Assignment – 5.1

Name: B.Rithwik

Hall Ticket No: 2303A52330

Batch: 35

Task 1: Privacy in API Usage

AI-generated code may expose API keys by hardcoding them.

Secure practice involves storing API keys in environment variables to prevent leakage.

Prompt: *Generate Python code to fetch weather data securely without exposing API keys in the code.*

AI-Generated Code:

```
import requests

API_KEY = "abcd1234MYSECRETKEY"

url = f"https://api.openweathermap.org/data/2.5/weather?q=London&appid={API_KEY}"

response = requests.get(url)

print(response.json())
```

Issues Identified:

API key is hardcoded in the source code.

Anyone accessing the file can misuse the API key.

Violates secure coding and privacy best practices.

Revised Secure Code (Ethical Version):

```
import os

import requests
```

```
api_key = os.getenv("WEATHER_API_KEY")

if not api_key:

   raise Exception("API key not found")

url = f"https://api.openweathermap.org/data/2.5/weather?q=London&appid={api_key}"

response = requests.get(url)

print(response.json())
```

Ethical Reflection

Developers must ensure that sensitive credentials are not exposed in source code. AI-generated code should always be reviewed and improved to follow secure development practices.

Task 2: Privacy & Security in File Handling

Storing passwords in plain text is unsafe.

Using hashing techniques protects sensitive data.

Prompt: *Generate a Python script that stores user data such as name, email, and password in a file.*

AI-Generated Code:

```
name = input("Enter name: ")

email = input("Enter email: ")

password = input("Enter password: ")

with open("users.txt", "a") as file:

   file.write(f"{name},{email},{password}\n")
```

Issues Identified:

- Passwords are stored in plain text.
- Anyone with file access can read sensitive information.
- High risk of data breach and privacy violation.
- Does not follow ethical data protection practices.

Revised Secure Code (Ethical Version):

```
import hashlib

name = input("Enter name: ")

email = input("Enter email: ")

password = input("Enter password: ")

hashed_password = hashlib.sha256(password.encode()).hexdigest()


with open("users.txt", "a") as file:

    file.write(f"{name},{email},{hashed_password}\n")
```

Ethical Reflection

Storing passwords securely is a fundamental developer responsibility. AI-generated code must be reviewed to ensure sensitive user data is protected using encryption or hashing techniques.

Task 3: Transparency in Algorithm Design

Armstrong number checking code should be well-commented and explained line-by-line.

Prompt: *Generate a Python function to check whether a number is an Armstrong number and explain the code line by line.*

AI-Generated Code:

```
def is_armstrong(number):

    digits = str(number)

    power = len(digits)

    total = 0

    for digit in digits:

        total += int(digit) ** power

    return total == number
```

AI Explanation (Line-by-Line)

- The number is converted to a string to count digits.
- The length of the number is stored as the power.

- A variable is initialized to store the sum.
- Each digit is raised to the power of total digits.
- The final sum is compared with the original number.

Transparency Analysis

- The explanation matches the actual code behavior.
- Each step of the algorithm is clearly described.
- The logic is simple and easy to understand.
- No hidden or unexplained operations are present.

Ethical Reflection

Transparent explanations help developers understand and trust AI-generated code. Clear documentation ensures accountability and prevents misuse of algorithms.

Task 4: Transparency in Algorithm Comparison

Bubble Sort and Quick Sort implementations should clearly explain differences in logic and efficiency.

Prompt: *Generate Python code for Quick Sort and Bubble Sort, and include comments explaining step-by-step how each algorithm works and how they differ.*

AI-Generated Code – Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

AI-Generated Code – Quick Sort

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
```

```
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)
```

Comparative Explanation

- Bubble Sort works by repeatedly swapping adjacent elements if they are in the wrong order.
- It is easy to understand but inefficient for large datasets.
- Quick Sort uses a divide-and-conquer approach by selecting a pivot and partitioning the list.
- Quick Sort is significantly faster for large inputs.

Efficiency Comparison

- Bubble Sort Time Complexity: $O(n^2)$
- Quick Sort Average Time Complexity: $O(n \log n)$

Ethical Reflection

Comparing algorithms transparently allows developers to make informed decisions. AI-generated code should clearly explain performance trade-offs to avoid misuse or inefficient implementations.

Task 5: Transparency in AI Recommendations

Recommendation systems must explain why suggestions are made to users.

Prompt: *Generate a simple product recommendation system in Python that also explains the reason for each recommendation.*

AI-Generated Code:

```
def recommend_products(user_interest):
    products = {
        "technology": ["Laptop", "Smartphone"],
        "fitness": ["Dumbbells", "Yoga Mat"],
```

```
    "music": ["Guitar", "Headphones"]
}
if user_interest in products:
    for product in products[user_interest]:
        print(f"Recommended: {product} because you are interested in {user_interest}")
else:
    print("No recommendations available")
```

Explainability Analysis

Each recommendation clearly states why it was suggested.

The system avoids black-box behavior.

Users can understand how their interests affect recommendations.

No hidden or biased decision logic is present.

Ethical Reflection

AI systems should always provide understandable explanations for their decisions. Explainable recommendations build user trust, reduce bias, and ensure accountability in AI-assisted decision-making.

Conclusion:

This lab demonstrates that while AI tools can assist in software development, they may generate insecure or non-transparent code. Developers are responsible for reviewing, correcting, and ethically improving AI-generated outputs. Secure coding, transparency, and accountability are essential principles of responsible AI usage.