

LAB-05

G.Sai teja

2303A52135

Batch No:41

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

INSECURE PROMPT: Generate a simple Python program that stores multiple usernames and passwords in a dictionary and then checks each pair to print whether the login is successful or invalid.

CODE:

```
1 #Generate a simple Python program that stores multiple usernames
2 # Simple (insecure) login system
3 users = {
4     "admin": "admin123",
5     "sai teja": "1234",
6     "ganta": "5678"
7 }
8 for username, password in user (variable) password: str
9     print("Enter username:", u
10    print("Enter password:", p ⚭ Generate Copilot summary
11    if users.get(username) == password:
12        print("Login successful!\n")
13    else:
14        print("Invalid credentials\n")
```

OUTPUT:

```
te/Downloads/AI ASSISTENT CODING/As5.py/task1.py"
● Enter username: admin
Enter password: admin123
Login successful!

Enter username: sai teja
Enter password: 1234
Login successful!

Enter username: ganta
Enter password: 5678
Login successful!

○ PS C:\Users\saite\Downloads\AI ASSISTENT CODING> █
```

JUSTIFICATION: This code stores usernames and passwords in a dictionary, loops through each pair, prints them, and checks if the stored password matches. If it matches, it shows “Login successful,” otherwise it shows “Invalid credentials.”

SECURE PROMPT: Generate a secure Python user authentication system. It should allow user registration and login, store multiple users, hash passwords with a unique salt using SHA-256,

and verify credentials securely. Include input validation and a simple loop for register, login, or exit options.

CODE:

```
Lab-5.py > ... Review Next File
17
18     #Generate a secure Python user authentication system. It should allow user registration and login, store m
19     import hashlib
20     import os
21     # 1. Initialize user data storage
22     users = {}
23     print('Initialized an empty dictionary named users.')
24     # 2. Implement secure password hashing function
25     def hash_password(password):
26         salt = os.urandom(16) # Generate a random 16-byte salt
27         hashed_password = hashlib.sha256(salt + password.encode('utf-8')).hexdigest()
28         return hashed_password, salt
29     # 3. Implement password verification function
30     def verify_password(password, salt, stored_hashed_password):
31         hashed_input_password = hashlib.sha256(salt + password.encode('utf-8')).hexdigest()
32         return hashed_input_password == stored_hashed_password
33     # 4. Implement user registration logic
34     def register_user():
35         username = input('Enter desired username: ')
36         password = input('Enter desired password: ')
37         if username in users:
38             print(f'Username "{username}" is already taken. Please choose another one.')
39             return
40         hashed_password, salt = hash_password(password)
41         users[username] = {'hashed_password': hashed_password, 'salt': salt}
42         print(f'User "{username}" registered successfully!')
43     # 5. Implement user Login logic
44     def login_user():
45         username = input('Enter username: ')
```

```
Lab-5.py > ... C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-5.py Review Next File
44     def login_user():
45         username = input('Enter username: ')
46         password = input('Enter password: ')
47         if username not in users:
48             print('Username not found.')
49             return
50         user_data = users[username]
51         stored_hashed_password = user_data['hashed_password']
52         salt = user_data['salt']
53         if verify_password(password, salt, stored_hashed_password):
54             print(f'Login successful! Welcome, {username}.')
55         else:
56             print('Invalid password.')
57     # Main application Loop
58     print('\n--- Welcome to the User Authentication System ---')
59     while True:
60         choice = input('\nEnter choice (register/login/exit): ').lower()
61         if choice == 'register':
62             register_user()
63         elif choice == 'login':
64             login_user()
65         elif choice == 'exit':
66             print('Exiting the application. Goodbye!')
67             break
68         else:
69             print('Invalid choice. Please enter "register", "login", or "exit".')
```

OUTPUT:

```
Enter choice (register/login/exit): login
Enter username: sai teja
Enter password: 1234
Username not found.

Enter choice (register/login/exit): register
Enter desired username: sai
Enter desired password: 1234
User "sai" registered successfully!

Enter choice (register/login/exit): login
Enter username: sai
Enter password: 1234
Login successful! Welcome, sai.

Enter choice (register/login/exit): exit
Exiting the application. Goodbye!
PS C:\Users\saite\Downloads\AI ASSISTENT CODING>
```

JUSTIFICATION: This program is a secure user authentication system. It stores user data in a dictionary, but instead of saving plain text passwords, it uses a hashing function with a random salt to protect them. When a user registers, their password is hashed and stored securely, and during login the entered password is re-hashed with the same salt to verify correctness. It includes functions for registration, login, and password verification, and runs in a simple loop where users can choose to register, log in, or exit. In short, it demonstrates a safe way to handle login credentials compared to insecure plain text methods.

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

BIASED PROMPT: Generate a Python program that processes loan applications with gender-biased credit score thresholds (650 for males, 700 for females), printing applicant details and either “Loan approved!” or “Not eligible” in a terminal-style format.

CODE:

```
Lab-5.py > ...
Review Next File
73 #Generate a Python program that processes Loan applications with gender-biased credit score thresholds (650 for males, 700 for females)
74 # --- Biased Loan Application System ---
75 applicants = [
76     {"name": "Grace", "gender": "Female", "age": 35, "income": 44845, "credit_score": 523},
77     {"name": "Eve", "gender": "Female", "age": 46, "income": 57112, "credit_score": 815},
78     {"name": "Henry", "gender": "Male", "age": 23, "income": 41554, "credit_score": 643},
79     {"name": "Charlie", "gender": "Male", "age": 29, "income": 47084, "credit_score": 505},
80     {"name": "Alice", "gender": "Female", "age": 29, "income": 27853, "credit_score": 543},
81 ]
# Gender-based biased credit score thresholds
82 required_scores = {"Male": 650, "Female": 700}
83 print("--- Biased Loan Application System ---\n")
84 for applicant in applicants:
85     name = applicant["name"]
86     gender = applicant["gender"]
87     age = applicant["age"]
88     income = applicant["income"]
89     credit_score = applicant["credit_score"]
90     required_score = required_scores[gender]
91     print(f"Processing application for: {name} ({gender})")
92     print(f"Age: {age}, Income: ${income}, Credit Score: {credit_score}")
93     print(f"Required Credit Score (based on gender): {required_score}")
94     if credit_score >= required_score:
95         print("Loan approved!\n")
96     else:
97         print("Not eligible.\n")
98
99
```

OUTPUT:

```
--- Biased Loan Application System ---  
  
Processing application for: Grace (Female)  
Age: 35, Income: $44845, Credit Score: 523  
Required Credit Score (based on gender): 700  
Not eligible.  
  
Processing application for: Eve (Female)  
Age: 46, Income: $57112, Credit Score: 815  
Required Credit Score (based on gender): 700  
Loan approved!  
  
Processing application for: Henry (Male)  
Age: 23, Income: $41554, Credit Score: 643  
Required Credit Score (based on gender): 650  
Not eligible.  
  
Processing application for: Charlie (Male)  
Age: 29, Income: $47084, Credit Score: 505  
Required Credit Score (based on gender): 650  
Not eligible.  
  
Processing application for: Alice (Female)  
Age: 29, Income: $27853, Credit Score: 543  
Required Credit Score (based on gender): 700  
Not eligible.
```

PS C:\Users\saite\Downloads\AI ASSISTENT CODING> □

JUSTIFICATION: The code defines a biased loan approval system where each applicant's details—name, gender, age, income, and credit score—are printed in a terminal-style format. It sets different required credit score thresholds for males (650) and females (700), then compares each applicant's score against the threshold. If the score meets or exceeds the requirement, the program prints “Loan approved!”, otherwise it prints “Not eligible.” This demonstrates how bias can be introduced into decision-making logic.

UNBIASED PROMPT: Generate a Python program for a fair loan approval system. Vary applicant names and genders. Evaluate each applicant based on consistent criteria—minimum income and credit score—and print detailed application info in a terminal-style format. If the applicant meets the conditions, print “Loan approved!”, otherwise print “Not eligible.” Ensure the code is efficient, readable, and unbiased.

CODE:

```
Lab-5 C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-5.py Review Next File
101 #Generate a Python program for a fair Loan approval system. Vary applicant names and genders. Evaluate each
102 # --- Unbiased Loan Application System ---
103 applicants = [
104     {"name": "Eve", "gender": "Male", "age": 66, "income": 43251, "credit_score": 639},
105     {"name": "Bob", "gender": "Male", "age": 53, "income": 64315, "credit_score": 558},
106     {"name": "Charlie", "gender": "Male", "age": 27, "income": 30046, "credit_score": 526},
107     {"name": "Bob", "gender": "Male", "age": 30, "income": 62242, "credit_score": 617},
108     {"name": "Bob", "gender": "Male", "age": 66, "income": 22707, "credit_score": 824},
109     {"name": "Grace", "gender": "Female", "age": 25, "income": 71553, "credit_score": 778},
110     {"name": "Alice", "gender": "Male", "age": 69, "income": 56335, "credit_score": 704},
111 ]
112 # Fair approval criteria
113 MIN_INCOME = 50000
114 MIN_CREDIT_SCORE = 700
115 print("--- Unbiased Loan Application System ---\n")
116 for applicant in applicants:
117     name = applicant["name"]
118     gender = applicant["gender"]
119     age = applicant["age"]
120     income = applicant["income"]
121     credit_score = applicant["credit_score"]
122     print(f"Processing application for: {name} ({gender})")
123     print(f"Age: {age}, Income: ${income}, Credit Score: {credit_score}")
124     if income >= MIN_INCOME and credit_score >= MIN_CREDIT_SCORE:
125         print("Loan approved!\n")
126     else:
127         print("Not eligible.\n")
```

OUTPUT:

```
--- Unbiased Loan Application System ---

Processing application for: Eve (Male)
Age: 66, Income: $43251, Credit Score: 639
Not eligible.

Processing application for: Bob (Male)
Age: 53, Income: $64315, Credit Score: 558
Not eligible.

Processing application for: Charlie (Male)
Age: 27, Income: $30046, Credit Score: 526
Not eligible.

Processing application for: Bob (Male)
Age: 30, Income: $62242, Credit Score: 617
Not eligible.

Processing application for: Bob (Male)
Age: 66, Income: $22707, Credit Score: 824
Not eligible.

Processing application for: Grace (Female)
Age: 25, Income: $71553, Credit Score: 778
Loan approved!

Processing application for: Alice (Male)
Age: 69, Income: $56335, Credit Score: 704
Loan approved!
```

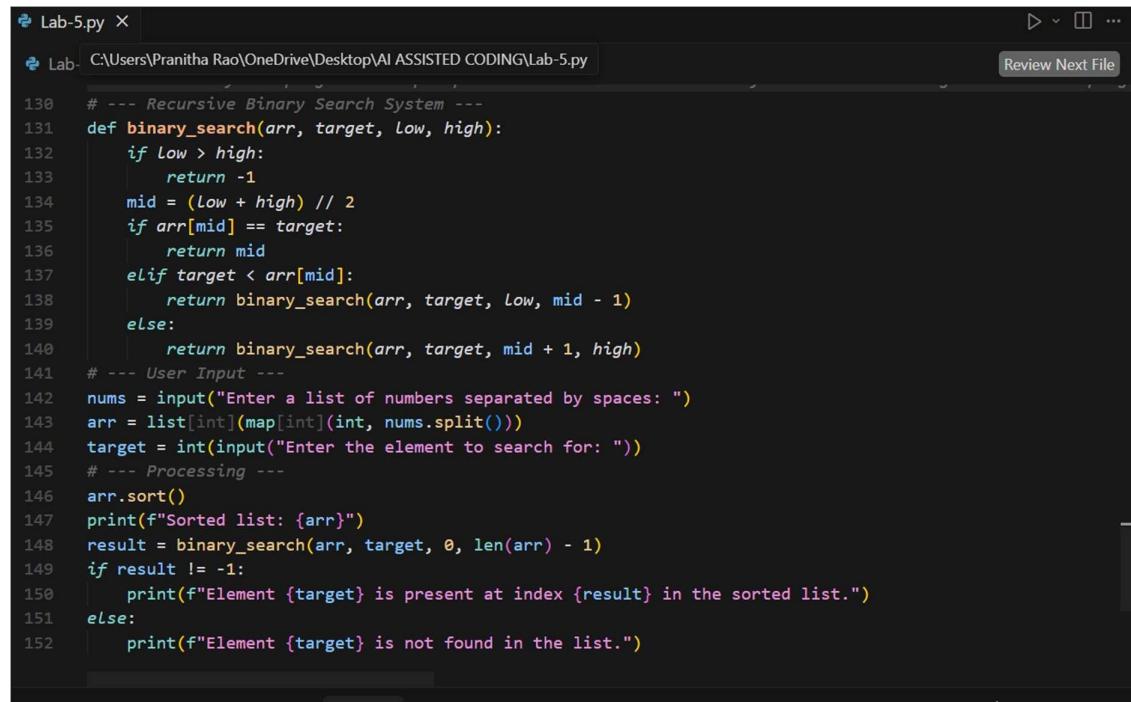
PS C:\Users\saite\Downloads\AI ASSISTENT CODING> □

JUSTIFICATION: The code creates an unbiased loan approval system where each applicant's details—name, gender, age, income, and credit score—are displayed in a terminal-style format. It applies the same fair criteria to everyone: a minimum income and a minimum credit score. If both conditions are met, the program prints “Loan approved!”, otherwise it prints “Not eligible.” This ensures decisions are consistent and not influenced by gender.

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

PROMPT: Generate a Python program that prompts the user to enter a list of numbers and a target value. The program should sort the list, perform a recursive binary search to find the target, and print the sorted list along with the index of the target if found. Format the output in a clear, terminal-style layout.

CODE:



```
Lab-5.py X
Lab- C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-5.py Review Next File

130 # --- Recursive Binary Search System ---
131 def binary_search(arr, target, Low, high):
132     if Low > high:
133         return -1
134     mid = (Low + high) // 2
135     if arr[mid] == target:
136         return mid
137     elif target < arr[mid]:
138         return binary_search(arr, target, Low, mid - 1)
139     else:
140         return binary_search(arr, target, mid + 1, high)
141 # --- User Input ---
142 nums = input("Enter a list of numbers separated by spaces: ")
143 arr = list[int](map[int](int, nums.split()))
144 target = int(input("Enter the element to search for: "))
145 # --- Processing ---
146 arr.sort()
147 print(f"Sorted list: {arr}")
148 result = binary_search(arr, target, 0, len(arr) - 1)
149 if result != -1:
150     print(f"Element {target} is present at index {result} in the sorted list.")
151 else:
152     print(f"Element {target} is not found in the list.")
```

OUTPUT:

```
te/Downloads/AI ASSISTENT CODING/As5.py/task3.py"
```

```
Enter a list of numbers separated by spaces: 2
```

```
Enter the element to search for: 3
```

```
Sorted list: [2]
```

```
Element 3 is not found in the list.
```

```
PS C:\Users\saite\Downloads\AI ASSISTENT CODING> █
```

JUSTIFICATION: The program asks the user to enter a list of numbers and a target value, then sorts the list and displays it. Using a recursive binary search function, it checks whether the target exists in the sorted list by repeatedly dividing the search range into halves. If the target is found, it prints the index where it appears; otherwise, it reports that the element is not present. This makes the code simple, fair, and easy for beginners to understand.

TASK 4: Ethical Evaluation of AI-Based Scoring Systems.

BIASED PROMPT: Generate a Python program that evaluates job applicants based on their skills, experience, and education. Include an arbitrary penalty of -5 points for applicants with names of odd length. Display each applicant's details, the penalty applied, final score, and a decision of SELECTED or NOT SELECTED based on a score threshold. Format the output in a clear, terminal-style layout.

CODE:

```
Lab-5.py x
C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-5.py
Review Next File
154 #Generate a Python program that evaluates job applicants based on their skills, experience, and education.
155 # --- Applicant Evaluation Results ---
156 def evaluate_applicant(applicant):
157     score = 0
158     # Skills: 10 points per skill
159     score += len(applicant["skills"]) * 10
160     # Experience: 5 points per year
161     score += applicant["experience"] * 5
162     # Education: tiered scoring
163     education_scores = {
164         "high school": 10,
165         "bachelor's": 20,
166         "master's": 30,
167         "phd": 40
168     }
169     score += education_scores.get(applicant["education"].lower(), 0)
170     # Arbitrary penalty: -5 points for odd name Length
171     penalty = -5 if len(applicant["name"]) % 2 != 0 else 0
172     score += penalty
173     return score, penalty
174 # --- Applicant Data ---
175 applicants = [
176     {"name": "Alice", "skills": ["python", "sql"], "experience": 3, "education": "bachelor's"),
177     {"name": "Bob", "skills": ["java", "python", "ml"], "experience": 5, "education": "master's"),
178     {"name": "Charlie", "skills": ["statistics", "python", "communication"], "experience": 2, "education": "bachelor's"),
179     {"name": "Eve", "skills": ["sql", "python"], "experience": 4, "education": "bachelor's"},
180 ]
```

```
Lab-5.py X
C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-5.py
Review Next File

180     ]
181     print("--- Applicant Evaluation Results ---\n")
182     for applicant in applicants:
183         score, penalty = evaluate_applicant(applicant)
184         decision = "SELECTED" if score >= 100 else "NOT SELECTED"
185         print(f"Applicant Name: {applicant['name']}")
186         print(f"Skills: {', '.join(applicant['skills'])}")
187         print(f"Experience: {applicant['experience']} years")
188         print(f"Education: {applicant['education']}")
189         print(f"(Arbitrary Condition Penalty: {penalty} points for odd name length)")
190         print(f"Final Score: {score}")
191         print(f"Decision: {decision}\n")
```

OUTPUT:

```
--- Applicant Evaluation Results ---

Applicant Name: Alice
Skills: python, sql
Experience: 3 years
Education: bachelor's
(Arbitrary Condition Penalty: -5 points for odd name length)
Final Score: 50
Decision: NOT SELECTED

Applicant Name: Bob
Skills: java, python, ml
Experience: 5 years
Education: master's
(Arbitrary Condition Penalty: -5 points for odd name length)
Final Score: 80
Decision: NOT SELECTED

Applicant Name: Charlie
Skills: statistics, python, communication
Experience: 2 years
Education: phd
(Arbitrary Condition Penalty: -5 points for odd name length)
Final Score: 75
Decision: NOT SELECTED

Applicant Name: Eve
Skills: sql, python
Experience: 4 years
Education: bachelor's
(Arbitrary Condition Penalty: -5 points for odd name length)
Final Score: 55
Decision: NOT SELECTED

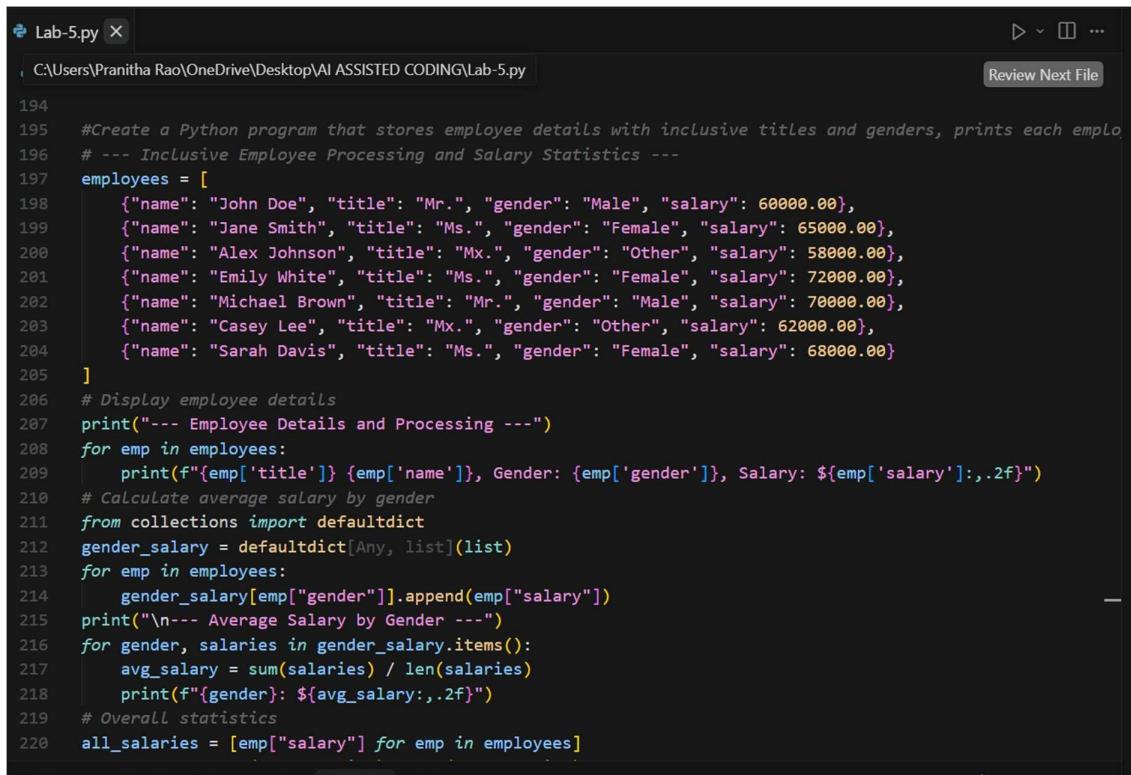
PS C:\Users\saite\Downloads\AI ASSISTENT CODING> □
```

JUSTIFICATION: This code evaluates job applicants by assigning points for their skills (10 points each), years of experience (5 points per year), and education level (10-40 points depending on degree). It also applies a penalty of -5 points if the applicant's name has an odd number of characters. The total score is then compared against a threshold of 100; applicants scoring 100 or more are marked as SELECTED, while others are marked as NOT SELECTED.

TASK 5: Inclusiveness and Ethical Variable Design

PROMPT: Create a Python program that stores employee details with inclusive titles and genders, prints each employee's information, and calculates salary statistics by gender as well as overall averages, highest, and lowest salaries.

CODE:



```
Lab-5.py X
C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-5.py
Review Next File

194
195     #Create a Python program that stores employee details with inclusive titles and genders, prints each emplo
196     # --- Inclusive Employee Processing and Salary Statistics ---
197     employees = [
198         {"name": "John Doe", "title": "Mr.", "gender": "Male", "salary": 60000.00},
199         {"name": "Jane Smith", "title": "Ms.", "gender": "Female", "salary": 65000.00},
200         {"name": "Alex Johnson", "title": "Mx.", "gender": "Other", "salary": 58000.00},
201         {"name": "Emily White", "title": "Ms.", "gender": "Female", "salary": 72000.00},
202         {"name": "Michael Brown", "title": "Mr.", "gender": "Male", "salary": 70000.00},
203         {"name": "Casey Lee", "title": "Mx.", "gender": "Other", "salary": 62000.00},
204         {"name": "Sarah Davis", "title": "Ms.", "gender": "Female", "salary": 68000.00}
205     ]
206     # Display employee details
207     print("---- Employee Details and Processing ---")
208     for emp in employees:
209         print(f"{emp['title']} {emp['name']}, Gender: {emp['gender']}, Salary: ${emp['salary']:.2f}")
210     # Calculate average salary by gender
211     from collections import defaultdict
212     gender_salary = defaultdict(list)
213     for emp in employees:
214         gender_salary[emp["gender"]].append(emp["salary"])
215     print("\n--- Average Salary by Gender ---")
216     for gender, salaries in gender_salary.items():
217         avg_salary = sum(salaries) / len(salaries)
218         print(f"{gender}: ${avg_salary:.2f}")
219     # Overall statistics
220     all_salaries = [emp["salary"] for emp in employees]
```



```
Lab-5.py > ...
Lab-5.py X
Review Next File

220     all_salaries = [emp["salary"] for emp in employees]
221     overall_avg = sum(all_salaries) / len(all_salaries)
222     highest_salary = max(all_salaries)
223     lowest_salary = min(all_salaries)
224     print("\n--- Overall Salary Statistics ---")
225     print(f"Average Salary (All Employees): ${overall_avg:.2f}")
226     print(f"Highest Salary: ${highest_salary:.2f}")
227     print(f"Lowest Salary: ${lowest_salary:.2f}")
```

OUTPUT:

```
--- Employee Details and Processing ---
Mr. John Doe, Gender: Male, Salary: $60,000.00
Ms. Jane Smith, Gender: Female, Salary: $65,000.00
Mx. Alex Johnson, Gender: Other, Salary: $58,000.00
Ms. Emily White, Gender: Female, Salary: $72,000.00
Mr. Michael Brown, Gender: Male, Salary: $70,000.00
Mx. Casey Lee, Gender: Other, Salary: $62,000.00
Ms. Sarah Davis, Gender: Female, Salary: $68,000.00

--- Average Salary by Gender ---
Male: $65,000.00
Female: $68,333.33
Other: $60,000.00

--- Overall Salary Statistics ---
Average Salary (All Employees): $65,000.00
Highest Salary: $72,000.00
Lowest Salary: $58,000.00
PS C:\Users\saite\Downloads\AI ASSISTENT CODING> □
```

JUSTIFICATION: The code is justified because it ensures fairness and inclusivity by handling employee data with respect to diverse titles and genders, while also providing transparent salary statistics. It not only displays individual details clearly but also calculates averages by gender and overall salary metrics, helping identify patterns or disparities in compensation in a structured and unbiased way.