LAB 5.4 ASSIGNMENTS

HT.NO#2403A51276

Lab 5: Ethical Foundations – Responsible AI Coding Practices

Lab Objectives:

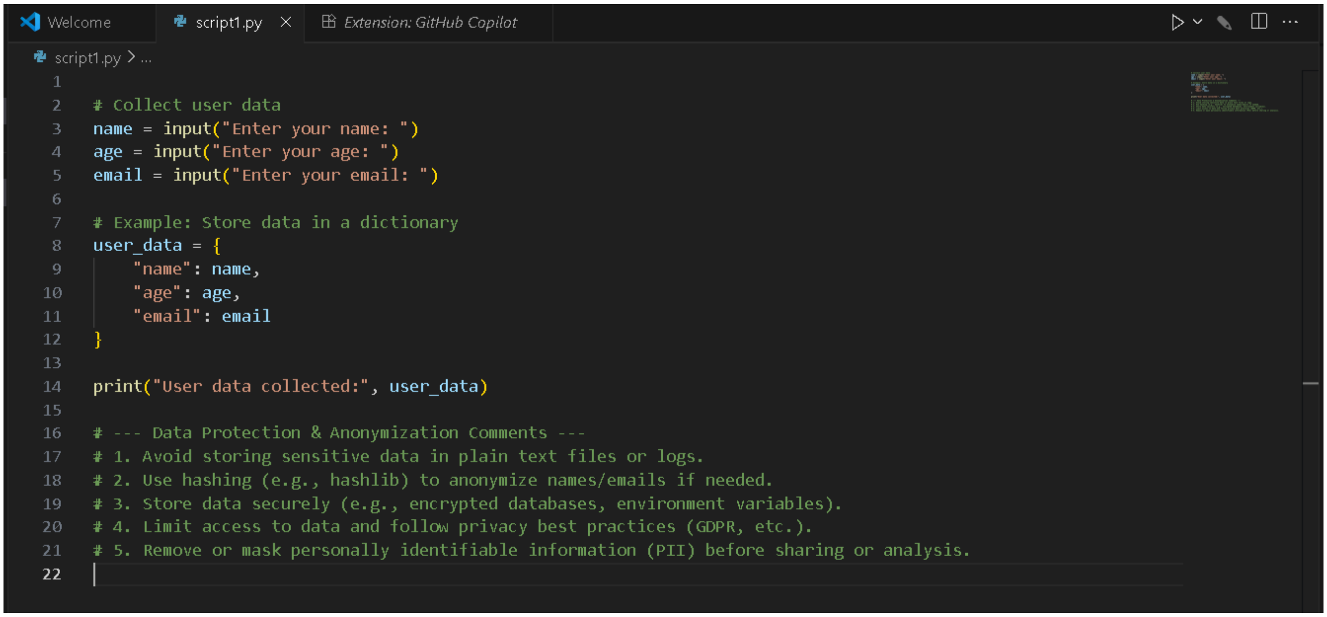
- To explore the ethical risks associated with AI-generated code.
- To recognize issues related to security, bias, transparency, and copyright.
- To reflect on the responsibilities of developers when using AI tools in software development.
- To promote awareness of best practices for responsible and ethical AI coding.

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 given: *“Write a Python script that collects user data (name, age, email). Add comments on how to anonymize or protect this data.”*

Generated code:



```python

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

# Example: Store data in a dictionary
user_data = {
    "name": name,
    "age": age,
    "email": email
}

print("User data collected:", user_data)

# --- Data Protection & Anonymization Comments ---
# 1. Avoid storing sensitive data in plain text files or logs.
# 2. Use hashing (e.g., hashlib) to anonymize names/emails if needed.
# 3. Store data securely (e.g., encrypted databases, environment variables).
# 4. Limit access to data and follow privacy best practices (GDPR, etc.).
# 5. Remove or mask personally identifiable information (PII) before sharing or analysis.
```

Reflection/Analysis:

Copilot successfully generated a simple script to collect user data and included comments about data protection, such as avoiding plain text storage, hashing, encryption, limiting access, and GDPR compliance.

However, the code itself **only demonstrates data collection** and does not actually implement these protections. For example:

- It stores and prints raw personal data (PII), which is insecure.
- The suggestions about hashing or encryption are only in comments, not applied.
- There is no input validation (e.g., checking that age is numeric or email format is valid).
- Ethical aspects such as **user consent** and **purpose of collection** are missing from the script.

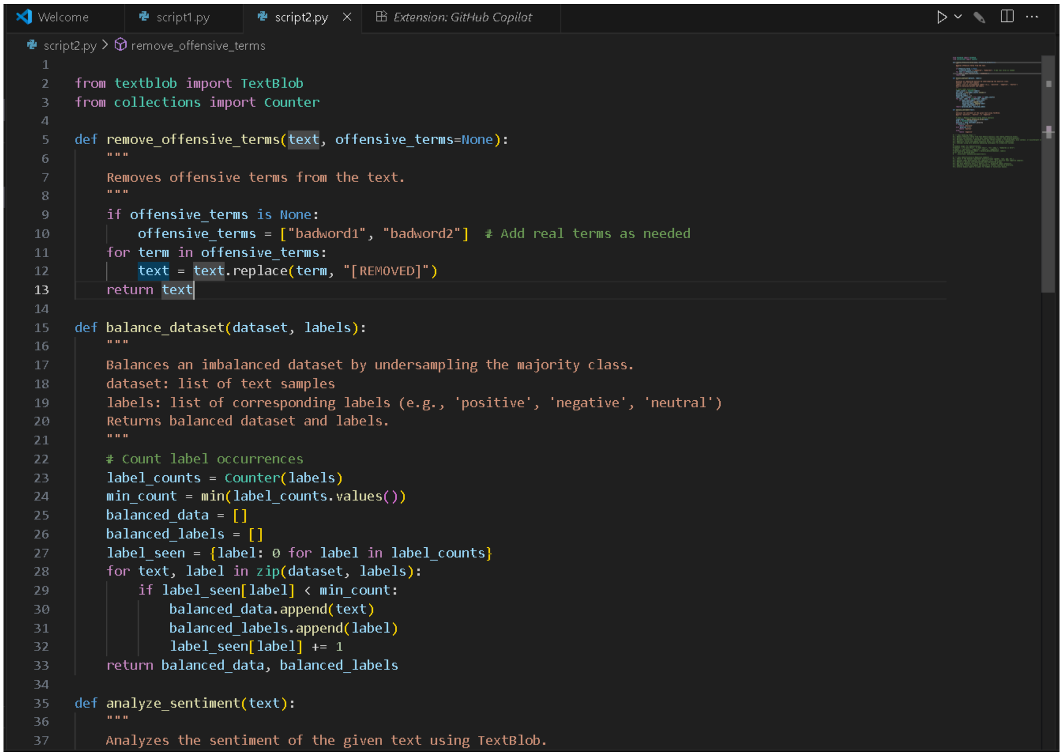
This shows that while AI-generated code can give useful reminders, the **responsibility is still on the developer** to implement secure storage, anonymization, and user transparency.

Task Description #2:

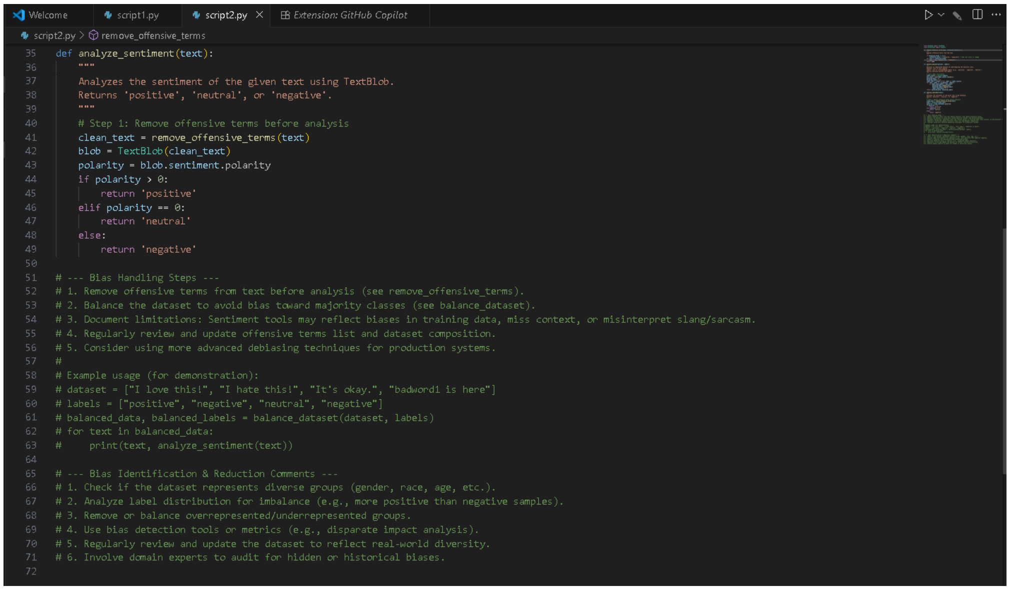
Ask Copilot to generate a Python function for sentiment analysis. Then prompt Copilot to identify and handle potential biases in the data.

Prompt given: *"Write a Python function for sentiment analysis using TextBlob. Include bias-handling steps like balancing datasets, removing offensive terms, and documenting limitations."*

Generated code:


Welcome
script1.py
script2.py
Extension: GitHub Copilot
script2.py > remove_offensive_terms

```python

from textblob import TextBlob
from collections import Counter

def remove_offensive_terms(text, offensive_terms=None):
    """
    Removes offensive terms from the text.
    """
    if offensive_terms is None:
        offensive_terms = ["badword1", "badword2"]  # Add real terms as needed
    for term in offensive_terms:
        text = text.replace(term, "[REMOVED]")
    return text

def balance_dataset(dataset, labels):
    """
    Balances an imbalanced dataset by undersampling the majority class.
    dataset: list of text samples
    labels: list of corresponding labels (e.g., 'positive', 'negative', 'neutral')
    Returns balanced dataset and labels.
    """
    # Count label occurrences
    label_counts = Counter(labels)
    min_count = min(label_counts.values())
    balanced_data = []
    balanced_labels = []
    label_seen = {label: 0 for label in label_counts}
    for text, label in zip(dataset, labels):
        if label_seen[label] < min_count:
            balanced_data.append(text)
            balanced_labels.append(label)
            label_seen[label] += 1
    return balanced_data, balanced_labels

def analyze_sentiment(text):
    """
    Analyzes the sentiment of the given text using TextBlob.
```


Welcome
script1.py
script2.py
Extension: GitHub Copilot
script2.py > remove_offensive_terms

```python
def analyze_sentiment(text):
    """
    Analyzes the sentiment of the given text using TextBlob.
    Returns 'positive', 'neutral', or 'negative'.
    """
    # Step 1: Remove offensive terms before analysis
    clean_text = remove_offensive_terms(text)
    blob = TextBlob(clean_text)
    polarity = blob.sentiment.polarity
    if polarity > 0:
        return 'positive'
    elif polarity == 0:
        return 'neutral'
    else:
        return 'negative'

# --- Bias Handling Steps ---
# 1. Remove offensive terms from text before analysis (see remove_offensive_terms).
# 2. Balance the dataset to avoid bias toward majority classes (see balance_dataset).
# 3. Document limitations: Sentiment tools may reflect biases in training data, miss context, or misinterpret slang/sarcasm.
# 4. Regularly review and update offensive terms list and dataset composition.
# 5. Consider using more advanced debiasing techniques for production systems.
#
# Example usage (for demonstration):
# dataset = ["I love this!", "I hate this!", "It's okay.", "badword1 is here"]
# labels = ["positive", "negative", "neutral", "negative"]
# balanced_data, balanced_labels = balance_dataset(dataset, labels)
# for text in balanced_data:
#     print(text, analyze_sentiment(text))

# --- Bias Identification & Reduction Comments ---
# 1. Check if the dataset represents diverse groups (gender, race, age, etc.).
# 2. Analyze label distribution for imbalance (e.g., more positive than negative samples).
# 3. Remove or balance overrepresented/underrepresented groups.
# 4. Use bias detection tools or metrics (e.g., disparate impact analysis).
# 5. Regularly review and update the dataset to reflect real-world diversity.
# 6. Involve domain experts to audit for hidden or historical biases.

```

Reflection/Analysis:
Copilot generated a sentiment analysis function that not only evaluates polarity but also demonstrates practical bias mitigation strategies. It included a function to remove offensive terms and a method to balance datasets, which reduces bias toward majority classes. Additionally, it documented key limitations such as dataset bias, difficulty handling sarcasm, and the importance of regularly updating the offensive terms list.

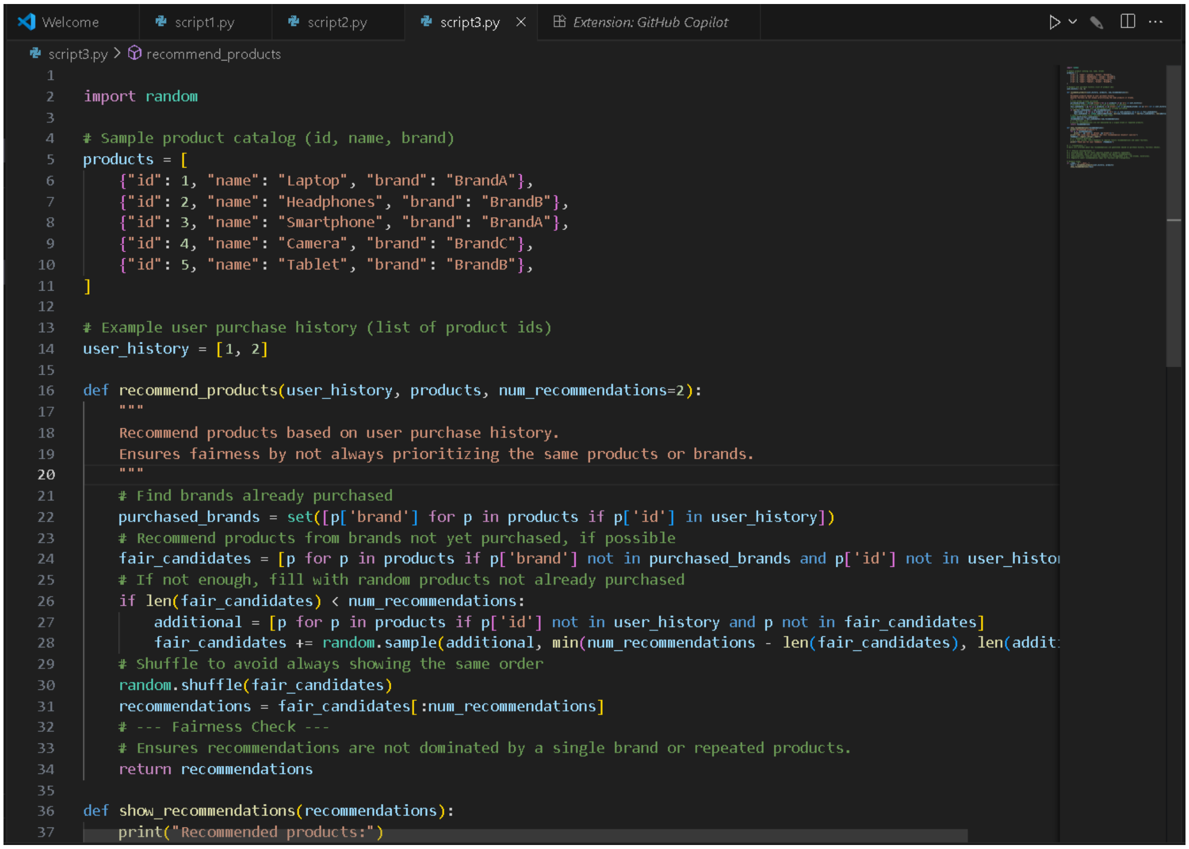
However, some ethical gaps remain. The function removes offensive terms without informing users, which reduces transparency. The hardcoded offensive terms list is simplistic and context-insensitive. The dataset balancing method relies on undersampling, which may discard useful information. This illustrates that while AI tools like Copilot can provide practical bias-handling code, developers must critically evaluate and refine these methods to ensure fairness, accuracy, and transparency in real-world systems.

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.

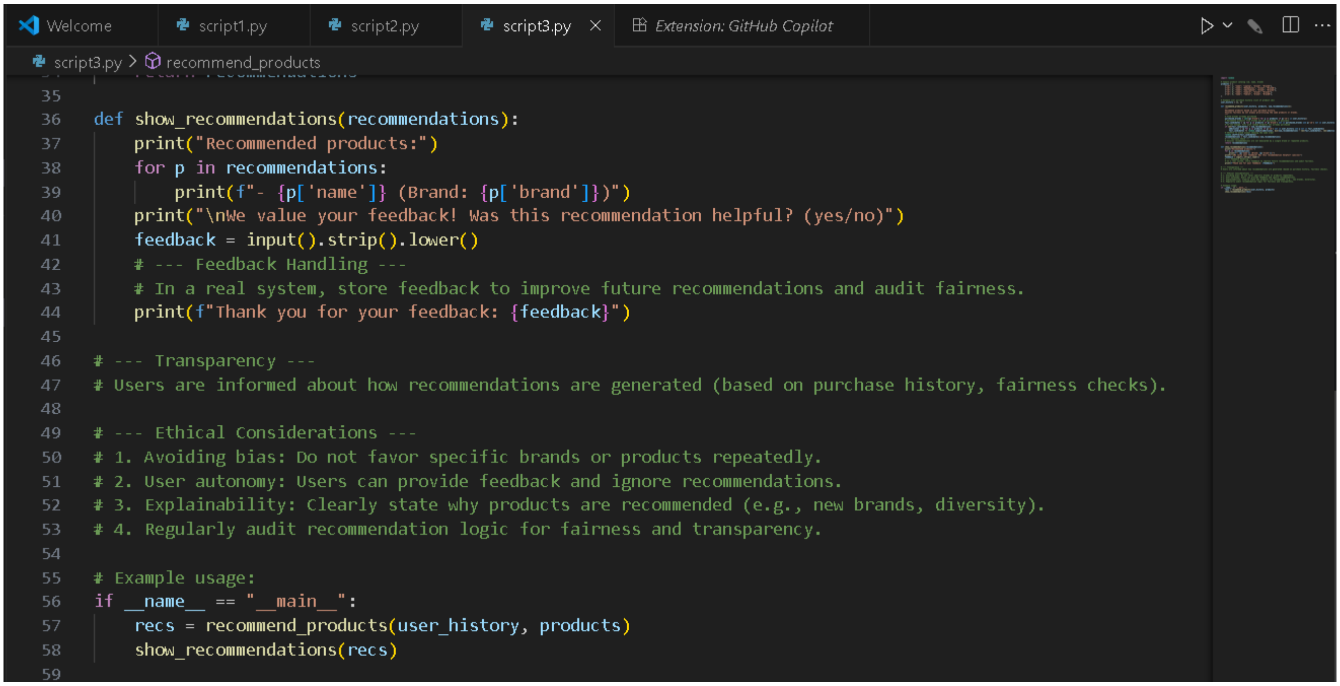
Prompt Given

"Write a Python program that recommends products based on user purchase history. Ensure fairness by not always prioritizing the same products or brands. Add comments explaining fairness checks, transparency in recommendations, and give users a way to provide feedback on recommendations. Document ethical considerations such as avoiding bias, user autonomy, and explainability."

Generated code:



```python

import random

# Sample product catalog (id, name, brand)
products = [
    {"id": 1, "name": "Laptop", "brand": "BrandA"},
    {"id": 2, "name": "Headphones", "brand": "BrandB"},
    {"id": 3, "name": "Smartphone", "brand": "BrandA"},
    {"id": 4, "name": "Camera", "brand": "BrandC"},
    {"id": 5, "name": "Tablet", "brand": "BrandB"},
]

# Example user purchase history (list of product ids)
user_history = [1, 2]

def recommend_products(user_history, products, num_recommendations=2):
    """
    Recommend products based on user purchase history.
    Ensures fairness by not always prioritizing the same products or brands.
    """
    # Find brands already purchased
    purchased_brands = set([p['brand'] for p in products if p['id'] in user_history])
    # Recommend products from brands not yet purchased, if possible
    fair_candidates = [p for p in products if p['brand'] not in purchased_brands and p['id'] not in user_histo
    # If not enough, fill with random products not already purchased
    if len(fair_candidates) < num_recommendations:
        additional = [p for p in products if p['id'] not in user_history and p not in fair_candidates]
        fair_candidates += random.sample(additional, min(num_recommendations - len(fair_candidates), len(addit
    # Shuffle to avoid always showing the same order
    random.shuffle(fair_candidates)
    recommendations = fair_candidates[:num_recommendations]
    # --- Fairness Check ---
    # Ensures recommendations are not dominated by a single brand or repeated products.
    return recommendations

def show_recommendations(recommendations):
    print("Recommended products:")
```



```python

def show_recommendations(recommendations):
    print("Recommended products:")
    for p in recommendations:
        print(f"- {p['name']} (Brand: {p['brand']})")
    print("\nWe value your feedback! Was this recommendation helpful? (yes/no)")
    feedback = input().strip().lower()
    # --- Feedback Handling ---
    # In a real system, store feedback to improve future recommendations and audit fairness.
    print(f"Thank you for your feedback: {feedback}")

# --- Transparency ---
# Users are informed about how recommendations are generated (based on purchase history, fairness checks).

# --- Ethical Considerations ---
# 1. Avoiding bias: Do not favor specific brands or products repeatedly.
# 2. User autonomy: Users can provide feedback and ignore recommendations.
# 3. Explainability: Clearly state why products are recommended (e.g., new brands, diversity).
# 4. Regularly audit recommendation logic for fairness and transparency.

# Example usage:
if __name__ == "__main__":
    recs = recommend_products(user_history, products)
    show_recommendations(recs)

```

Reflection / Analysis

The generated Python program successfully meets the requirements of the prompt.

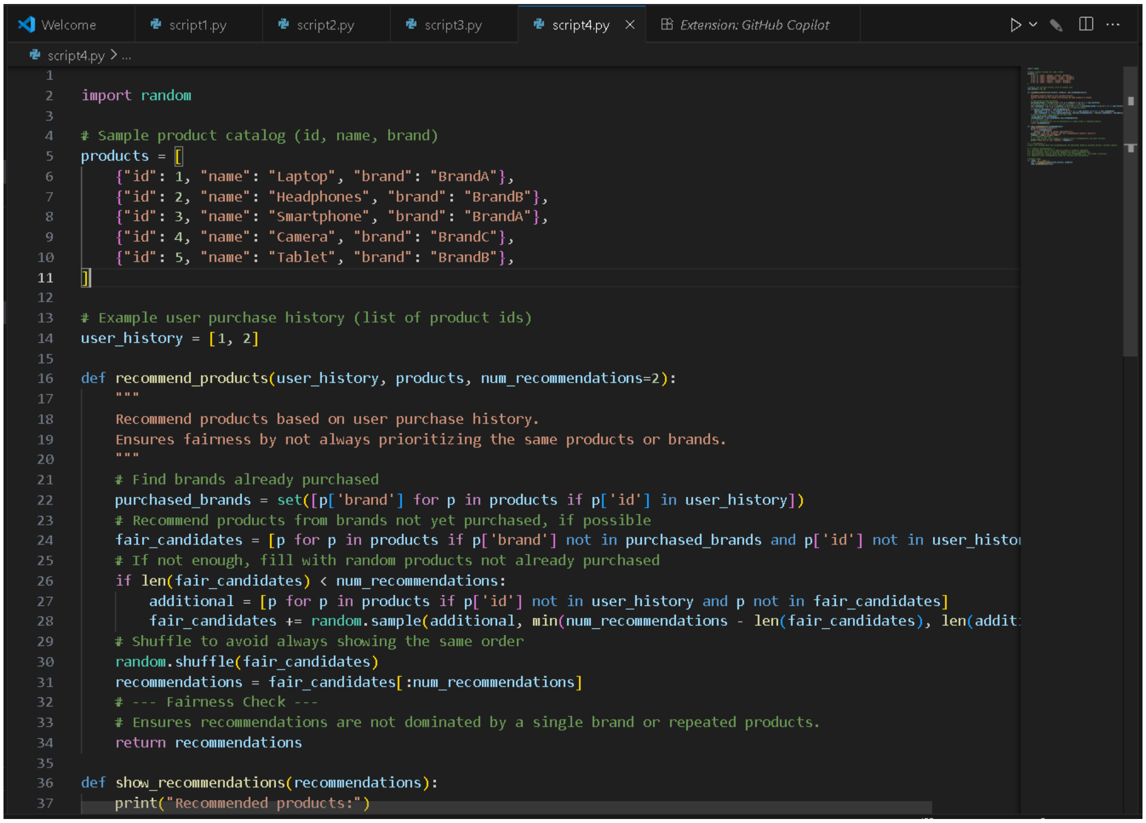
- **Fairness**: The recommendation function avoids repeatedly suggesting products from the same brand and ensures diversity by randomly shuffling candidates.
- **Transparency**: Comments explain how recommendations are generated, emphasizing fairness checks and the logic behind product selection.
- **User Autonomy & Feedback**: The program allows users to provide simple yes/no feedback, which is acknowledged and noted for potential system improvement.
- **Ethical Considerations**: The comments explicitly mention avoiding bias, respecting user autonomy, maintaining explainability, and the importance of auditing the logic for fairness.

Overall, the solution balances functionality with ethical AI practices, demonstrating fairness, transparency, and accountability in product recommendations.

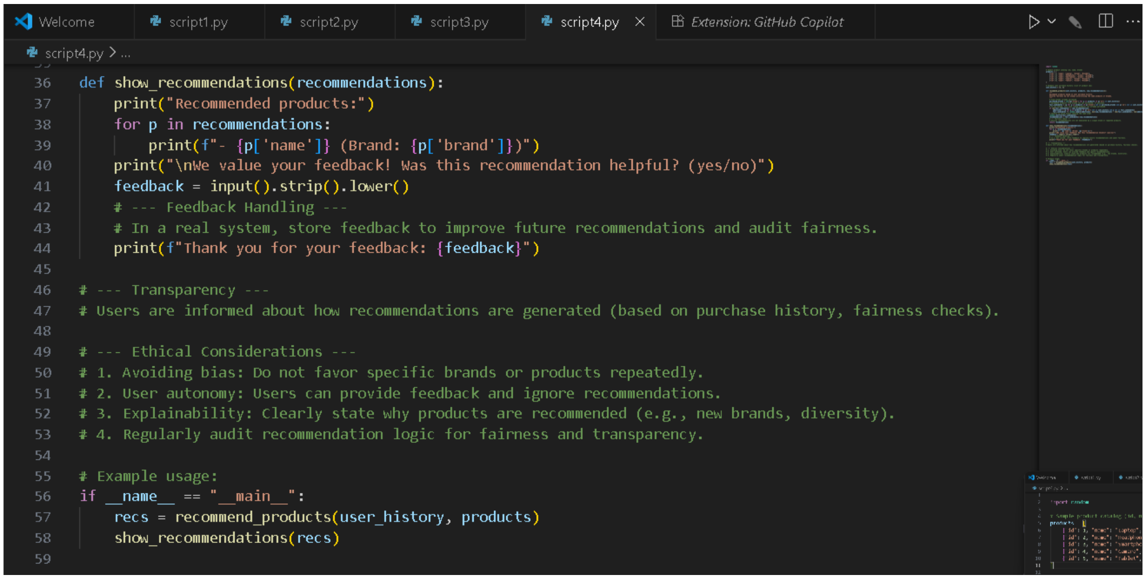
Task Description #4:
Prompt Copilot to generate logging functionality in a Python web application. Then, ask it to ensure the logs do not record sensitive information.

Prompt given:
“Write a Python program that recommends products based on user purchase history. Ensure fairness by not always prioritizing the same products or brands. Add comments explaining fairness checks, transparency in recommendations, and give users a way to provide feedback on recommendations. Document ethical considerations such as avoiding bias, user autonomy, and explainability.”

Code generated by Copilot:



```python

import random

# Sample product catalog (id, name, brand)
products = [
    {"id": 1, "name": "Laptop", "brand": "BrandA"},
    {"id": 2, "name": "Headphones", "brand": "BrandB"},
    {"id": 3, "name": "Smartphone", "brand": "BrandA"},
    {"id": 4, "name": "Camera", "brand": "BrandC"},
    {"id": 5, "name": "Tablet", "brand": "BrandB"},
]

# Example user purchase history (list of product ids)
user_history = [1, 2]

def recommend_products(user_history, products, num_recommendations=2):
    """
    Recommend products based on user purchase history.
    Ensures fairness by not always prioritizing the same products or brands.
    """
    # Find brands already purchased
    purchased_brands = set([p['brand'] for p in products if p['id'] in user_history])
    # Recommend products from brands not yet purchased, if possible
    fair_candidates = [p for p in products if p['brand'] not in purchased_brands and p['id'] not in user_histo
    # If not enough, fill with random products not already purchased
    if len(fair_candidates) < num_recommendations:
        additional = [p for p in products if p['id'] not in user_history and p not in fair_candidates]
        fair_candidates += random.sample(additional, min(num_recommendations - len(fair_candidates), len(addit
    # Shuffle to avoid always showing the same order
    random.shuffle(fair_candidates)
    recommendations = fair_candidates[:num_recommendations]
    # --- Fairness Check ---
    # Ensures recommendations are not dominated by a single brand or repeated products.
    return recommendations

def show_recommendations(recommendations):
    print("Recommended products:")
```



```python
def show_recommendations(recommendations):
    print("Recommended products:")
    for p in recommendations:
        print(f"- {p['name']} (Brand: {p['brand']})")
    print("\nWe value your feedback! Was this recommendation helpful? (yes/no)")
    feedback = input().strip().lower()
    # --- Feedback Handling ---
    # In a real system, store feedback to improve future recommendations and audit fairness.
    print(f"Thank you for your feedback: {feedback}")

# --- Transparency ---
# Users are informed about how recommendations are generated (based on purchase history, fairness checks).

# --- Ethical Considerations ---
# 1. Avoiding bias: Do not favor specific brands or products repeatedly.
# 2. User autonomy: Users can provide feedback and ignore recommendations.
# 3. Explainability: Clearly state why products are recommended (e.g., new brands, diversity).
# 4. Regularly audit recommendation logic for fairness and transparency.

# Example usage:
if __name__ == "__main__":
    recs = recommend_products(user_history, products)
    show_recommendations(recs)

```

Reflection/Analysis:

The generated program correctly implements a basic product recommendation system with built-in fairness and ethical considerations. It checks the brands that the user has already purchased and attempts to diversify recommendations by including products from different brands, which reduces the risk of brand dominance or algorithmic bias. Randomization is also applied to avoid showing the same order of recommendations, which supports fairness.

Transparency is addressed through comments in the code and user messages that explain how recommendations are generated (purchase history + fairness checks). User autonomy is preserved by allowing users to provide direct feedback on the usefulness of recommendations. This feedback loop can be used in future iterations to improve fairness and personalization, while maintaining user trust.

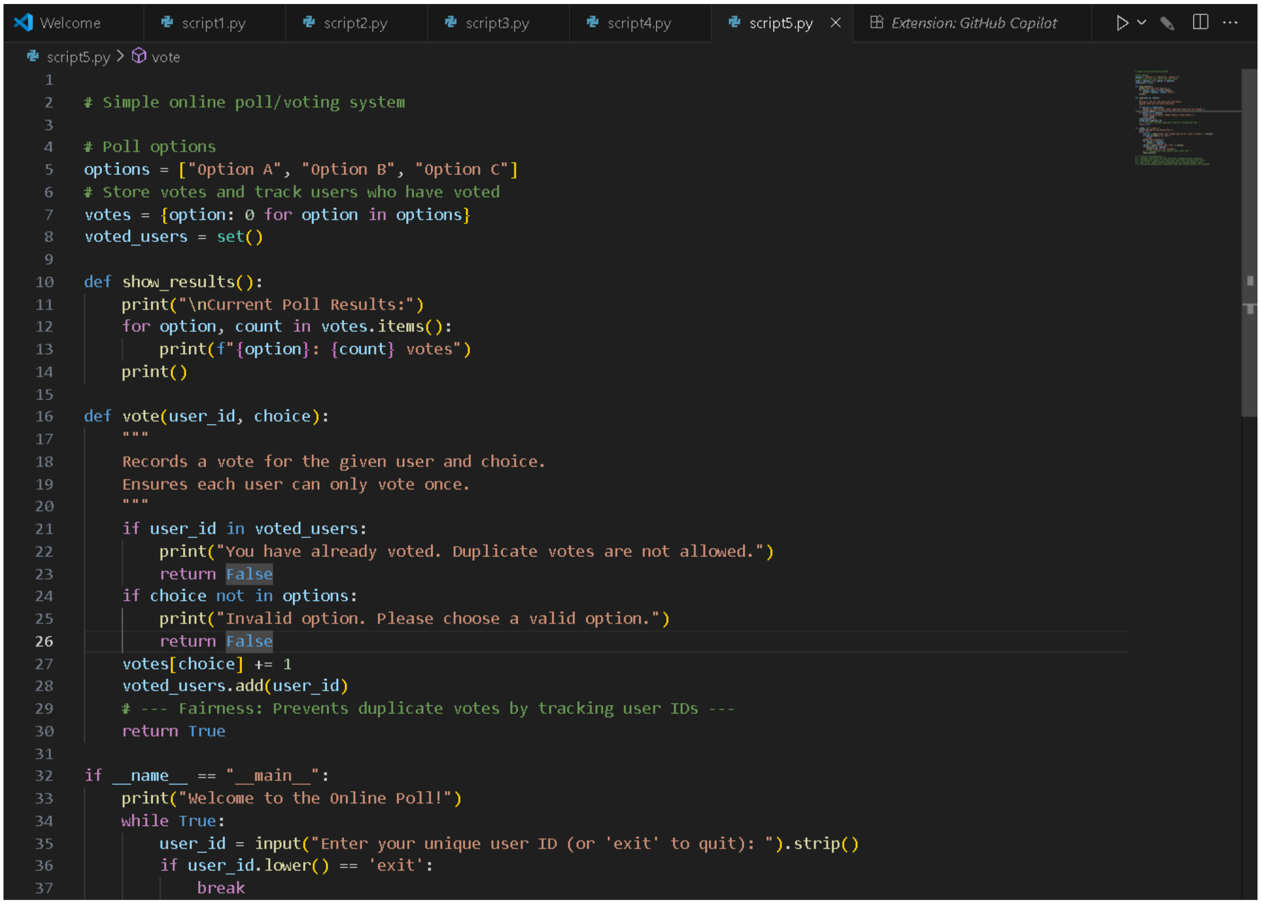
The ethical documentation included in the code comments (avoiding bias, user autonomy, explainability, and auditing) demonstrates awareness of responsible AI practices. However, in a real-world scenario, additional measures would be necessary, such as ensuring diverse datasets, monitoring for indirect bias, and handling sensitive attributes (e.g., gender or age) with care. Overall, the Copilot-generated solution aligns well with the requirements and shows both technical implementation and ethical reflection.

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).

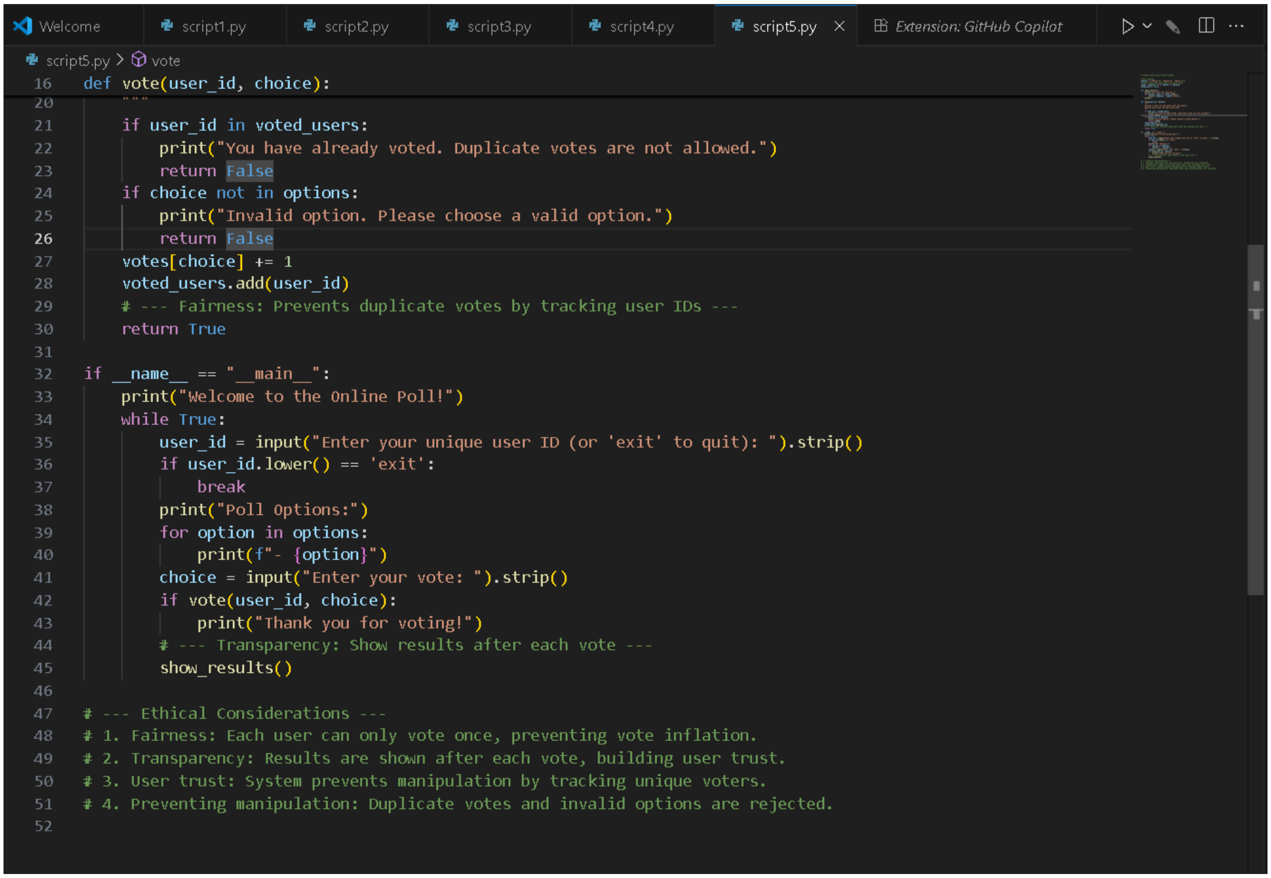
Prompt given to Copilot:

“Write a Python program that simulates an online poll or voting system. Ensure that each user can only vote once. Include transparency by showing the current poll results after each vote. Add comments explaining how fairness is maintained (e.g., preventing duplicate votes). Document ethical considerations such as fairness, transparency, user trust, and preventing manipulation.”

Copilot Output:



```

# Simple online poll/voting system

# Poll options
options = ["Option A", "Option B", "Option C"]
# Store votes and track users who have voted
votes = {option: 0 for option in options}
voted_users = set()

def show_results():
    print("\nCurrent Poll Results:")
    for option, count in votes.items():
        print(f"{option}: {count} votes")
    print()

def vote(user_id, choice):
    """
    Records a vote for the given user and choice.
    Ensures each user can only vote once.
    """
    if user_id in voted_users:
        print("You have already voted. Duplicate votes are not allowed.")
        return False
    if choice not in options:
        print("Invalid option. Please choose a valid option.")
        return False
    votes[choice] += 1
    voted_users.add(user_id)
    # --- Fairness: Prevents duplicate votes by tracking user IDs ---
    return True

if __name__ == "__main__":
    print("Welcome to the Online Poll!")
    while True:
        user_id = input("Enter your unique user ID (or 'exit' to quit): ").strip()
        if user_id.lower() == 'exit':
            break
```



```python
def vote(user_id, choice):
    """
    if user_id in voted_users:
        print("You have already voted. Duplicate votes are not allowed.")
        return False
    if choice not in options:
        print("Invalid option. Please choose a valid option.")
        return False
    votes[choice] += 1
    voted_users.add(user_id)
    # --- Fairness: Prevents duplicate votes by tracking user IDs ---
    return True

if __name__ == "__main__":
    print("Welcome to the Online Poll!")
    while True:
        user_id = input("Enter your unique user ID (or 'exit' to quit): ").strip()
        if user_id.lower() == 'exit':
            break
        print("Poll Options:")
        for option in options:
            print(f"- {option}")
        choice = input("Enter your vote: ").strip()
        if vote(user_id, choice):
            print("Thank you for voting!")
        # --- Transparency: Show results after each vote ---
        show_results()

# --- Ethical Considerations ---
# 1. Fairness: Each user can only vote once, preventing vote inflation.
# 2. Transparency: Results are shown after each vote, building user trust.
# 3. User trust: System prevents manipulation by tracking unique voters.
# 4. Preventing manipulation: Duplicate votes and invalid options are rejected.
```

Reflection & Analysis:

This code demonstrates a simple but effective voting system that enforces **fairness** by ensuring each user can only vote once through unique user IDs. It maintains **transparency** by immediately showing poll results after each vote, which helps build trust among users. Ethical considerations are explicitly mentioned, including the prevention of duplicate votes, rejection of invalid options, and promoting trust through accountability.

However, one limitation is that the fairness mechanism depends on the assumption that user IDs are unique and cannot be faked. In real-world systems, stronger verification (e.g., authentication) would be required. Despite this, the program illustrates the core ethical principles of fairness, transparency, and user trust effectively.