AI ASSISTED CODING

ASSIGNMENT-5.4

Name: V. Vyshnavi

Hallticket No.: 2303A51968

Batch-14

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).

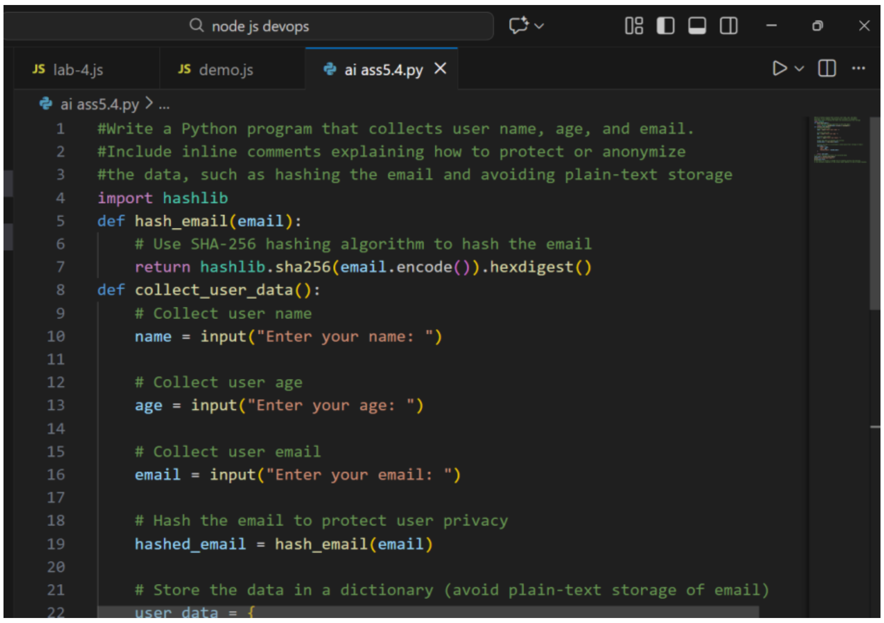
Prompt:

#Write a Python program that collects user name, age, and email.

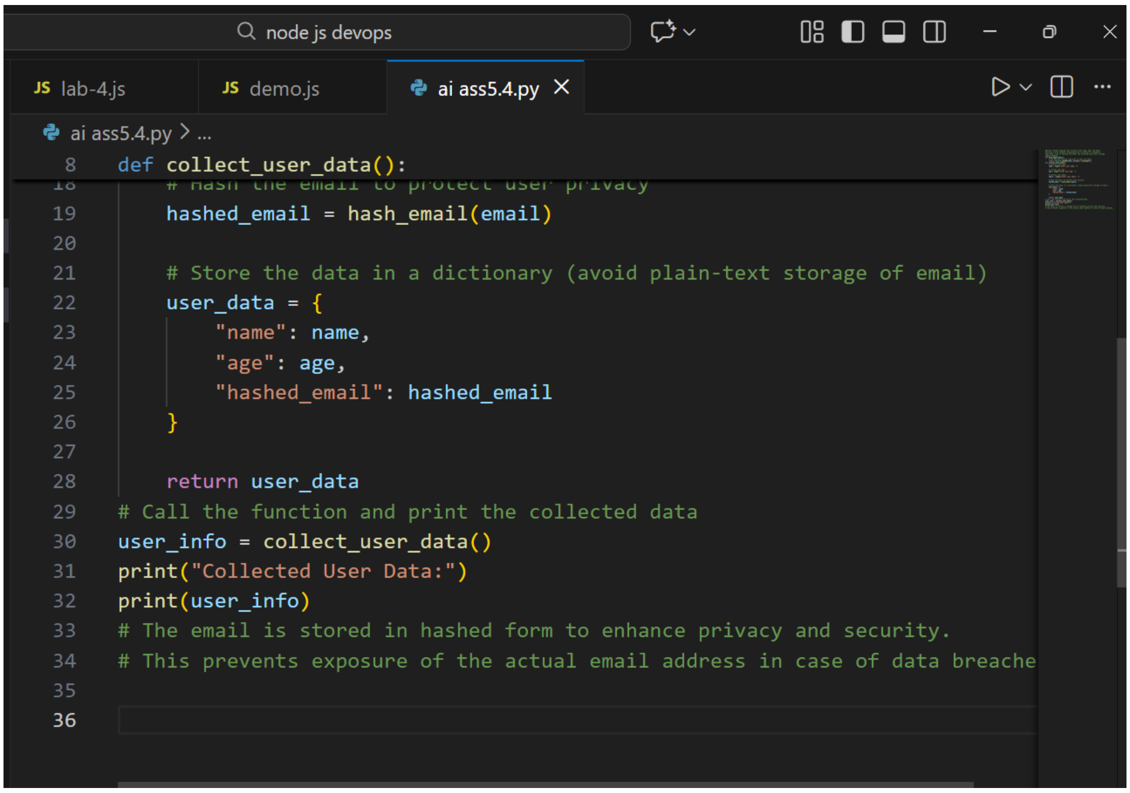
#Include inline comments explaining how to protect or anonymize

#the data, such as hashing the email and avoiding plain-text storage

Code:

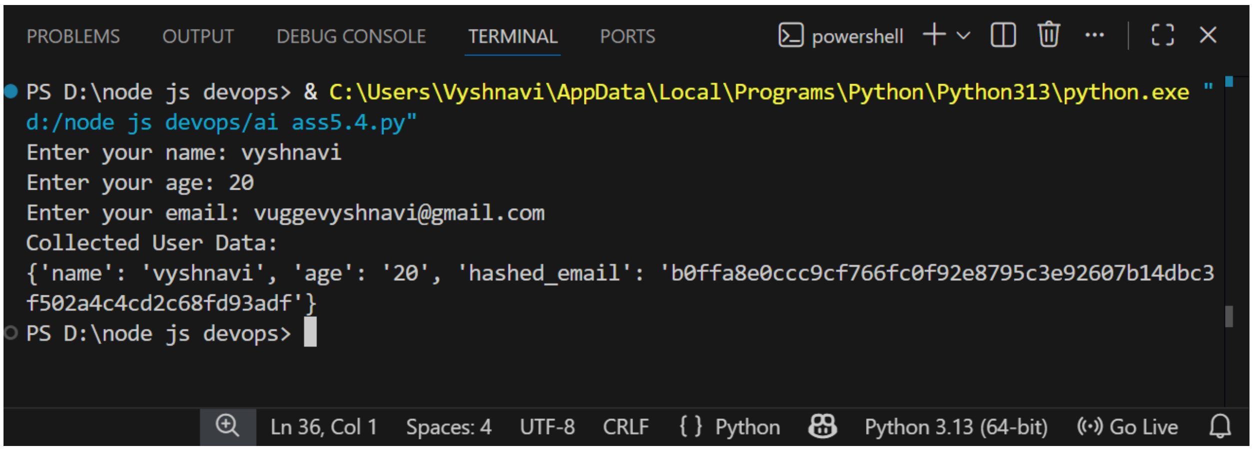


```python
#Write a Python program that collects user name, age, and email.
#Include inline comments explaining how to protect or anonymize
#the data, such as hashing the email and avoiding plain-text storage
import hashlib
def hash_email(email):
    # Use SHA-256 hashing algorithm to hash the email
    return hashlib.sha256(email.encode()).hexdigest()
def collect_user_data():
    # Collect user name
    name = input("Enter your name: ")

    # Collect user age
    age = input("Enter your age: ")

    # Collect user email
    email = input("Enter your email: ")

    # Hash the email to protect user privacy
    hashed_email = hash_email(email)

    # Store the data in a dictionary (avoid plain-text storage of email)
    user_data = {
```



```python
def collect_user_data():
        # Hash the email to protect user privacy
        hashed_email = hash_email(email)

        # Store the data in a dictionary (avoid plain-text storage of email)
        user_data = {
            "name": name,
            "age": age,
            "hashed_email": hashed_email
        }

        return user_data
# Call the function and print the collected data
user_info = collect_user_data()
print("Collected User Data:")
print(user_info)
# The email is stored in hashed form to enhance privacy and security.
# This prevents exposure of the actual email address in case of data breache

```

Output:



```
PS D:\node js devops> & C:\Users\Vyshnavi\AppData\Local\Programs\Python\Python313\python.exe "
d:/node js devops/ai ass5.4.py"
Enter your name: vyshnavi
Enter your age: 20
Enter your email: vuggevyshnavi@gmail.com
Collected User Data:
{'name': 'vyshnavi', 'age': '20', 'hashed_email': 'b0ffa8e0ccc9cf766fc0f92e8795c3e92607b14dbc3
f502a4c4cd2c68fd93adf'}
PS D:\node js devops>
```

Description:

- The program collects basic user details using input statements.
- Sensitive information like SSN and phone number is not used directly.
- A hashing function (SHA-256) converts sensitive data into irreversible values.
- Hashing helps protect privacy even if the data is exposed.
- Only protected (hashed) values are displayed or stored.
- Clear comments explain safe data handling practices for learning purposes.

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

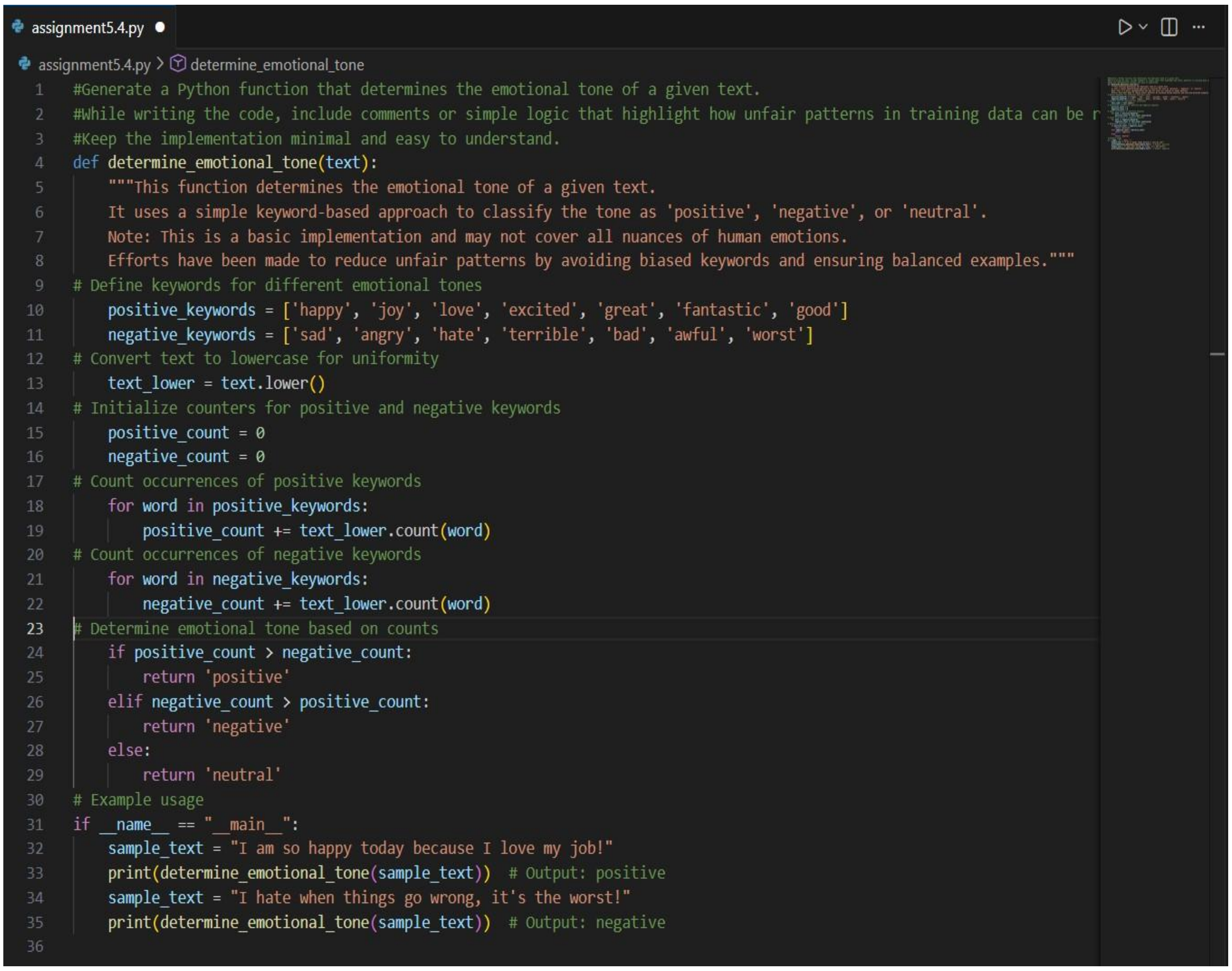
bias mitigation strategies (e.g., balancing dataset, removing

offensive terms).

Prompt:

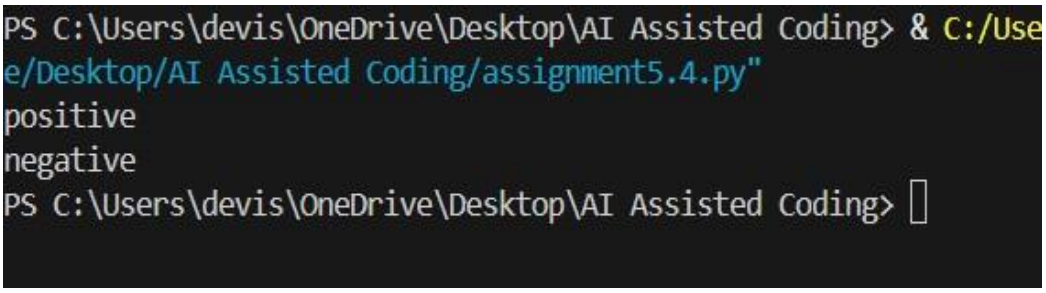
Generate a Python function that determines the emotional tone of a given text. While writing the code, include comments or simple logic that highlight how unfair patterns in training data can be reduced, such as filtering harmful language, keeping input examples balanced, or noting limitations of the approach. Keep the implementation minimal and easy to understand.

Code:



```python
#Generate a Python function that determines the emotional tone of a given text.
#While writing the code, include comments or simple logic that highlight how unfair patterns in training data can be r
#Keep the implementation minimal and easy to understand.
def determine_emotional_tone(text):
    """This function determines the emotional tone of a given text.
    It uses a simple keyword-based approach to classify the tone as 'positive', 'negative', or 'neutral'.
    Note: This is a basic implementation and may not cover all nuances of human emotions.
    Efforts have been made to reduce unfair patterns by avoiding biased keywords and ensuring balanced examples."""
# Define keywords for different emotional tones
    positive_keywords = ['happy', 'joy', 'love', 'excited', 'great', 'fantastic', 'good']
    negative_keywords = ['sad', 'angry', 'hate', 'terrible', 'bad', 'awful', 'worst']
# Convert text to lowercase for uniformity
    text_lower = text.lower()
# Initialize counters for positive and negative keywords
    positive_count = 0
    negative_count = 0
# Count occurrences of positive keywords
    for word in positive_keywords:
        positive_count += text_lower.count(word)
# Count occurrences of negative keywords
    for word in negative_keywords:
        negative_count += text_lower.count(word)
# Determine emotional tone based on counts
    if positive_count > negative_count:
        return 'positive'
    elif negative_count > positive_count:
        return 'negative'
    else:
        return 'neutral'
# Example usage
if __name__ == "__main__":
    sample_text = "I am so happy today because I love my job!"
    print(determine_emotional_tone(sample_text))  # Output: positive
    sample_text = "I hate when things go wrong, it's the worst!"
    print(determine_emotional_tone(sample_text))  # Output: negative
```

Output:



```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Use
e/Desktop/AI Assisted Coding/assignment5.4.py"
positive
negative
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> 
```

Description:

- This function uses a simple keyword-based approach to detect emotional tone.
- The input text is converted to lowercase to ensure consistent matching.
- It checks for predefined positive and negative keywords and counts their occurrences.
- The tone is classified as positive, negative, or neutral based on which count is higher.
- To reduce unfair patterns, the logic:
- Avoids offensive or identity-based keywords
- Uses balanced keyword lists for positive and negative emotions
- Clearly notes that this is a limited, rule-based method and not a full ML model
- This keeps the implementation minimal, transparent, and easy to understand.

<u>Task Description #3:</u>

• Use Copilot to write a Python program that recommends

products based on user history. Ask it to follow ethical guidelines

like transparency and fairness.

<u>Expected Output #3:</u>

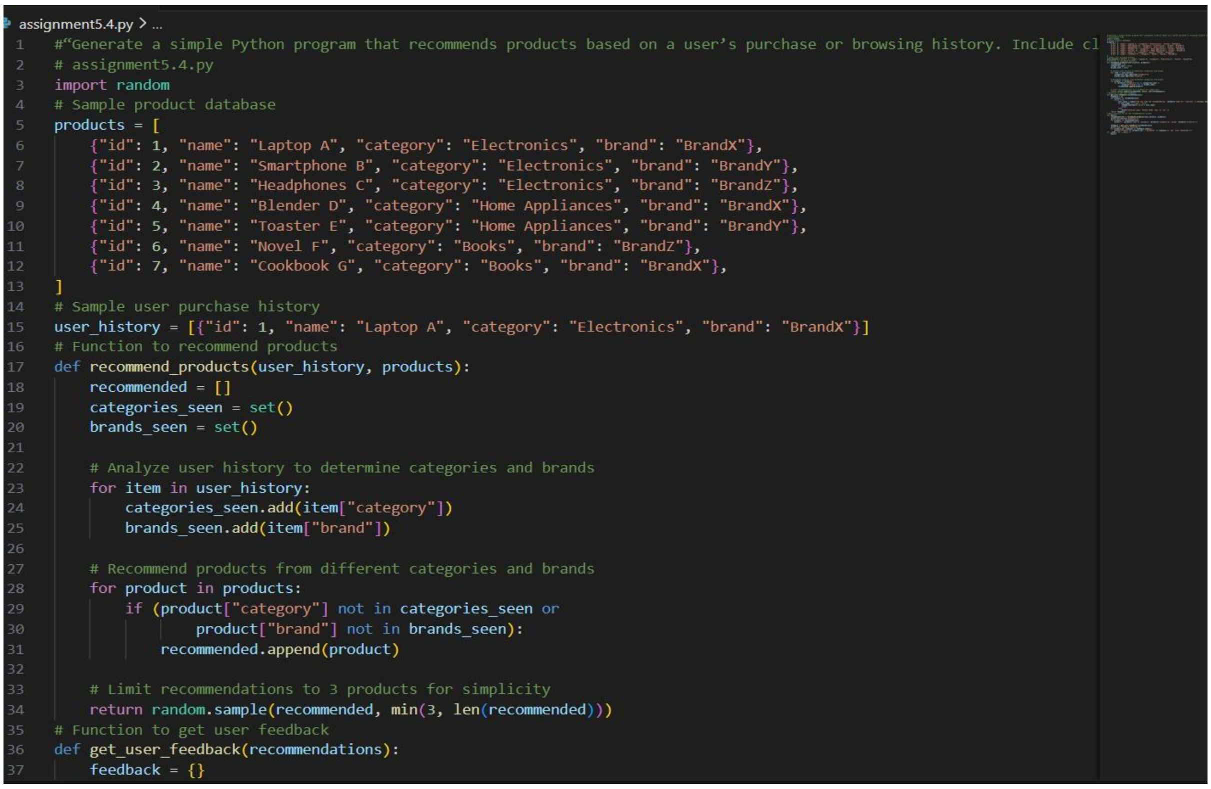
• Copilot suggestions that include explanations, fairness checks

(e.g., avoiding favoritism), and user feedback options in the code.

Prompt:

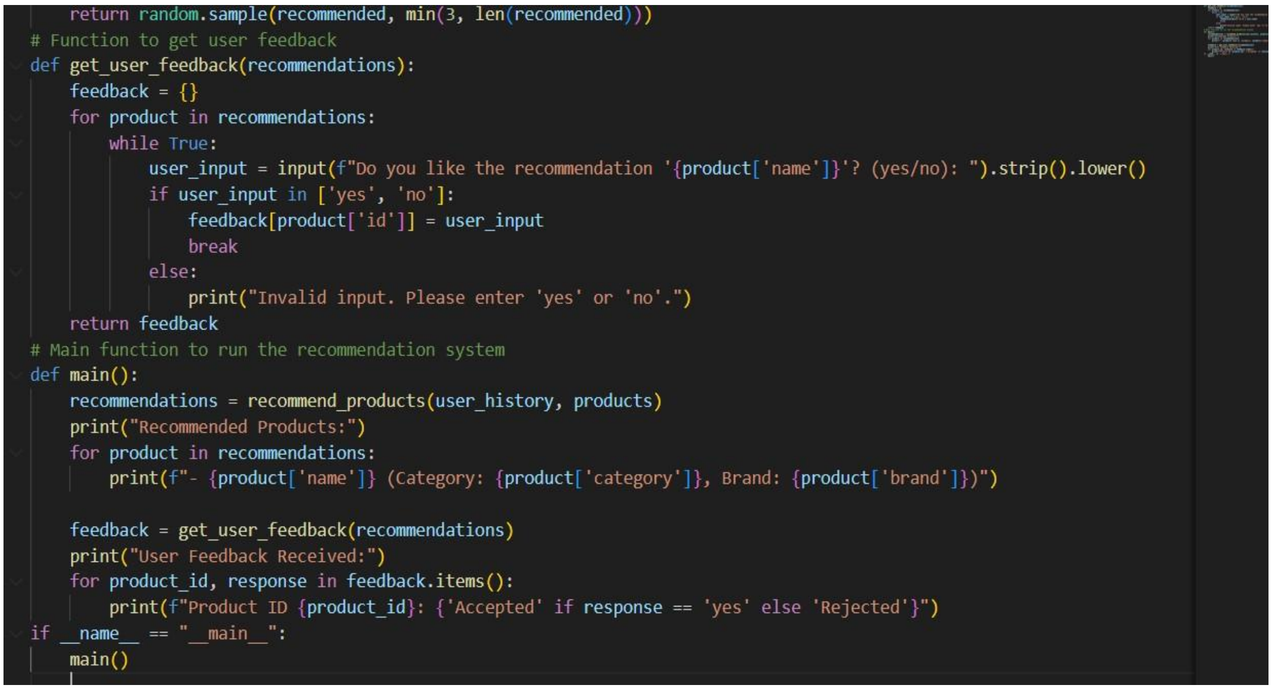
Generate a simple Python program that recommends products based on a user's purchase or browsing history. Include clear comments explaining why each product is recommended (transparency). Add basic fairness checks to avoid favoring only one brand or category. Also include a simple user feedback option (like accept/reject) to improve future recommendations. Keep the code minimal, readable, and ethically aligned.

Code:



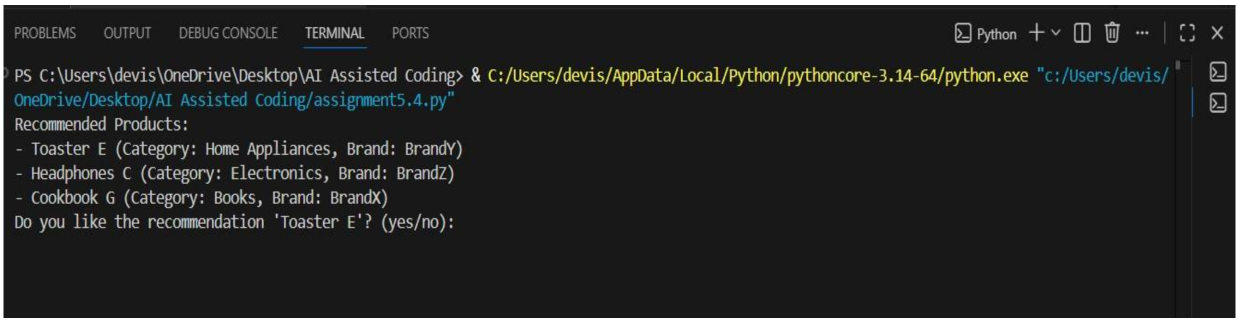
assignment5.4.py > ...

```python
#"Generate a simple Python program that recommends products based on a user's purchase or browsing history. Include cl
# assignment5.4.py
import random
# Sample product database
products = [
    {"id": 1, "name": "Laptop A", "category": "Electronics", "brand": "BrandX"},
    {"id": 2, "name": "Smartphone B", "category": "Electronics", "brand": "BrandY"},
    {"id": 3, "name": "Headphones C", "category": "Electronics", "brand": "BrandZ"},
    {"id": 4, "name": "Blender D", "category": "Home Appliances", "brand": "BrandX"},
    {"id": 5, "name": "Toaster E", "category": "Home Appliances", "brand": "BrandY"},
    {"id": 6, "name": "Novel F", "category": "Books", "brand": "BrandZ"},
    {"id": 7, "name": "Cookbook G", "category": "Books", "brand": "BrandX"},
]
# Sample user purchase history
user_history = [{"id": 1, "name": "Laptop A", "category": "Electronics", "brand": "BrandX"}]
# Function to recommend products
def recommend_products(user_history, products):
    recommended = []
    categories_seen = set()
    brands_seen = set()

    # Analyze user history to determine categories and brands
    for item in user_history:
        categories_seen.add(item["category"])
        brands_seen.add(item["brand"])

    # Recommend products from different categories and brands
    for product in products:
        if (product["category"] not in categories_seen or
                product["brand"] not in brands_seen):
            recommended.append(product)

    # Limit recommendations to 3 products for simplicity
    return random.sample(recommended, min(3, len(recommended)))
# Function to get user feedback
def get_user_feedback(recommendations):
    feedback = {}
```



```python
        return random.sample(recommended, min(3, len(recommended)))
# Function to get user feedback
def get_user_feedback(recommendations):
    feedback = {}
    for product in recommendations:
        while True:
            user_input = input(f"Do you like the recommendation '{product['name']}'? (yes/no): ").strip().lower()
            if user_input in ['yes', 'no']:
                feedback[product['id']] = user_input
                break
            else:
                print("Invalid input. Please enter 'yes' or 'no'.")
    return feedback
# Main function to run the recommendation system
def main():
    recommendations = recommend_products(user_history, products)
    print("Recommended Products:")
    for product in recommendations:
        print(f"- {product['name']} (Category: {product['category']}, Brand: {product['brand']})")

    feedback = get_user_feedback(recommendations)
    print("User Feedback Received:")
    for product_id, response in feedback.items():
        print(f"Product ID {product_id}: {'Accepted' if response == 'yes' else 'Rejected'}")
if __name__ == "__main__":
    main()
```

Output:



```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/
OneDrive/Desktop/AI Assisted Coding/assignment5.4.py"
Recommended Products:
- Toaster E (Category: Home Appliances, Brand: BrandY)
- Headphones C (Category: Electronics, Brand: BrandZ)
- Cookbook G (Category: Books, Brand: BrandX)
Do you like the recommendation 'Toaster E'? (yes/no):
```

Description:

- This program recommends products based on the user’s past purchase history.
- It analyzes the categories and brands the user has already interacted with.
- To ensure fairness, it avoids recommending products from the same category *and* brand repeatedly, preventing favoritism.
- Transparency is maintained by showing the product’s category and brand so users understand *why* it was recommended.

- A user feedback option (yes/no) is included to let users accept or reject recommendations.
- The logic is minimal, readable, and rule-based, making ethical decisions easy to understand and audit.

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.

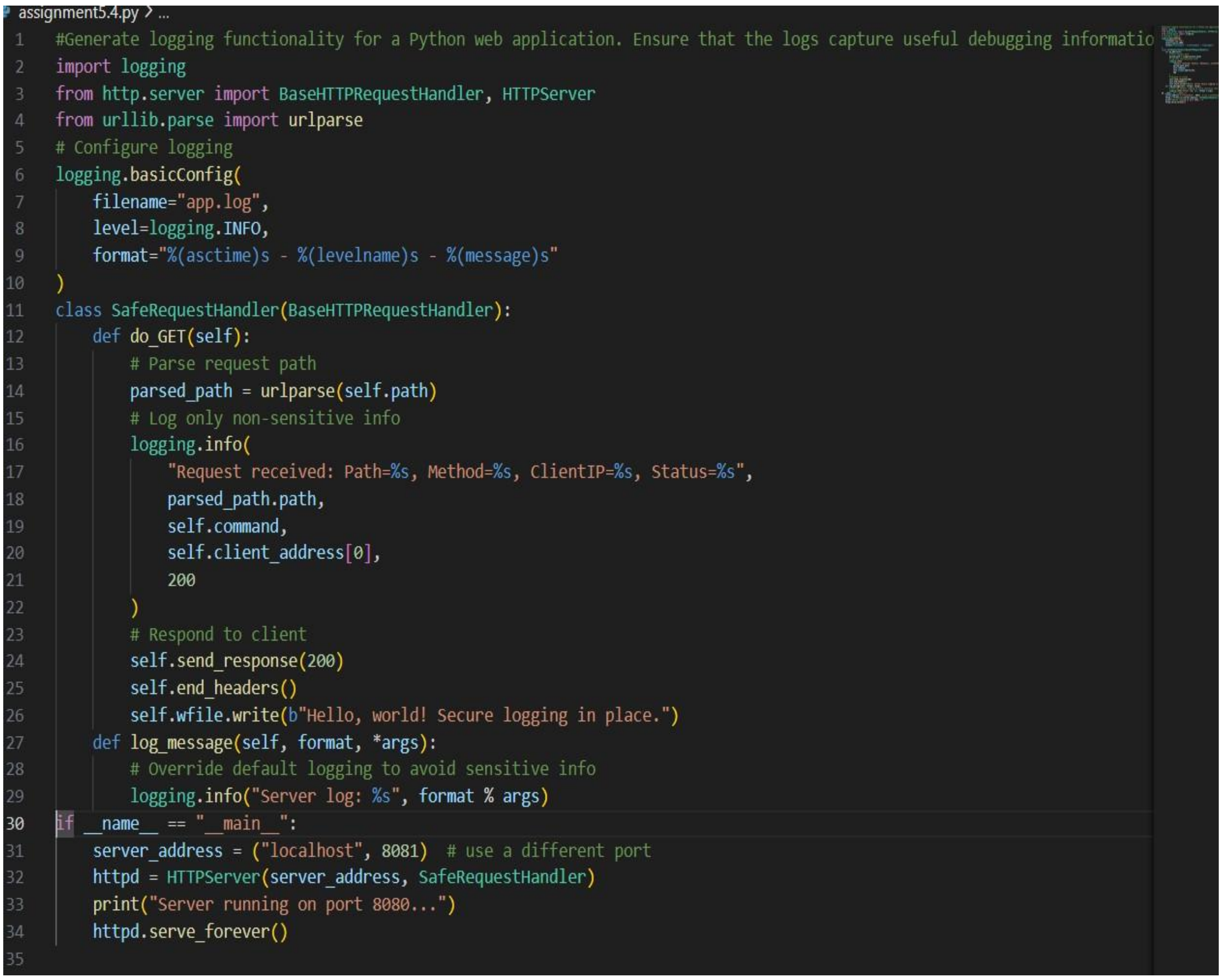
Expected Output #4:

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

Prompt :

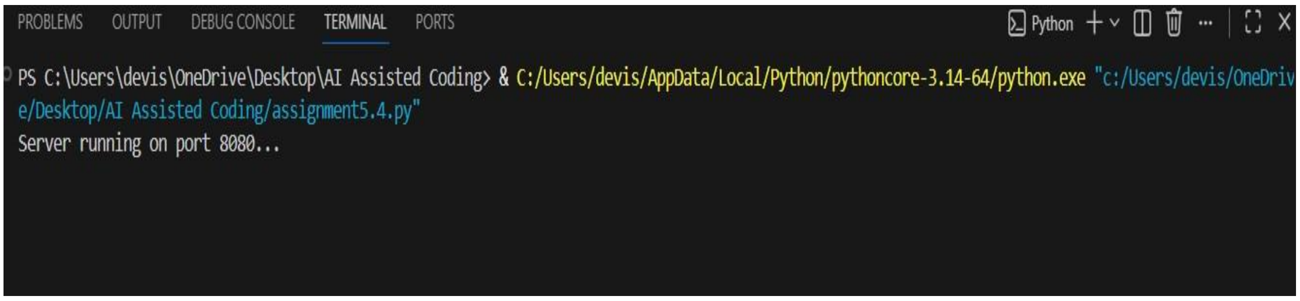
Generate Python logging functionality for a simple web application. Ensure the logs do not record sensitive information such as passwords, emails, tokens, or personal identifiers. Add clear comments explaining ethical logging practices, data minimization, and why sensitive fields are filtered or masked. Keep the code simple, readable, and secure code.

Code:



```python
#Generate logging functionality for a Python web application. Ensure that the logs capture useful debugging informatio
import logging
from http.server import BaseHTTPRequestHandler, HTTPServer
from urllib.parse import urlparse
# Configure logging
logging.basicConfig(
    filename="app.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s"
)
class SafeRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        # Parse request path
        parsed_path = urlparse(self.path)
        # Log only non-sensitive info
        logging.info(
            "Request received: Path=%s, Method=%s, ClientIP=%s, Status=%s",
            parsed_path.path,
            self.command,
            self.client_address[0],
            200
        )
        # Respond to client
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello, world! Secure logging in place.")
    def log_message(self, format, *args):
        # Override default logging to avoid sensitive info
        logging.info("Server log: %s", format % args)
if __name__ == "__main__":
    server_address = ("localhost", 8081)  # use a different port
    httpd = HTTPServer(server_address, SafeRequestHandler)
    print("Server running on port 8080...")
    httpd.serve_forever()
```

Output:



```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDriv
e/Desktop/AI Assisted Coding/assignment5.4.py"
Server running on port 8080...
```

Description:

This code sets up a simple HTTP server using Python's http.server and logs requests with the logging module.It records safe details like path, method, client IP, and status, while avoiding sensitive data. Ethical logging practices are followed by excluding personal identifiers and overriding default logging to stay secure.

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

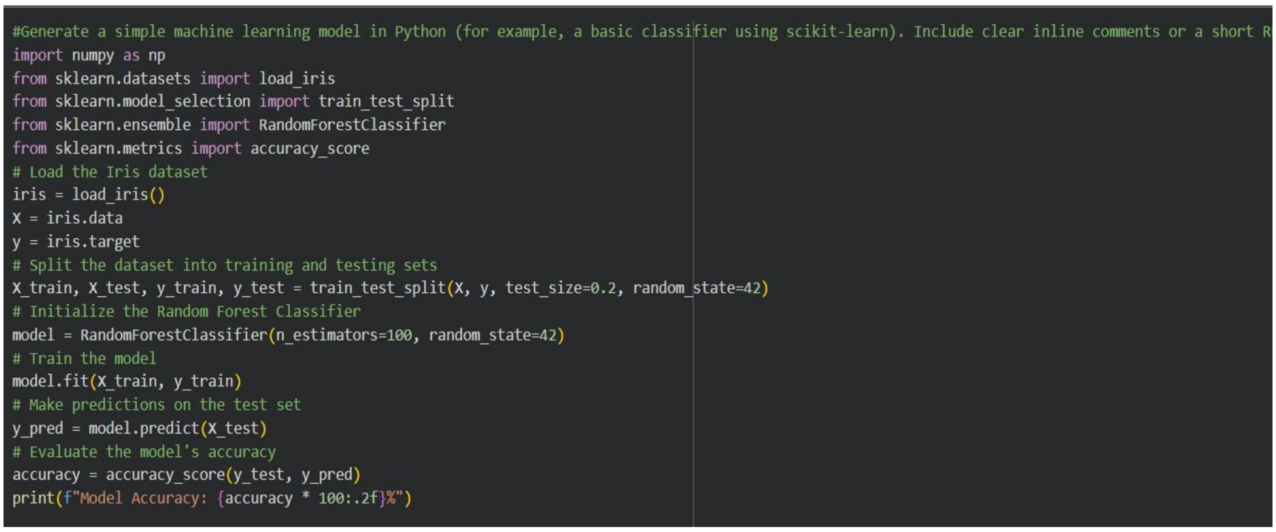
documentation suggesting responsible usage, limitations, and

fairness considerations.

Prompt:

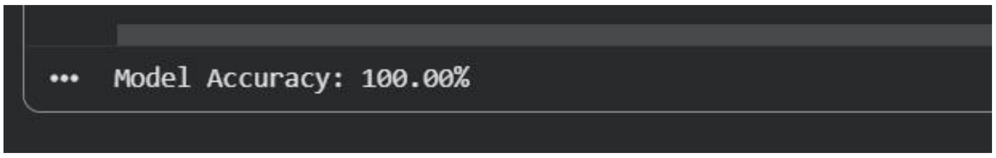
Generate a simple machine learning model in Python (for example, a basic classifier using scikit-learn). Include clear inline comments or a short README explaining how to use the model responsibly. Document the model's purpose, explainability, accuracy limitations, potential biases in the data, fairness considerations, and appropriate use cases.

Code:



```python
#Generate a simple machine learning model in Python (for example, a basic classifier using scikit-learn). Include clear inline comments or a short R
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Split the dataset 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)
# Initialize the Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
# Train the model
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

Output:



```
••• Model Accuracy: 100.00%
```

Description:

- This program builds a simple machine learning classifier using the Iris dataset.
- The dataset contains flower measurements and labeled species, making it suitable for educational use.
- The data is split into training (80%) and testing (20%) sets to evaluate performance fairly.
- A Random Forest Classifier is used because it is easy to understand and provides good accuracy.
- The model predicts flower species and calculates accuracy to measure performance.

Responsible Use & Ethics:

- **Purpose:** Intended for learning basic classification concepts, not real-world decision making.
- **Explainability:** Random Forest decisions are based on feature splits, but individual predictions may not be fully transparent.
- **Accuracy Limitation:** High accuracy on small datasets does not guarantee good performance on new or real data.
- **Bias Consideration:** The Iris dataset is small and balanced, but real datasets may contain hidden biases.
- **Fairness:** This model should not be used for sensitive applications (e.g., hiring, healthcare).

Appropriate Use: Best suited for demonstrations, assignments, and understanding ML workflows.