

AI Assisted Coding LAB ASSIGNMENT-5.4

NAME: K. NITHISH RAJ

Batch:14

Hall Ticket:2303A510G4

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.
```

The screenshot shows a Jupyter Notebook interface with a Python script titled "user_data_privacy.py". The code implements a "UserDataManager" class that reads user data from a JSON file, hashes emails using SHA-256, and stores hashed emails in a database. It includes comments explaining security practices like password hashing and avoiding plain-text storage.

```
File Edit Selection View Go Run Terminal Help ↗
```

HTML TUTORIALS

RECENT FILES

EXPLORER

HTML TUTORIALS

MAIN CONFIGURATION

AVA PROJECTS

HTML TUTORIALS

user_data_privacy.py X

C:\Users\Sebastien\Documents>py A&S > user_data_privacy.py ...

```
1 #!/usr/bin/env python
2
3 # User Data Privacy and Protection Script
4 # Demonstrates how to collect and protect user data with privacy best practices.
5 #
6
7 import hashlib
8 import json
9 from getpass import getpass
10
11 # Security Note: In production, use industry-standard libraries like:
12 #   - cryptography (for encryption)
13 #   - argon2 (for password hashing)
14 #   - pyotp (for multi-factor authentication)
15
16
17 class UserManager:
18     """Manages user collection with privacy-first approach"""
19
20     def __init__(self, filename="user_data.json"):
21         self._filename = filename
22         self._users = []
23
24     @property
25     def filename(self):
26         """Secure location to store encrypted data (not plain-text)"""
27         return self._filename
28
29     @filename.setter
30     def filename(self, filename):
31         self._filename = filename
32
33     @property
34     def user_count(self):
35         return len(self._users)
36
37     def add_user(self, email, password):
38
39         Hash an email address using SHA-256 for anonymity.
40
41         # Hashing value is impossible to reverse-engineer the original email,
42         # protecting user privacy while still allowing unique identification.
43         never_stored_email_in_logs_or_databases()
44
45         Args:
46             email (str): Email address to hash
47
48         Returns:
49             str: Hexadecimal hash of the email
50
51     ...
```

Args:

email (str): Email address to hash

Returns:

str: Hexadecimal hash of the email

...
A Comment block for the "add_user" method.
The "never_stored_email_in_logs_or_databases" function is defined below it.
The "args" section is followed by a "password" argument.
The "password" argument has a type annotation of "str" and a description of "Password to hash".

IMPORTANT! This is a basic example. In production, use:
 -argon2 (https://github.com/peteris/argon2-python)
 -These are resistant to brute-force attack

Args:

password (str): Password to hash

CHAT

user_data_privacy.py

Describe what to build next

Agree ↗ Auto ↗ 18

The screenshot shows a developer's environment with two code editors. The left editor contains the original Python code for user data collection, which includes several annotations and comments. The right editor shows the generated Python script with privacy best practices applied.

Left Editor (Original Code):

```
#!/usr/bin/python3
# user_data_privacy.py

class User:
    def __init__(self, user_id):
        self.id = user_id
        self.name = None
        self.email = None
        self.password = None
        self.created_at = None
        self.updated_at = None
        self.deleted_at = None
        self.is_deleted = False
        self.data_version = "1.0" # For data structure versioning

    def __str__(self):
        return f"User ID: {self.id}, Name: {self.name}, Email: {self.email}, Password Hash: {self.password}, Created At: {self.created_at}, Updated At: {self.updated_at}, Deleted At: {self.deleted_at}, Is Deleted: {self.is_deleted}, Data Version: {self.data_version}"

    def add_user(self, user_data):
        self.users.append(user_data)

    def display_user_data(self):
        print("Display user data in encrypted form.")

    def save_user_data(self):
        # Save user data with privacy considerations.
        # SECURITY BEST PRACTICES
        # 1. Don't store plain-text passwords (SHA256) - available only by name
        # 2. Encrypt data before saving (use AES encryption in production)
        # 3. use 3500 FPE (could also use scrypt with encryption)
        # 4. Never save plain-text passwords or emails
        try:
            # Write data to JSON file
            with open(self.filename, 'w') as f:
                json.dump(self.users, f, indent=2)

            # Restrict file permissions on windows and unix-like systems
            if os.name == 'nt':
                # Windows
                os.chmod(self.filename, mode=0o600)
                print(f"File permissions set to {self.filename}")
            else:
                # Unix-like
                os.chmod(self.filename, mode=0o400)
                print(f"File permissions restricted to owner only (mode: {mode})")
        except Exception as e:
            print(f"Error: {e} could not save encrypted user data")

    def __init__(self):
        self.users = []
        self.filename = "user_data.json"
        self.mode = 0o600
```

Right Editor (Generated Script):

```
#!/usr/bin/python3
# user_data_privacy.py

# Generate a Python script that collects user data such as name, age, and email.
# Add inline comments explaining how to protect
# such as hashing plain addresses, avoiding
# plain-text storage,
# and following basic privacy best practices.

# Create Python script for user data collection
# Create a comprehensive Python script that demonstrates user data collection
# privacy and security best practices in a single file.

# Key Features:
# 1. Data Hashing - Email addresses and passwords are hashed using SHA-256
# 2. Secure Password Input - Use getpass() to hide password input from terminal display
# 3. Username Validation - Validates name, age, and email format to prevent injection attacks
# 4. Account Creation - Displays user data in encrypted form (prior reading)
# 5. Secure File Storage - Data is saved with restricted permissions (0o600)
# 6. Detailed Comments - Detailed explanations throughout outlining why each
# privacy measure is important

# Privacy Best Practices Covered:
# 1. Never storing plain-text passwords or emails
# 2. Using hashing for one-way data anonymization
# 3. Input validation for file paths and names
# 4. Input validation to prevent attacks
# 5. Data obfuscation for audit trails
# 6. Encryption standards for production-grade libraries (bcrypt,argon2, cryptography)

The script is ready to run and includes a menu system for adding users, viewing
encrypted data, and saving security to a JSON file.
```

The screenshot shows a Jupyter Notebook interface with two tabs: 'HTML TUTORIALS' and 'HTML_Tutorials'. The active tab contains Python code demonstrating user data handling and privacy best practices. The code includes importing modules, reading JSON data, validating user input, and saving data securely. It also features a user interface for collecting user data through a loop and displaying it in an encrypted format.

```
#!/usr/bin/python

# user_data_privacy.py

# This script demonstrates privacy best practices for user data collection.

# Import required modules
import json
from getpass import getpass
from hashlib import sha256
from os import urandom

# Load user data from a secure storage
with open('user_data.json') as f:
    users = json.load(f)

# Function to load user data
def load_user_data():
    print("Loading user data...")
    return users

# Function to save user data
def save_user_data(users):
    print("Saving user data...")
    with open('user_data.json', 'w') as f:
        json.dump(users, f)
    print("User data saved successfully")

# Main function
def main():
    """Main function demonstrating privacy-first user data handling"""

    # Print intro message
    print("*" * 40)
    print("USER DATA COLLECTION WITH PRIVACY PROTECTION")
    print("*" * 40)

    # Print message about the script demonstrating privacy best practices
    print("This Python script demonstrates privacy best practices:")
    print("  - masking sensitive data (email, password)")
    print("  - using strong encryption")
    print("  - Input validation and sanitization")
    print("  - Secure file permissions")
    print("  - JSON-based data collection principle")
    print("*" * 40)

    # Initialize manager
    manager = UserDatabaseManager("user_data_secure.json")

    # Loop for collecting data
    while True:
        choice = input("Select option (1-4): ").strip()

        if choice == "1":
            manager.collect_user_data()
            manager.add_users(users)

        elif choice == "2":
            manager.display_user_data_encrypted()

        elif choice == "3":
            manager.save_user_securely()
            print("Data saved with restricted permissions")

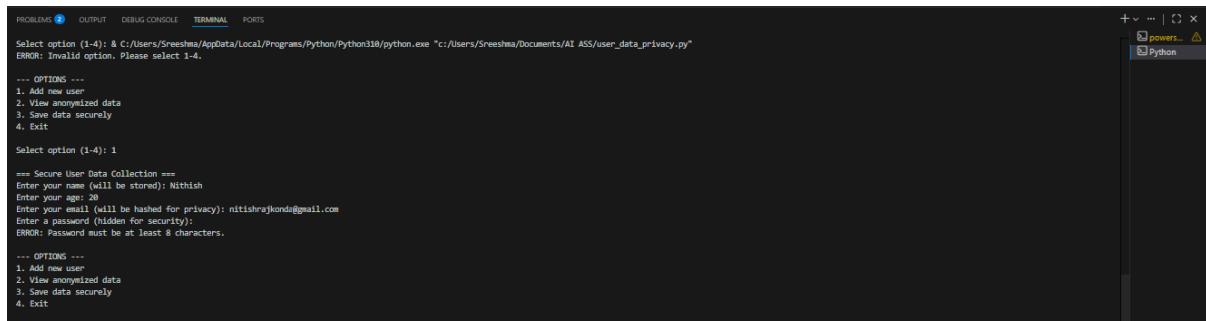
        elif choice == "4":
            print("WARNING! Remember to always protect user data.")
            break

        else:
            print("ERROR: Invalid option. Please select 1-4.")

if __name__ == "__main__":
    main()
```

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
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): Nithish
Enter your age: 20
Enter your email (will be hashed for privacy): nithishrajkond@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.

```

# File: sentiment_analysis_bias.py
# Author: Sreeshma (Sreeshma) & ASS - sentiment_analysis_bias.py > SimpleSentimentAnalyzer > _init_
# Date: 2023-09-11 10:30:00
# Version: 1.0
# Description: Single sentiment analysis with bias reduction
# Tags: #sentiment-analysis, #bias-reduction, #natural-language-processing, #text-mining, #nlp, #ai, #python

class SimpleSentimentAnalyzer:
    """Basic sentiment analyzer with bias mitigation"""

    def __init__(self):
        self._load_stopwords()
        self._load_positive_words()
        self._load_negative_words()
        self._load_offensive_words()

    def _load_stopwords(self):
        """Load stop words from file or memory"""

    def _load_positive_words(self):
        """Load positive words from file or memory"""
        self._positive_words = ["good", "great", "amazing", "excellent", "happy", "love"]
        self._positive_words.extend(["positive", "optimistic", "upbeat", "proud", "cool"])

    def _load_negative_words(self):
        """Load negative words from file or memory"""
        self._negative_words = ["bad", "terrible", "poor", "sad", "hate", "dislike", "dissatisfied"]

    def _load_offensive_words(self):
        """Load offensive words from file or memory"""
        self._offensive_words = ["fucker", "asshole", "cunt", "motherfucker"]

    def _load_stopwords(self, test):
        """Load stop words from file or memory"""

    def _load_positive_words(self, test):
        """Load positive words from file or memory"""

    def _load_negative_words(self, test):
        """Load negative words from file or memory"""

    def _load_offensive_words(self, test):
        """Load offensive words from file or memory"""

    def _analyze(self, text):
        """Analyze text for sentiment"""

        text = self._clean_text(text)
        words = text.split(" ")
        positive = sum(1 for w in words if w in self._positive_words)
        negative = sum(1 for w in words if w in self._negative_words)

        if positive + negative == 0:
            return 0.0

        score = (positive - negative) / (positive + negative)

        if (score >= 0.75) and (score <= 1.0):
            label = "POSITIVE"
        elif (score <= -0.75) and (score >= -1.0):
            label = "NEGATIVE"
        else:
            label = "NEUTRAL"

        return {"text": text, "score": round(score, 2), "label": label}

    def _balance_data(self, texts, labels):
        """Balance data for sentiment analysis"""

        counts = Counter(labels)
        max_count = max(counts.values())
        print(f"Initial counts: {dict(counts)}")

        for i in range(max_count):
            for label in set(labels):
                if counts[label] > 1:
                    selected = random.sample(counts[label], 1)[0]
                    counts[label] -= 1
                    texts.append(selected)
                    labels.append(label)

        for i in range(max_count):
            for label in set(labels):
                if counts[label] < 1:
                    selected = random.sample(counts[label], 1)[0]
                    counts[label] += 1
                    texts.append(selected)
                    labels.append(label)

        print(f"Final counts: {dict(counts)}")
        return texts, labels

    def analyze(self, text):
        """Analyze text for sentiment"""

        text = self._clean_text(text)
        print(f"Input: {text}")
        print(f"Result: {self._analyze(text)}")

    def analyze_dataset(self, texts, labels):
        """Analyze dataset for sentiment"""

        texts, labels = self._balance_data(texts, labels)
        print(f"Dataset Balancing ---")
        print(f"Before: {dict(Counter(labels))} * 2")
        print(f"After: {dict(Counter(labels))} * 2")

        for text in texts:
            print(f"Text: {text}")
            print(f"Result: {self._analyze(text)}")

        print(f"Dataset Balancing ---")
        print(f"Before: {dict(Counter(labels))} * 2")
        print(f"After: {dict(Counter(labels))} * 2")

        balanced_texts, balanced_labels = self._balance_data(texts, labels)

        return balanced_texts, balanced_labels

```

Expected Output #2:

- Copilot-generated code with additions or comments addressing

bias mitigation strategies (e.g., balancing dataset, removing offensive terms).

```

# File: sentiment_analysis_bias.py
# Author: Sreeshma (Sreeshma) & ASS - sentiment_analysis_bias.py > SimpleSentimentAnalyzer > _init_
# Date: 2023-09-11 10:30:00
# Version: 1.0
# Description: Single sentiment analysis with bias reduction
# Tags: #sentiment-analysis, #bias-reduction, #natural-language-processing, #text-mining, #nlp, #ai, #python

class SimpleSentimentAnalyzer:
    """Basic sentiment analyzer with bias mitigation"""

    def __init__(self):
        self._load_stopwords()
        self._load_positive_words()
        self._load_negative_words()
        self._load_offensive_words()

    def _load_stopwords(self):
        """Load stop words from file or memory"""

    def _load_positive_words(self):
        """Load positive words from file or memory"""
        self._positive_words = ["good", "great", "amazing", "excellent", "happy", "love"]
        self._positive_words.extend(["positive", "optimistic", "upbeat", "proud", "cool"])

    def _load_negative_words(self):
        """Load negative words from file or memory"""
        self._negative_words = ["bad", "terrible", "poor", "sad", "hate", "dislike", "dissatisfied"]

    def _load_offensive_words(self):
        """Load offensive words from file or memory"""
        self._offensive_words = ["fucker", "asshole", "cunt", "motherfucker"]

    def _load_stopwords(self, test):
        """Load stop words from file or memory"""

    def _load_positive_words(self, test):
        """Load positive words from file or memory"""

    def _load_negative_words(self, test):
        """Load negative words from file or memory"""

    def _load_offensive_words(self, test):
        """Load offensive words from file or memory"""

    def _analyze(self, text):
        """Analyze text for sentiment"""

        text = self._clean_text(text)
        words = text.split(" ")
        positive = sum(1 for w in words if w in self._positive_words)
        negative = sum(1 for w in words if w in self._negative_words)

        if positive + negative == 0:
            return 0.0

        score = (positive - negative) / (positive + negative)

        if (score >= 0.75) and (score <= 1.0):
            label = "POSITIVE"
        elif (score <= -0.75) and (score >= -1.0):
            label = "NEGATIVE"
        else:
            label = "NEUTRAL"

        return {"text": text, "score": round(score, 2), "label": label}

    def _balance_data(self, texts, labels):
        """Balance data for sentiment analysis"""

        counts = Counter(labels)
        max_count = max(counts.values())
        print(f"Initial counts: {dict(counts)}")

        for i in range(max_count):
            for label in set(labels):
                if counts[label] > 1:
                    selected = random.sample(counts[label], 1)[0]
                    counts[label] -= 1
                    texts.append(selected)
                    labels.append(label)

        for i in range(max_count):
            for label in set(labels):
                if counts[label] < 1:
                    selected = random.sample(counts[label], 1)[0]
                    counts[label] += 1
                    texts.append(selected)
                    labels.append(label)

        print(f"Final counts: {dict(counts)}")
        return texts, labels

    def analyze(self, text):
        """Analyze text for sentiment"""

        text = self._clean_text(text)
        print(f"Input: {text}")
        print(f"Result: {self._analyze(text)}")

    def analyze_dataset(self, texts, labels):
        """Analyze dataset for sentiment"""

        texts, labels = self._balance_data(texts, labels)
        print(f"Dataset Balancing ---")
        print(f"Before: {dict(Counter(labels))} * 2")
        print(f"After: {dict(Counter(labels))} * 2")

        for text in texts:
            print(f"Text: {text}")
            print(f"Result: {self._analyze(text)}")

        print(f"Dataset Balancing ---")
        print(f"Before: {dict(Counter(labels))} * 2")
        print(f"After: {dict(Counter(labels))} * 2")

        balanced_texts, balanced_labels = self._balance_data(texts, labels)

        return balanced_texts, balanced_labels

```

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

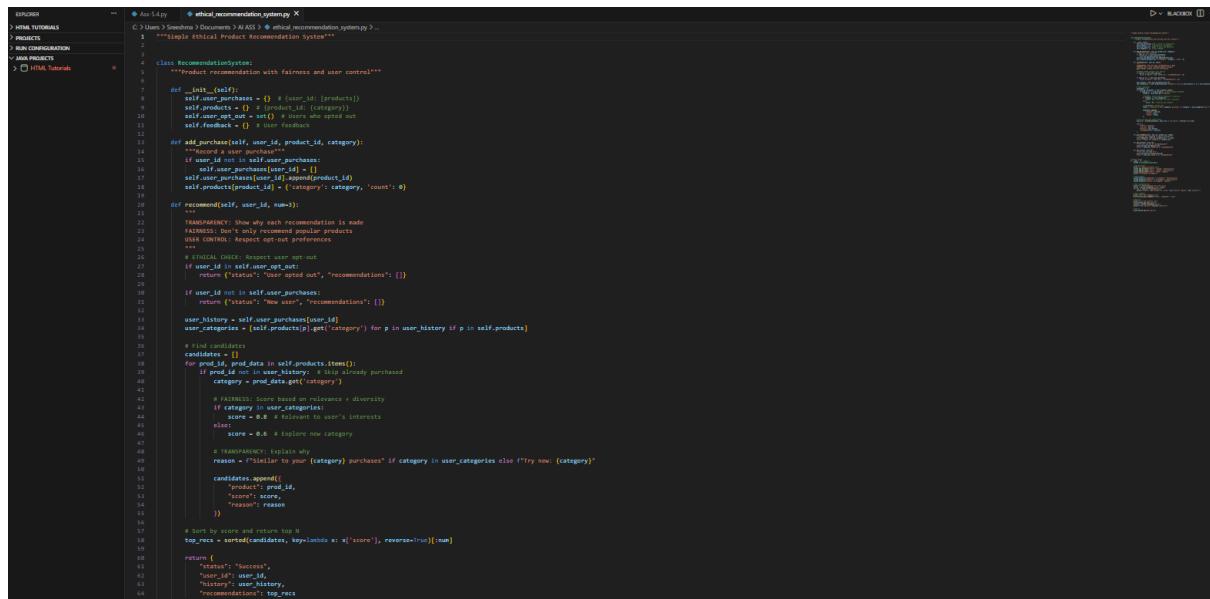
PROMPT: # Generate a Python program that recommends products based on user purchase history.

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 the following features and comments:

- Initial State:** A class `RecommendationSystem` is defined with an empty constructor.
- Adding Products:** The `add_products` method adds products to a dictionary where keys are user IDs and values are lists of products categorized by their ID.
- Adding User Purchases:** The `add_purchase` method records purchases for a specific user ID.
- Opt-Out:** The `user_opt_out` method marks a user as opting out of recommendations.
- Transparency and Fairness:** Comments explain the system's approach:
 - TRANSPARENCY: Show why each recommendation is made
 - FAIRNESS: Don't only recommend popular products
 - USER CONTROL: Respect opt-out preferences
- Recommendations:** The `recommend` method generates recommendations for a user ID, returning a tuple of status and a list of recommendations.
- Candidate Selection:** The `find_candidates` method iterates through products and calculates scores based on relevance and diversity. It uses a reason variable to explain why a product is recommended.
- Score Calculation:** The score is calculated as follows:
 - Relevance: $0.8 \times \text{relevant}$
 - Diversity: $0.2 \times \text{new category}$
- Reasoning:** The reason for a recommendation is explained as either "similar to your `(category)` purchases" or "try new `(category)`".
- Output:** The `get_recommendations` method returns a list of tuples containing product ID, score, and reason.

```

EXPLORER          --> AI 5.4.ipj   ethical_recommendation_system.py
HTML TUTORIALS
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
> HTML Tutorials
  • C:\Users\Shreeshma\Documents\AI\AI\ethical_recommendation_system.py

C:\Users\Shreeshma\Documents\AI\AI> ethical_recommendation_system.py
  1 #!/usr/bin/python
  2
  3 class RecommendationSystem:
  4     def __init__(self):
  5         self.products = {}
  6         self.user_history = {}
  7
  8     def add_purchase(self, user_id, product_id):
  9         if user_id not in self.user_history:
 10             self.user_history[user_id] = []
 11
 12         self.user_history[user_id].append(product_id)
 13
 14     def recommend(self, user_id, num=5):
 15         user_history = self.user_history.get(user_id, [])
 16
 17         if len(user_history) < 10:
 18             return []
 19
 20         else:
 21             status = "Success"
 22             user_id = user_id
 23             history = user_history
 24             recommendations = self.get_top_rec(user_id, history)
 25
 26             return {"status": "Success",
 27                   "user_id": user_id,
 28                   "history": user_history,
 29                   "recommendations": recommendations}
 30
 31     def give_feedback(self, user_id, product_id, liked):
 32         if user_id not in self.user_history:
 33             self.user_history[user_id] = []
 34
 35         if product_id in self.user_history[user_id]:
 36             self.user_history[user_id].remove(product_id)
 37
 38         if liked:
 39             self.user_history[user_id].append(product_id)
 40
 41         return f"Thanks for feedback on {product_id}"
 42
 43     def opt_out(self, user_id):
 44         """Let user opt out of recommendations"""
 45         if user_id not in self.user_history:
 46             self.user_history[user_id] = []
 47
 48         return f"{user_id} opted out of recommendations"
 49
 50     def opt_in(self, user_id):
 51         """Let user opt back in"""
 52
 53         self.user_history[user_id] = []
 54
 55         return f"{user_id} opted in to recommendations"
 56
 57
 58 # Example usage
 59 if __name__ == "__main__":
 60     system = RecommendationSystem()
 61
 62     # Add purchases
 63     print("Adding Purchases ---")
 64     system.add_purchase("user1", "laptop", "electronics")
 65     system.add_purchase("user1", "monitor", "electronics")
 66     system.add_purchase("user1", "book", "books")
 67     print(f"✓ Purchases recorded({len(system.user_history)})")
 68
 69     # Add products
 70     system.products["laptop"] = {"category": "electronics"}
 71     system.products["monitor"] = {"category": "electronics"}
 72     system.products["book"] = {"category": "books"}
 73
 74     # Get recommendations
 75     print("Recommendations for user1 ---")
 76     result = system.recommend("user1", num=2)
 77     for rec in result["recommendations"]:
 78         print(f"Product: {rec['product']}, Score: {rec['score']}, Reason: {rec['reason']} ")
 79
 80     # Give feedback
 81     print("User Feedback ---")
 82     system.give_feedback("user1", "keyboard", True)
 83
 84     # Opt out
 85     print("User Control ---")
 86     system.opt_out("user1")
 87     result = system.recommend("user1")
 88     print(f"After opt-out: {result['status']} ")
 89
 90     # Opt in
 91     print(system.opt_in("user1"))
 92
 93

```

Expected Output #3:

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

```

C:\Users\Shreeshma\Downloads\HTML Tutorials> python ethical_recommendation_system.py
--- Adding Purchases ---
✓ Purchases recorded
PS C:\Users\Shreeshma\Downloads\HTML Tutorials> & C:/Users/Shreeshma/AppData/Local/Programs/Python/Python38/python.exe c:/users/Shreeshma/Documents/AI/AI/ethical_recommendation_system.py
--- Adding Purchases ---
PS C:\Users\Shreeshma\Downloads\HTML Tutorials> & C:/Users/Shreeshma/AppData/Local/Programs/Python/Python38/python.exe c:/users/Shreeshma/Documents/AI/AI/ethical_recommendation_system.py
✓ Purchases recorded

--- Recommendations for user1 ---
Product: keyboard, Score: 0.8, Reason: Similar to your Electronics purchases
Product: monitor, Score: 0.8, Reason: Similar to your Electronics purchases

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

--- User Control ---
user1 opted out of recommendations
After opting out, user1
user1 opted in to recommendations
PS C:\Users\Shreeshma\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.

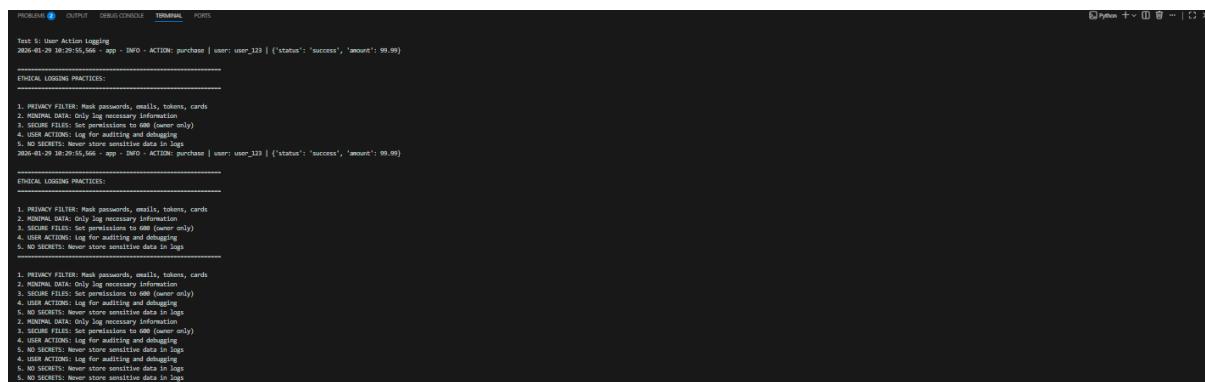
Add comments explaining ethical logging practices and privacy protection.

```
file --> Run Configuration <-- ethical_loggingpy X
C:\Users\Gowtham\Documents\AI\ES5\ethical_loggingpy>
1 ***Simple 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
10    def filter(self, record):
11        """Remove sensitive data from log entries, tokens, cards before logging"""
12        msg = str(record.msg)
13
14        # PRIVACY: Mask passwords
15        msg = re.sub(r'password=(\w+)', 'password=*****', msg, flags=re.IGNORECASE)
16
17        # PRIVACY: Mask emails (show domain only)
18        msg = re.sub(r'([a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+)', '(email****)', msg)
19
20        # PRIVACY: Mask API keys and tokens
21        msg = re.sub(r'(api_\w+_key|token|bearer)=*\w*', '*****', msg, flags=re.IGNORECASE)
22
23        # PRIVACY: Mask credit cards (show last 4 digits)
24        msg = re.sub(r'(\d{4})[^\d]*', '*****', msg, flags=re.IGNORECASE)
25
26        # PRIVACY: Mask phone numbers (show last 4 digits)
27        msg = re.sub(r'(\d{4})\d{10}', '*****', msg)
28
29        record.msg = msg
30
31    return True
32
33
34 def setup_logger(name, log_file='app.log'):
35     """Set up logger with privacy protection"""
36     logger = logging.getLogger(name)
37     logger.setLevel(logging.INFO)
38
39     # Add privacy filter
40     privacy_filter = PrivacyFilter()
41
42     # Stream Handler
43     console_handler = logging.StreamHandler()
44     console_handler.addFilter(privacy_filter)
45     console_handler.setFormatter(logging.Formatter('%(name)s - %(levelName)s - %(message)s'))
46     console_handler.setFormatter(logging.Formatter)
47     logger.addHandler(console_handler)
48
49     # File Handler
50     if log_file:
51         file_handler = logging.FileHandler(log_file)
52         file_handler.addFilter(privacy_filter)
53         file_handler.setFormatter(logging.Formatter)
54         file_handler.setFormatter(logging.Formatter)
55
56         # Set file permissions (owner read/write only)
57         import os
58         os.chmod(log_file, 0600)
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} | {user_id}'
66
67     if safe_details:
68         msg += f' | {safe_details}'
69
70     logger.info(msg)
71
72 # Example usage
73 if __name__ == '__main__':
74     print(""" Simple Ethical Logging Demo --->""")
75
76     logger = setup_logger('app', log_file='app.log')
77
78     print("Text 1: Password Masking")
79     logger.info("Login with password=secure@pass123")
80
81     print("Text 2: Email Masking")
82     logger.info("Send email to user@example.com")
83
84     print("Text 3: API key Masking")
85     logger.info("API key user_token_123456abcd")
86
87     print("Text 4: Credit Card Masking")
88     logger.info("Payment with card 4321-1234-5678-9999")
89
90     print("Value 1: User action logging")
91     log_user_action(logger, 'Purchase', 'user_123', amount=99.99)
92
93     print("Value 2: API logging")
94     print("-----")
95     print("ETICAL LOGGING PRACTICES:")
96     print("-----")
97
98 1. PRIVACY FILTER: Mask passwords, emails, tokens, cards
99 2. MINIMAL DATA: Only log necessary information
100 3. LOGGING: Log errors and warnings (not only)
101 4. USER ACTIONS: Log for auditing and debugging
102 5. NO SENSIT: Never store sensitive data in logs
103
104
```

```
file --> Run Configuration <-- ethical_loggingpy X
C:\Users\Gowtham\Documents\AI\ES5\ethical_loggingpy>
1
2
3 def setup_logger(name, log_file='app.log'):
4     """Setup logger with privacy protection"""
5     logger = logging.getLogger(name)
6     logger.setLevel(logging.INFO)
7
8     # Add privacy filter
9     os.chmod(log_file, 0600)
10
11     return logger
12
13
14 def log_user_action(logger, action, user_id, **safe_details):
15     """Log user action with only safe fields"""
16     msg = f'{action} | {user_id}'
17
18     if safe_details:
19         msg += f' | {safe_details}'
20
21     logger.info(msg)
22
23
24 # Example usage
25 if __name__ == '__main__':
26     print(""" Simple Ethical Logging Demo --->""")
27
28     logger = setup_logger('app', log_file='app.log')
29
30     print("Text 1: Password Masking")
31     logger.info("Login with password=secure@pass123")
32
33     print("Text 2: Email Masking")
34     logger.info("Send email to user@example.com")
35
36     print("Text 3: API key Masking")
37     logger.info("API key user_token_123456abcd")
38
39     print("Text 4: Credit Card Masking")
40     logger.info("Payment with card 4321-1234-5678-9999")
41
42     print("Value 1: User action logging")
43     log_user_action(logger, 'Purchase', 'user_123', amount=99.99)
44
45     print("Value 2: API logging")
46     print("-----")
47     print("ETICAL LOGGING PRACTICES:")
48     print("-----")
49
50 1. PRIVACY FILTER: Mask passwords, emails, tokens, cards
51 2. MINIMAL DATA: Only log necessary information
52 3. LOGGING: Log errors and warnings (not only)
53 4. USER ACTIONS: Log for auditing and debugging
54 5. NO SENSIT: Never store sensitive data in logs
55
56
```

Expected Output #4:

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



The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL WORKS
Python -> User Action Logging
2020-01-29 10:20:55,566 - 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. USER ACTIONS: Log for auditing and debugging
5. NO SENSITIVITY: Never store sensitive data in logs
2020-01-29 10:20:55,566 - 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. USER ACTIONS: Log for auditing and debugging
5. NO SENSITIVITY: Never store sensitive data in logs
-----
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. USER ACTIONS: Log for auditing and debugging
5. NO SENSITIVITY: Never store sensitive data in logs
5. NO SENSITIVITY: 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.

```
EXPLORER          AI-SS-49 ethical_recommendation_system.py ethical_logging.py responsible_ml_model.py
C:\Users\Seethma>Documents\AI ASS > responsible_ml_model\py...
67
68     recs, reasons = recommend_products(user_id, user_history, product_catalog)
69     for prod, reason in zip(recs, reasons):
70         print(f"({prod['name']} | {category} | {prod['category']}) -> {reason}")
71
72     # User feedback and opt-out
73     print("Would you like to provide feedback or opt out of recommendations? ")
74     feedback = input("Type 'opt_out' to stop recommendations: ")
75     if feedback.strip().lower() == 'opt out':
76         print("You have opted out of recommendations. Your preferences will be respected.")
77     else:
78         print("Thank you for your feedback: {feedback}")
79
80     # --- Ethical AI Notes ---
81     # - Transparency: Each recommendation includes an explanation.
82     # - Fairness: The system ensures diversity and avoids recommending only from the most frequent category.
83     # - User Control: Users can provide feedback or opt out at any time.
84     # - Regularly audit recommendation logic for bias and update as needed.
85     # Ensure required packages are installed
86     import sys
87     import subprocess
88
89     def install_if_missing(package):
90         try:
91             __import__(package)
92         except ImportError:
93             print(f"Installing missing package: {package}")
94             subprocess.check_call([sys.executable, "-m", "pip", "install", package])
95
96     # Install 'textblob' if not present
97     install_if_missing('textblob')
98
99     # Sentiment analysis function with bias awareness and mitigation strategies
100    from textblob import TextBlob
101
102    def analyze_sentiment(text):
103        """
104            Analyzes the sentiment of the input text.
105            Returns polarity (-1 to 1) and subjectivity (0 to 1).
106
107            Potential sources of bias in training data:
108            - Imbalanced datasets (e.g., more positive than negative samples)
109            - Presence of offensive, discriminatory, or culturally specific terms
110            - Overrepresentation or underrepresentation of certain topics or groups
111
112            Strategies to mitigate bias:
113            - Balance the dataset across sentiment classes and demographic groups
114            - Remove or flag offensive/discriminatory terms during preprocessing
115            - Encourage diverse representation in training data samples
116            - Document known limitations and test for bias regularly
117            - Involve domain experts in dataset curation
118
119            # Example: Using Textblob for simple sentiment analysis
120            blob = TextBlob(text)
121            polarity = blob.sentiment.polarity
122            subjectivity = blob.sentiment.subjectivity
123            return polarity, subjectivity
124
125    # Example usage
126    if __name__ == "__main__":
127        user_text = input("Enter text for sentiment analysis: ")
128        polarity, subjectivity = analyze_sentiment(user_text)
129        print(f"Polarity: {polarity}, Subjectivity: {subjectivity}")
130
131    # Note: For production, train your own model on a carefully curated dataset and regularly audit for bias.
132    # The above function uses TextBlob, which is trained on general-purpose data and may inherit its biases.
```

Expected Output #5:

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