AI Assisted Coding LAB ASSIGNMENT-5.4

NAME: K.MAHESH REDDY

Batch:14

Hall Ticket:2303A510H1

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.

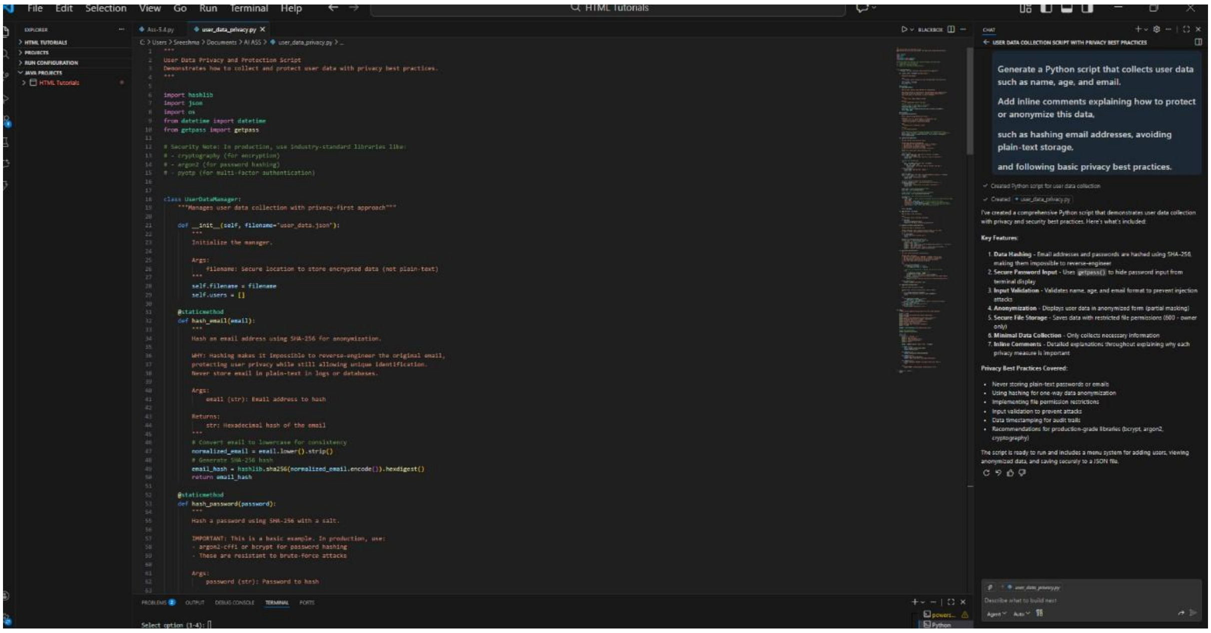
PROMPT

# 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.



```python
"""
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

```

```
Select option (1-4):
```

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

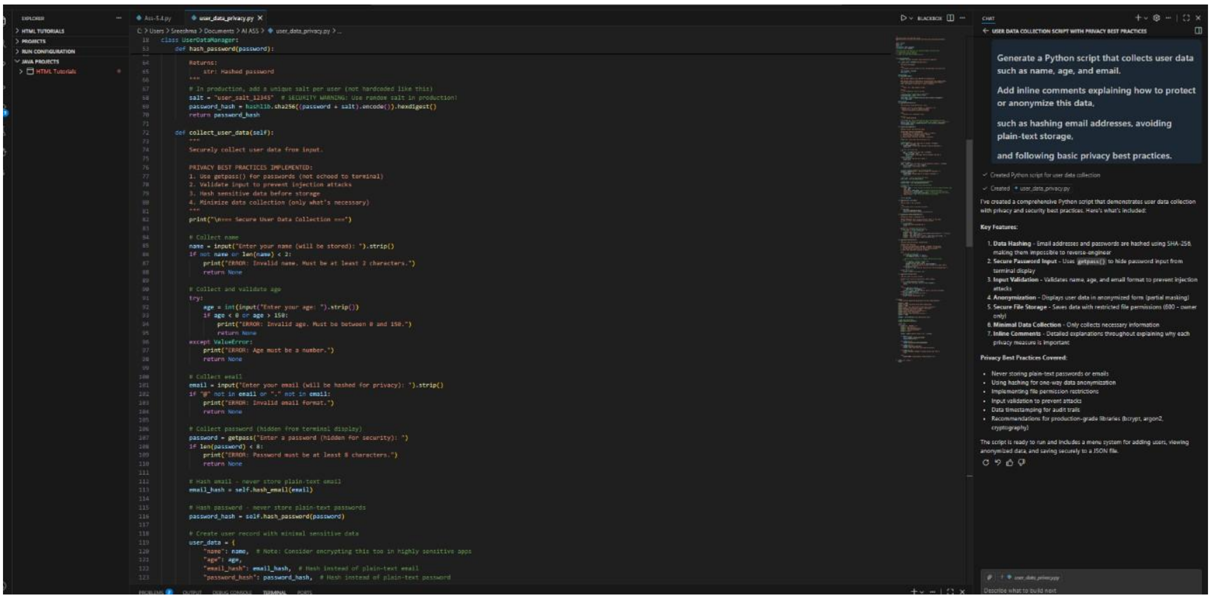
Privacy Best Practices Covered:

- 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



```python
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
```

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

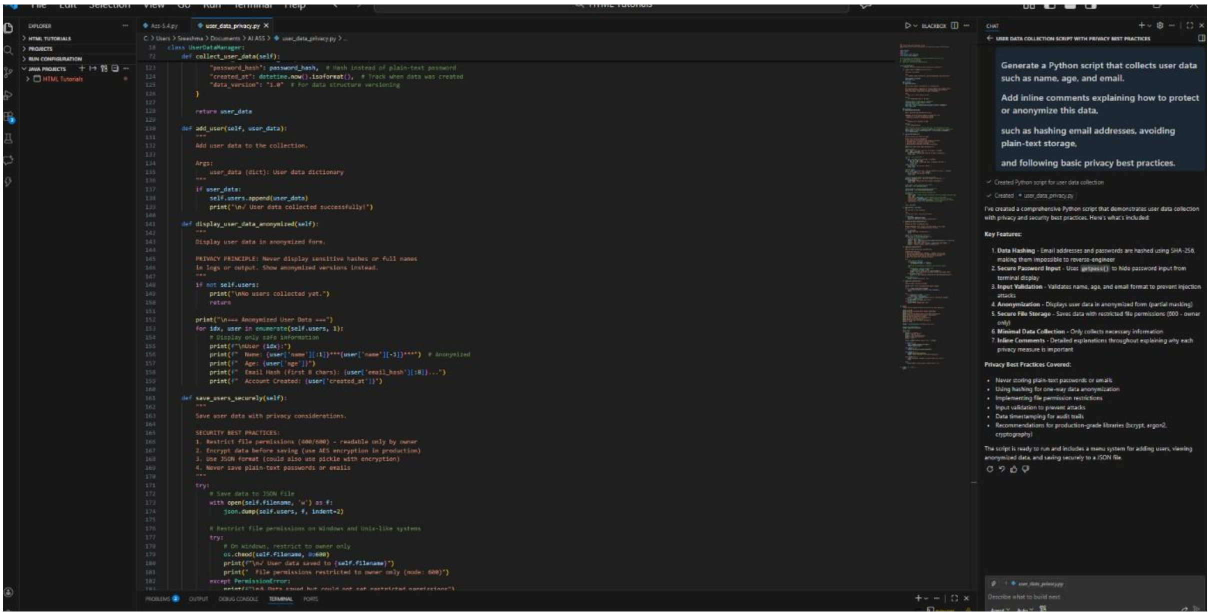
Privacy Best Practices Covered:

- 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



```
class UserDataManager:
    def collect_user_data(self):
            "password_hash": password_hash,  # Hash instead of plain-text password
            "created_at": datetime.now().isoformat(),  # Track when data was created
            "data_version": "1.0"  # For data structure versioning
        }

        return user_data

    def add_user(self, user_data):
        """
        Add user data to the collection.

        Args:
            user_data (dict): User data dictionary
        """
        if user_data:
            self.users.append(user_data)
            print("\n✓ User data collected successfully!")

    def display_user_data_anonymized(self):
        """
        Display user data in anonymized form.

        PRIVACY PRINCIPLE: Never display sensitive hashes or full names
        in logs or output. Show anonymized versions instead.
        """
        if not self.users:
            print("\nNo users collected yet.")
            return

        print("\n=== Anonymized User Data ===")
        for idx, user in enumerate(self.users, 1):
            # Display only safe information
            print(f"\nUser {idx}:")
            print(f"  Name: {user['name'][:1]}***{user['name'][-1]}***")  # Anonymized
            print(f"  Age: {user['age']}")
            print(f"  Email Hash (first 8 chars): {user['email_hash'][:8]}...")
            print(f"  Account Created: {user['created_at']}")

    def save_users_securely(self):
        """
        Save user data with privacy considerations.

        SECURITY BEST PRACTICES:
        1. Restrict file permissions (400/600) - readable only by owner
        2. Encrypt data before saving (use AES encryption in production)
        3. Use JSON format (could also use pickle with encryption)
        4. Never save plain-text passwords or emails
        """
        try:
            # Save 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
            try:
                # On Windows, restrict to owner only
                os.chmod(self.filename, 0o600)
                print(f"\n✓ User data saved to {self.filename}")
                print("  File permissions restricted to owner only (mode: 600)")
            except PermissionError:
```

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:

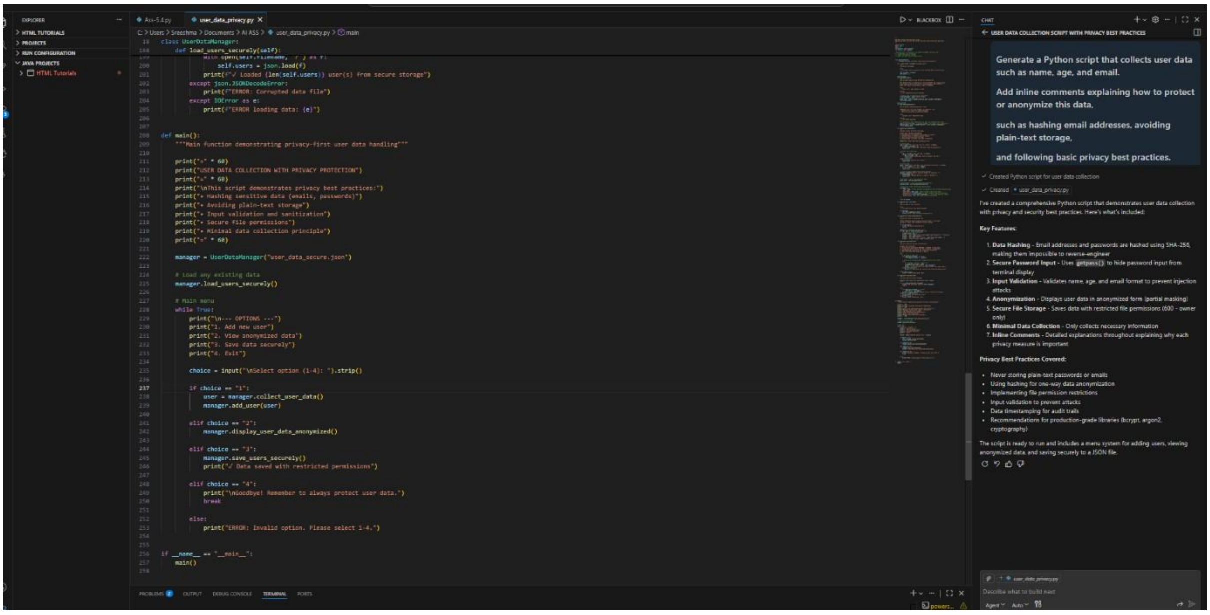
Key Features:

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.



```
class UserDataManager:
    def load_users_securely(self):
                self.users = json.load(f)
            print(f"✓ Loaded {len(self.users)} user(s) from secure storage")
        except json.JSONDecodeError:
            print(f"ERROR: Corrupted data file")
        except IOError as e:
            print(f"ERROR loading data: {e}")


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

    print("=" * 60)
    print("USER DATA COLLECTION WITH PRIVACY PROTECTION")
    print("=" * 60)
    print("\nThis script demonstrates privacy best practices:")
    print("• Hashing sensitive data (emails, passwords)")
    print("• Avoiding plain-text storage")
    print("• Input validation and sanitization")
    print("• Secure file permissions")
    print("• Minimal data collection principle")
    print("=" * 60)

    manager = UserDataManager("user_data_secure.json")

    # Load any existing data
    manager.load_users_securely()

    # Main menu
    while True:
        print("\n--- OPTIONS ---")
        print("1. Add new user")
        print("2. View anonymized data")
        print("3. Save data securely")
        print("4. Exit")

        choice = input("\nSelect option (1-4): ").strip()

        if choice == "1":
            user = manager.collect_user_data()
            manager.add_user(user)

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

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

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

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


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

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

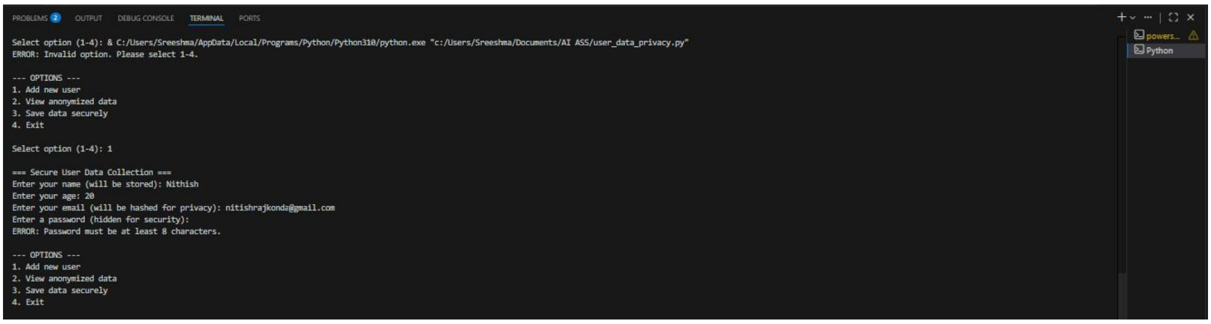
Privacy Best Practices Covered:

- 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.

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).



```
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): nitishrajkonda@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.

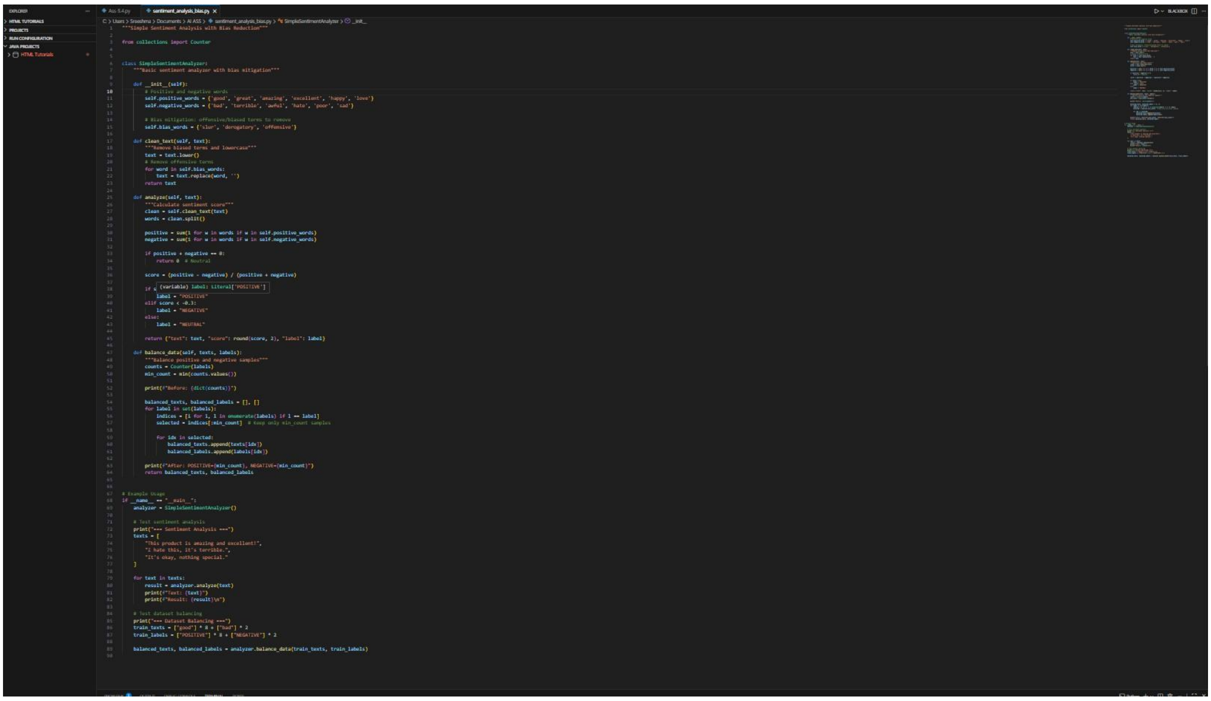
Then prompt Copilot to identify and handle potential biases in the data.

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.



```python
"""Simple Sentiment Analysis with Bias Reduction"""

from collections import Counter


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

    def __init__(self):
        # Positive and negative words
        self.positive_words = {'good', 'great', 'amazing', 'excellent', 'happy', 'love'}
        self.negative_words = {'bad', 'terrible', 'awful', 'hate', 'poor', 'sad'}

        # Bias mitigation: offensive/biased terms to remove
        self.bias_words = {'slur', 'derogatory', 'offensive'}

    def clean_text(self, text):
        """Remove biased terms and lowercase"""
        text = text.lower()
        # Remove offensive terms
        for word in self.bias_words:
            text = text.replace(word, '')
        return text

    def analyze(self, text):
        """Calculate sentiment score"""
        clean = self.clean_text(text)
        words = clean.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  # Neutral

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

        if s (variable) label: Literal['POSITIVE']
            label = "POSITIVE"
        elif score < -0.3:
            label = "NEGATIVE"
        else:
            label = "NEUTRAL"

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

    def balance_data(self, texts, labels):
        """Balance positive and negative samples"""
        counts = Counter(labels)
        min_count = min(counts.values())

        print(f"Before: {dict(counts)}")

        balanced_texts, balanced_labels = [], []
        for label in set(labels):
            indices = [i for i, l in enumerate(labels) if l == label]
            selected = indices[:min_count]  # Keep only min_count samples

            for idx in selected:
                balanced_texts.append(texts[idx])
                balanced_labels.append(labels[idx])

        print(f"After: POSITIVE={min_count}, NEGATIVE={min_count}")
        return balanced_texts, balanced_labels


# Example Usage
if __name__ == "__main__":
    analyzer = SimpleSentimentAnalyzer()

    # Test sentiment analysis
    print("=== Sentiment Analysis ===")
    texts = [
        "This product is amazing and excellent!",
        "I hate this, it's terrible.",
        "It's okay, nothing special."
    ]

    for text in texts:
        result = analyzer.analyze(text)
        print(f"Text: {text}")
        print(f"Result: {result}\n")

    # Test dataset balancing
    print("=== Dataset Balancing ===")
    train_texts = ["good"] * 8 + ["bad"] * 2
    train_labels = ["POSITIVE"] * 8 + ["NEGATIVE"] * 2

    balanced_texts, balanced_labels = analyzer.balance_data(train_texts, train_labels)
```

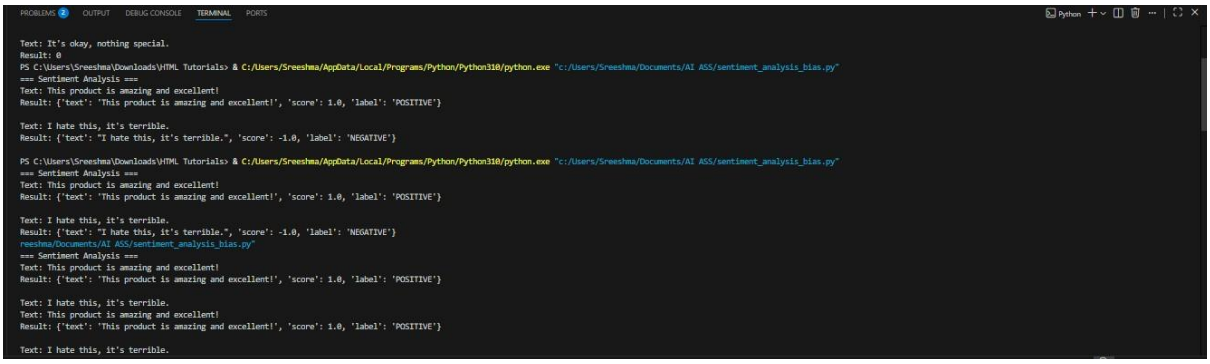
Ass 5.4.py
sentiment_analysis_bias.py
C: > Users > Sreeshma > Documents > AI ASS > sentiment_analysis_bias.py > SimpleSentimentAnalyzer > __init__

Expected Output #2:

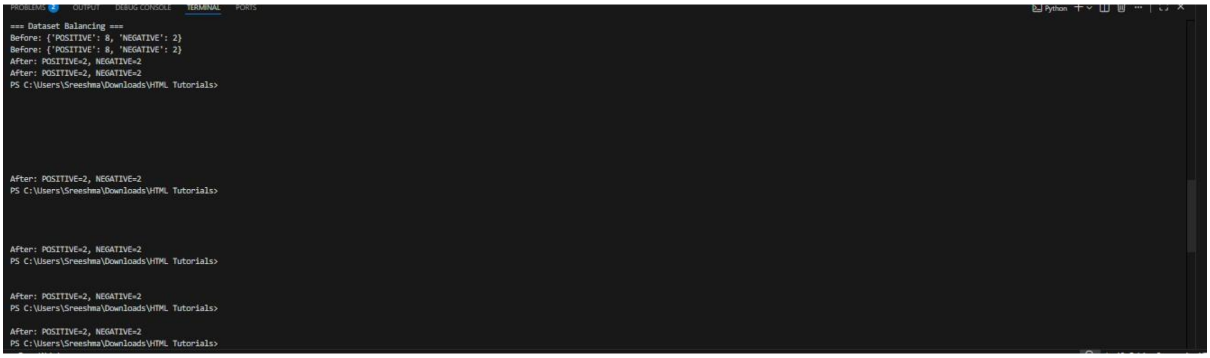
- Copilot-generated code with additions or comments addressing

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

offensive terms).



```
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'}
reeshma/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.
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.
```



```
=== Dataset Balancing ===
Before: {'POSITIVE': 8, 'NEGATIVE': 2}
Before: {'POSITIVE': 8, 'NEGATIVE': 2}
After: POSITIVE=2, NEGATIVE=2
After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreeshma\Downloads\HTML Tutorials>




After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreeshma\Downloads\HTML Tutorials>



After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreeshma\Downloads\HTML Tutorials>


After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreeshma\Downloads\HTML Tutorials>


After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreeshma\Downloads\HTML Tutorials>
```

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

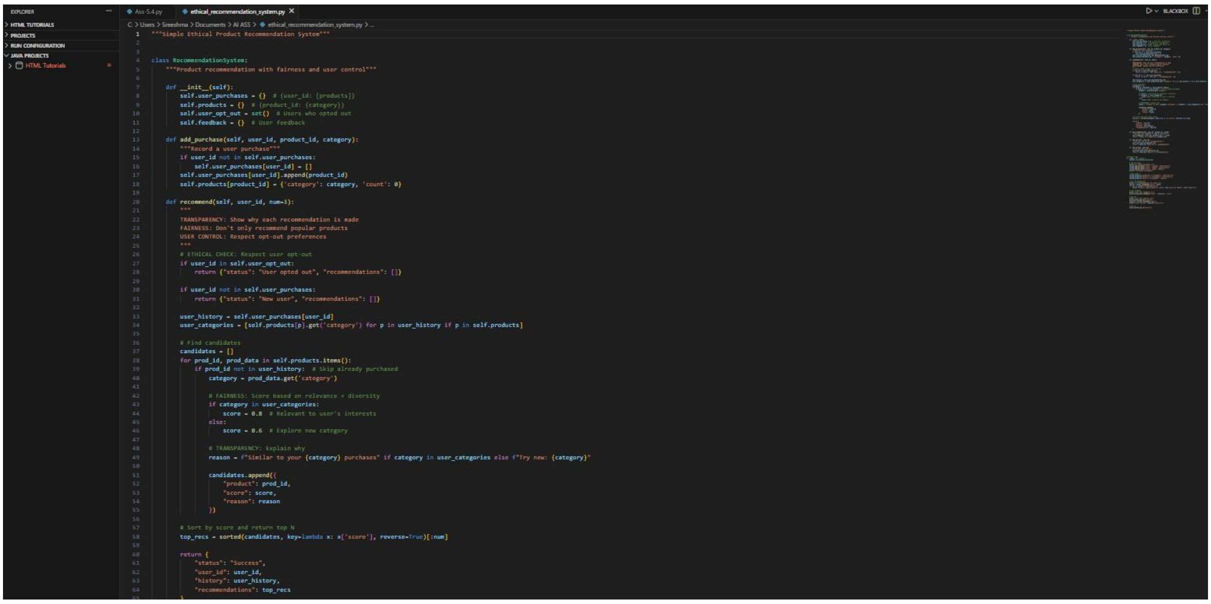
PROMPT: # Generate a Python program that recommends products based on user purchase history.

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

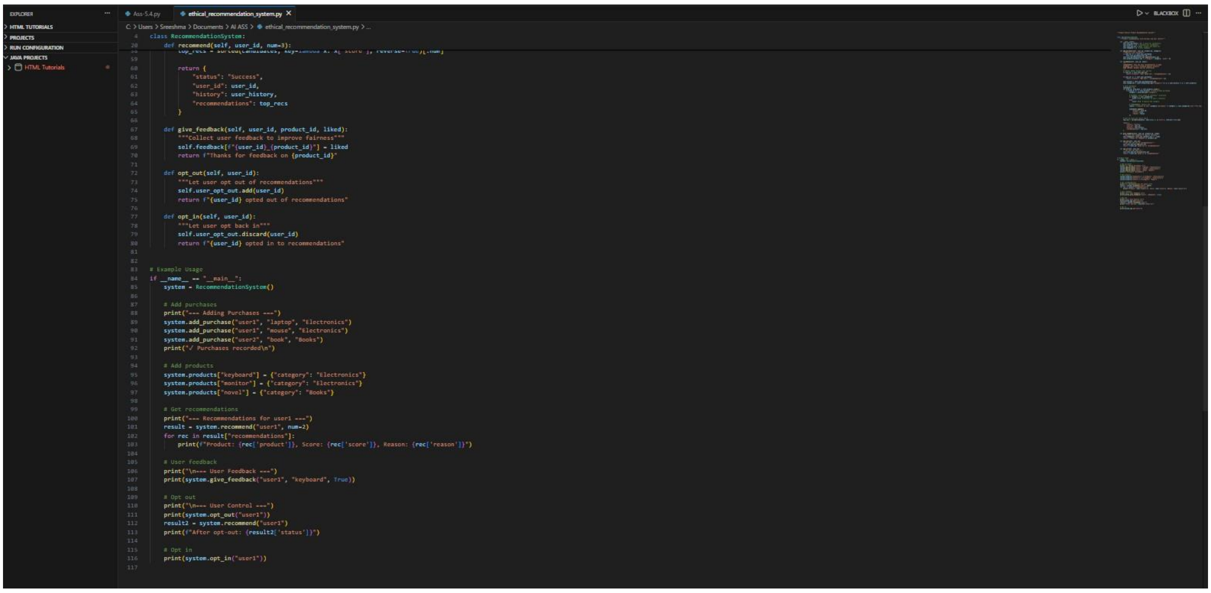
# Add comments explaining how recommendations are generated,

# avoid favoritism toward only popular products,

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


EXPLORER
HTML TUTORIALS
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
HTML Tutorials
Ass-5.4.py
ethical_recommendation_system.py
C: > Users > Sreeshma > Documents > AI ASS > ethical_recommendation_system.py > ...
BLACKBOX

```python
"""Simple Ethical Product Recommendation System"""


class RecommendationSystem:
    """Product recommendation with fairness and user control"""

    def __init__(self):
        self.user_purchases = {}  # {user_id: [products]}
        self.products = {}  # {product_id: {category}}
        self.user_opt_out = set()  # Users who opted out
        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.products[product_id] = {'category': category, 'count': 0}

    def recommend(self, user_id, num=3):
        """
        TRANSPARENCY: Show why each recommendation is made
        FAIRNESS: Don't only recommend popular products
        USER CONTROL: Respect opt-out preferences
        """
        # ETHICAL CHECK: Respect user opt-out
        if user_id in self.user_opt_out:
            return {"status": "User opted out", "recommendations": []}

        if user_id not in self.user_purchases:
            return {"status": "New user", "recommendations": []}

        user_history = self.user_purchases[user_id]
        user_categories = [self.products[p].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')

                # FAIRNESS: Score based on relevance + diversity
                if category in user_categories:
                    score = 0.8  # Relevant to user's interests
                else:
                    score = 0.6  # Explore new category

                # TRANSPARENCY: Explain why
                reason = f"Similar to your {category} purchases" if category in user_categories else f"Try new: {category}"

                candidates.append({
                    "product": prod_id,
                    "score": score,
                    "reason": reason
                })

        # Sort by score and return top N
        top_recs = sorted(candidates, key=lambda x: x['score'], reverse=True)[:num]

        return {
            "status": "Success",
            "user_id": user_id,
            "history": user_history,
            "recommendations": top_recs
        }
```



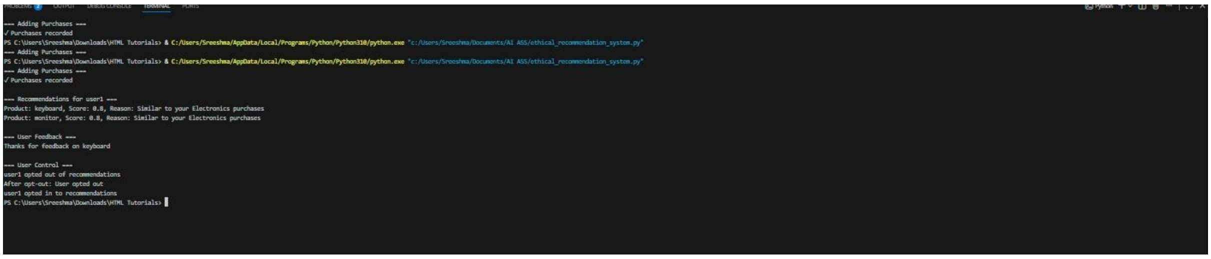
Ass-5.4.py
ethical_recommendation_system.py
C: > Users > Sreeshma > Documents > AI ASS > ethical_recommendation_system.py > ...

```python
class RecommendationSystem:
    def recommend(self, user_id, num=3):

        return {
            "status": "Success",
            "user_id": user_id,
            "history": user_history,
            "recommendations": top_recs
        }

    def give_feedback(self, user_id, product_id, liked):
        """Collect user feedback to improve fairness"""
        self.feedback[f"{user_id}_{product_id}"] = liked
        return f"Thanks for feedback on {product_id}"

    def opt_out(self, user_id):
        """Let user opt out of recommendations"""
        self.user_opt_out.add(user_id)
        return f"{user_id} opted out of recommendations"

    def opt_in(self, user_id):
        """Let user opt back in"""
        self.user_opt_out.discard(user_id)
        return f"{user_id} opted in to recommendations"


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

    # Add purchases
    print("=== Adding Purchases ===")
    system.add_purchase("user1", "laptop", "Electronics")
    system.add_purchase("user1", "mouse", "Electronics")
    system.add_purchase("user2", "book", "Books")
    print("✓ Purchases recorded\n")

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

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

    # User feedback
    print("\n=== User Feedback ===")
    print(system.give_feedback("user1", "keyboard", True))

    # Opt out
    print("\n=== User Control ===")
    print(system.opt_out("user1"))
    result2 = system.recommend("user1")
    print(f"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\Sreeshma\Downloads\HTML Tutorials> & C:/Users/Sreeshma/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Sreeshma/Documents/AI ASS/ethical_recommendation_system.py"
=== Adding Purchases ===
PS C:\Users\Sreeshma\Downloads\HTML Tutorials> & C:/Users/Sreeshma/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Sreeshma/Documents/AI ASS/ethical_recommendation_system.py"
=== Adding Purchases ===
✓ 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 opt-out: User opted out
user1 opted in to recommendations
PS C:\Users\Sreeshma\Downloads\HTML Tutorials> 
```

Task Description #4:

- Prompt Copilot to generate logging functionality in a Python web

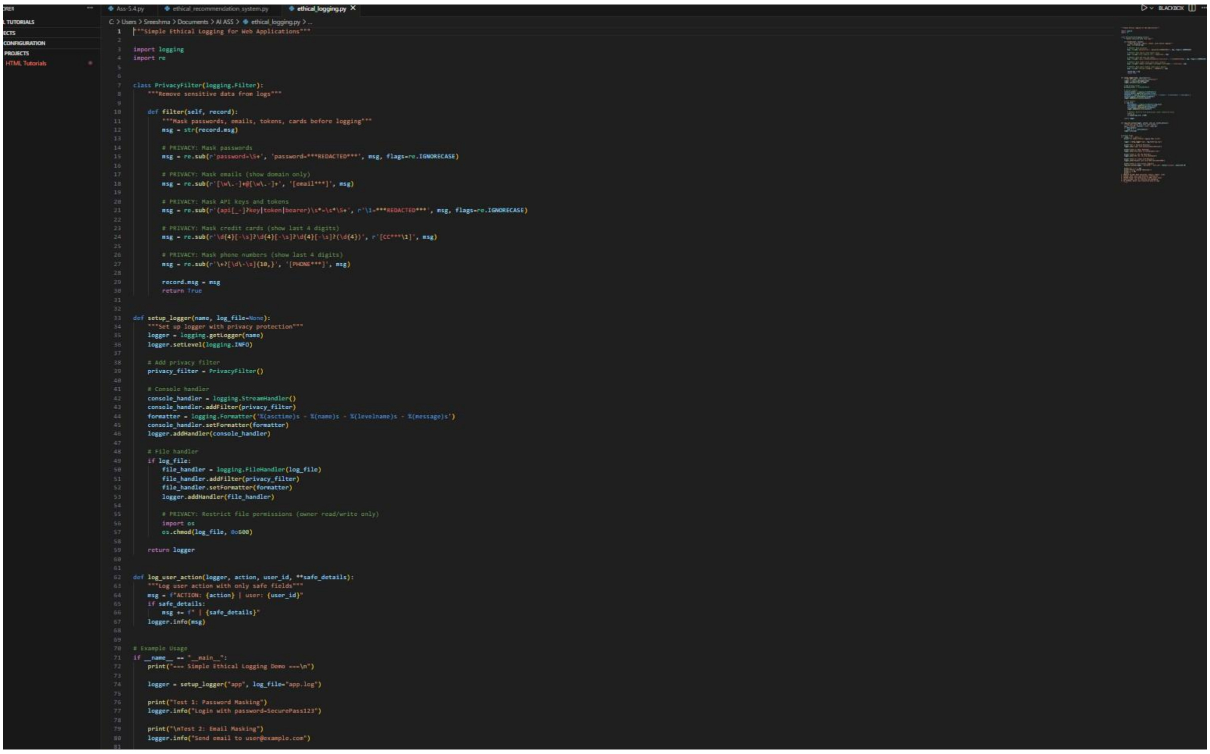
application. Then, ask it to ensure the logs do not record sensitive information.

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

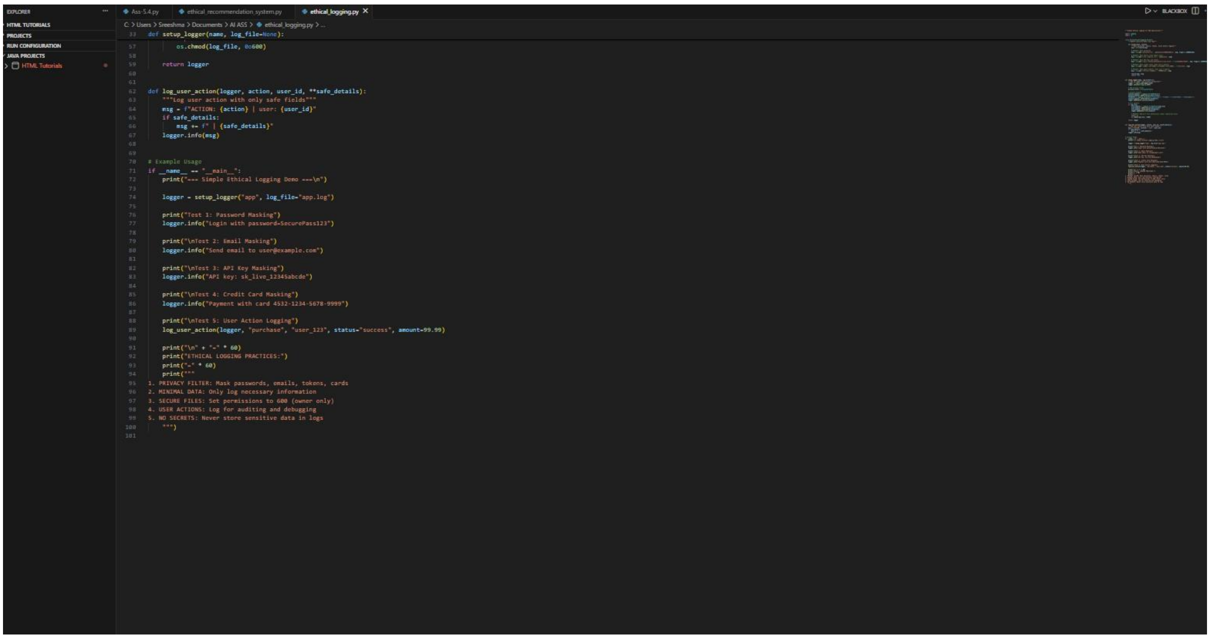
# Ensure logs do NOT store sensitive information such as passwords,

# emails, or personal identifiers.

# Add comments explaining ethical logging practices and privacy protection.


Ass-5.4.py
ethical_recommendation_system.py
ethical_logging.py
C: > Users > Sreeshma > Documents > AI ASS > ethical_logging.py > ...
TUTORIALS
PROJECTS
CONFIGURATION
HTML Tutorials
BLACKBOX

```python
"""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")
```

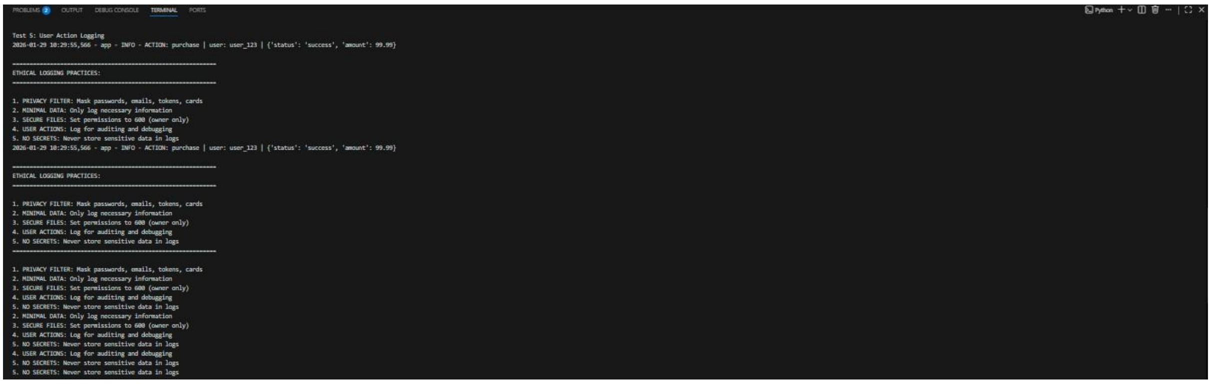


```python
def setup_logger(name, log_file=None):
            os.chmod(log_file, 0o600)

    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")
```

```python

    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.



```
Test 5: User Action Logging
2026-01-29 10:29:55,566 - 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 SECRETS: Never store sensitive data in logs
2026-01-29 10:29:55,566 - 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 SECRETS: 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 SECRETS: Never store sensitive data in logs
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
4. USER ACTIONS: Log for auditing and debugging
5. NO SECRETS: Never store sensitive data in logs
5. NO SECRETS: Never store sensitive data in logs
```

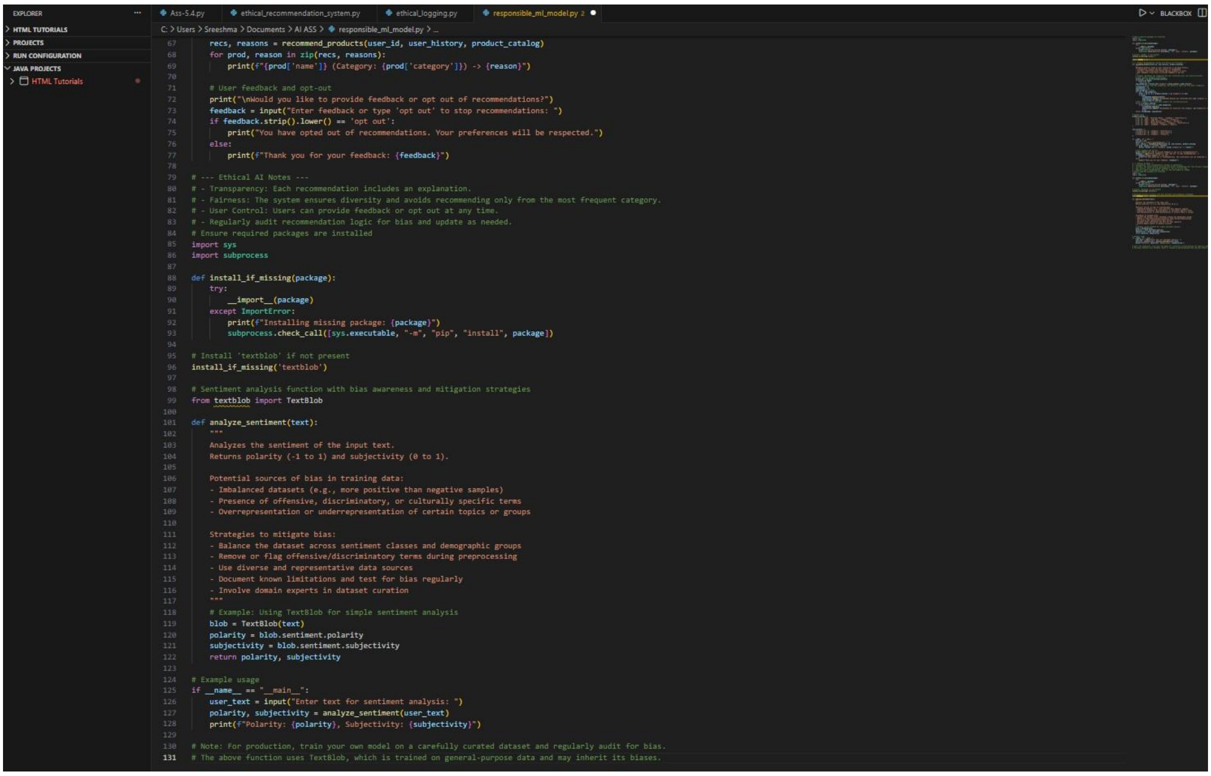
Task Description #5:

- 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).

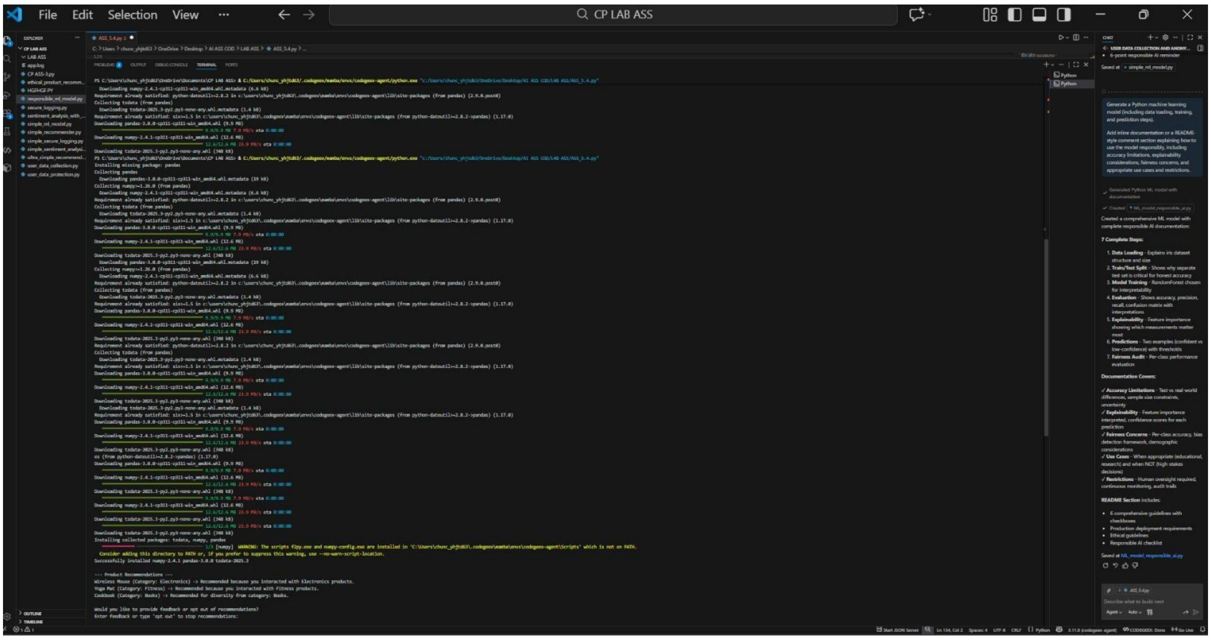
Add inline documentation or a README-style comment section explaining how to use the model responsibly, including accuracy limitations, explainability considerations, fairness concerns, and appropriate use cases and restrictions.



```python
    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.



```
Downloading tzdata-2025.3-py2.py3-none-any.whl (348 kB)
Installing collected packages: tzdata, numpy, pandas
   1/3 [numpy]  WARNING: The scripts f2py.exe and numpy-config.exe are installed in 'C:\Users\chunc_yhjtd63\.codegeex\mamba\envs\codegeex-agent\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed numpy-2.4.1 pandas-3.0.0 tzdata-2025.3

--- Product Recommendations ---
Wireless Mouse (Category: Electronics) -> Recommended because you interacted with Electronics products.
Yoga Mat (Category: Fitness) -> Recommended because you interacted with Fitness products.
Cookbook (Category: Books) -> Recommended for diversity from category: Books.

Would you like to provide feedback or opt out of recommendations?
Enter feedback or type 'opt out' to stop recommendations:
```

- 6-point responsible AI reminder

Saved at simple_ml_model.py

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.

✓ Generated Python ML model with documentation

✓ Created ML_model_responsible_ai.py

Created a comprehensive ML model with complete responsible AI documentation:

7 Complete Steps:

1. **Data Loading** - Explains iris dataset structure and size
2. **Train/Test Split** - Shows why separate test set is critical for honest accuracy
3. **Model Training** - RandomForest chosen for interpretability
4. **Evaluation** - Shows accuracy, precision, recall, confusion matrix with interpretations
5. **Explainability** - Feature importance showing which measurements matter most
6. **Predictions** - Two examples (confident vs low-confidence) with thresholds
7. **Fairness Audit** - Per-class performance evaluation

Documentation Covers:

✓ **Accuracy Limitations** - Test vs real-world differences, sample size constraints, uncertainty
✓ **Explainability** - Feature importance interpreted, confidence scores for each prediction
✓ **Fairness Concerns** - Per-class accuracy, bias detection framework, demographic considerations
✓ **Use Cases** - When appropriate (educational, research) and when NOT (high-stakes decisions)
✓ **Restrictions** - Human oversight required, continuous monitoring, audit trails

README Section includes:

- 6 comprehensive guidelines with checkboxes
- Production deployment requirements
- Ethical guidelines
- Responsible AI checklist

Saved at ML_model_responsible_ai.py