

AI Assisted Coding LAB ASSIGNMENT-5.4

NAME: G.SAI GANESH

Batch:13

Hall Ticket:2303A51848

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

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

```
# Add inline comments explaining how to protect or  
anonymize this data,
```

```
# such as hashing email addresses, avoiding plain-text  
storage,
```

```
# and following basic privacy best practices.
```

File Edit Selection View Go Run terminal Help

src/main/java/com/rtm/privacypolicy/UserDataPrivacy.java

```

1 // C:\Users\3\Documents>javac -d . com.rtm.privacypolicy>...
2
3 /**
4  * User Data Privacy and Protection Script
5  * Demonstrates how to collect and protect user data with privacy best practices.
6  */
7
8 package com.rtm.privacypolicy;
9
10 import java.io.*;
11 import java.util.*;
12 import org.json.*;
13 import org.json.simple.parser.JSONParser;
14 import org.json.simple.JSONObject;
15 import org.json.simple.parser.ParseException;
16 import org.json.simple.JSONArray;
17
18 /**
19  * Import Java
20  */
21
22 import java.security.*;
23 import java.util.*;
24
25 /**
26  * Import Libraries
27  */
28
29 import java.util.*;
30 import java.util.*;
31
32 /**
33  * Import Hashlib
34  */
35
36 import java.util.*;
37
38 /**
39  * Import Base64
40  */
41
42 import java.util.*;
43
44 /**
45  * Import Getpass
46  */
47
48 import java.util.*;
49
50 /**
51  * Security Note: In production, use industry-standard libraries like:
52  */
53
54 import java.util.*;
55
56 /**
57  * - cryptography (for encryption)
58  * - argon2 (for password hashing)
59  * - bcrypt (for multi-factor authentication)
60  */
61
62
63 /**
64  * Class UserDataManager:
65  *   Manages user data collection with privacy-first approach
66  */
67
68 class UserDataManager {
69
70     /**
71      * Initialize the manager.
72      */
73
74     String filename;
75
76     /**
77      * Initialize the manager.
78      */
79     void initialize() {
80         if (has_email_email()) {
81             System.out.println("Email address found!");
82         }
83     }
84
85     /**
86      * Hash an email address using SHA-256 for anonymization.
87      */
88
89     String hash_email(String email) {
90         try {
91             byte[] hashedBytes = MessageDigest.getInstance("SHA-256").digest(email.getBytes());
92             return bytesToHex(hashedBytes);
93         } catch (NoSuchAlgorithmException e) {
94             e.printStackTrace();
95         }
96         return null;
97     }
98
99     /**
100      * Hash a password using SHA-256 with a salt.
101      */
102
103     String hash_password(String password) {
104
105         byte[] salt = new byte[16];
106         SecureRandom random = new SecureRandom();
107         random.nextBytes(salt);
108
109         String hashedPassword = password + salt;
110
111         try {
112             MessageDigest digest = MessageDigest.getInstance("SHA-256");
113             byte[] hashedBytes = digest.digest(hashedPassword.getBytes("UTF-8"));
114             return bytesToHex(hashedBytes);
115         } catch (NoSuchAlgorithmException | UnsupportedEncodingException e) {
116             e.printStackTrace();
117         }
118         return null;
119     }
120
121     /**
122      * Collect user data from input.
123      */
124
125     void collect_user_data(Map<String, String> user_data) {
126
127         /**
128          * PRIVACY BEST PRACTICES IMPLEMENTED:
129          */
130
131         /**
132          * 1. Use getpass() For passwords (not echoed to terminal)
133          * 2. Validate input for common injection attacks
134          * 3. Hash sensitive data before storing
135          * 4. Minimize data collection (only what's necessary)
136          */
137
138         System.out.println("Welcome to Secure User Data Collection!!!");
139
140         // Collect name
141         String name = input("Enter your name (will be stored): ").strip();
142         if (name.length() < 2) {
143             print("ERROR: Invalid name. Must be at least 2 characters.");
144             return;
145         }
146
147         // Collect age
148         int age = input("Enter your age: ").strip();
149         if (age < 0 || age > 100) {
150             print("ERROR: Invalid age. Must be between 0 and 100.");
151             return;
152         }
153
154         // Collect email
155         String email = input("Enter your email (will be hashed for privacy): ").strip();
156         if (!email.contains("@")) {
157             print("ERROR: Invalid email format.");
158             return;
159         }
160
161         // Collect password (hidden from terminal display)
162         String password = getpass("Enter a password (hidden for security): ");
163         if (password.length() < 8) {
164             print("ERROR: Password must be at least 8 characters.");
165             return;
166         }
167
168         // Create user record with hashed sensitive data
169         user_data = [
170             {
171                 "name": name, // Note: Consider encrypting this too in highly sensitive apps
172                 "age": age,
173                 "email_hash": email.hashCode(), // Hash instead of plain-text email
174                 "password_hash": password.hashCode() // Hash instead of plain-text password
175             }
176         ];
177     }
178
179     /**
180      * Converts bytes to hex.
181      */
182
183     String bytesToHex(byte[] bytes) {
184         StringBuilder hexString = new StringBuilder();
185         for (byte b : bytes) {
186             String hex = Integer.toHexString(b & 0xFF);
187             if (hex.length() == 1) {
188                 hexString.append('0');
189             }
190             hexString.append(hex);
191         }
192         return hexString.toString();
193     }
194
195     /**
196      * Main method
197      */
198
199     public static void main(String[] args) {
200         UserDataManager manager = new UserDataManager();
201         manager.collect_user_data(user_data);
202     }
203 }

```

File Output Debug Console Terminal Ports

Select option (1-4): []

File Edit Selection View Go Run terminal Help

src/main/java/com/rtm/privacypolicy/UserDataPrivacy.java

```

1 // C:\Users\3\Documents>javac -d . com.rtm.privacypolicy>...
2
3 /**
4  * User Data Privacy and Protection Script
5  * Demonstrates how to collect and protect user data with privacy best practices.
6  */
7
8 package com.rtm.privacypolicy;
9
10 import java.io.*;
11 import java.util.*;
12 import org.json.*;
13 import org.json.simple.parser.JSONParser;
14 import org.json.simple.JSONObject;
15 import org.json.simple.JSONArray;
16 import org.json.simple.parser.ParseException;
17
18 /**
19  * Import Java
20  */
21
22 import java.util.*;
23 import java.util.*;
24
25 /**
26  * Import Libraries
27  */
28
29 import java.util.*;
30 import java.util.*;
31
32 /**
33  * Import Hashlib
34  */
35
36 import java.util.*;
37
38 /**
39  * Import Base64
40  */
41
42 import java.util.*;
43
44 /**
45  * Import Getpass
46  */
47
48 import java.util.*;
49
50 /**
51  * Security Note: In production, use industry-standard libraries like:
52  */
53
54 import java.util.*;
55
56 /**
57  * - cryptography (for encryption)
58  * - argon2 (for password hashing)
59  * - bcrypt (for multi-factor authentication)
60  */
61
62
63 /**
64  * Class UserDataManager:
65  *   Manages user data collection with privacy-first approach
66  */
67
68 class UserDataManager {
69
70     /**
71      * Initialize the manager.
72      */
73
74     String filename;
75
76     /**
77      * Initialize the manager.
78      */
79     void initialize() {
80         if (has_email_email()) {
81             System.out.println("Email address found!");
82         }
83     }
84
85     /**
86      * Hash an email address using SHA-256 for anonymization.
87      */
88
89     String hash_email(String email) {
90         try {
91             byte[] hashedBytes = MessageDigest.getInstance("SHA-256").digest(email.getBytes());
92             return bytesToHex(hashedBytes);
93         } catch (NoSuchAlgorithmException e) {
94             e.printStackTrace();
95         }
96         return null;
97     }
98
99     /**
100      * Hash a password using SHA-256 with a salt.
101      */
102
103     String hash_password(String password) {
104
105         byte[] salt = new byte[16];
106         SecureRandom random = new SecureRandom();
107         random.nextBytes(salt);
108
109         String hashedPassword = password + salt;
110
111         try {
112             MessageDigest digest = MessageDigest.getInstance("SHA-256");
113             byte[] hashedBytes = digest.digest(hashedPassword.getBytes("UTF-8"));
114             return bytesToHex(hashedBytes);
115         } catch (NoSuchAlgorithmException | UnsupportedEncodingException e) {
116             e.printStackTrace();
117         }
118         return null;
119     }
120
121     /**
122      * Collect user data from input.
123      */
124
125     void collect_user_data(Map<String, String> user_data) {
126
127         /**
128          * PRIVACY BEST PRACTICES IMPLEMENTED:
129          */
130
131         /**
132          * 1. Use getpass() For passwords (not echoed to terminal)
133          * 2. Validate input for common injection attacks
134          * 3. Hash sensitive data before storing
135          * 4. Minimize data collection (only what's necessary)
136          */
137
138         System.out.println("Welcome to Secure User Data Collection!!!");
139
140         // Collect name
141         String name = input("Enter your name (will be stored): ").strip();
142         if (name.length() < 2) {
143             print("ERROR: Invalid name. Must be at least 2 characters.");
144             return;
145         }
146
147         // Collect age
148         int age = input("Enter your age: ").strip();
149         if (age < 0 || age > 100) {
150             print("ERROR: Invalid age. Must be between 0 and 100.");
151             return;
152         }
153
154         // Collect email
155         String email = input("Enter your email (will be hashed for privacy): ").strip();
156         if (!email.contains("@")) {
157             print("ERROR: Invalid email format.");
158             return;
159         }
160
161         // Collect password (hidden from terminal display)
162         String password = getpass("Enter a password (hidden for security): ");
163         if (password.length() < 8) {
164             print("ERROR: Password must be at least 8 characters.");
165             return;
166         }
167
168         // Create user record with hashed sensitive data
169         user_data = [
170             {
171                 "name": name, // Note: Consider encrypting this too in highly sensitive apps
172                 "age": age,
173                 "email_hash": email.hashCode(), // Hash instead of plain-text email
174                 "password_hash": password.hashCode() // Hash instead of plain-text password
175             }
176         ];
177     }
178
179     /**
180      * Converts bytes to hex.
181      */
182
183     String bytesToHex(byte[] bytes) {
184         StringBuilder hexString = new StringBuilder();
185         for (byte b : bytes) {
186             String hex = Integer.toHexString(b & 0xFF);
187             if (hex.length() == 1) {
188                 hexString.append('0');
189             }
190             hexString.append(hex);
191         }
192         return hexString.toString();
193     }
194
195     /**
196      * Main method
197      */
198
199     public static void main(String[] args) {
200         UserDataManager manager = new UserDataManager();
201         manager.collect_user_data(user_data);
202     }
203 }

```

File Output Debug Console Terminal Ports

Select option (1-4): []

The screenshot shows a developer's environment with two code editors and a terminal window.

Code Editors:

- Left Editor:** Displays Python code for a user data manager. It includes methods for adding users, displaying user data, and saving user data securely. The code uses SHA-256 hashing for password storage and includes annotations for security best practices like password complexity and secure file handling.
- Right Editor:** Displays a Python script for user data collection, annotated with security best practices such as using secure input, validating user data, and avoiding plain-text storage.

Terminal:

- Shows the command `python3 user_data_collector.py` being run.
- Output indicates the script is collecting user data and generating a file named `user_data_collected.json`.
- Annotations explain the use of `argparse` for secure input, `hashlib` for password hashing, and `json` for data serialization.

The screenshot shows a dual-monitor setup. The left monitor displays a code editor with Python code for user data privacy, while the right monitor shows a terminal window.

Left Monitor (Code Editor):

```
EDITOR
> HTML_TUTORIALS
  |- PROJECTS
  |- DOCUMENTATION
  > JAVA_PROJECTS
    > HTML_Tutorials

--> file:///C:/Users/Sneha/Desktop/PyAss/  user_data_privacy.py <
  C:\Users\Sneha\Desktop\PyAss\user_data_privacy.py | main
  1  class UserDatabase:
  2      def __init__(self):
  3          self.users = {}
  4
  5      def add_user(self, name, password):
  6          self.users[name] = password
  7
  8      def get_user(self, name):
  9          return self.users.get(name)
 10
 11      def list_users(self):
 12          print("Loaded [an] user(s) user(s) from secure storage")
 13
 14      def save_users(self):
 15          print("Saving user data to file")
 16
 17      def load_users(self):
 18          print("Loading user data from file")
 19
 20      def delete_user(self, name):
 21          del self.users[name]
 22
 23      def update_user(self, name, new_password):
 24          self.users[name] = new_password
 25
 26      def validate_email(self, email):
 27          print("Input validation and sanitization")
 28
 29      def check_permissions(self, permission):
 30          print("Secure file permissions")
 31
 32      def collect_user_data(self):
 33          print("Collecting user data collection principle")
 34
 35      def __del__(self):
 36          print("User database destroyed")
 37
 38
 39  def main():
 40      """Main function demonstrating privacy-first user data handling"""
 41
 42      print("+" * 40)
 43      print("USER DATA COLLECTION WITH PRIVACY PROTECTION")
 44      print("+" * 40)
 45      print("This script demonstrates privacy best practices:")
 46      print("  - masking sensitive data (email, passwords)")
 47      print("  - using secure storage (JSON file, SHA-256 hashing)")
 48      print("  - input validation and sanitization")
 49      print("  - secure file permissions")
 50      print("  - collecting user data collection principle")
 51      print("+" * 40)
 52
 53      manager = UserDatabaseManager("user_data_secure.json")
 54
 55      # Add any existing data
 56      manager.add_users_securely()
 57
 58      # Main menu
 59
 60      while True:
 61          print("1. Create New User")
 62          print("2. View Encrypted Data")
 63          print("3. Update User Securely")
 64          print("4. Exit")
 65
 66          choice = input("Select option (1-4): ")
 67
 68          if choice == "1":
 69              user = manager.collect_user_data()
 70              manager.add_user(user)
 71
 72          elif choice == "2":
 73              manager.display_user_data_encrypted()
 74
 75          elif choice == "3":
 76              manager.save_users_securely()
 77              print("Data saved with restricted permissions")
 78
 79          elif choice == "4":
 80              print("Goodbye! Remember to always protect user data.")
 81              break
 82
 83          else:
 84              print("Error: Invalid option. Please select 1-4.")
 85
 86
 87  if __name__ == "__main__":
 88      main()
```

Right Monitor (Terminal):

```
CMD
> BLACKBOX [-]
+--> user_data_privacy.py <
  C:\Users\Sneha\Desktop\PyAss\user_data_privacy.py | main
  1  class UserDatabase:
  2      def __init__(self):
  3          self.users = {}
  4
  5      def add_user(self, name, password):
  6          self.users[name] = password
  7
  8      def get_user(self, name):
  9          return self.users.get(name)
 10
 11      def list_users(self):
 12          print("Loaded [an] user(s) user(s) from secure storage")
 13
 14      def save_users(self):
 15          print("Saving user data to file")
 16
 17      def load_users(self):
 18          print("Loading user data from file")
 19
 20      def delete_user(self, name):
 21          del self.users[name]
 22
 23      def update_user(self, name, new_password):
 24          self.users[name] = new_password
 25
 26      def validate_email(self, email):
 27          print("Input validation and sanitization")
 28
 29      def check_permissions(self, permission):
 30          print("Secure file permissions")
 31
 32      def collect_user_data(self):
 33          print("Collecting user data collection principle")
 34
 35      def __del__(self):
 36          print("User database destroyed")
 37
 38
 39  def main():
 40      """Main function demonstrating privacy-first user data handling"""
 41
 42      print("+" * 40)
 43      print("USER DATA COLLECTION WITH PRIVACY PROTECTION")
 44      print("+" * 40)
 45      print("This script demonstrates privacy best practices:")
 46      print("  - masking sensitive data (email, passwords)")
 47      print("  - using secure storage (JSON file, SHA-256 hashing)")
 48      print("  - input validation and sanitization")
 49      print("  - secure file permissions")
 50      print("  - collecting user data collection principle")
 51      print("+" * 40)
 52
 53      manager = UserDatabaseManager("user_data_secure.json")
 54
 55      # Add any existing data
 56      manager.add_users_securely()
 57
 58      # Main menu
 59
 60      while True:
 61          print("1. Create New User")
 62          print("2. View Encrypted Data")
 63          print("3. Update User Securely")
 64          print("4. Exit")
 65
 66          choice = input("Select option (1-4): ")
 67
 68          if choice == "1":
 69              user = manager.collect_user_data()
 70              manager.add_user(user)
 71
 72          elif choice == "2":
 73              manager.display_user_data_encrypted()
 74
 75          elif choice == "3":
 76              manager.save_users_securely()
 77              print("Data saved with restricted permissions")
 78
 79          elif choice == "4":
 80              print("Goodbye! Remember to always protect user data.")
 81              break
 82
 83          else:
 84              print("Error: Invalid option. Please select 1-4.")
 85
 86
 87  if __name__ == "__main__":
 88      main()
```

The terminal window shows the command `user_data_privacy.py | main` being run, indicating the script is ready to execute.

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).

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + - | X powers... Python

Select option (1-4): & C:/Users/Sreeshma/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Sreeshma/Documents/AI ASS/user_data_privacy.py"
ERROR: Invalid option. Please select 1-4.

--- OPTIONS ---
1. Add new user
2. View anonymized data
3. Save data securely
4. Exit

Select option (1-4): 1

== Secure User Data Collection ==
Enter your name (will be stored): Nitish
Enter your age: 20
Enter your email (will be hashed for privacy): nitishrajkond@gmail.com
Enter a password (hidden for security):
ERROR: Password must be at least 8 characters.

--- OPTIONS ---
1. Add new user
2. View anonymized data
3. Save data securely
4. Exit
```

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: # Generate a Python function for sentiment analysis.

Add comments or code to identify and reduce potential biases in the data,

such as removing offensive terms, balancing positive and negative samples,

and avoiding biased language in predictions.

```

class SentimentAnalyzer:
    def __init__(self):
        self.VOCABULARY = set()
        self.POSITIVE_WORDS = {'good', 'great', 'amazing', 'excellent', 'happy', 'love'}
        self.NEGATIVE_WORDS = {'bad', 'terrible', 'awful', 'hate', 'poor', 'sad'}
        self.OFFENSIVE_TERMS = {'f***', 'd***tory', 'offensive'}
        self.SWORE_WORDS = ('c...', 'd...', 's...')

    def clean_text(self, text):
        text = text.lower()
        for word in self.SWORE_WORDS:
            text = text.replace(word, 'swear')
        return text

    def analyze(self, text):
        score = 0
        positive = 0
        negative = 0
        for word in self.POSITIVE_WORDS:
            if word in text:
                positive += 1
        for word in self.NEGATIVE_WORDS:
            if word in text:
                negative -= 1
        if positive > negative:
            score = positive - negative
        else:
            score = negative - positive
        return score

    def balance_data(self, texts, labels):
        counts = Counter(labels)
        max_count = max(counts.values())
        print(f"Max count: {max_count}")
        balanced_texts, balanced_labels = [], []
        for label in set(labels):
            if counts[label] < max_count:
                selected = random.sample(texts, max_count - counts[label])
                for t in selected:
                    balanced_texts.append(t)
                    balanced_labels.append(label)
        print(f"After: {len(balanced_texts)}, {len(balanced_labels)}")
        return balanced_texts, balanced_labels

    def __str__(self):
        return f"Sentiment Analyzer\n---\n{self.VOCABULARY}\n---\n{self.POSITIVE_WORDS}\n---\n{self.NEGATIVE_WORDS}\n---\n{self.OFFENSIVE_TERMS}\n---\n{self.SWORE_WORDS}\n---\n"

```

Expected Output #2:

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ×

Text: It's okay, nothing special.
Result: 0
PS C:\Users\Sreeshma\Downloads\HTML Tutorials> & C:/Users/Sreeshma/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Sreeshma/Documents/AI ASS/sentiment_analysis_bias.py"
==> Sentiment Analysis
Text: This product is amazing and excellent!
Result: {"text": "This product is amazing and excellent!", "score": 1.0, "label": "POSITIVE"}

Text: I hate this, it's terrible.
Result: {"text": "I hate this, it's terrible.", "score": -1.0, "label": "NEGATIVE"}

PS C:\Users\Sreeshma\Downloads\HTML Tutorials> & C:/Users/Sreeshma/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Sreeshma/Documents/AI ASS/sentiment_analysis_bias.py"
==> Sentiment Analysis
Text: This product is amazing and excellent!
Result: {"text": "This product is amazing and excellent!", "score": 1.0, "label": "POSITIVE"}

Text: I hate this, it's terrible.
Result: {"text": "I hate this, it's terrible.", "score": -1.0, "label": "NEGATIVE"}
==> Dataset Balancing
Before: {"POSITIVE": 8, "NEGATIVE": 2}
After: {"POSITIVE": 8, "NEGATIVE": 2}
Before: {"POSITIVE": 8, "NEGATIVE": 2}
After: {"POSITIVE": 2, "NEGATIVE": 2}
After: {"POSITIVE": 2, "NEGATIVE": 2}

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ×

==> Dataset Balancing
Before: {"POSITIVE": 8, "NEGATIVE": 2}
Before: {"POSITIVE": 8, "NEGATIVE": 2}
After: {"POSITIVE": 2, "NEGATIVE": 2}
After: {"POSITIVE": 2, "NEGATIVE": 2}
After: {"POSITIVE": 2, "NEGATIVE": 2}

```

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

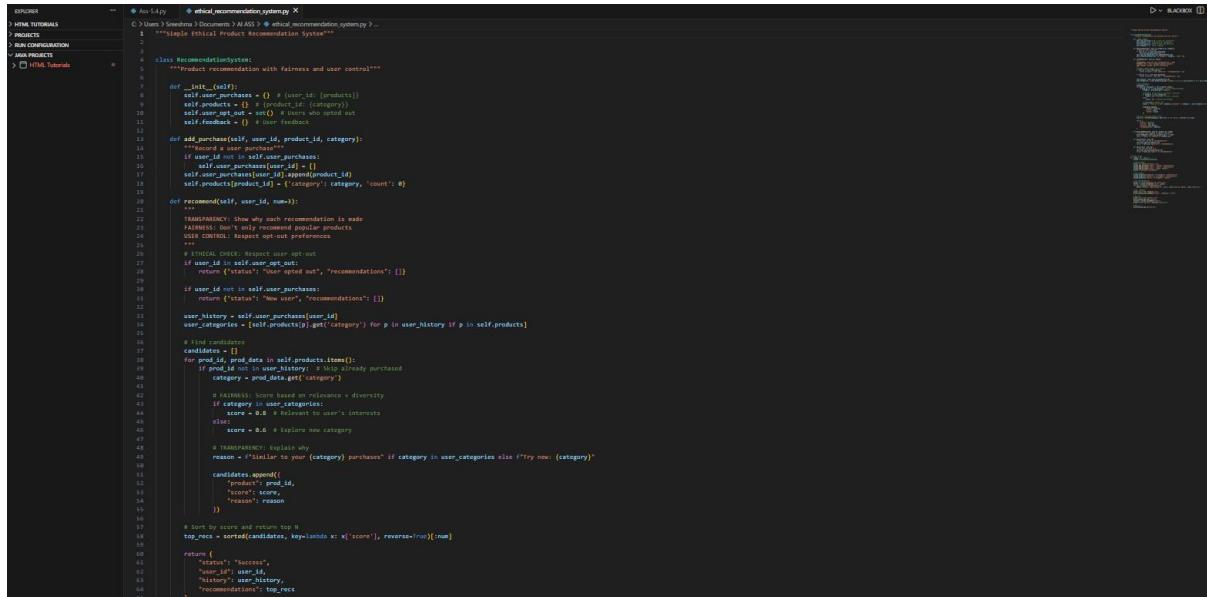
PROMPT: # Generate a Python program that recommends products based on user purchase history.

Follow ethical AI guidelines such as transparency, fairness, and user control.

Add comments explaining how recommendations are generated,

avoid favoritism toward only popular products,

and allow users to give feedback or opt out of recommendations.



The screenshot shows a code editor with a Python file named `ethical_recommendation_system.py`. The code implements a recommendation system with ethical considerations. It includes functions for adding purchases, recommending products, and calculating scores based on popularity, relevance, and diversity. The code is annotated with comments explaining its logic and ethical principles like transparency and fairness.

```
#!/usr/bin/env python3
# ethical_recommendation_system.py

class RecommendationSystem:
    """Implements ethical product recommendation System"""

    def __init__(self):
        self.user_purchases = {} # (user_id: [products])
        self.products = {} # product_id: (category)
        self.user_categories = {} # user_id: [categories]
        self.feedback = {} # user_feedback

    def add_purchase(self, user_id, product_id, category):
        """Record a user purchase"""
        if user_id not in self.user_purchases:
            self.user_purchases[user_id] = []
        self.user_purchases[user_id].append(product_id)
        self.user_categories[user_id].append(category)
        self.products[product_id] = {'category': category, 'count': 0}

    def recommend(self, user_id, num=10):
        """TRANSPARENCY: show why each recommendation is made
        Fairness: Don't only recommend popular products
        User control: Respect user's opt-out
        """
        if user_id not in self.user_purchases:
            return {"status": "User opted out", "recommendations": []}
        if user_id not in self.user_categories:
            return {"status": "New user", "recommendations": []}

        user_history = self.user_purchases[user_id]
        user_categories = self.products.get('category') for p in user_history if p in self.products

        # Find candidates
        candidates = []
        for prod_id, prod_data in self.products.items():
            if prod_id not in user_history: # Skip already purchased
                category = prod_data.get('category')
                if category not in user_categories: # relevance + diversity
                    if category in user_categories:
                        score = 0.8 # Relevant to user's interests
                    else:
                        score = 0.6 # Explore new category
                else:
                    reason = "Similar to your [category] purchases" if category in user_categories else "[try now] (category)"
                    candidates.append({
                        "product": prod_id,
                        "score": score,
                        "reason": reason
                    })
        top_rec = sorted(candidates, key=lambda x: x['score'], reverse=True)[num]

        return {
            "user_id": user_id,
            "user_ip": user_ip,
            "history": user_history,
            "recommendations": top_rec
        }
```

```

# Example usage
if __name__ == "__main__":
    system = RecommendationSystem()

    # Add purchases
    print("--- Adding Purchases ---")
    system.add_purchase("user1", "laptop", "electronics")
    system.add_purchase("user1", "monitor", "electronics")
    system.add_purchase("user2", "book", "books")
    print("Purchases recorded")

    # Add products
    system.products["keyboard"] = {"category": "Electronics"}
    system.products["monitor"] = {"category": "Electronics"}
    system.products["novel1"] = {"category": "Books"}

    # Get recommendations
    print("--- Recommendations for user1 ---")
    result1 = system.recommend("user1", num=2)
    for rec in result1["recommendations"]:
        print(f"Product: {rec['product']}, Score: {rec['score']}, Reason: {rec['reason']}")

    # Give feedback
    print("--- User Feedback ---")
    print(system.give_feedback("user1", "keyboard", True))

    # Opt out
    print("--- User Control ---")
    print(system.opt_out("user1"))
    result2 = system.recommend("user1")
    print("After opt-out: ", result2["status"])

    # Opt in
    print(system.opt_in("user1"))

```

Expected Output #3:

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

```

--- Adding Purchases ---
'Purchases recorded
PS C:\Users\greenba\Downloads\HTML Tutorials> & C:/Users/greenba/AppData/Local/Programs/Python/Python310/python.exe "C:/Users/greenba/Documents/AI AI/ethical_recommendation_system.py"
PS C:\Users\greenba\Downloads\HTML Tutorials> & C:/Users/greenba/AppData/Local/Programs/Python/Python310/python.exe "C:/Users/greenba/Documents/AI AI/ethical_recommendation_system.py"

--- Adding Purchases ---
'Purchases recorded
PS C:\Users\greenba\Downloads\HTML Tutorials>

--- Recommendations for user1 ---
Product: keyboard, Score: 8.8, Reason: Stellar to your Electronics purchases
Product: monitor, Score: 8.8, Reason: Stellar to your Electronics purchases

--- User Feedback ---
Thanks for feedback on keyboard

--- User Control ---
user opted out of recommendations
After opt-out: user opted out
user opted out of recommendations
PS C:\Users\greenba\Downloads\HTML Tutorials>

```

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: # Generate logging functionality for a Python web application.

Ensure logs do NOT store sensitive information such as passwords,

emails, or personal identifiers.

Add comments explaining ethical logging practices and privacy protection.

```
BUKU -- ethical_logging.py ethical recommendation systempy ethical_loggingpy X
C:\Users>Somedemo>Documents>AI\ADS>ethical_loggingpy >...
1 # Example Ethical Logging for web Applications
2
3 import logging
4 import re
5
6
7 class PrivacyFilter(logging.Filter):
8     """Remove sensitive data from logs"""
9     def filter(self, record):
10         """Mask password, emails, tokens, cards before logging"""
11         msg = self.filter(record)
12
13         # PRIVATE: Mask password
14         msg = re.sub(r'password=\w+', 'password=***REDACTED***', msg, flags=re.IGNORECASE)
15
16         # PRIVATE: Mask emails (show domain only)
17         msg = re.sub(r'[\w.-]+\@[^\.\w.-]+[\.\w.-]*', '[REDACTED]', msg)
18
19         # PRIVATE: Mask API keys and tokens
20         msg = re.sub(r'[\w.-]+\:[\w.-]+\:[\w.-]+\:[\w.-]+\:[\w.-]+\:[\w.-]+', '*****REDACTED*****', msg, flags=re.IGNORECASE)
21
22         # PRIVATE: Mask credit card numbers (show last 4 digits)
23         msg = re.sub(r'(\d{4})\d{12}', '(\REDACTED)\d{12}', msg)
24
25         # PRIVATE: Mask phone numbers (show last 4 digits)
26         msg = re.sub(r'(\d{4})\d{9}', '(\REDACTED)\d{9}', msg)
27
28         record.msg = msg
29
30     return True
31
32
33 def setup_logger(name, log_file='app.log'):
34     """Setup logger with privacy protection"""
35     logger = logging.getLogger(name)
36     logger.setLevel(logging.INFO)
37
38     # Add privacy filter
39     privacy_filter = PrivacyFilter()
40
41     # Console Handler
42     console_handler = logging.StreamHandler()
43     console_handler.addFilter(privacy_filter)
44     formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
45     console_handler.setFormatter(formatter)
46     logger.addHandler(console_handler)
47
48     # File Handler
49     if log_file:
50         file_handler = logging.FileHandler(log_file)
51         file_handler.addFilter(privacy_filter)
52         file_handler.setLevel(logging.DEBUG)
53         file_handler.setFormatter(formatter)
54         logger.addHandler(file_handler)
55
56         # PRIVATE: Restrict file permissions (owner read/write only)
57         import os
58         os.chmod(log_file, 0o600)
59
60     return logger
61
62
63 def log_user_action(logger, action, user_id, **safe_details):
64     """Log user action with only safe fields"""
65     msg = f'ACTION: {action} | user: {user_id}'
66     if safe_details:
67         msg += f' | {safe_details}'
68     logger.info(msg)
69
70
71 # Example usage
72 if __name__ == "__main__":
73     print("---- simple Ethical logging demo ----")
74
75     logger = setup_logger('app', log_file='app.log')
76
77     print("Test 1: Password Masking")
78     logger.info("login with password=SecurePass123")
79
80     print("Test 2: Email Masking")
81     logger.info("send email to user@example.com")
82
```

```
BUKU -- ethical_logging.py ethical recommendation systempy ethical_loggingpy X
C:\Users>Somedemo>Documents>AI\ADS>ethical_loggingpy >...
12 def setup_logger(name, log_file='app.log'):
13
14     def log_user_action(logger, action, user_id, **safe_details):
15         """Log user action with only safe fields"""
16         msg = f'ACTION: {action} | user: {user_id}'
17         if safe_details:
18             msg += f' | {safe_details}'
19         logger.info(msg)
20
21
22 # Example usage
23 if __name__ == "__main__":
24     print("---- simple Ethical logging demo ----")
25
26     logger = setup_logger('app', log_file='app.log')
27
28     print("Test 1: Password Masking")
29     logger.info("login with password=SecurePass123")
30
31     print("Test 2: Email Masking")
32     logger.info("Send email to user@example.com")
33
34     print("Test 3: API Key Masking")
35     logger.info("API key: sk_1live_1234abcde")
36
37     print("Test 4: Credit Card Masking")
38     logger.info("Payment with card 4321-1234-5678-9999")
39
40     print("Test 5: User Action Logging")
41     log_user_action(logger, "purchase", "user_123", status="success", amount=99.99)
42
43     print("User ID: " + str(user_id))
44     print("----- LOGGING PRACTICES -----")
45     print("1. REMOVE FILTER: Mask passwords, emails, tokens, cards")
46     print("2. MINIMAL DATA: Only log necessary information")
47     print("3. SECURE PWD: Set permissions to 600 (owner only)")
48     print("4. LOGGING PRACTICES: Log for development, testing, and QA")
49     print("5. NO SENSITIVE: Never store sensitive data in logs")
50
51
```

Expected Output #4:

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PAGES
Test 5: User Action Logging
2024-01-29 18:20:55,566 - app - INFO - ACTION: purchase | user: user_123 | {'status': 'success', 'amount': 99.99}

ETHICAL LOGGING PRACTICES:
1. PRIVACY FILTER: Mask passwords, emails, tokens, cards
2. MINIMAL DATA: Only log necessary information
3. SECURE FILES: Set permissions to 600 (owner only)
4. AUDIT ACTIONS: Log for auditing and debugging
5. NO SECRETS: Never store sensitive data in logs
2024-01-29 18:20:55,566 - app - INFO - ACTION: purchase | user: user_123 | {'status': 'success', 'amount': 99.99}

ETHICAL LOGGING PRACTICES:
1. PRIVACY FILTER: Mask passwords, emails, tokens, cards
2. MINIMAL DATA: Only log necessary information
3. SECURE FILES: Set permissions to 600 (owner only)
4. AUDIT ACTIONS: Log for auditing and debugging
5. NO SECRETS: Never store sensitive data in logs
2024-01-29 18:20:55,566 - app - INFO - ACTION: purchase | user: user_123 | {'status': 'success', 'amount': 99.99}
5. NO SECRETS: Never store sensitive data in logs
5. NO SECRETS: Never store sensitive data in logs
5. NO SECRETS: Never store sensitive data in logs

```

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: Generate a Python machine learning model (including data loading, training, and prediction steps).

Add inline documentation or a README-style comment section explaining how to use the model responsibly, including accuracy limitations, explainability considerations, fairness concerns, and appropriate use cases and restrictions.

```

    ...
    # User feedback and opt-out
    print("Would you like to provide feedback or opt out of recommendations?")
    feedback = input("Enter feedback or type 'opt out' to stop recommendations:")
    if feedback.strip().lower() == 'opt out':
        print("You have opted out of recommendations. Your preferences will be respected.")
    else:
        print("Thank you for your feedback: {feedback}")
    ...

    # --- Ethical AI Notes ---
    # - Transparency: Each recommendation includes an explanation.
    # - Fairness: The system ensures diversity and avoids recommending only from the most frequent category.
    # - User Control: Users can provide feedback or opt out at any time.
    # - Regularly audit recommendation logic for bias and update as needed.
    # All required packages are installed
    import sys
    import subprocess

    def install_if_missing(package):
        try:
            import_(package)
        except ImportError:
            print(f"Installing missing package: {package}")
            subprocess.check_call([sys.executable, "-m", "pip", "install", package])

    # Install 'textblob' if not present
    install_if_missing('textblob')

    # Sentiment analysis function with bias awareness and mitigation strategies
    from textblob import TextBlob
    ...

    def analyze_sentiment(text):
        """
        Analyzes the sentiment of the input text.
        Returns polarity (-1 to 1) and subjectivity (0 to 1).
        """

        Potential sources of bias in training data:
        - Imbalanced datasets (e.g., more positive than negative samples)
        - Presence of offensive, discriminatory, or culturally specific terms
        - Overrepresentation or underrepresentation of certain topics or groups
        ...

        Strategies to mitigate bias:
        - Balance the dataset across sentiment classes and demographic groups
        - Remove or mitigate discriminatory tones during preprocessing
        - Use diverse and representative data sources
        - Document known limitations and test for bias regularly
        - Involve domain experts in dataset curation
        ...

        # Example: Using TextBlob for simple sentiment analysis
        blob = TextBlob(text)
        polarity = blob.sentiment.polarity
        subjectivity = blob.sentiment.subjectivity
        return polarity, subjectivity

    ...

    # Example usage
    if __name__ == "__main__":
        user_text = input("Enter text for sentiment analysis: ")
        polarity, subjectivity = analyze_sentiment(user_text)
        print(f"Polarity: {polarity}, Subjectivity: {subjectivity}")

    # Note: For production, train your own model on a carefully curated dataset and regularly audit for bias.
    ...

```

Expected Output #5:

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

The screenshot shows a code editor with a Python file named `AISS-5.4.py`. The code is identical to the one shown in the previous screenshot. On the right side of the screen, there is a large sidebar titled "Python Model: Responsible ML Model". This sidebar contains several sections of text and checkboxes related to responsible AI practices. Some of the visible text includes:

- Generate a Python machine learning script with responsible AI training, testing, and prediction steps.**
- Add inline documentation or a README:** Add a detailed description of how to use the model responsibly, including responsible AI best practices, fairness considerations, fairness controls, and appropriate data and methods.
- Comprehensive Model:** Create a comprehensive ML model with complete documentation for responsible AI.
- Code Review:**
 - 1. Data Loading: Checks if the dataset is clean and suitable for the task.
 - 2. Model Training: Checks for overfitting and underfitting.
 - 3. Model Evaluation: Checks for reasonable thresholds for classification.
 - 4. Model Deployment: Checks for security and performance.
- Fairness:** Feature importance analysis and fairness requirements.
- Predictability:** Feature importance analysis and fairness requirements.
- Interpretability:** Feature importance analysis and fairness requirements.
- Accuracy:** Feature importance analysis and fairness requirements.
- Model Comparison:** Feature importance analysis and fairness requirements.
- User Guide:** When appropriate, include a user guide and API for high-level decisions.
- Readme:** Includes a comprehensive overview of responsible AI practices.

At the bottom of the sidebar, there is a section titled "README Section Includes" with a list of items:

- A comprehensive guide with responsible AI best practices.
- Production deployment requirements.
- Ethical guidelines.
- Responsible AI checklist.

At the very bottom of the sidebar, it says "Send to AI Model: responsible_ml.py" and has a "Send" button.