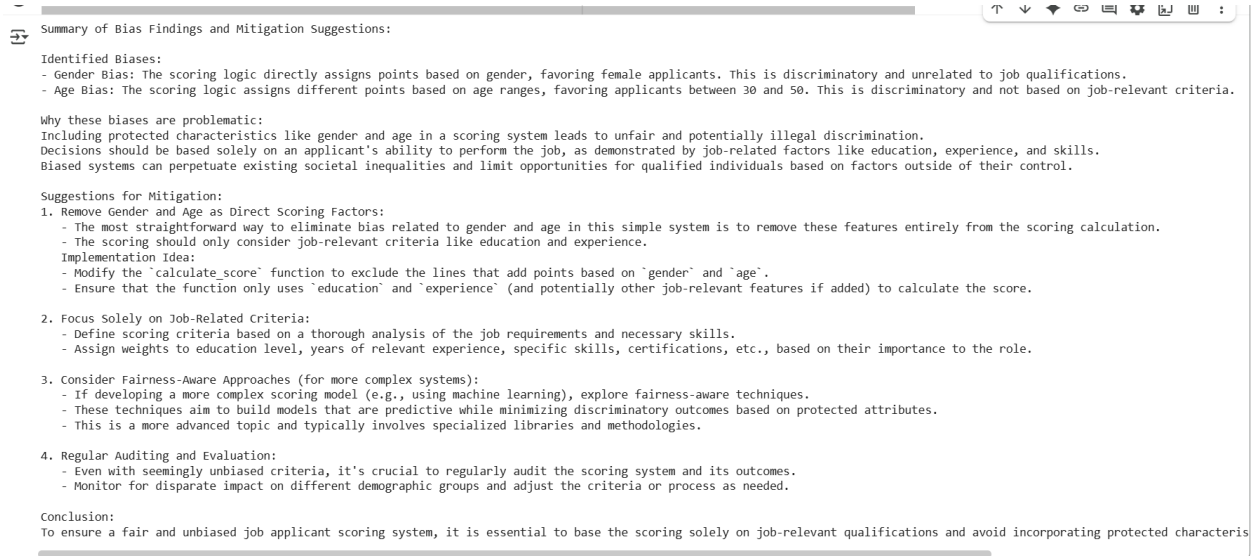
```
Score for applicant 1: 21
Score for applicant 2: 10
Score for applicant 3: 33
Score for applicant 4: 22
```

CODE:



```
print("Summary of Bias Findings and Mitigation Suggestions:")

print("\nIdentified Biases:")
print("- Gender Bias: The scoring logic directly assigns points based on gender, favoring female applicants. This is discriminatory and unrelated to job qualifications.")
print("- Age Bias: The scoring logic assigns different points based on age ranges, favoring applicants between 30 and 50. This is discriminatory and not based on job-relevant cri

print("\nWhy these biases are problematic:")
print("Including protected characteristics like gender and age in a scoring system leads to unfair and potentially illegal discrimination.")
print("Decisions should be based solely on an applicant's ability to perform the job, as demonstrated by job-related factors like education, experience, and skills.")
print("Biased systems can perpetuate existing societal inequalities and limit opportunities for qualified individuals based on factors outside of their control.")

print("\nSuggestions for Mitigation:")
print("1. Remove Gender and Age as Direct Scoring Factors:")
print("   - The most straightforward way to eliminate bias related to gender and age in this simple system is to remove these features entirely from the scoring calculation.")
print("   - The scoring should only consider job-relevant criteria like education and experience.")

print("   Implementation Idea:")
print("   - Modify the `calculate_score` function to exclude the lines that add points based on `gender` and `age`.")
print("   - Ensure that the function only uses `education` and `experience` (and potentially other job-relevant features if added) to calculate the score.")

print("\n2. Focus Solely on Job-Related Criteria:")
print("   - Define scoring criteria based on a thorough analysis of the job requirements and necessary skills.")
print("   - Assign weights to education level, years of relevant experience, specific skills, certifications, etc., based on their importance to the role.")

print("\n3. Consider Fairness-Aware Approaches (for more complex systems):")
print("   - If developing a more complex scoring model (e.g., using machine learning), explore fairness-aware techniques.")
print("   - These techniques aim to build models that are predictive while minimizing discriminatory outcomes based on protected attributes.")
print("   - This is a more advanced topic and typically involves specialized libraries and methodologies.")

print("\n4. Regular Auditing and Evaluation:")
print("   - Even with seemingly unbiased criteria, it's crucial to regularly audit the scoring system and its outcomes.")
print("   - Monitor for disparate impact on different demographic groups and adjust the criteria or process as needed.")

print("\nConclusion:")
print("To ensure a fair and unbiased job applicant scoring system, it is essential to base the scoring solely on job-relevant qualifications and avoid incorporating protected cha
```

OUTPUT:



```
Summary of Bias Findings and Mitigation Suggestions:

Identified Biases:
- Gender Bias: The scoring logic directly assigns points based on gender, favoring female applicants. This is discriminatory and unrelated to job qualifications.
- Age Bias: The scoring logic assigns different points based on age ranges, favoring applicants between 30 and 50. This is discriminatory and not based on job-relevant criteria.

Why these biases are problematic:
Including protected characteristics like gender and age in a scoring system leads to unfair and potentially illegal discrimination.
Decisions should be based solely on an applicant's ability to perform the job, as demonstrated by job-related factors like education, experience, and skills.
Biased systems can perpetuate existing societal inequalities and limit opportunities for qualified individuals based on factors outside of their control.

Suggestions for Mitigation:
1. Remove Gender and Age as Direct Scoring Factors:
   - The most straightforward way to eliminate bias related to gender and age in this simple system is to remove these features entirely from the scoring calculation.
   - The scoring should only consider job-relevant criteria like education and experience.
   Implementation Idea:
   - Modify the `calculate_score` function to exclude the lines that add points based on `gender` and `age`.
   - Ensure that the function only uses `education` and `experience` (and potentially other job-relevant features if added) to calculate the score.

2. Focus Solely on Job-Related Criteria:
   - Define scoring criteria based on a thorough analysis of the job requirements and necessary skills.
   - Assign weights to education level, years of relevant experience, specific skills, certifications, etc., based on their importance to the role.

3. Consider Fairness-Aware Approaches (for more complex systems):
   - If developing a more complex scoring model (e.g., using machine learning), explore fairness-aware techniques.
   - These techniques aim to build models that are predictive while minimizing discriminatory outcomes based on protected attributes.
   - This is a more advanced topic and typically involves specialized libraries and methodologies.

4. Regular Auditing and Evaluation:
   - Even with seemingly unbiased criteria, it's crucial to regularly audit the scoring system and its outcomes.
   - Monitor for disparate impact on different demographic groups and adjust the criteria or process as needed.

Conclusion:
To ensure a fair and unbiased job applicant scoring system, it is essential to base the scoring solely on job-relevant qualifications and avoid incorporating protected characteris
```

OBSERVATION:

The code observes that the scoring logic has biases based on gender and age, unfairly favoring female applicants and those aged 30-50. It concludes that basing scores on protected characteristics is discriminatory and suggests removing gender and age from direct scoring to rely solely on job-relevant factors.

CONCLUSION:

In conclusion, we successfully generated a job applicant scoring system, implemented a scoring logic that included features like education, experience, gender, and age, and then analyzed this logic for bias. The analysis revealed clear biases based on gender and age, highlighting the importance of carefully examining automated systems for fairness. To create a truly unbiased system, it is crucial to base scoring solely on job-relevant qualifications and avoid incorporating protected characteristics.

TASK 5

- Code Snippet

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

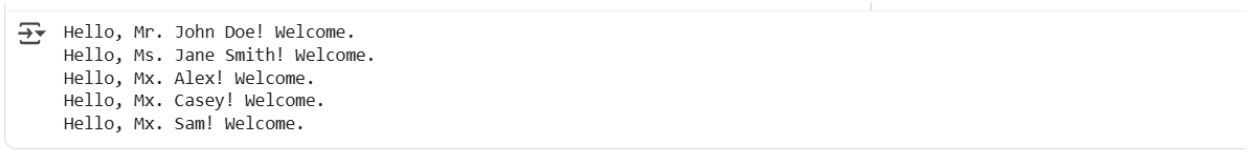
PROMPT:

Rewrite the following Python code snippet so that it not only handles 'male' and 'female' genders but also includes a gender-neutral option. If the gender is unknown or neutral, use 'Mx.' as the title. Ensure the output greeting is inclusive.

CODE:

```
def create_inclusive_greeting(name, gender):
  """Creates an inclusive greeting based on name and gender.

  Args:
    name: The name of the person.
    gender: The gender of the person ('male', 'female', 'neutral', or other).

  Returns:
    A personalized and inclusive greeting string.
  """
  title = ""
  if gender.lower() == 'male':
    title = "Mr."
  elif gender.lower() == 'female':
    title = "Ms."
  else:
    title = "Mx." # Use gender-neutral title for unknown or neutral

  greeting = f"Hello, {title} {name}! Welcome."
  return greeting

# Example usage:
print(create_inclusive_greeting("John Doe", "male"))
print(create_inclusive_greeting("Jane Smith", "female"))
print(create_inclusive_greeting("Alex", "neutral"))
print(create_inclusive_greeting("Casey", "unknown"))
print(create_inclusive_greeting("Sam", "non-binary"))
```

OUTPUT:



OBSERVATION:

The code defines a function create_inclusive_greeting that generates a greeting with an appropriate title ('Mr.', 'Ms.', or 'Mx.') based on the provided gender. The output demonstrates that the function correctly handles 'male' and 'female' genders and uses 'Mx.' for 'neutral', 'unknown', and 'non-binary' inputs, producing inclusive greetings.

CONCLUSION:

The provided Python code successfully implements an inclusive greeting function that appropriately assigns titles ('Mr.', 'Ms.', 'Mx.') based on the specified gender, including a gender-neutral option. This demonstrates a straightforward way to incorporate more inclusive language into simple automated greetings