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.

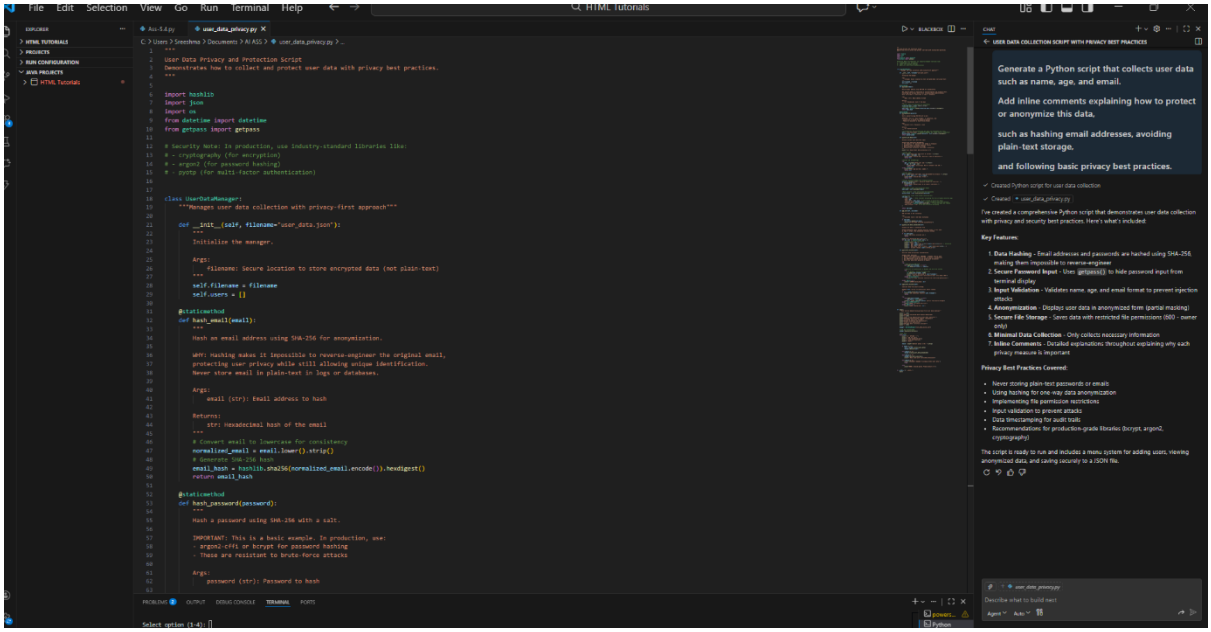
PROMPT

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

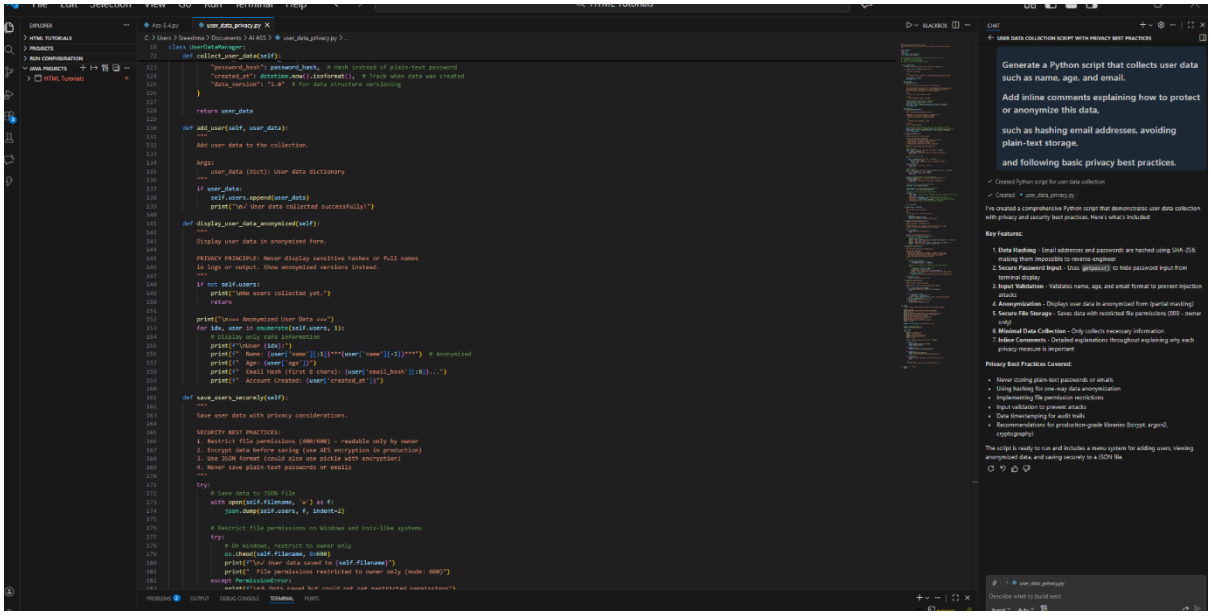
Add inline comments explaining how to protect or anonymize this data,

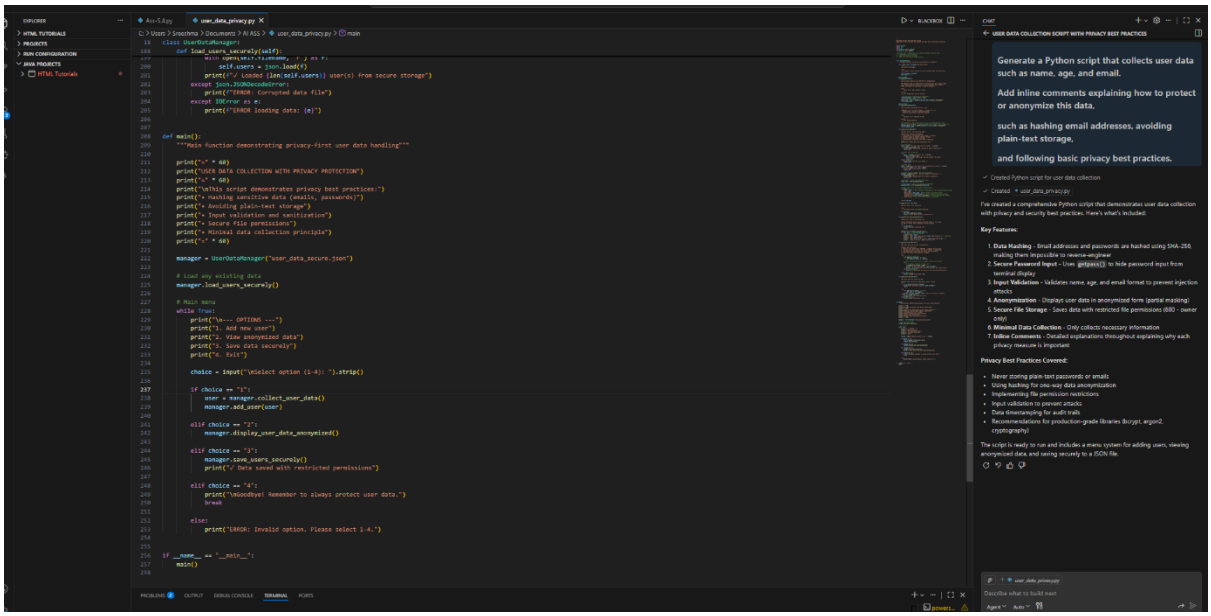
such as hashing email addresses, avoiding plain-text storage,

and following basic privacy best practices.


File Edit Selection View Go Run Terminal Help
HTML Tutorials
EXPLORER
HTML TUTORIALS
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
HTML Tutorials
Ass-5.4.py
user_data_privacy.py
C: > Users > Sreelekha > Documents > AI ASS > user_data_privacy.py > ...
"""
User Data Privacy and Protection Script
Demonstrates how to collect and protect user data with privacy best practices.
"""
import hashlib
import json
import os
from datetime import datetime
from getpass import getpass
# Security Note: In production, use industry-standard libraries like:
# - cryptography (for encryption)
# - argon2 (for password hashing)
# - pyotp (for multi-factor authentication)
class UserDataManager:
"""Manages user data collection with privacy-first approach"""
def __init__(self, filename="user_data.json"):
"""
Initialize the manager.
Args:
filename: Secure location to store encrypted data (not plain-text)
"""
self.filename = filename
self.users = []
@staticmethod
def hash_email(email):
"""
Hash an email address using SHA-256 for anonymization.
WHY: Hashing makes it impossible to reverse-engineer the original email,
protecting user privacy while still allowing unique identification.
Never store email in plain-text in logs or databases.
Args:
email (str): Email address to hash
Returns:
str: Hexadecimal hash of the email
"""
# Convert email to lowercase for consistency
normalized_email = email.lower().strip()
# Generate SHA-256 hash
email_hash = hashlib.sha256(normalized_email.encode()).hexdigest()
return email_hash
@staticmethod
def hash_password(password):
"""
Hash a password using SHA-256 with a salt.
IMPORTANT: This is a basic example. In production, use:
- argon2-cffi or bcrypt for password hashing
- These are resistant to brute-force attacks
Args:
password (str): Password to hash
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Select option (1-4):
powers...
Python
CHAT
USER DATA COLLECTION SCRIPT WITH PRIVACY BEST PRACTICES
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.
Created Python script for user data collection
Created + user_data_privacy.py
I've created a comprehensive Python script that demonstrates user data collection with privacy and security best practices. Here's what's included:
Key Features:
1. Data Hashing - Email addresses and passwords are hashed using SHA-256, making them impossible to reverse-engineer
2. Secure Password Input - Uses getpass() to hide password input from terminal display
3. Input Validation - Validates name, age, and email format to prevent injection attacks
4. Anonymization - Displays user data in anonymized form (partial masking)
5. Secure File Storage - Saves data with restricted file permissions (600 - owner only)
6. Minimal Data Collection - Only collects necessary information
7. Inline Comments - Detailed explanations throughout explaining why each privacy measure is important
Privacy Best Practices Covered:
Never storing plain-text passwords or emails
Using hashing for one-way data anonymization
Implementing file permission restrictions
Input validation to prevent attacks
Data timestamping for audit trails
Recommendations for production-grade libraries (bcrypt, argon2, cryptography)
The script is ready to run and includes a menu system for adding users, viewing anonymized data, and saving securely to a JSON file.
user_data_privacy.py
Describe what to build next
Agent
Auto


EXPLORER
HTML TUTORIALS
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
HTML Tutorials
Ass-5.4.py
user_data_privacy.py
C: > Users > Sreelekha > Documents > AI ASS > user_data_privacy.py > ...
class UserDataManager:
def hash_password(password):
Returns:
str: Hashed password
"""
# In production, add a unique salt per user (not hardcoded like this)
salt = "user_salt_12345" # SECURITY WARNING: Use random salt in production!
password_hash = hashlib.sha256((password + salt).encode()).hexdigest()
return password_hash
def collect_user_data(self):
"""
Securely collect user data from input.
PRIVACY BEST PRACTICES IMPLEMENTED:
1. Use getpass() for passwords (not echoed to terminal)
2. Validate input to prevent injection attacks
3. Hash sensitive data before storage
4. Minimize data collection (only what's necessary)
"""
print("\n=== Secure User Data Collection ===")
# Collect name
name = input("Enter your name (will be stored): ").strip()
if not name or len(name) < 2:
print("ERROR: Invalid name. Must be at least 2 characters.")
return None
# Collect and validate age
try:
age = int(input("Enter your age: ").strip())
if age < 0 or age > 150:
print("ERROR: Invalid age. Must be between 0 and 150.")
return None
except ValueError:
print("ERROR: Age must be a number.")
return None
# Collect email
email = input("Enter your email (will be hashed for privacy): ").strip()
if '@' not in email or '.' not in email:
print("ERROR: Invalid email format.")
return None
# Collect password (hidden from terminal display)
password = getpass("Enter a password (hidden for security): ")
if len(password) < 8:
print("ERROR: Password must be at least 8 characters.")
return None
# Hash email - never store plain-text email
email_hash = self.hash_email(email)
# Hash password - never store plain-text passwords
password_hash = self.hash_password(password)
# Create user record with minimal sensitive data
user_data = {
"name": name, # Note: Consider encrypting this too in highly sensitive apps
"age": age,
"email_hash": email_hash, # Hash instead of plain-text email
"password_hash": password_hash, # Hash instead of plain-text password
CHAT
USER DATA COLLECTION SCRIPT WITH PRIVACY BEST PRACTICES
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.
Created Python script for user data collection
Created + user_data_privacy.py
I've created a comprehensive Python script that demonstrates user data collection with privacy and security best practices. Here's what's included:
Key Features:
1. Data Hashing - Email addresses and passwords are hashed using SHA-256, making them impossible to reverse-engineer
2. Secure Password Input - Uses getpass() to hide password input from terminal display
3. Input Validation - Validates name, age, and email format to prevent injection attacks
4. Anonymization - Displays user data in anonymized form (partial masking)
5. Secure File Storage - Saves data with restricted file permissions (600 - owner only)
6. Minimal Data Collection - Only collects necessary information
7. Inline Comments - Detailed explanations throughout explaining why each privacy measure is important
Privacy Best Practices Covered:
Never storing plain-text passwords or emails
Using hashing for one-way data anonymization
Implementing file permission restrictions
Input validation to prevent attacks
Data timestamping for audit trails
Recommendations for production-grade libraries (bcrypt, argon2, cryptography)
The script is ready to run and includes a menu system for adding users, viewing anonymized data, and saving securely to a JSON file.
user_data_privacy.py
Describe what to build next



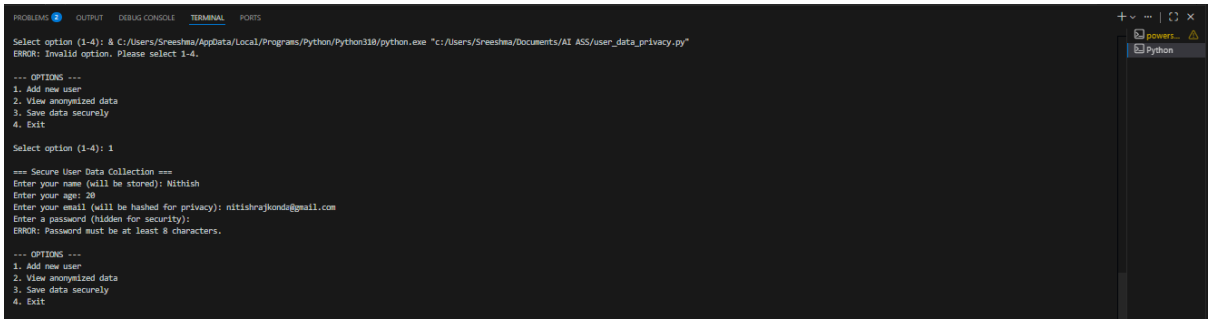


Expected Output #1:

• 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).



Task Description #2:

• Ask Copilot to generate a Python function for sentiment analysis.

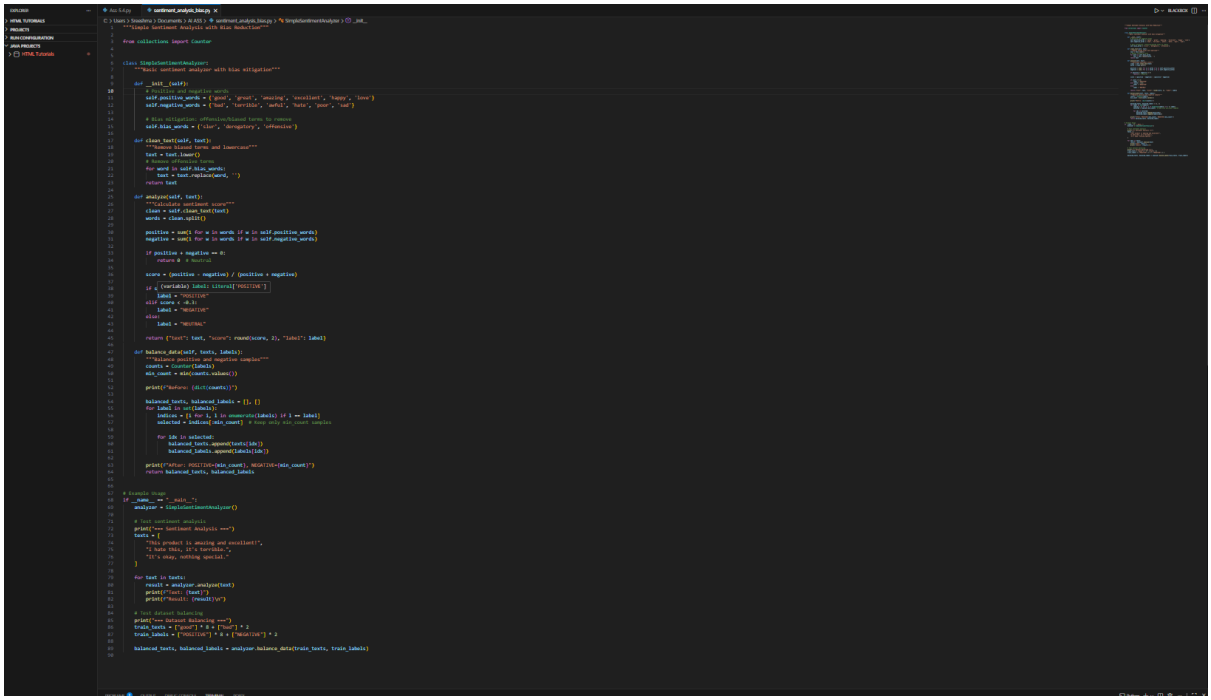
Then prompt Copilot to identify and handle potential biases in the data.

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.

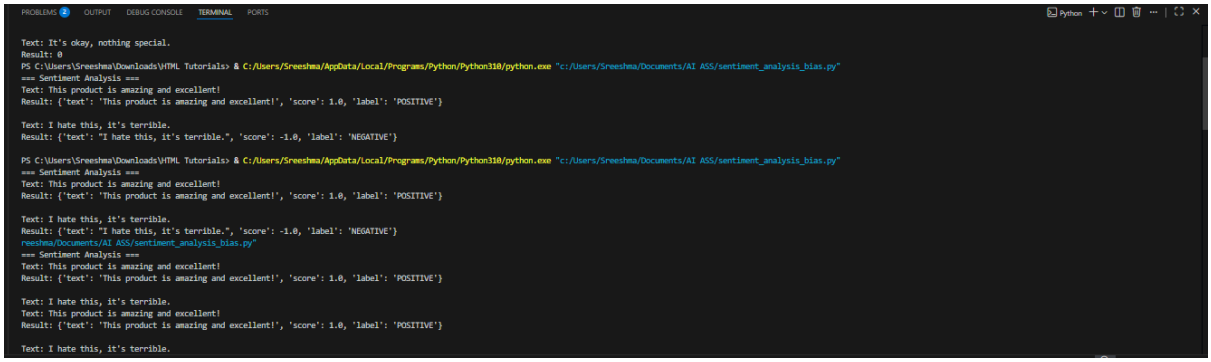


Expected Output #2:

• Copilot-generated code with additions or comments addressing

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

offensive terms).



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

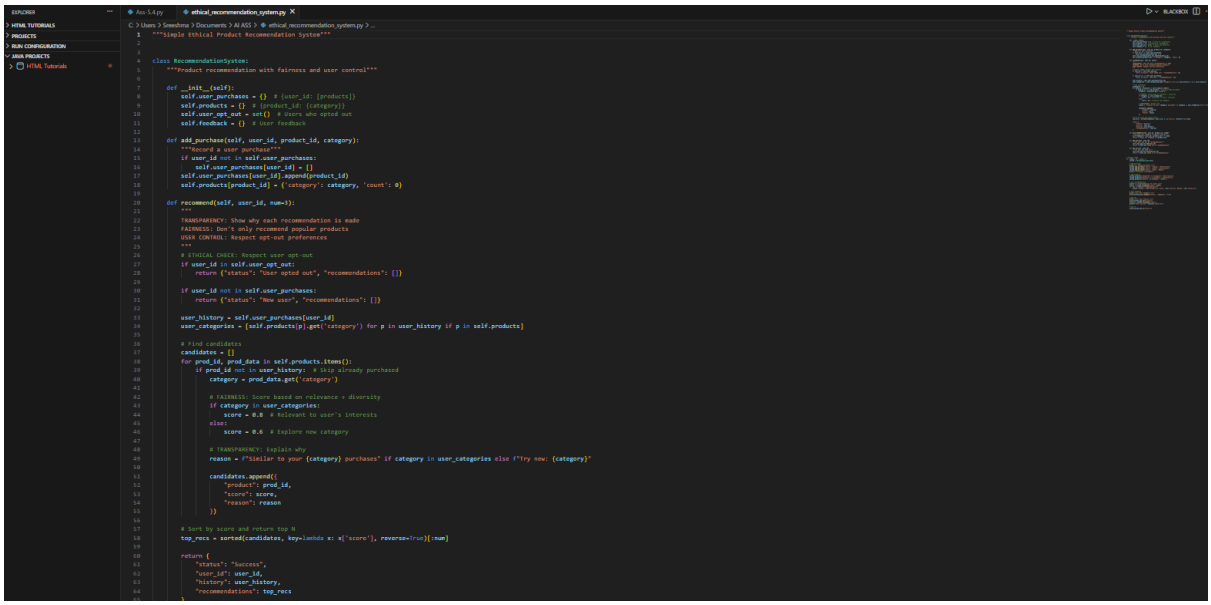
PROMPT: # Generate a Python program that recommends products based on user purchase history.

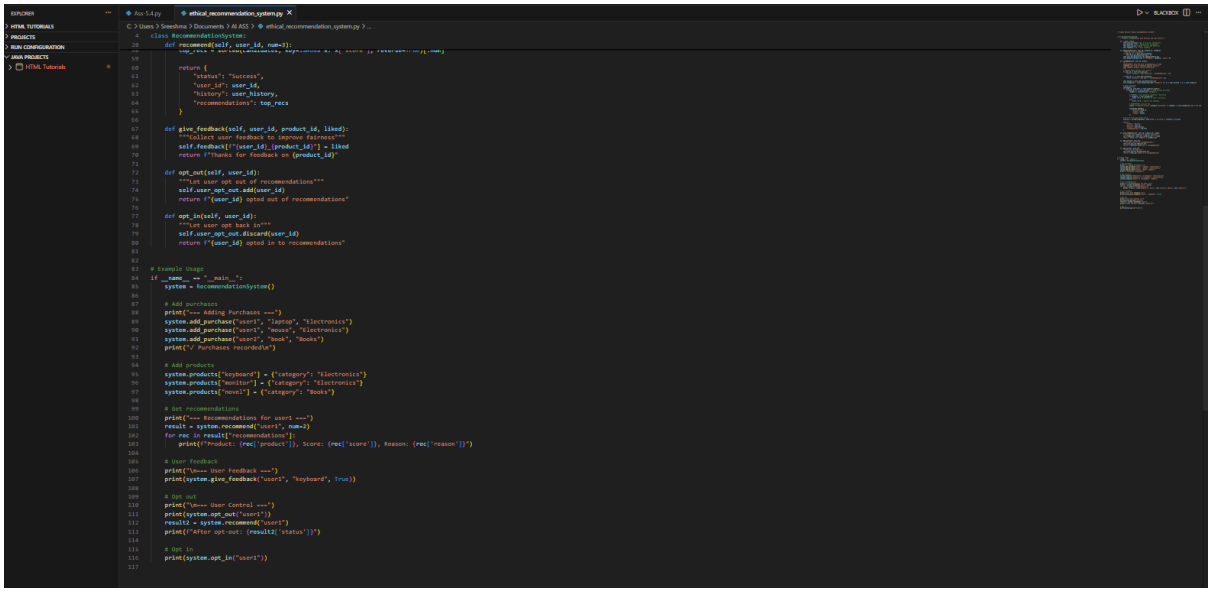
Follow ethical AI guidelines such as transparency, fairness, and user control.

Add comments explaining how recommendations are generated,

avoid favoritism toward only popular products,

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

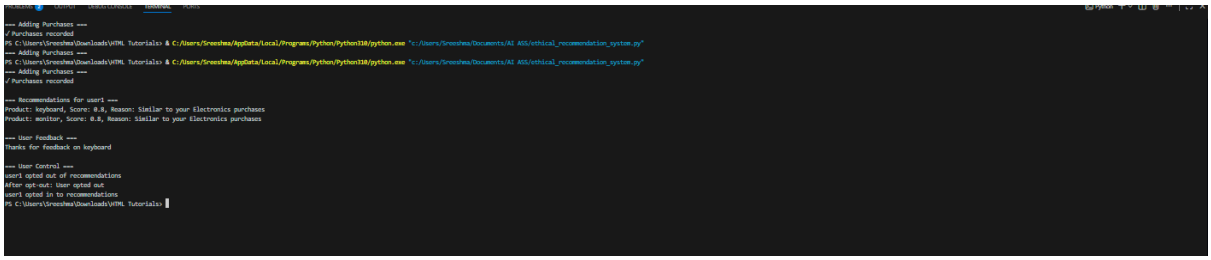




Expected Output #3:

• Copilot suggestions that include explanations, fairness checks

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



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.

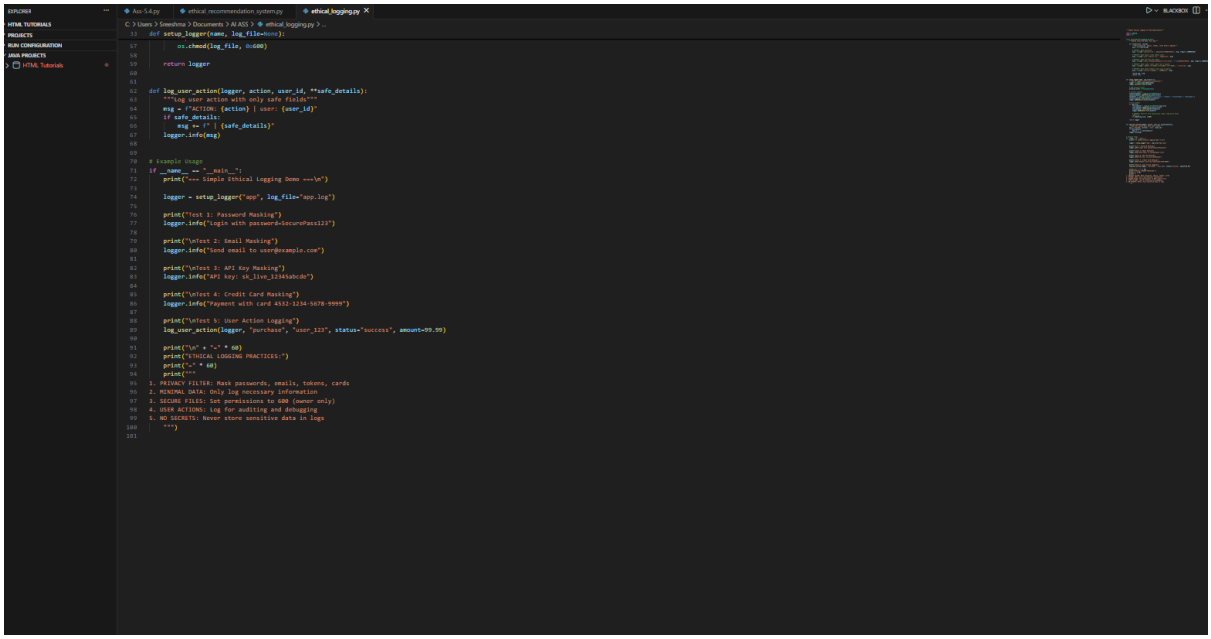
PROMPT: # Generate logging functionality for a Python web application.

Ensure logs do NOT store sensitive information such as passwords,

emails, or personal identifiers.

Add comments explaining ethical logging practices and privacy protection.

```
"""Simple Ethical Logging for Web Applications"""

import logging
import re


class PrivacyFilter(logging.Filter):
    """Remove sensitive data from logs"""

    def filter(self, record):
        """Mask passwords, emails, tokens, cards before logging"""
        msg = str(record.msg)

        # PRIVACY: Mask passwords
        msg = re.sub(r'password=\S+', 'password=***REDACTED***', msg, flags=re.IGNORECASE)

        # PRIVACY: Mask emails (show domain only)
        msg = re.sub(r'[\w\.-]+@[\w\.-]+', '[email***]', msg)

        # PRIVACY: Mask API keys and tokens
        msg = re.sub(r'(api[_-]?key|token|bearer)\s*=\s*\S+', r'\1=***REDACTED***', msg, flags=re.IGNORECASE)

        # PRIVACY: Mask credit cards (show last 4 digits)
        msg = re.sub(r'\d{4}[-\s]?\d{4}[-\s]?\d{4}[-\s]?(\d{4})', r'[CC***\1]', msg)

        # PRIVACY: Mask phone numbers (show last 4 digits)
        msg = re.sub(r'\+?[\d\-\s]{10,}', '[PHONE***]', msg)

        record.msg = msg
        return True


def setup_logger(name, log_file=None):
    """Set up logger with privacy protection"""
    logger = logging.getLogger(name)
    logger.setLevel(logging.INFO)

    # Add privacy filter
    privacy_filter = PrivacyFilter()

    # Console handler
    console_handler = logging.StreamHandler()
    console_handler.addFilter(privacy_filter)
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    console_handler.setFormatter(formatter)
    logger.addHandler(console_handler)

    # File handler
    if log_file:
        file_handler = logging.FileHandler(log_file)
        file_handler.addFilter(privacy_filter)
        file_handler.setFormatter(formatter)
        logger.addHandler(file_handler)

        # PRIVACY: Restrict file permissions (owner read/write only)
        import os
        os.chmod(log_file, 0o600)

    return logger


def log_user_action(logger, action, user_id, **safe_details):
    """Log user action with only safe fields"""
    msg = f"ACTION: {action} | user: {user_id}"
    if safe_details:
        msg += f" | {safe_details}"
    logger.info(msg)


# Example usage
if __name__ == "__main__":
    print("=== Simple Ethical Logging Demo ===\n")

    logger = setup_logger("app", log_file="app.log")

    print("Test 1: Password Masking")
    logger.info("Login with password=SecurePass123")

    print("\nTest 2: Email Masking")
    logger.info("Send email to user@example.com")
```



```
def setup_logger(name, log_file=None):
        os.chmod(log_file, 0o600)

    return logger


def log_user_action(logger, action, user_id, **safe_details):
    """Log user action with only safe fields"""
    msg = f"ACTION: {action} | user: {user_id}"
    if safe_details:
        msg += f" | {safe_details}"
    logger.info(msg)


# Example usage
if __name__ == "__main__":
    print("=== Simple Ethical Logging Demo ===\n")

    logger = setup_logger("app", log_file="app.log")

    print("Test 1: Password Masking")
    logger.info("Login with password=SecurePass123")

    print("\nTest 2: Email Masking")
    logger.info("Send email to user@example.com")

    print("\nTest 3: API Key Masking")
    logger.info("API key: sk_live_12345abcde")

    print("\nTest 4: Credit Card Masking")
    logger.info("Payment with card 4532-1234-5678-9999")

    print("\nTest 5: User Action Logging")
    log_user_action(logger, "purchase", "user_123", status="success", amount=99.99)

    print("\n" + "-" * 60)
    print("ETHICAL LOGGING PRACTICES:")
    print("-" * 60)
    print("""
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 SECRETS: Never store sensitive data in logs
    """)
```

Expected Output #4:

• Logging code that avoids saving personal identifiers (e.g.,

passwords, emails), and includes comments about ethical logging

practices.

Task Description #5:

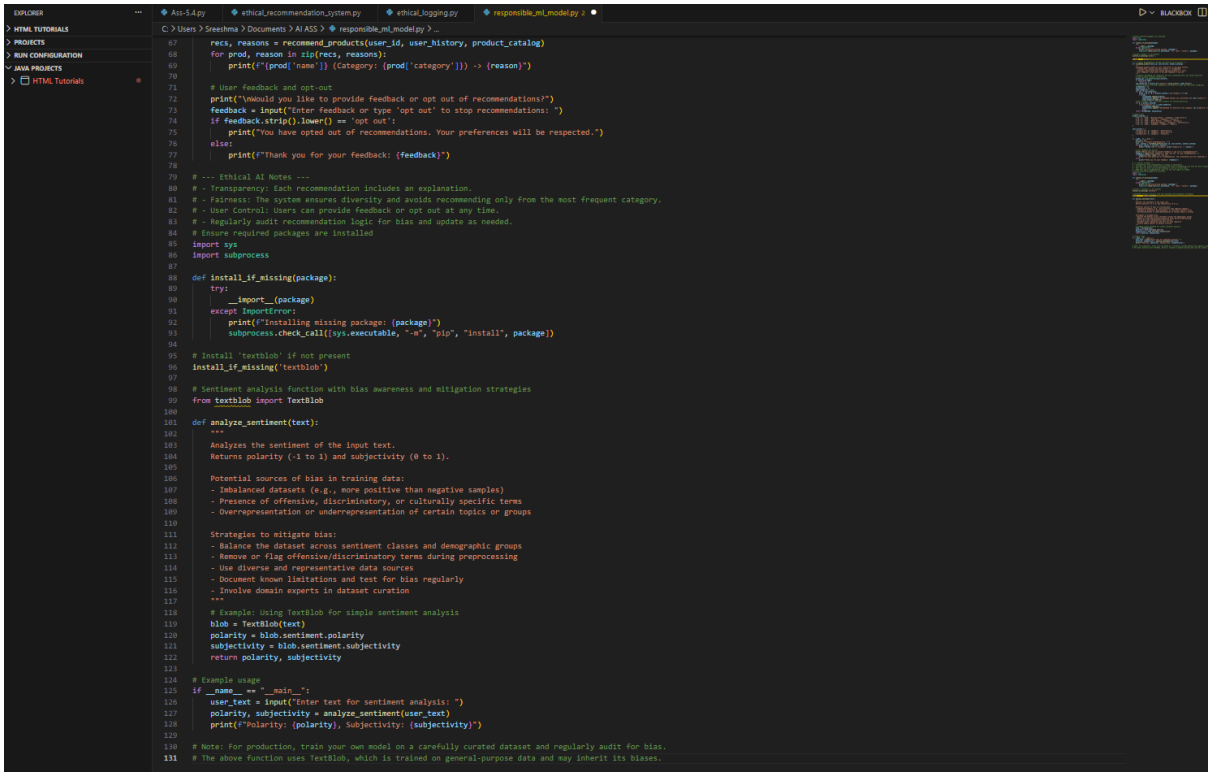
• Ask Copilot to generate a machine learning model. Then, prompt

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.



```
    recs, reasons = recommend_products(user_id, user_history, product_catalog)
    for prod, reason in zip(recs, reasons):
        print(f"{prod['name']} (Category: {prod['category']}) -> {reason}")

    # User feedback and opt-out
    print("\nWould 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(f"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.
# Ensure 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 flag offensive/discriminatory terms 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.
# 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.