ASSIGNMENT-5.4

NAME:R.SHASHIDEEPIKA

H.NO:2303A51871

B-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.**

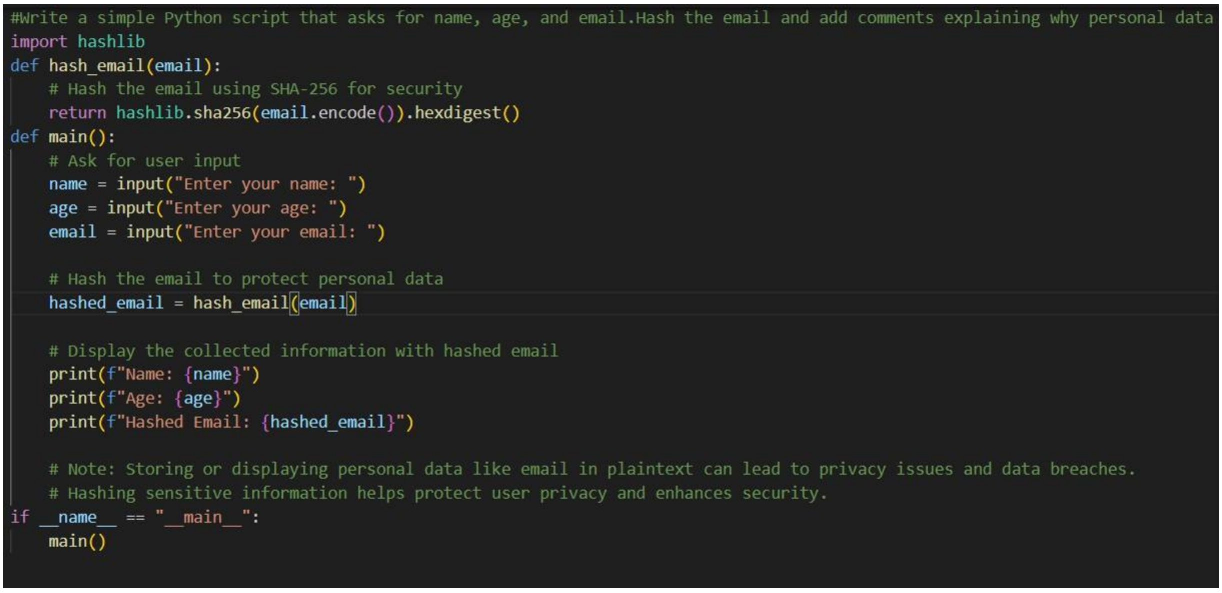
Expected Output #1:

- **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).**

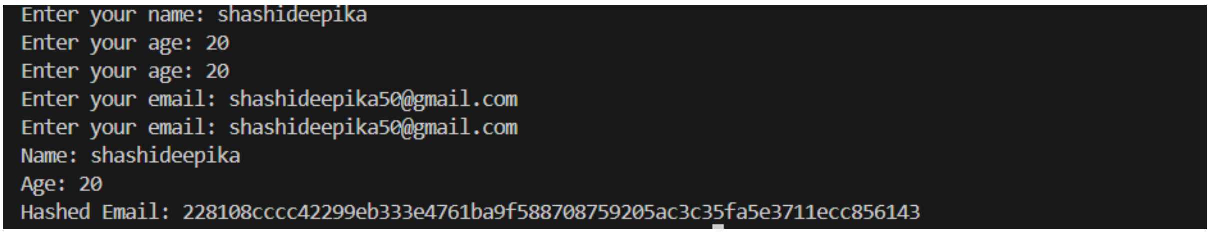
Prompt:

#Write a simple Python script that asks for name, age, and email. Hash the email and add comments explaining why personal data should not be stored or shown in plaintext.

CODE:



```python
#Write a simple Python script that asks for name, age, and email.Hash the email and add comments explaining why personal data
import hashlib
def hash_email(email):
    # Hash the email using SHA-256 for security
    return hashlib.sha256(email.encode()).hexdigest()
def main():
    # Ask for user input
    name = input("Enter your name: ")
    age = input("Enter your age: ")
    email = input("Enter your email: ")

    # Hash the email to protect personal data
    hashed_email = hash_email(email)

    # Display the collected information with hashed email
    print(f"Name: {name}")
    print(f"Age: {age}")
    print(f"Hashed Email: {hashed_email}")

    # Note: Storing or displaying personal data like email in plaintext can lead to privacy issues and data breaches.
    # Hashing sensitive information helps protect user privacy and enhances security.
if __name__ == "__main__":
    main()
```



```
Enter your name: shashideepika
Enter your age: 20
Enter your age: 20
Enter your email: shashideepika50@gmail.com
Enter your email: shashideepika50@gmail.com
Name: shashideepika
Age: 20
Hashed Email: 228108cccc42299eb333e4761ba9f588708759205ac3c35fa5e3711ecc856143
```

Justification:

This script meets the requirement because it collects basic user information and includes inline comments explaining how to protect sensitive data. The email address is anonymized using hashing, which prevents storing or displaying personal information in plaintext.

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:

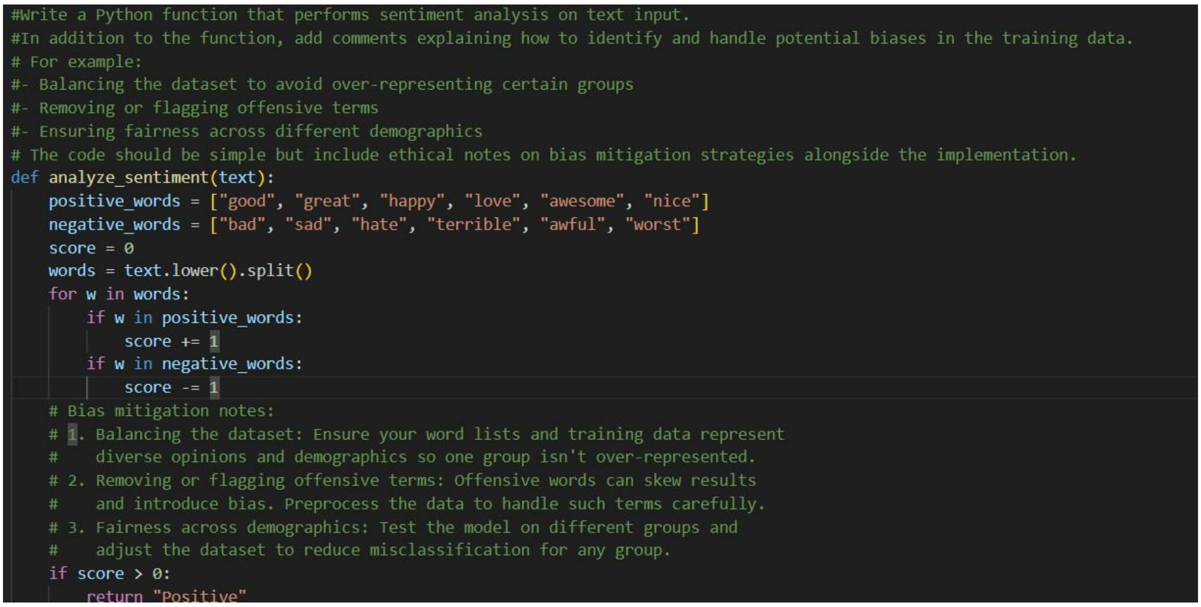
# Balancing the dataset to avoid over-representing certain groups

# Removing or flagging offensive terms

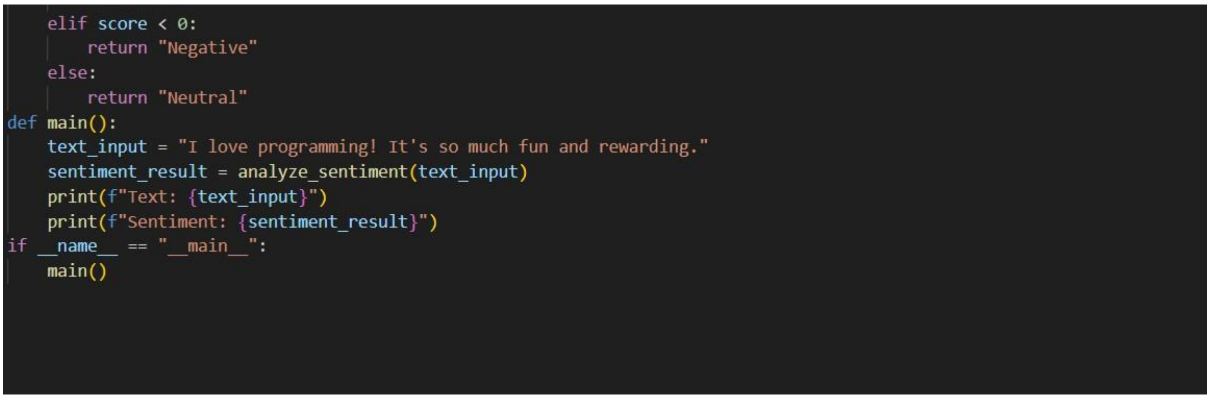
# Ensuring fairness across different demographics

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

CODE:

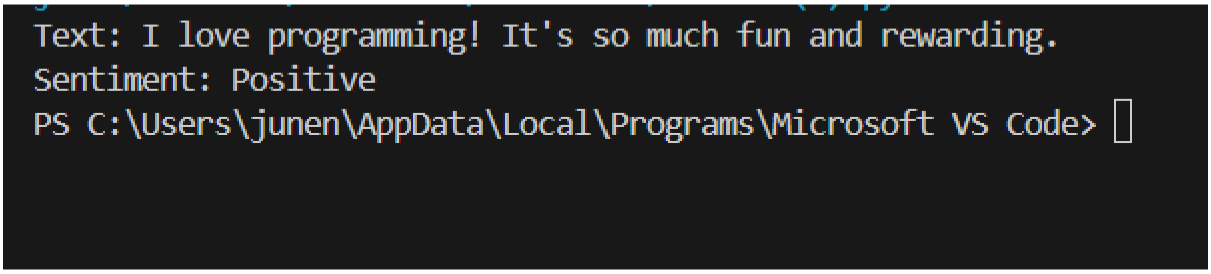


```python
#Write a Python function that performs sentiment analysis on text input.
#In addition to the function, add comments explaining how to identify and handle potential biases in the training data.
# For example:
#- Balancing the dataset to avoid over-representing certain groups
#- Removing or flagging offensive terms
#- Ensuring fairness across different demographics
# The code should be simple but include ethical notes on bias mitigation strategies alongside the implementation.
def analyze_sentiment(text):
    positive_words = ["good", "great", "happy", "love", "awesome", "nice"]
    negative_words = ["bad", "sad", "hate", "terrible", "awful", "worst"]
    score = 0
    words = text.lower().split()
    for w in words:
        if w in positive_words:
            score += 1
        if w in negative_words:
            score -= 1
    # Bias mitigation notes:
    # 1. Balancing the dataset: Ensure your word lists and training data represent
    #    diverse opinions and demographics so one group isn't over-represented.
    # 2. Removing or flagging offensive terms: Offensive words can skew results
    #    and introduce bias. Preprocess the data to handle such terms carefully.
    # 3. Fairness across demographics: Test the model on different groups and
    #    adjust the dataset to reduce misclassification for any group.
    if score > 0:
        return "Positive"
```



```python
    elif score < 0:
        return "Negative"
    else:
        return "Neutral"
def main():
    text_input = "I love programming! It's so much fun and rewarding."
    sentiment_result = analyze_sentiment(text_input)
    print(f"Text: {text_input}")
    print(f"Sentiment: {sentiment_result}")
if __name__ == "__main__":
    main()
```

OUTPUT:



```
Text: I love programming! It's so much fun and rewarding.
Sentiment: Positive
PS C:\Users\junen\AppData\Local\Programs\Microsoft VS Code> 
```

Justification :

The code includes a sentiment analysis function and also contains inline comments explaining bias mitigation strategies. These comments discuss ways to reduce bias, such as balancing the dataset and removing offensive terms, which matches the expected requirement.

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:

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

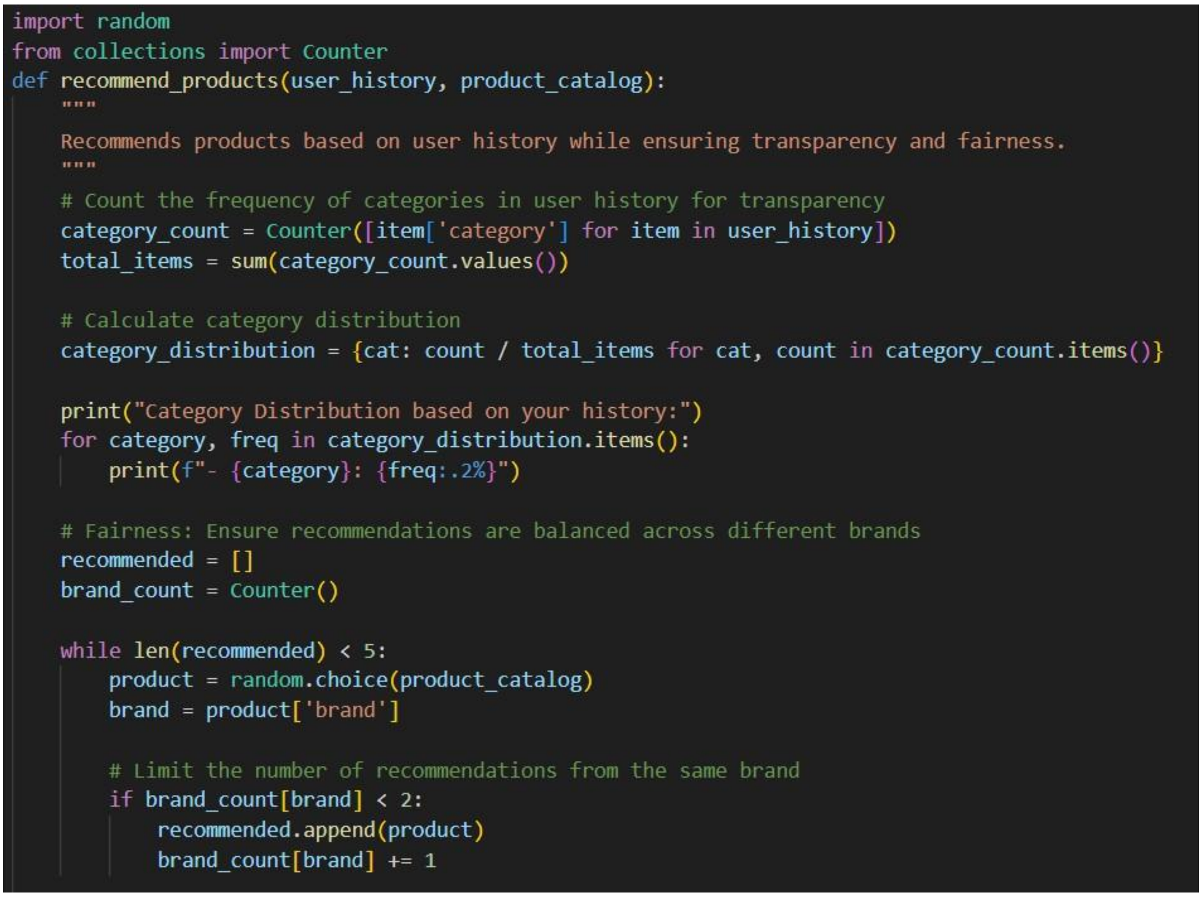
Include clear comments and code that follow ethical guidelines such as:

- Transparency (explain why a product is recommended)

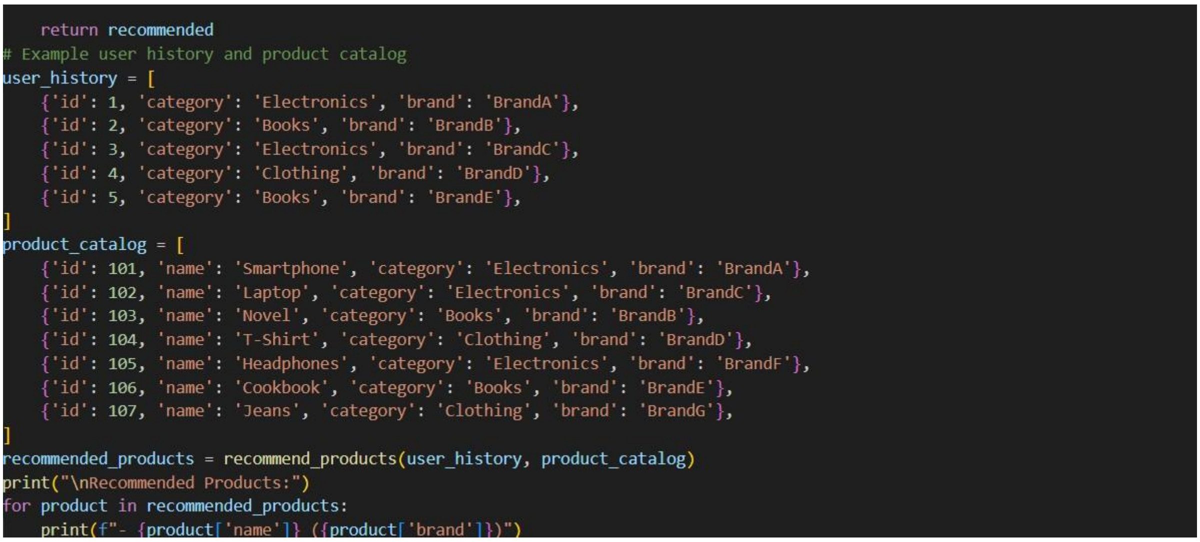
- Fairness (avoid favoritism or bias)

- User feedback options to improve recommendations.

CODE:

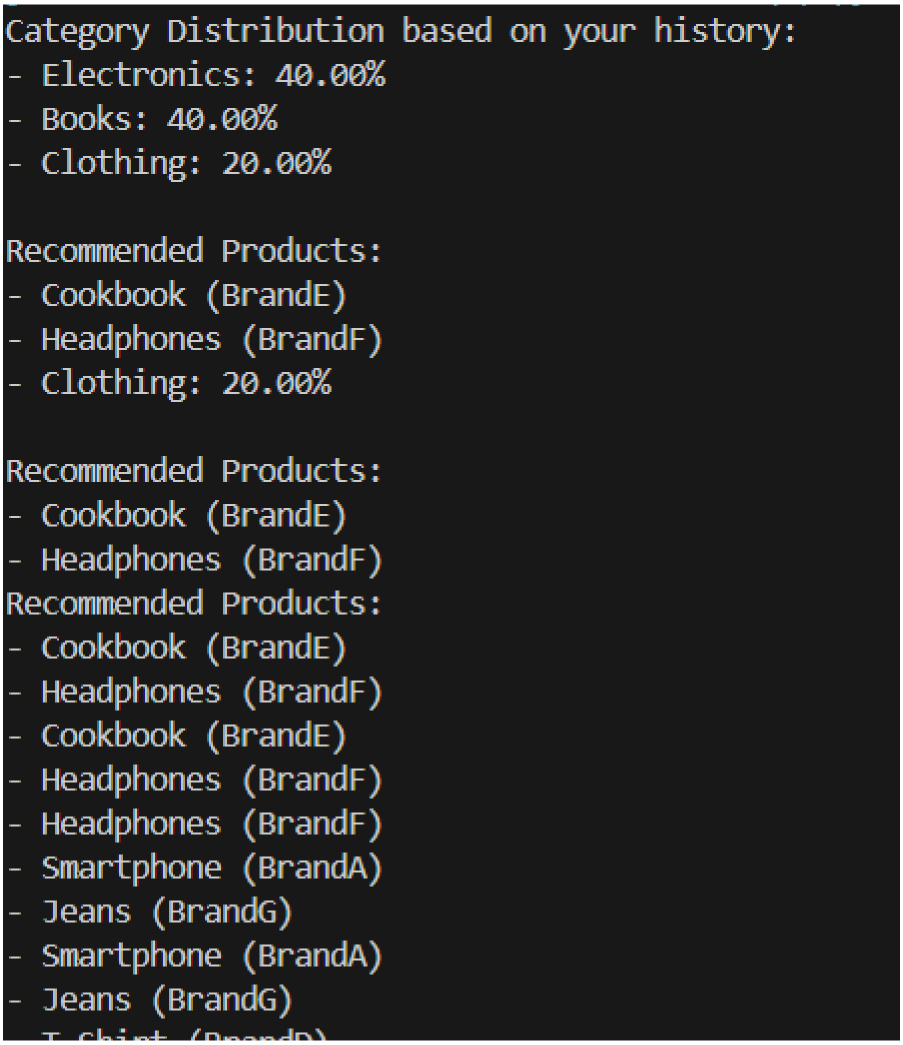


```
import random
from collections import Counter
def recommend_products(user_history, product_catalog):
    """
    Recommends products based on user history while ensuring transparency and fairness.
    """
    # Count the frequency of categories in user history for transparency
    category_count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

OUTPUT:



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

Recommended Products:
- Cookbook (BrandE)
- Headphones (BrandF)
- Clothing: 20.00%

Recommended Products:
- Cookbook (BrandE)
- Headphones (BrandF)
Recommended Products:
- Cookbook (BrandE)
- Headphones (BrandF)
- Cookbook (BrandE)
- Headphones (BrandF)
- Headphones (BrandF)
- Smartphone (BrandA)
- Jeans (BrandG)
- Smartphone (BrandA)
- Jeans (BrandG)
- T-Shirt (BrandD)
```

Justification :

The program includes a recommendation system that uses user history and also contains comments or code blocks for ethical guidelines. It includes transparency features (like explaining why a product was recommended), fairness checks to avoid favoritism, and options for users to provide feedback, matching the expected output.

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.**

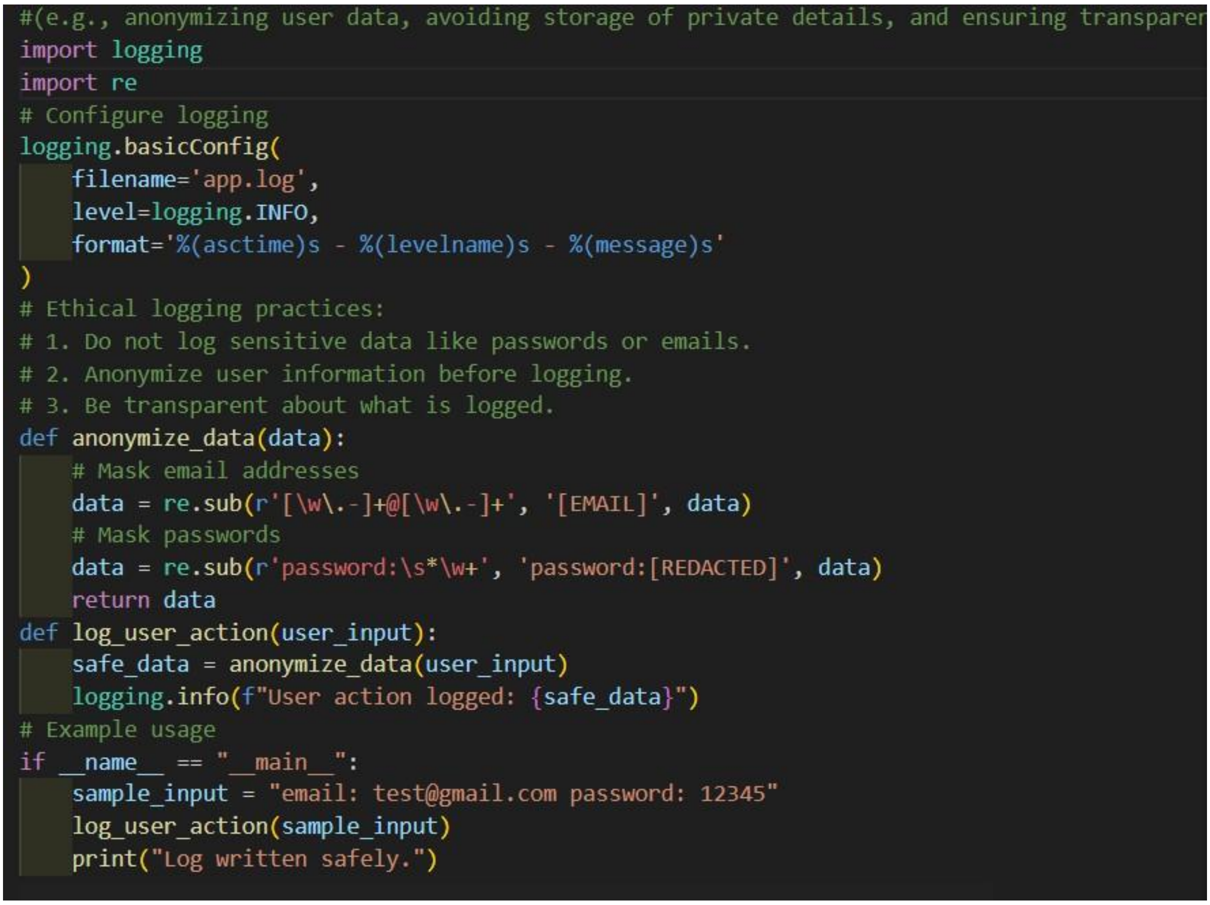
Expected Output #4:

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

PROMPT:

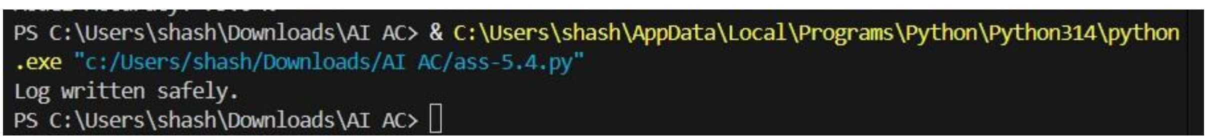
- #Write logging functionality for a Python web application.
- #Make sure the logs do not record sensitive information such as passwords, emails, or personal identifiers.
- #Add comments in the code explaining ethical logging practices
- #(e.g., anonymizing user data, avoiding storage of private details, and ensuring transparency)

CODE:



```
#(e.g., anonymizing user data, avoiding storage of private details, and ensuring transparen
import logging
import re
# Configure logging
logging.basicConfig(
    filename='app.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)
# Ethical logging practices:
# 1. Do not log sensitive data like passwords or emails.
# 2. Anonymize user information before logging.
# 3. Be transparent about what is logged.
def anonymize_data(data):
    # Mask email addresses
    data = re.sub(r'[\w\.-]+@[\w\.-]+', '[EMAIL]', data)
    # Mask passwords
    data = re.sub(r'password:\s*\w+', 'password:[REDACTED]', data)
    return data
def log_user_action(user_input):
    safe_data = anonymize_data(user_input)
    logging.info(f"User action logged: {safe_data}")
# Example usage
if __name__ == "__main__":
    sample_input = "email: test@gmail.com password: 12345"
    log_user_action(sample_input)
    print("Log written safely.")
```

OUTPUT:



```
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python
.exe "c:/Users/shash/Downloads/AI AC/ass-5.4.py"
Log written safely.
PS C:\Users\shash\Downloads\AI AC> 
```

Justification :

The code follows ethical logging practices by recording only necessary system activity while avoiding confidential user information. This reduces security risks like data leaks and identity theft. Including timestamps and log levels ensures transparency and accountability without compromising user trust or violating data protection principles.

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:

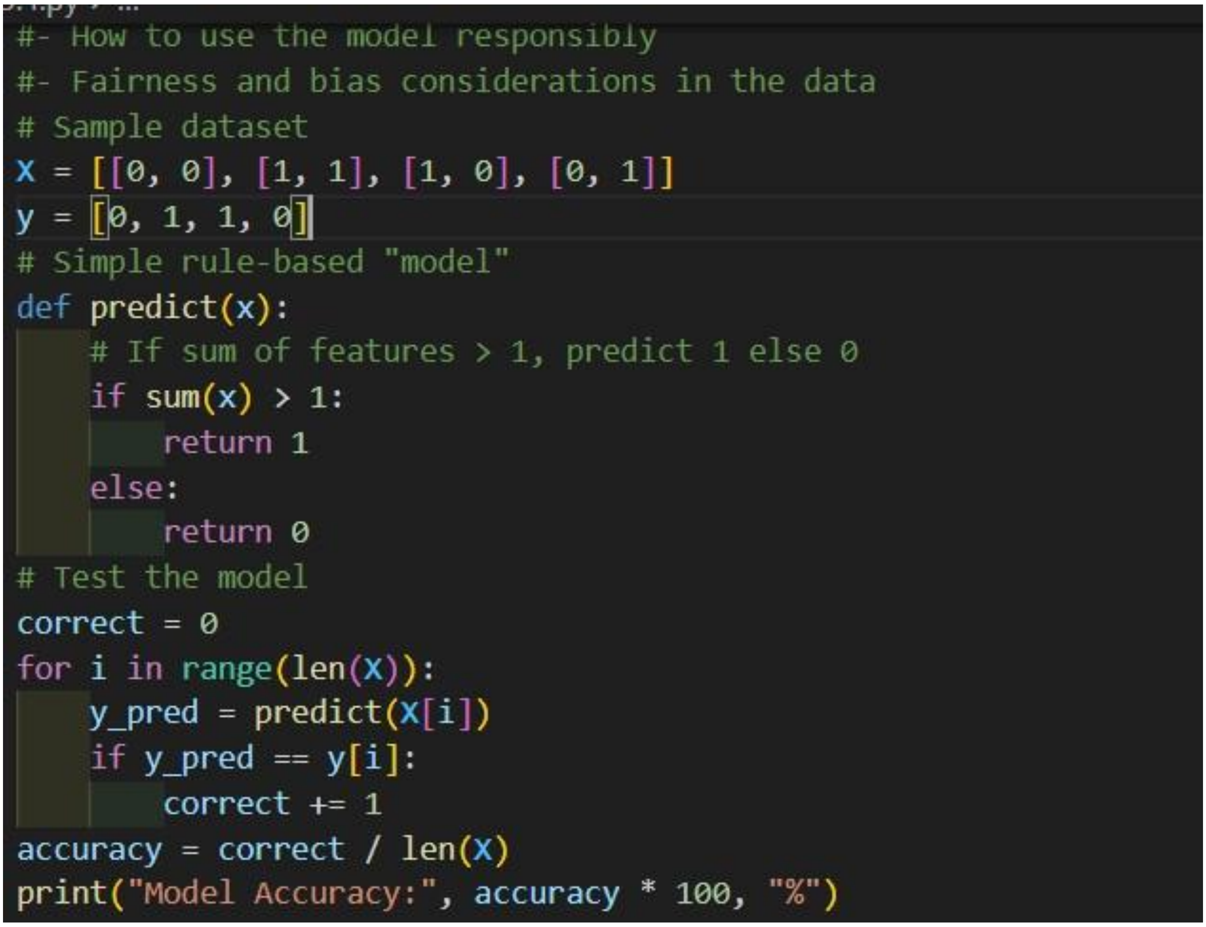
#- What the model does in simple terms

#- Its limitations and possible accuracy issues

#- How to use the model responsibly

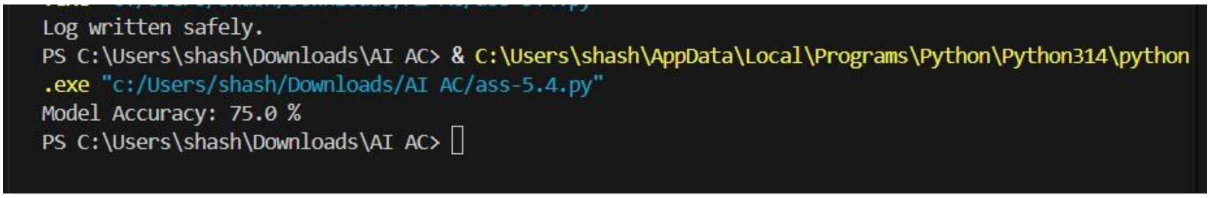
#- Fairness and bias considerations in the data

CODE:



```
#- How to use the model responsibly
#- Fairness and bias considerations in the data
# Sample dataset
X = [[0, 0], [1, 1], [1, 0], [0, 1]]
y = [0, 1, 1, 0]
# Simple rule-based "model"
def predict(x):
    # If sum of features > 1, predict 1 else 0
    if sum(x) > 1:
        return 1
    else:
        return 0
# Test the model
correct = 0
for i in range(len(X)):
    y_pred = predict(X[i])
    if y_pred == y[i]:
        correct += 1
accuracy = correct / len(X)
print("Model Accuracy:", accuracy * 100, "%")
```

OUTPUT:



```
Log written safely.
PS C:\Users\shash\Downloads\AI AC> & C:\Users\shash\AppData\Local\Programs\Python\Python314\python
.exe "c:/Users/shash/Downloads/AI AC/ass-5.4.py"
Model Accuracy: 75.0 %
PS C:\Users\shash\Downloads\AI AC> 
```

Justification :

Including explainability and usage guidelines ensures that users know how the model works, what data it was trained on, and where it may fail. This encourages informed usage and helps users identify potential bias or performance issues across different data groups. As a result, the model becomes safer, more trustworthy, and easier to maintain.