

AI Assisted Coding LAB ASSIGNMENT-5.4

NAME: K.Harish

Batch:13

Hall Ticket:2303A51858

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.

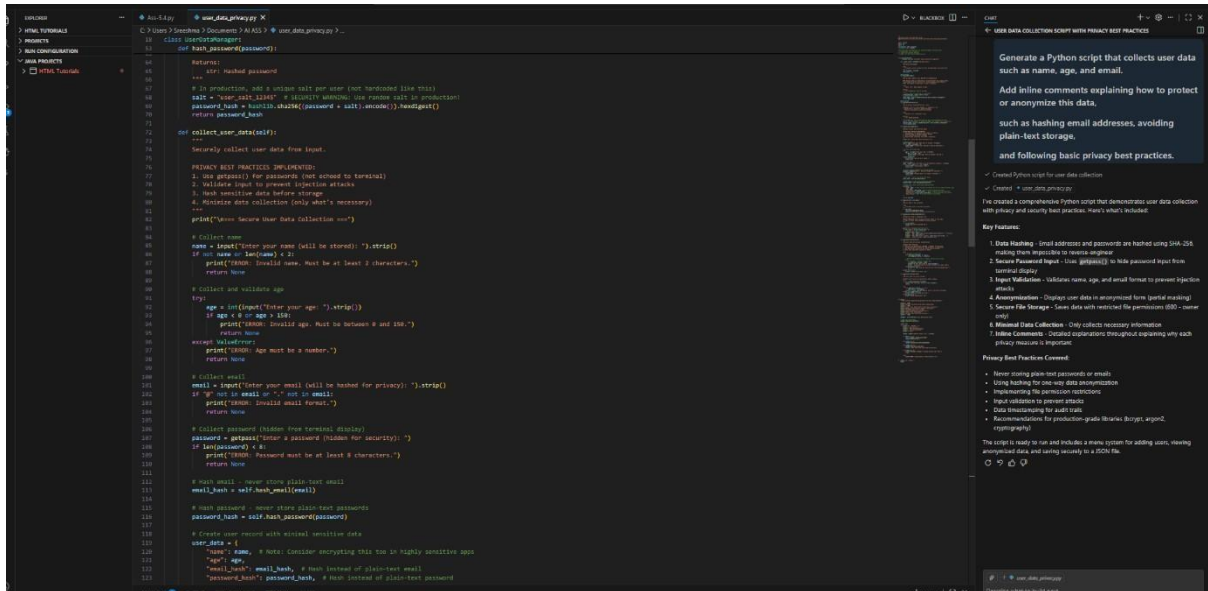
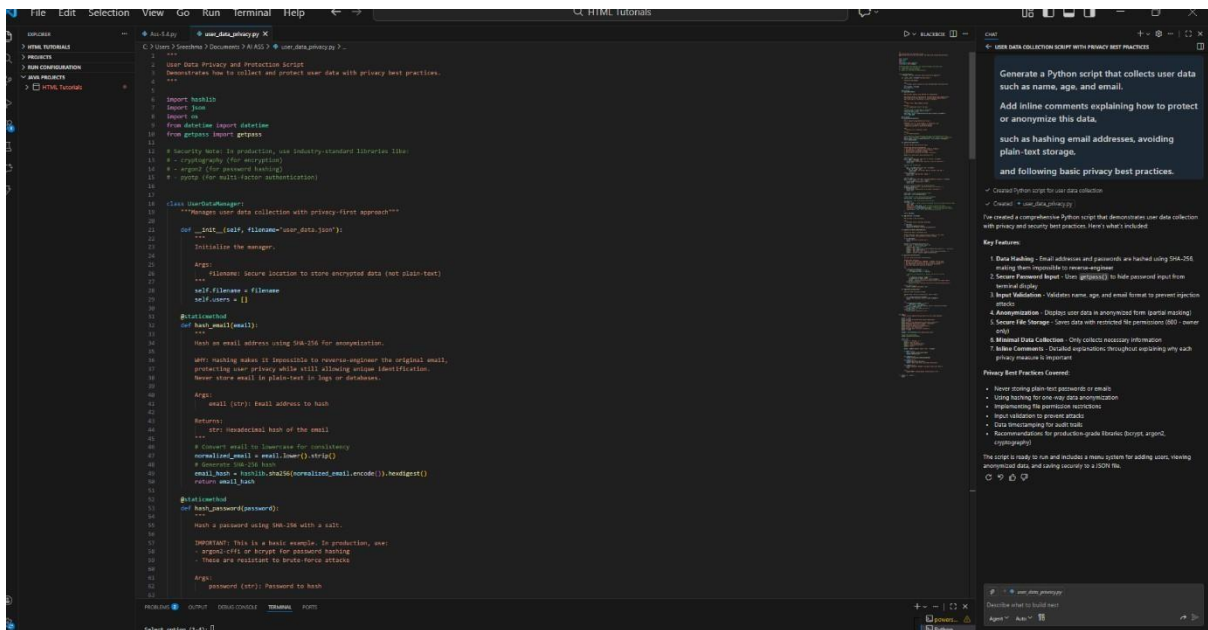
PROMPT

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.

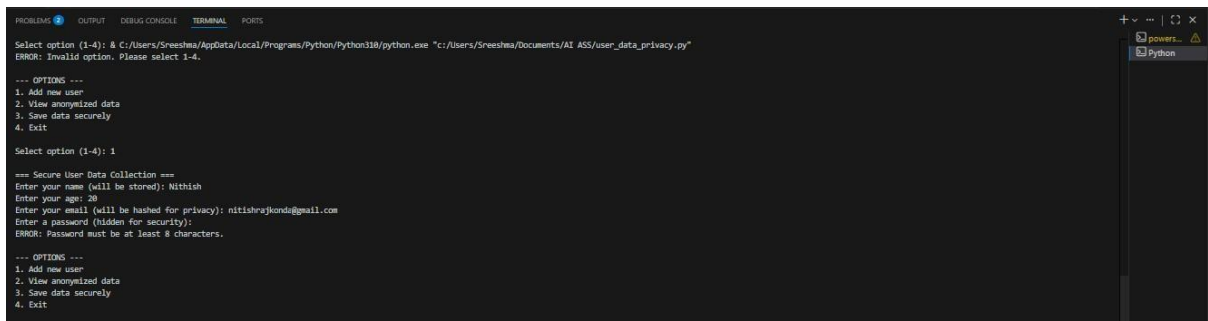


```
100 def collect_user_data(self):
101     """Collect user data from secure storage"""
102     # Create a secure storage instance
103     manager = UserDataManager('user_data_secure.json')
104     # Load existing data
105     manager.load_existing_data()
106     # Add new user
107     print("\nAdd new user")
108     name = input("Enter user name: ")
109     email = input("Enter user email: ")
110     password = input("Enter user password: ")
111     # Hash the password
112     hashed_password = self.hash_password(password)
113     # Create a new user dictionary
114     user_data = {
115         'name': name,
116         'email': email,
117         'password': hashed_password,
118         'created_at': datetime.now().isoformat(),
119     }
120     # Add the user data to the collection
121     manager.add_user_data(user_data)
122     # Display the user data
123     manager.display_user_data_anonymized()
124     # Save the user data with restricted permissions
125     manager.save_user_data_securely()
126     print("\nUser data collected successfully!")
127
128 def display_user_data_anonymized(self):
129     """Display user data in anonymized form"""
130     # Load existing data
131     manager = UserDataManager('user_data_secure.json')
132     manager.load_existing_data()
133     # Display the user data
134     print("\nUser data collected successfully!")
135     # Anonymize the user data
136     manager.anonymize_user_data()
137     # Display the anonymized user data
138     manager.display_user_data_anonymized()
139
140 def hash_password(self, password):
141     """Hash a password using SHA-256"""
142     # Create a SHA-256 hash object
143     sha256_hash = hashlib.sha256()
144     # Update the hash object with the password
145     sha256_hash.update(password.encode('utf-8'))
146     # Get the hexadecimal representation of the hash
147     hashed_password = sha256_hash.hexdigest()
148     return hashed_password
149
150 def main():
151     """Main function demonstrating privacy-first user data handling"""
152     # Create a secure storage instance
153     manager = UserDataManager('user_data_secure.json')
154     # Load existing data
155     manager.load_existing_data()
156     # Add new user
157     print("\nAdd new user")
158     name = input("Enter user name: ")
159     email = input("Enter user email: ")
160     password = input("Enter user password: ")
161     # Hash the password
162     hashed_password = self.hash_password(password)
163     # Create a new user dictionary
164     user_data = {
165         'name': name,
166         'email': email,
167         'password': hashed_password,
168         'created_at': datetime.now().isoformat(),
169     }
170     # Add the user data to the collection
171     manager.add_user_data(user_data)
172     # Display the user data
173     manager.display_user_data_anonymized()
174     # Save the user data with restricted permissions
175     manager.save_user_data_securely()
176     print("\nUser data collected successfully!")
177
178 if __name__ == "__main__":
179     main()
180
```

```
100 def collect_user_data(self):
101     """Collect user data from secure storage"""
102     # Create a secure storage instance
103     manager = UserDataManager('user_data_secure.json')
104     # Load existing data
105     manager.load_existing_data()
106     # Add new user
107     print("\nAdd new user")
108     name = input("Enter user name: ")
109     email = input("Enter user email: ")
110     password = input("Enter user password: ")
111     # Hash the password
112     hashed_password = self.hash_password(password)
113     # Create a new user dictionary
114     user_data = {
115         'name': name,
116         'email': email,
117         'password': hashed_password,
118         'created_at': datetime.now().isoformat(),
119     }
120     # Add the user data to the collection
121     manager.add_user_data(user_data)
122     # Display the user data
123     manager.display_user_data_anonymized()
124     # Save the user data with restricted permissions
125     manager.save_user_data_securely()
126     print("\nUser data collected successfully!")
127
128 def display_user_data_anonymized(self):
129     """Display user data in anonymized form"""
130     # Load existing data
131     manager = UserDataManager('user_data_secure.json')
132     manager.load_existing_data()
133     # Display the user data
134     print("\nUser data collected successfully!")
135     # Anonymize the user data
136     manager.anonymize_user_data()
137     # Display the anonymized user data
138     manager.display_user_data_anonymized()
139
140 def hash_password(self, password):
141     """Hash a password using SHA-256"""
142     # Create a SHA-256 hash object
143     sha256_hash = hashlib.sha256()
144     # Update the hash object with the password
145     sha256_hash.update(password.encode('utf-8'))
146     # Get the hexadecimal representation of the hash
147     hashed_password = sha256_hash.hexdigest()
148     return hashed_password
149
150 def main():
151     """Main function demonstrating privacy-first user data handling"""
152     # Create a secure storage instance
153     manager = UserDataManager('user_data_secure.json')
154     # Load existing data
155     manager.load_existing_data()
156     # Add new user
157     print("\nAdd new user")
158     name = input("Enter user name: ")
159     email = input("Enter user email: ")
160     password = input("Enter user password: ")
161     # Hash the password
162     hashed_password = self.hash_password(password)
163     # Create a new user dictionary
164     user_data = {
165         'name': name,
166         'email': email,
167         'password': hashed_password,
168         'created_at': datetime.now().isoformat(),
169     }
170     # Add the user data to the collection
171     manager.add_user_data(user_data)
172     # Display the user data
173     manager.display_user_data_anonymized()
174     # Save the user data with restricted permissions
175     manager.save_user_data_securely()
176     print("\nUser data collected successfully!")
177
178 if __name__ == "__main__":
179     main()
180
```

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/Sreesha/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Sreesha/Documents/AI_ASS/user_data_privacy.py"
ERROR: Invalid option. Please select 1-4.

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

Select option (1-4): 1

=== Secure User Data Collection ===
Enter your name (will be stored): Nitish
Enter your age: 20
Enter your email (will be hashed for privacy): nitishrajkonde@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.

[illegible]

Expected Output #2:

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Text: It's okay, nothing special.
Result: 0
PS C:\Users\Sreesha\Downloads\VHML Tutorials> & C:\Users\Sreesha\AppData\Local\Programs\Python\Python118\python.exe "c:\Users\Sreesha\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\Sreesha\Downloads\VHML Tutorials> & C:\Users\Sreesha\AppData\Local\Programs\Python\Python118\python.exe "c:\Users\Sreesha\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'}
sreesha\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.

```

```

PS C:\Users\Sreesha> NEW-PF -table c:\pf\1\ORIGINAL -PDIR
new 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\Sreesha\Downloads\HTML_Tutorials>

After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreesha\Downloads\HTML_Tutorials>

After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreesha\Downloads\HTML_Tutorials>

After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreesha\Downloads\HTML_Tutorials>

After: POSITIVE=2, NEGATIVE=2
PS C:\Users\Sreesha\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.

```
1 # ethical_recommendation_system.py
2 """Single Ethical Product Recommendation System"""
3
4 class RecommendationSystem:
5     """Product recommendation with fairness and user control"""
6
7     def __init__(self):
8         self.user_purchases = {} # {user_id: [products]}
9         self.products = {} # {product_id: [category]}
10        self.user_opt_out = set() # Users who opted out
11        self.feedback = {} # {user_id: feedback}
12
13    def add_purchase(self, user_id, product_id, category):
14        """Record a user purchase"""
15        if user_id not in self.user_purchases:
16            self.user_purchases[user_id] = []
17        self.user_purchases[user_id].append(product_id)
18        self.products[product_id] = {'category': category, 'count': 0}
19
20    def recommend(self, user_id, num=5):
21        """
22        TRANSPARENCY: Show why each recommendation is made
23        FAIRNESS: Don't only recommend popular products
24        USER CONTROL: Respect opt-out preferences
25        """
26        # ETHICAL CHECK: Respect user opt-out
27        if user_id in self.user_opt_out:
28            return ("status": "User opted out", "recommendations": [])
29
30        if user_id not in self.user_purchases:
31            return ("status": "New user", "recommendations": [])
32
33        user_history = self.user_purchases[user_id]
34        user_categories = [self.products[p].get('category') for p in user_history if p in self.products]
35
36        # Find candidates
37        candidates = []
38        for prod_id, prod_data in self.products.items():
39            if prod_id not in user_history: # Skip already purchased
40                category = prod_data.get('category')
41
42                # FAIRNESS: Score based on relevance + diversity
43                if category in user_categories:
44                    score = 0.8 # Relevant to user's interests
45                else:
46                    score = 0.6 # Explore new category
47
48                # TRANSPARENCY: Explain why
49                reason = f"Similar to your {category} purchases" if category in user_categories else f"Try new {category}"
50
51                candidates.append({
52                    "product": prod_id,
53                    "score": score,
54                    "reason": reason
55                })
56
57        # Sort by score and return top N
58        top_rec = sorted(candidates, key=lambda x: x['score'], reverse=True)[:num]
59
60        return {
61            "status": "Success",
62            "user_id": user_id,
63            "history": user_history,
64            "recommendations": top_rec
65        }
```

```
class RecommendationSystem:
    def __init__(self, user_id, item_id):
        self.user_id = user_id
        self.item_id = item_id
        self.history = []
        self.recommendations = []
        self.top_rec = []

    def recommend(self, user_id, item_id):
        """Recommend items based on user history and item popularity"""
        return {
            "status": "success",
            "user_id": user_id,
            "history": self.history,
            "recommendations": self.recommendations,
            "top_rec": self.top_rec
        }

    def give_feedback(self, user_id, product_id, liked):
        """Collect user feedback to improve fairness"""
        self.feedback(user_id, product_id, liked)
        return "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"

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

    # Add purchases
    print("=== Adding Purchases ===")
    system.add_purchase("user1", "laptop", "electronics")
    system.add_purchase("user2", "mouse", "electronics")
    system.add_purchase("user3", "book", "books")
    print(f"/ Purchases recorded")

    # Add products
    system.products["keyboard"] = {"category": "electronics"}
    system.products["monitor"] = {"category": "electronics"}
    system.products["book"] = {"category": "books"}

    # Get recommendations
    print("=== Recommendations for user1 ===")
    result = system.recommend("user1", None)
    for rec in result["recommendations"]:
        print(f"Product: {rec['product']}, Score: {rec['score']}, Reason: {rec['reason']}")

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

    # Opt out
    print("=== User Control ===")
    print(system.opt_out("user1"))
    result2 = system.recommend("user1")
    print(f"After opt-out: {result2['recommendations']}")

    # 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
--- Adding Purchases ---
PS C:\Users\freemba\Downloads\VTM_Tutorial> . C:\Users\freemba\Documents\AI_Alt\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\freemba\Downloads\VTM_Tutorial>
```

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.

```
1 #!/usr/bin/env python3
2 """Simple ethical logging for web applications"""
3
4 import logging
5
6 # Remove sensitive data from logs
7
8 class PrivacyFilter(logging.Filter):
9     """Remove sensitive data from logs"""
10
11     def filter(self, record):
12         """Mask passwords, emails, tokens, cards before logging"""
13         msg = str(record.msg)
14
15         # PRIVACY: Mask passwords
16         msg = re.sub(r'password=', 'password=***REDACTED***', msg, flags=re.IGNORECASE)
17
18         # PRIVACY: Mask emails (show domain only)
19         msg = re.sub(r'[a-zA-Z0-9]{1,30}@[a-zA-Z0-9]{2,6}\.com', '[REDACTED]', msg)
20
21         # PRIVACY: Mask API keys and tokens
22         msg = re.sub(r'key=[a-zA-Z0-9]{16,32}', 'key=***REDACTED***', msg, flags=re.IGNORECASE)
23
24         # PRIVACY: Mask credit cards (show last 4 digits)
25         msg = re.sub(r'card=[a-zA-Z0-9]{16,19}', 'card=****[REDACTED]', msg)
26
27         # PRIVACY: Mask phone numbers (show last 4 digits)
28         msg = re.sub(r'phone=[a-zA-Z0-9]{10,15}', 'phone=****[REDACTED]', msg)
29
30         record.msg = msg
31         return True
32
33 def setup_logger(name, log_file=None):
34     """Set up logger with privacy protection"""
35     logger = logging.getLogger(name)
36     logger.setLevel(logging.INFO)
37
38     # Add privacy filter
39     privacy_filter = PrivacyFilter()
40
41     # Console handler
42     console_handler = logging.StreamHandler()
43     console_handler.addFilter(privacy_filter)
44     formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
45     console_handler.setFormatter(formatter)
46     logger.addHandler(console_handler)
47
48     # File handler
49     if log_file:
50         file_handler = logging.FileHandler(log_file)
51         file_handler.addFilter(privacy_filter)
52         file_handler.setFormatter(formatter)
53         logger.addHandler(file_handler)
54
55     # Restrict file permissions (owner read/write only)
56     import os
57     os.chmod(log_file, 0o600)
58
59     return logger
60
61 def log_user_action(logger, action, user_id, **safe_details):
62     """Log user action with only safe fields"""
63     msg = '%(action)s | user: %(user_id)s'
64     if safe_details:
65         msg += ' | ' + ', '.join(f'%(key)s: %(value)s' for key, value in safe_details.items())
66     logger.info(msg)
67
68 # Example usage
69 if __name__ == '__main__':
70     print('--- Simple Ethical Logging Demo ---\n')
71
72     logger = setup_logger('app', log_file='app.log')
73
74     print('Test 1: Password Masking')
75     logger.info('login with password=securepass123')
76
77     print('Test 2: Email Masking')
78     logger.info('send email to user@example.com')
79
80     print('Test 3: API Key Masking')
81     logger.info('API key: sk_live_1234567890')
82
83     print('Test 4: Credit Card Masking')
84     logger.info('payment with card 4111 1111 1111 1111')
85
86     print('Test 5: User Action Logging')
87     log_user_action(logger, 'purchase', 'user_123', status='success', amount=99.99)
88
89     print('\n' + '-' * 50)
90     print('ETHICAL LOGGING PRACTICES:')
91     print('-' * 50)
92
93     # PRIVACY FILTER: Mask passwords, emails, tokens, cards
94     # MINIMAL DATA: Only log necessary information
95     # SECURE FILES: Set permissions to owner (owner only)
96     # USE ACTIONS: Log for auditing and debugging
97     # NO SENSIT: Never store sensitive data in logs
98     print('\n')
```

```
101
102
103 # Example usage
104 if __name__ == '__main__':
105     print('--- Simple Ethical Logging Demo ---\n')
106
107     logger = setup_logger('app', log_file='app.log')
108
109     print('Test 1: Password Masking')
110     logger.info('login with password=securepass123')
111
112     print('Test 2: Email Masking')
113     logger.info('send email to user@example.com')
114
115     print('Test 3: API Key Masking')
116     logger.info('API key: sk_live_1234567890')
117
118     print('Test 4: Credit Card Masking')
119     logger.info('payment with card 4111 1111 1111 1111')
120
121     print('Test 5: User Action Logging')
122     log_user_action(logger, 'purchase', 'user_123', status='success', amount=99.99)
123
124     print('\n' + '-' * 50)
125     print('ETHICAL LOGGING PRACTICES:')
126     print('-' * 50)
127
128     # PRIVACY FILTER: Mask passwords, emails, tokens, cards
129     # MINIMAL DATA: Only log necessary information
130     # SECURE FILES: Set permissions to owner (owner only)
131     # USE ACTIONS: Log for auditing and debugging
132     # NO SENSIT: Never store sensitive data in logs
133     print('\n')
```

Expected Output #4:

- ```
Task 5: User Action Logging
2024-01-29 10:20:16,866 - user | 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 FIELDS: Set permissions to own (owner-only)
4. USER ACTIONS: Log for auditing and debugging
5. NO SECRETS: Never store sensitive data in logs

2024-01-29 10:20:16,866 - user | 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 FIELDS: Set permissions to own (owner-only)
4. USER ACTIONS: Log for auditing and debugging
5. NO SECRETS: Never store sensitive data in logs

2024-01-29 10:20:16,866 - user | 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 FIELDS: Set permissions to own (owner-only)
4. USER ACTIONS: Log for auditing and debugging
5. NO SECRETS: Never store sensitive data in logs
```

- 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).

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.

```
Anki4day # ethical_recommendation_system.py # ethica_logging.py # responsible_ml_model.py :
C:\Users\Sreshan> Document > AIADG > responsible_ml_model.py
PROJECTS
VITAL TUTORIALS
RUN COMPUTATION
JAVA PROJECTS
HTML Tutorials

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

[illegible]