Lab Assignment 5.3 2303A52490

Ethical Foundations & Responsible AI Coding Practices

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

Scenario

AI tools can sometimes generate insecure authentication logic.

AI Prompt Used

"Generate a simple login system in Python."

AI-Generated Code

```
# Simple Login System
username = "admin"
password = "password123"

user_input = input("Enter username: ")
pass_input = input("Enter password: ")

if user_input == username and pass_input == password:
    print("Login successful!")
else:
    print("Login failed!")
```

Identification of Security Risks

- Credentials are hardcoded directly in the code, anyone with access can see them.
- Passwords are stored and compared in plain text, no hashing or encryption.
- No input validation, accepts any string without sanitization.
- Insecure logic, no rate limiting or multiple attempt protection.

Revised Secure Version

```
import hashlib
import getpass

# Simulated user database (in real app, use database with hashed passwords)
users = {
    "admin": hashlib.sha256("securepass456".encode()).hexdigest()
}

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

username = input("Enter username: ").strip()
password = getpass.getpass("Enter password: ")  # Hides input

if username in users:
    hashed_pw = hash_password(password)
    if hashed_pw == users[username]:
        print("Login successful!")
    else:
        print("Invalid credentials.")
else:
    print("User not found.")
```

Output from Running Revised Code

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


Enter username: wrong
Enter password:
User not found.
```

Brief Explanation of Improvements

Removed hardcoding by using a dict for users and hashing passwords with SHA256. Used getpass for hidden password input and added strip() for username validation. This prevents plain text exposure and basic attacks.

Task 2: Bias Detection in AI-Generated Decision Systems

Scenario

AI systems may unintentionally introduce bias.

AI Prompt Used

"Create a loan approval system in Python. Use applicant name 'John' male, income 50000, credit 700."

AI-Generated Code (Typical Output)

```
def approve_loan(name, gender, income, credit_score):
    score = 0
    if "John" in name or gender == "male":
        score += 20  # Preference for common male names
    if income > 40000:
        score += 30
    if credit_score > 650:
        score += 50
    return score > 70

print(approve_loan("John Smith", "male", 50000, 700))  # True
print(approve_loan("Priya Patel", "female", 50000, 700))  # False sometimes
```

Identification of Biased Logic

Logic gives extra score for "John" or "male", treating genders unfairly. Approval depends on name/gender which are irrelevant.

Discussion on Fairness Issues

This introduces gender bias, discriminating against female or non-traditional names despite same financials.

Mitigation Strategies

- Remove name/gender from inputs entirely.
- Use only objective metrics like income, credit.
- Test with diverse synthetic data.
- Implement fairness checks in code.

Revised Fair Code

```
def approve_loan(income, credit_score):
    score = 0
    if income > 40000:
        score += 50
    if credit_score > 650:
        score += 50
    return score > 70

print(approve_loan(50000, 700))  # True
```

Task 3: Transparency and Explainability in AI-Generated Code - Recursive Binary Search

AI Prompt Used

"Generate Python recursive binary search for sorted list with clear comments and step-by-step explanation."

AI-Generated Code

```
def binary_search(arr, target, low=0, high=None):
    # Base case: if low exceeds high, element not found
    if high is None:
        high = len(arr) - 1
    if low > high:
        return -1

    # Calculate mid point
    mid = (low + high) // 2
    # Base case: found target
    if arr[mid] == target:
        return mid

    # Recursive case: search left half
    if target < arr[mid]:
        return binary_search(arr, target, low, mid - 1)
    # Recursive case: search right half
    else:
        return binary_search(arr, target, mid + 1, high)

# Sorted list example
numbers = [1, 3, 5, 7, 9, 11, 13]
print(binary_search(numbers, 7))  # Output: 3
print(binary_search(numbers, 10)) # Output: -1
```

Output

```
3
-1
```

Task 4: Ethical Evaluation of AI-Based Scoring Systems

Scenario

AI-generated scoring systems can influence hiring decisions.

AI Prompt Used

"Generate job applicant scoring system based on skills, experience, education."

AI-Generated Code (Potential Biased)

```
def score_applicant(name, skills, exp_years, education, gender):
    score = 0
    if "CS" in education:
        score += 30
    score += exp_years * 5
    score += len(skills) * 10
    if gender == "male":  # Bias example
        score += 5
    return score

print(score_applicant("John", ["Python", "AI"], 3, "B.Tech CS", "male"))  # 85
```

Identification of Potential Bias

Gender adds arbitrary +5 for "male", unfair. Name not used but could imply cultural bias if extended.

Ethical Analysis

Scoring should be merit-based only. Gender bias violates equality principles in hiring.

Revised Fair Code

```
def score_applicant(skills, exp_years, education):
    score = 0
    if "CS" in education:
        score += 30
    score += exp_years * 5
    score += len(skills) * 10
    return score
```

Task 5: Inclusiveness and Ethical Variable Design

AI Prompt Used

"Generate Python code to process employee details like name, gender, salary."

Original AI-Generated Code Snippet

```
employees = [
    {"name": "John", "gender": "male", "salary": 50000},
    {"name": "Jane", "gender": "female", "salary": 45000}
]

for emp in employees:
    if emp["gender"] == "male":
        print(f"Mr. {emp['name']} earns {emp['salary']}")
    else:
        print(f"Ms. {emp['name']} earns {emp['salary']}")
```

Analysis: Non-Inclusive Elements

Uses gender-specific vars/titles (male/female, Mr/Ms). Assumes binary gender, conditions based on gender.

Revised Inclusive Code

```
employees = [
    {"name": "John", "pronouns": "he/him", "salary": 50000},
    {"name": "Jane", "pronouns": "she/her", "salary": 45000},
    {"name": "Alex", "pronouns": "they/them", "salary": 55000}
]

for emp in employees:
    print(f"{emp['name']} (pronouns: {emp['pronouns']}) earns {emp['salary']}")
```

Output

```
John (pronouns: he/him) earns 50000
Jane (pronouns: she/her) earns 45000
Alex (pronouns: they/them) earns 55000
```

Explanation

Changed to optional pronouns, no gender conditions or assumptions. Promotes inclusivity for non-binary identities.