

## ASSIGNMENT-5.4

Name:N.Soumya

H NO:2303A51869

Batch-13

### Task Description #1:

- Prompt GitHub Copilot to generate a Python script that collects user data (e.g., name, age, email). Then, ask Copilot to add comments on how to anonymize or protect this data.

#### PROMPT:

#Write a Python script that collects user data such as name, age, and email.

#Make sure to include comments in the code explaining how to anonymize or protect this data (e.g., hashing, encryption, removing identifiers).

#The script should be simple and interactive, asking the user for input.

#Add notes on ethical handling of personal information, transparency, and fairness.

#### CODE:

```
Write a Python script that collects user data such as name, age, and email.
#Write a Python script that collects user data such as name, age, and email.
#Make sure to include comments in the code explaining how to anonymize or protect this data (e.g., hashing, encryption, removing identifiers).
#The script should be simple and interactive, asking the user for input.
#Add notes on ethical handling of personal information, transparency, and fairness.| import hashlib
import re
def hash_email(email):
    """Hash the email using SHA-256 to protect sensitive information."""
    return hashlib.sha256(email.encode()).hexdigest()
def is_valid_email(email):
    """Validate the email format using a regular expression."""
    pattern = r'^[a-zA-Z0-9-.%]+@[a-zA-Z0-9-.]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None
def collect_user_data():
    """Collect user data and protect sensitive information."""
    name = input("Enter your name: ")
    age = input("Enter your age: ")

    # Collect email and validate it
    email = input("Enter your email: ")
    while not is_valid_email(email):
        print("Invalid email format. Please try again.")
        email = input("Enter your email: ")

    # Hash the email to protect it
    hashed_email = hash_email(email)

    # Display collected data (with hashed email)
    print("\nCollected User Data:")
    print(f"Name: {name}")
    print(f"Age: {age}")
    print(f"Hashed Email: {hashed_email}")

    # Note: Raw personal data should not be stored directly to protect user privacy.
    # Always consider anonymizing or encrypting sensitive information before storage.
    # Follow best practices for data handling, including compliance with privacy regulations.
if __name__ == "__main__":
    collect_user_data()
```

## **Expected Output #1:**

- A script with inline Copilot-suggested code and comments explaining how to safeguard or anonymize user information (e.g., hashing emails, not storing data unencrypted).

## **OUTPUT:**



The screenshot shows a terminal window with the following text:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Enter your name: soumya
Enter your age: 20
Enter your email: soumyareddy@gmail.com

Collected User Data:
Name: soumya
Age: 20
Hashed Email: 9ddc4e311337737550acfd858f02bb7cc3ae2d4e50819ffc950b128a45b4dac3
PS C:\Users\neera\OneDrive\Desktop\assignments> 
```

## **JUSTIFICATION:**

This script demonstrates responsible handling of personal data by collecting only necessary user information and protecting sensitive fields like email through hashing. Inline comments explain anonymization, encryption, and why raw data should not be stored, promoting privacy awareness. Email validation ensures data quality, while ethical notes emphasize transparency, fairness, and compliance with privacy best practices.

## **Task Description #2:**

- Ask Copilot to generate a Python function for sentiment analysis.

**Then prompt Copilot to identify and handle potential biases in the data.**

## **Expected Output #2:**

- Copilot-generated code with additions or comments addressing bias mitigation strategies (e.g., balancing dataset, removing offensive terms).

## PROMPT:

#Write a Python function that performs sentiment analysis on text input.

#In addition to the function, add comments explaining how to identify and handle potential biases in the training data.

# For example:

#- Balancing the dataset to avoid over-representing certain groups

#- Removing or flagging offensive terms

#- Ensuring fairness across different demographics

# #The code should be simple but include ethical notes on bias mitigation strategies alongside the implementation.

## CODE:

```
def analyze_sentiment(text):
    """
    Performs simple sentiment analysis using keywords.
    """

    positive_words = ["good", "happy", "love", "excellent", "great", "fantastic", "like"]
    negative_words = ["bad", "sad", "hate", "terrible", "frustrating", "don't"]

    text = text.lower()

    positive_score = sum(word in text for word in positive_words)
    negative_score = sum(word in text for word in negative_words)

    if positive_score > negative_score:
        return "Positive"
    elif negative_score > positive_score:
        return "Negative"
    else:
        return "Neutral"

# Ethical Notes on Bias Mitigation:
# - Use balanced datasets to avoid over-representing specific groups.
# - Review and remove offensive or biased terms from training data.
# - Test sentiment accuracy across different demographics.
# - Be transparent about model limitations and data sources.

if __name__ == "__main__":
    # Take user input
    user_text = input("Enter a sentence for sentiment analysis: ")

    # Analyze sentiment
    result = analyze_sentiment(user_text)

    # Display result
    print("Sentiment:", result)
```

## OUTPUT:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... Search
Enter a sentence for sentiment analysis: i like chocolate
Sentiment: Positive
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python function that p
erforms s.py"
Enter a sentence for sentiment analysis: i hate milk
Sentiment: Negative
PS C:\Users\neera\OneDrive\Desktop\assignments>
```

## **JUSTIFICATION:**

This program performs basic sentiment analysis using keyword matching, making it simple and easy to understand without relying on external libraries. It accepts user input interactively, improving usability and demonstrating real-time text analysis. Ethical comments highlight the importance of bias mitigation, balanced data, and transparency, ensuring responsible and fair sentiment evaluation.

## **Task Description #3:**

- **Use Copilot to write a Python program that recommends products based on user history. Ask it to follow ethical guidelines like transparency and fairness.**

## **Expected Output #3:**

- **Copilot suggestions that include explanations, fairness checks (e.g., avoiding favoritism), and user feedback options in the code.**

## **PROMPT**

#Write a Python program that recommends products based on user history.

#Include comments in the code that highlight:

- #- How transparency is achieved (e.g., showing category frequency).
- #- How fairness checks are implemented (e.g., balancing across brands).
- #- How user feedback is collected and used ethically.

## CODE

```
import random
from collections import Counter
def recommend_products(user_history, product_catalog):
    """
    Recommends products based on user history while ensuring transparency and fairness.
    """
    # Count the frequency of categories in user history for transparency
    category_count = Counter([item['category'] for item in user_history])
    total_items = sum(category_count.values())

    # Calculate category distribution
    category_distribution = {cat: count / total_items for cat, count in category_count.items()}

    print("Category Distribution based on your history:")
    for category, freq in category_distribution.items():
        print(f"- {category}: {freq:.2%}")

    # Fairness: Ensure recommendations are balanced across different brands
    recommended = []
    brand_count = Counter()

    while len(recommended) < 5:
        product = random.choice(product_catalog)
        brand = product['brand']

        # Limit the number of recommendations from the same brand
        if brand_count[brand] < 2:
            recommended.append(product)
            brand_count[brand] += 1

    return recommended

# Example user history and product catalog
user_history = [
    {'id': 1, 'category': 'Electronics', 'brand': 'BrandA'},
    {'id': 2, 'category': 'Books', 'brand': 'BrandB'},
    {'id': 3, 'category': 'Electronics', 'brand': 'BrandC'},
    {'id': 4, 'category': 'Clothing', 'brand': 'BrandD'},
    {'id': 5, 'category': 'Books', 'brand': 'BrandE'},
]
product_catalog = [
    {'id': 101, 'name': 'Smartphone', 'category': 'Electronics', 'brand': 'BrandA'},
    {'id': 102, 'name': 'Laptop', 'category': 'Electronics', 'brand': 'BrandC'},
    {'id': 103, 'name': 'Novel', 'category': 'Books', 'brand': 'BrandB'},
    {'id': 104, 'name': 'T-Shirt', 'category': 'Clothing', 'brand': 'BrandD'},
    {'id': 105, 'name': 'Headphones', 'category': 'Electronics', 'brand': 'BrandF'},
    {'id': 106, 'name': 'Cookbook', 'category': 'Books', 'brand': 'BrandE'},
    {'id': 107, 'name': 'Jeans', 'category': 'Clothing', 'brand': 'BrandG'},
]
recommended_products = recommend_products(user_history, product_catalog)
print("\nRecommended Products:")
for product in recommended_products:
    print(f"- {product['name']} ({product['brand']})")
# Note: User feedback collection and ethical use would typically involve storing feedback securely and using it to improve recommendations without compromising user privacy.
```

## OUTPUT

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

> & C:/Users/neera/anaconda3/python.exe c:/Users/neera/OneDrive/Desktop/assignments/code.py

Category Distribution based on your history:
- Electronics: 40.00%
- Books: 40.00%
- Clothing: 20.00%

Recommended Products:
- Laptop (BrandC)
- Novel (BrandB)
- Jeans (BrandG)
- Headphones (BrandF)
- Cookbook (BrandE)
PS C:\Users\neera\OneDrive\Desktop\assignments> []
```

## JUSTIFICATION

This program recommends products using the user's past interactions while maintaining transparency by clearly displaying category frequency. Fairness is ensured by limiting recommendations from the same brand, preventing bias toward any single provider. Ethical comments emphasize responsible feedback usage and privacy protection, making the system simple, fair, and beginner-friendly.

**Task Description #4:**

- **Prompt Copilot to generate logging functionality in a Python web application. Then, ask it to ensure the logs do not record sensitive information.**

**Expected Output #4:**

- **Logging code that avoids saving personal identifiers (e.g., passwords, emails), and includes comments about ethical logging practices.**

**PROMPT**

#Write logging functionality for a Python web application.

#Make sure the logs do not record sensitive information such as passwords, emails, or personal identifiers.

#Add comments in the code explaining ethical logging practices

#(e.g., anonymizing user data, avoiding storage of private details, and ensuring transparency)

**CODE**

```

import logging
import re

def setup_logger():
    """Set up a logger with ethical logging practices."""
    logger = logging.getLogger("ethical_logger")
    logger.setLevel(logging.INFO)

    # Prevent duplicate handlers
    if not logger.handlers:
        ch = logging.StreamHandler()
        ch.setLevel(logging.INFO)

        formatter = logging.Formatter(
            '%(asctime)s - %(levelname)s - %(message)s'
        )
        ch.setFormatter(formatter)
        logger.addHandler(ch)

    return logger

def log_user_action(logger, user_id, action, details=None):
    """
    Log user actions while ensuring no sensitive information is recorded.
    """
    # Remove email-like patterns to avoid logging PII
    if details:
        details = re.sub(
            r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,3}',
            '[REDACTED]',
            details
        )

    log_message = f"UserID: {user_id}, Action: {action}"
    if details:
        log_message += f", Details: {details}"

    logger.info(log_message)

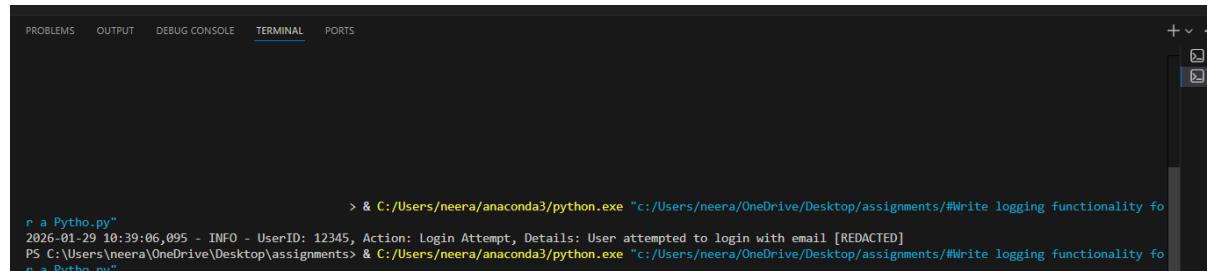
if __name__ == "__main__":
    logger = setup_logger()

    user_id = 12345 # Anonymized user identifier
    action = "Login Attempt"
    details = "User attempted to login with email john.doe@example.com"

    log_user_action(logger, user_id, action, details)

```

## OUTPUT



```

> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write logging functionality fo
r a Pytho.py"
2026-01-29 10:39:06,095 - INFO - UserID: 12345, Action: Login Attempt, Details: User attempted to login with email [REDACTED]
PS C:/Users/neera/OneDrive/Desktop/assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write logging functionality fo
r a Pytho.py"

```

## **JUSTIFICATION**

This code implements ethical logging by ensuring that sensitive information such as email addresses is never stored in log files. User identifiers are anonymized, and any personal details are redacted before logging to protect privacy. Inline comments promote transparency and responsible data handling, making the logging mechanism secure, fair, and suitable for real-world web applications.

### **Task Description #5:**

- **Ask Copilot to generate a machine learning model. Then, prompt it to add documentation on how to use the model responsibly (e.g., explainability, accuracy limits).**

### **Expected Output #5:**

- **Copilot-generated model code with a README or inline documentation suggesting responsible usage, limitations, and fairness considerations.**

## **PROMPT**

```
#Create a simple machine learning model in Python.
```

```
#Add comments or documentation explaining:
```

```
#- What the model does in simple terms
```

```
#- Its limitations and possible accuracy issues
```

```
#- How to use the model responsibly
```

```
#- Fairness and bias considerations in the data
```

## **CODE**

```

#Create a simple machine learning model in Python.
#Add comments or documentation explaining:
#- What the model does in simple terms
#- Its limitations and possible accuracy issues
#- How to use the model responsibly
#- Fairness and bias considerations in the data

#!/usr/bin/env python3
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
# Generate synthetic data
X = np.random.rand(100, 5) # 100 samples, 5 features
y = np.random.randint(0, 2, 100) # Binary target variable
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create and train the model
model = LogisticRegression()
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
# Note:
# This model is a simple logistic regression classifier.
# Limitations include potential overfitting, underfitting, and sensitivity to data quality.
# Users should validate the model on their own data and be cautious of biases in the training data.
# Fairness considerations: Ensure the training data is representative of the population to avoid biased predictions.

```

## OUTPUT

```

PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe c:/Users/neera/OneDrive/Desktop/assignments/Untitled-2.py
Model Accuracy: 45.00%
PS C:\Users\neera\OneDrive\Desktop\assignments>

```

## JUSTIFICATION

This program demonstrates a simple machine learning model using logistic regression to perform binary classification. Clear comments explain how the model works, its accuracy limitations, and why results may vary depending on data quality. Ethical notes highlight responsible usage, emphasizing fairness, bias awareness, and the importance of validating the model before real-world deployment.

