

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.
```

File Edit Selection View Go Run Terminal Help

HTML TUTORIALS

HTML TUTORIALS

Run Configuration

Java Projects

HTML Tutorials

C:\Users\Seethra\Documents\AIASS\user_data_privacy.py

```

1 # User Data Privacy and Protection Script
2 # Demonstrates how to collect and protect user data with privacy best practices.
3 #
4 #
5 import hashlib
6 import json
7 import os
8 from datetime import datetime
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 # - python (for multi-factor authentication)
15
16
17 class UserWithEmailAndPassword:
18     """Manages user data collection with privacy-first approach"""
19
20     def __init__(self, filename='user_data.json'):
21         self.filename = filename
22         self.users = []
23
24     def __enter__(self):
25         """Initializes the manager."""
26
27     def __exit__(self, exc_type, exc_val, exc_tb):
28         """Closes the manager, securely saving data to file."""
29
30         self.filename = self.filename + '_encrypted'
31         self.users = []
32
33     @staticmethod
34     def hash_email(email):
35         """Hashes email address using SHA-256 for anonymization.
36
37         Warning: Hashing makes it impossible to reverse-engineer the original email,
38         protecting user privacy while still allowing unique identification.
39         Never store plain text emails or logins!
40
41         Args:
42             email (str): Email address to hash.
43
44         Returns:
45             str: Hexadecimal hash of the email.
46
47         """
48         normalized_email = email.lower().strip()
49         email_hash = hashlib.sha256(normalized_email.encode()).hexdigest()
50         return email_hash
51
52     @staticmethod
53     def hash_password(password):
54         """Hashes a password using SHA-256 with a salt.
55
56         IMPORTANT: This is a basic example. In production, use:
57         - argon2-cffi or bcrypt for password hashing
58         - PBKDF2 is resistant to brute-force attacks
59
60         Args:
61             password (str): Password to hash.
62
63         Returns:
64             str: Hashed password.
65
66         """
67         salt = os.urandom(16) # SECURITY WARNING: Use random salts in production!
68         password_hash = hashlib.pbkdf2_hmac('sha256', password + salt, encode(salt), 100000)
69         return password_hash
70
71     def collect_user_data(self):
72         """Securely collect user data from input.
73
74         PREVACY BEST PRACTICES IMPLEMENTED:
75         1. use getpass() for passwords (not echoed to terminal)
76         2. use len() to validate user input instead of regex
77         3. Hash sensitive data before storage
78         4. Minimize data collection (only what's necessary)
79
80         print("Welcome to Secure User Data Collection!")
81
82         # Collect name
83         name = input("Enter your name (will be stored): ").strip()
84         if len(name) < 3:
85             print("Error: Invalid name. Must be at least 3 characters.")
86             return None
87
88         # Collect and validate age
89         try:
90             age = int(input("Enter your age: ").strip())
91             if age < 10 or age > 100:
92                 print("Error: Invalid age. Must be between 10 and 100.")
93                 return None
94
95             except ValueError:
96                 print("Error: Age must be a number.")
97                 return None
98
99         # Collect email
100         email = input("Enter your email (will be hashed for privacy): ").strip()
101         if '@' not in email:
102             print("Error: Invalid email format.")
103             return None
104
105         # Collect password (hidden from terminal display)
106         password = getpass("Enter a password (hidden for security): ")
107         if len(password) < 8:
108             print("Error: Password must be at least 8 characters.")
109             return None
110
111         # Hash email - never store plain text email
112         email_hash = self.hash_email(email)
113
114         # Hash password - never store plain text passwords
115         password_hash = self.hash_password(password)
116
117         # Create user record with minimal sensitive data
118         user_data = {
119             'name': name,
120             'age': age,
121             'email': email,
122             'password_hash': password_hash
123         }
124
125         # Write user data to JSON file
126         with open(self.filename, 'w') as f:
127             json.dump(user_data, f)
128
129         print(f"User data successfully saved to {self.filename}!")
130
131     def __str__(self):
132         """Returns a string representation of the user data collection object.
133
134         Returns:
135             str: A summary of the user data collected.
136
137         """
138         return f"User data collection object containing {len(self.users)} users."
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL FOCUS

Select option (1-4): []

File Edit Selection View Go Run Terminal Help

HTML TUTORIALS

PROJECTS

Run Configuration

Java Projects

HTML Tutorials

C:\Users\Seethra\Documents\AIASS\user_data_privacy.py

```

1 # User Data Privacy and Protection Script
2 # Demonstrates how to collect and protect user data with privacy best practices.
3 #
4 #
5 import hashlib
6 import json
7 import os
8 from datetime import datetime
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 # - python (for multi-factor authentication)
15
16
17 class UserWithEmailAndPassword:
18     """Manages user data collection with privacy-first approach"""
19
20     def __init__(self, password):
21         self.password = password
22
23     def __enter__(self):
24         """Initializes the manager, prompting for password input."""
25
26     def __exit__(self, exc_type, exc_val, exc_tb):
27         """Closes the manager, securely saving data to file."""
28
29         self.collect_user_data()
30
31     @staticmethod
32     def hash_email(email):
33         """Hashes email address using SHA-256 for anonymization.
34
35         Warning: Hashing makes it impossible to reverse-engineer the original email,
36         protecting user privacy while still allowing unique identification.
37         Never store plain text emails or logins!
38
39         Args:
40             email (str): Email address to hash.
41
42         Returns:
43             str: Hexadecimal hash of the email.
44
45         """
46         normalized_email = email.lower().strip()
47         email_hash = hashlib.sha256(normalized_email.encode()).hexdigest()
48         return email_hash
49
50     @staticmethod
51     def hash_password(password):
52         """Hashes a password using SHA-256 with a salt.
53
54         IMPORTANT: This is a basic example. In production, use:
55         - argon2-cffi or bcrypt for password hashing
56         - PBKDF2 is resistant to brute-force attacks
57
58         Args:
59             password (str): Password to hash.
60
61         Returns:
62             str: Hashed password.
63
64         """
65         salt = os.urandom(16) # SECURITY WARNING: Use random salts in production!
66         password_hash = hashlib.pbkdf2_hmac('sha256', password + salt, encode(salt), 100000)
67         return password_hash
68
69     def collect_user_data(self):
70         """Securely collect user data from input.
71
72         PREVACY BEST PRACTICES IMPLEMENTED:
73         1. use getpass() for passwords (not echoed to terminal)
74         2. use len() to validate user input instead of regex
75         3. Hash sensitive data before storage
76         4. Minimize data collection (only what's necessary)
77
78         print("Welcome to Secure User Data Collection!")
79
80         # Collect name
81         name = input("Enter your name (will be stored): ").strip()
82         if len(name) < 3:
83             print("Error: Invalid name. Must be at least 3 characters.")
84             return None
85
86         # Collect and validate age
87         try:
88             age = int(input("Enter your age: ").strip())
89             if age < 10 or age > 100:
90                 print("Error: Invalid age. Must be between 10 and 100.")
91                 return None
92
93             except ValueError:
94                 print("Error: Age must be a number.")
95                 return None
96
97         # Collect email
98         email = input("Enter your email (will be hashed for privacy): ").strip()
99         if '@' not in email:
100             print("Error: Invalid email format.")
101             return None
102
103         # Collect password (hidden from terminal display)
104         password = getpass("Enter a password (hidden for security): ")
105         if len(password) < 8:
106             print("Error: Password must be at least 8 characters.")
107             return None
108
109         # Hash email - never store plain text email
110         email_hash = self.hash_email(email)
111
112         # Hash password - never store plain text passwords
113         password_hash = self.hash_password(password)
114
115         # Create user record with minimal sensitive data
116         user_data = {
117             'name': name,
118             'age': age,
119             'email': email,
120             'password_hash': password_hash
121         }
122
123         # Write user data to JSON file
124         with open(self.filename, 'w') as f:
125             json.dump(user_data, f)
126
127         print(f"User data successfully saved to {self.filename}!")
128
129     def __str__(self):
130         """Returns a string representation of the user data collection object.
131
132         Returns:
133             str: A summary of the user data collected.
134
135         """
136         return f"User data collection object containing {len(self.users)} users."
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL FOCUS

Select option (1-4): []

```

user_data_collector.py
1 #!/usr/bin/python
2
3 import json
4 import hashlib
5 import argparse
6
7 class UserDatabaseManager:
8     def __init__(self):
9         self.users = []
10
11     def collect_user_data(self):
12         hashed_passwords = []
13         for user in self.users:
14             user['password'] = hashlib.sha256(user['password']).hexdigest()
15             user['data_version'] = '1.0' # For data structure versioning
16
17         return self.users
18
19     def add_user(self, user_data):
20         self.users.append(user_data)
21
22     def display_user_data_anonymized(self):
23         user_data = {}
24
25         for user in self.users:
26             user_data[user['name']] = {
27                 'age': user['age'],
28                 'email': user['email'].replace('@', '.').lower(),
29                 'password': user['password'],
30                 'data_version': user['data_version']
31             }
32
33         return user_data
34
35     def save_users(self, filename='users.json'):
36
37         users_to_save = []
38
39         for user in self.users:
40             user['password'] = hashlib.sha256(user['password']).hexdigest()
41             users_to_save.append(user)
42
43         with open(filename, 'w') as f:
44             json.dump(users_to_save, f)
45
46     def load_user_data(self):
47         try:
48             with open('users.json') as f:
49                 users = json.load(f)
50
51             self.users = []
52             for user in users:
53                 self.add_user(user)
54
55         except FileNotFoundError:
56             print("No user data found")
57
58
59 if __name__ == '__main__':
60     parser = argparse.ArgumentParser()
61     parser.add_argument('--action', choices=['collect', 'add', 'display', 'save'])
62     parser.add_argument('--user', type=dict, required=True)
63
64     args = parser.parse_args()
65
66     manager = UserDatabaseManager()
67
68     if args.action == 'collect':
69         manager.collect_user_data()
70
71     elif args.action == 'add':
72         manager.add_user(args.user)
73
74     elif args.action == 'display':
75         manager.display_user_data_anonymized()
76
77     elif args.action == 'save':
78         manager.save_users()
79
80     else:
81         print("Unknown action. Please select 1-4.")
82
83
84 if __name__ == "__main__":
85     main()

```



```

user_data_processor.py
1 #!/usr/bin/python
2
3 import json
4 import hashlib
5 import argparse
6
7 class UserDatabaseManager:
8     def __init__(self):
9         self.users = []
10
11     def collect_user_data(self):
12         hashed_passwords = []
13         for user in self.users:
14             user['password'] = hashlib.sha256(user['password']).hexdigest()
15             user['data_version'] = '1.0' # For data structure versioning
16
17         return self.users
18
19     def add_user(self, user_data):
20         self.users.append(user_data)
21
22     def display_user_data_anonymized(self):
23         user_data = {}
24
25         for user in self.users:
26             user_data[user['name']] = {
27                 'age': user['age'],
28                 'email': user['email'].replace('@', '.').lower(),
29                 'password': user['password'],
30                 'data_version': user['data_version']
31             }
32
33         return user_data
34
35     def save_users(self, filename='users.json'):
36
37         users_to_save = []
38
39         for user in self.users:
40             user['password'] = hashlib.sha256(user['password']).hexdigest()
41             users_to_save.append(user)
42
43         with open(filename, 'w') as f:
44             json.dump(users_to_save, f)
45
46     def load_user_data(self):
47         try:
48             with open('users.json') as f:
49                 users = json.load(f)
50
51             self.users = []
52             for user in users:
53                 self.add_user(user)
54
55         except FileNotFoundError:
56             print("No user data found")
57
58
59 if __name__ == '__main__':
60     parser = argparse.ArgumentParser()
61     parser.add_argument('--action', choices=['collect', 'add', 'display', 'save'])
62     parser.add_argument('--user', type=dict, required=True)
63
64     args = parser.parse_args()
65
66     manager = UserDatabaseManager()
67
68     if args.action == 'collect':
69         manager.collect_user_data()
70
71     elif args.action == 'add':
72         manager.add_user(args.user)
73
74     elif args.action == 'display':
75         manager.display_user_data_anonymized()
76
77     elif args.action == 'save':
78         manager.save_users()
79
80     else:
81         print("Unknown action. Please select 1-4.")
82
83
84 if __name__ == "__main__":
85     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): Nitish
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.

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 Python + - x

*** 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>
```

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

```

COPilot -- AI Assistant for ethical recommendation system.py
C:\Users\Shreya\Documents>AI Assistant for ethical recommendation system.py
...
20     def recommend(self, user_id, num=3):
21         user_rec = self.get_recommendations(user_id)
22         return {
23             "status": "success",
24             "user_id": user_id,
25             "history": user.history,
26             "recommendations": top_rec
27         }
28
29     def give_feedback(self, user_id, product_id, liked):
30         user_rec = self.get_recommendations(user_id)
31         self.feedback["user_id"] [product_id] = liked
32         return f"Thanks for feedback on {product_id}"
33
34     def opt_out(self, user_id):
35         """Let user opt out of recommendations"""
36         self.user_opt_out.append(user_id)
37         return f"(user_id) opted out of recommendations"
38
39     def opt_in(self, user_id):
40         """Let user opt back in"""
41         self.user_opt_out.remove(user_id)
42         return f"(user_id) opted in to recommendations"
43
44     # Example Usage
45     if __name__ == "__main__":
46         system = RecommendationSystem()
47
48         # Add purchases
49         print("Adding Purchases ---")
50         system.add_purchase("user1", "laptop", "Electronics(*")
51         system.add_purchase("user1", "monitor", "Electronics(*)")
52         system.add_purchase("user1", "book", "Books")
53         print("Purchases recorded")
54
55         # Add products
56         system.add_product("monitor", {"category": "Electronics"})
57         system.products["monitor"] = {"category": "Electronics"}
58         system.add_product("book", {"category": "Books"})
59
60         # Get recommendations
61         print("Getting recommendations for user1 ---")
62         result1 = system.recommend("user1", num=2)
63         for rec in result1["recommendation"]:
64             print(f"Product: {rec['product']}, Score: {rec['score']}, Reason: {rec['reason']} ")
65
66         # User Feedback
67         user_rec = self.get_recommendations("user1")
68         print(system.user_feedback("user1", "keyboard", True))
69
70         # Opt Out
71         print("User Opt Out ---")
72         print(system.opt_out("user1"))
73         result2 = system.recommend("user1")
74         print(f"After opt-out: {result2['status']} ")
75
76         # Opt In
77         print(system.opt_in("user1"))
78
79

```

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
/ Purchases recorded
-- Adding Purchases ---
/ Purchases recorded
-- Recommendations for user1 ---
Product: keyboard, Score: 0.8, Reason: Stellar 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
user1 opted in to recommendations
user1 opted in to recommendations
-- Purchases recorded

```

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.

The image shows a code editor with three tabs open, all titled "ethical_logging.py". The tabs are located in the top left corner of the interface.

Top Tab Content:

```
#!/usr/bin/python
# ethical recommendation system
# ethical_logging.py

C:\Users\5-Seneca\Documents>cd ASIS > ethical_logging.py

#<<<Simple ethical logging for web Application>>>

1 import logging
2 import re
3
4
5 class PrivacyFilter(logging.Filter):
6     """Remove sensitive data from logs"""
7
8     def filter(self, record):
9         """Mask passwords, emails, tokens, cards before logging"""
10        msg = record.getMessage()
11
12        # REMOVE passwords
13        msg = re.sub(r'password|User', 'password***REDACTED***', msg, flags=re.IGNORECASE)
14
15        # REMOVE email emails
16        msg = re.sub(r'([Ww].+)[@][Ww].+', '[email***]', msg)
17
18        # REMOVE most API keys and tokens
19        msg = re.sub(r'(api_)?[Aa][Bb][Cc][Dd][Ee][Ff][Gg][Hh][Ii][Jj][Kk][Ll][Mm][Nn][Oo][Pp][Qq][Rr][Ss][Tt][Uu][Vv][Ww][Xx][Yy][Zz]([0-9]{1,4})', '***REDACTED***', msg, flags=re.IGNORECASE)
20
21        # REMOVE Mask credit cards (show last 4 digits)
22        msg = re.sub(r'([4-9][0-9]{12}[0-9]{3})', '([C***]1)', msg)
23
24        # REMOVE Mask phone numbers (show last 4 digits)
25        msg = re.sub(r'([+][1-9][0-9]{12}[0-9]{3})', '(PHONE***)', msg)
26
27        record.msg = msg
28
29    return True
30
31
32 def setup_logger(name, log_file_name):
33     """Set up logger with writer protection"""
34     logger = logging.getLogger(name)
35     logger.setLevel(logging.INFO)
36
37     # Add privacy filter
38     privacy_filter = PrivacyFilter()
39
40     # Create handler
41     console_handler = logging.StreamHandler()
42     console_handler.addFilter(privacy_filter)
43     formatter = logging.Formatter('%(asctime)s : %(name)s : %(levelname)s : %(message)s')
44     console_handler.setFormatter(formatter)
45     logger.addHandler(console_handler)
46
47     # File Handler
48     if log_file:
49         file_handler = logging.FileHandler(log_file)
50         file_handler.addFilter(privacy_filter)
51         file_handler.setFormatter(formatter)
52         logger.addHandler(file_handler)
53
54     # Finally, restrict file permissions (owner read/write only)
55     if log_file:
56         os.chmod(log_file, 0600)
57
58    return logger
59
60
61 def log_user_action(logger, action, user_id, **safe_details):
62     """Log user action with only safe fields"""
63     msg = f'{action} | {user_id}'
64     if safe_details:
65         msg += f' | {safe_details}'
66
67     logger.info(msg)
68
69
70 # Example Usage
71 if __name__ == "__main__":
72     print("... Simple ethical logging demo ...")
73
74     logger = setup_logger("app", log_file="app.log")
75
76     print("Test 1: Password Masking")
77     logger.info("Login with password=SecurePass123")
78
79     print("Test 2: Email Masking")
80     logger.info("Send email to user@example.com")
81
82
83 def log_user_action(logger, action, user_id, **safe_details):
84     """Log user action with only safe fields"""
85     msg = f'{action} | {user_id}'
86     if safe_details:
87         msg += f' | {safe_details}'
88     logger.info(msg)
89
90
91 # Example Usage
92 if __name__ == "__main__":
93     print("... Simple ethical Logging demo ...\\n")
94
95     logger = setup_logger("app", log_file="app.log")
96
97     print("Test 1: Password Masking")
98     logger.info("Login with password=SecurePass123")
99
100    print("Test 2: Email Masking")
101    logger.info("Send email to user@example.com")
102
103    print("Test 3: API Key Masking")
104    logger.info("API key st_live_3243434343434343")
105
106    print("Test 4: Credit Card Masking")
107    logger.info("Payment with card 4532-1234-5678-9999")
108
109    print("Test 5: User Action Logging")
110    log_user_action(logger, "purchase", "user_123", status="success", amount=99.99)
111
112    print("...n" + " " * 80)
113    print("PRACTICE: Only log necessary information")
114    print("PRACTICE: Set permissions to 'info' (audit only)")
115    print("PRACTICE: Actions should validate and sanitize")
116    print("PRACTICE: Never store sensitive data in logs")
117
118
```

Middle Tab Content:

```
#!/usr/bin/python
# ethical recommendation system
# ethical_logging.py

C:\Users\5-Seneca\Documents>cd ASIS > ethical_logging.py

#<<<Simple ethical logging for web Application>>>

1 import logging
2 import re
3
4
5 class PrivacyFilter(logging.Filter):
6     """Remove sensitive data from logs"""
7
8     def filter(self, record):
9         """Mask passwords, emails, tokens, cards before logging"""
10        msg = record.getMessage()
11
12        # REMOVE passwords
13        msg = re.sub(r'password|User', 'password***REDACTED***', msg, flags=re.IGNORECASE)
14
15        # REMOVE email emails
16        msg = re.sub(r'([Ww].+)[@][Ww].+', '[email***]', msg)
17
18        # REMOVE most API keys and tokens
19        msg = re.sub(r'(api_)?[Aa][Bb][Cc][Dd][Ee][Ff][Gg][Hh][Ii][Jj][Kk][Ll][Mm][Nn][Oo][Pp][Qq][Rr][Ss][Tt][Uu][Vv][Ww][Xx][Yy][Zz]([0-9]{1,4})', '***REDACTED***', msg, flags=re.IGNORECASE)
20
21        # REMOVE Mask credit cards (show last 4 digits)
22        msg = re.sub(r'([4-9][0-9]{12}[0-9]{3})', '([C***]1)', msg)
23
24        # REMOVE Mask phone numbers (show last 4 digits)
25        msg = re.sub(r'([+][1-9][0-9]{12}[0-9]{3})', '(PHONE***)', msg)
26
27        record.msg = msg
28
29    return True
30
31
32 def setup_logger(name, log_file_name):
33     """Set up logger with writer protection"""
34     logger = logging.getLogger(name)
35     logger.setLevel(logging.INFO)
36
37     # Add privacy filter
38     privacy_filter = PrivacyFilter()
39
40     # Create handler
41     console_handler = logging.StreamHandler()
42     console_handler.addFilter(privacy_filter)
43     formatter = logging.Formatter('%(asctime)s : %(name)s : %(levelname)s : %(message)s')
44     console_handler.setFormatter(formatter)
45     logger.addHandler(console_handler)
46
47     # File Handler
48     if log_file:
49         file_handler = logging.FileHandler(log_file)
50         file_handler.addFilter(privacy_filter)
51         file_handler.setFormatter(formatter)
52         logger.addHandler(file_handler)
53
54     # Finally, restrict file permissions (owner read/write only)
55     if log_file:
56         os.chmod(log_file, 0600)
57
58    return logger
59
60
61 def log_user_action(logger, action, user_id, **safe_details):
62     """Log user action with only safe fields"""
63     msg = f'{action} | {user_id}'
64     if safe_details:
65         msg += f' | {safe_details}'
66
67     logger.info(msg)
68
69
70 # Example Usage
71 if __name__ == "__main__":
72     print("... Simple ethical logging demo ...")
73
74     logger = setup_logger("app", log_file="app.log")
75
76     print("Test 1: Password Masking")
77     logger.info("Login with password=SecurePass123")
78
79     print("Test 2: Email Masking")
80     logger.info("Send email to user@example.com")
81
82
83 def log_user_action(logger, action, user_id, **safe_details):
84     """Log user action with only safe fields"""
85     msg = f'{action} | {user_id}'
86     if safe_details:
87         msg += f' | {safe_details}'
88     logger.info(msg)
89
90
91 # Example Usage
92 if __name__ == "__main__":
93     print("... Simple ethical Logging demo ...\\n")
94
95     logger = setup_logger("app", log_file="app.log")
96
97     print("Test 1: Password Masking")
98     logger.info("Login with password=SecurePass123")
99
100    print("Test 2: Email Masking")
101    logger.info("Send email to user@example.com")
102
103    print("Test 3: API Key Masking")
104    logger.info("API key st_live_3243434343434343")
105
106    print("Test 4: Credit Card Masking")
107    logger.info("Payment with card 4532-1234-5678-9999")
108
109    print("Test 5: User Action Logging")
110    log_user_action(logger, "purchase", "user_123", status="success", amount=99.99)
111
112    print("...n" + " " * 80)
113    print("PRACTICE: Only log necessary information")
114    print("PRACTICE: Set permissions to 'info' (audit only)")
115    print("PRACTICE: Actions should validate and sanitize")
116    print("PRACTICE: Never store sensitive data in logs")
117
118
```

Bottom Tab Content:

```
#!/usr/bin/python
# ethical recommendation system
# ethical_logging.py

C:\Users\5-Seneca\Documents>cd ASIS > ethical_logging.py

#<<<Simple ethical logging for web Application>>>

1 import logging
2 import re
3
4
5 class PrivacyFilter(logging.Filter):
6     """Remove sensitive data from logs"""
7
8     def filter(self, record):
9         """Mask passwords, emails, tokens, cards before logging"""
10        msg = record.getMessage()
11
12        # REMOVE passwords
13        msg = re.sub(r'password|User', 'password***REDACTED***', msg, flags=re.IGNORECASE)
14
15        # REMOVE email emails
16        msg = re.sub(r'([Ww].+)[@][Ww].+', '[email***]', msg)
17
18        # REMOVE most API keys and tokens
19        msg = re.sub(r'(api_)?[Aa][Bb][Cc][Dd][Ee][Ff][Gg][Hh][Ii][Jj][Kk][Ll][Mm][Nn][Oo][Pp][Qq][Rr][Ss][Tt][Uu][Vv][Ww][Xx][Yy][Zz]([0-9]{1,4})', '***REDACTED***', msg, flags=re.IGNORECASE)
20
21        # REMOVE Mask credit cards (show last 4 digits)
22        msg = re.sub(r'([4-9][0-9]{12}[0-9]{3})', '([C***]1)', msg)
23
24        # REMOVE Mask phone numbers (show last 4 digits)
25        msg = re.sub(r'([+][1-9][0-9]{12}[0-9]{3})', '(PHONE***)', msg)
26
27        record.msg = msg
28
29    return True
30
31
32 def setup_logger(name, log_file_name):
33     """Set up logger with writer protection"""
34     logger = logging.getLogger(name)
35     logger.setLevel(logging.INFO)
36
37     # Add privacy filter
38     privacy_filter = PrivacyFilter()
39
40     # Create handler
41     console_handler = logging.StreamHandler()
42     console_handler.addFilter(privacy_filter)
43     formatter = logging.Formatter('%(asctime)s : %(name)s : %(levelname)s : %(message)s')
44     console_handler.setFormatter(formatter)
45     logger.addHandler(console_handler)
46
47     # File Handler
48     if log_file:
49         file_handler = logging.FileHandler(log_file)
50         file_handler.addFilter(privacy_filter)
51         file_handler.setFormatter(formatter)
52         logger.addHandler(file_handler)
53
54     # Finally, restrict file permissions (owner read/write only)
55     if log_file:
56         os.chmod(log_file, 0600)
57
58    return logger
59
60
61 def log_user_action(logger, action, user_id, **safe_details):
62     """Log user action with only safe fields"""
63     msg = f'{action} | {user_id}'
64     if safe_details:
65         msg += f' | {safe_details}'
66
67     logger.info(msg)
68
69
70 # Example Usage
71 if __name__ == "__main__":
72     print("... Simple ethical logging demo ...")
73
74     logger = setup_logger("app", log_file="app.log")
75
76     print("Test 1: Password Masking")
77     logger.info("Login with password=SecurePass123")
78
79     print("Test 2: Email Masking")
80     logger.info("Send email to user@example.com")
81
82
83 def log_user_action(logger, action, user_id, **safe_details):
84     """Log user action with only safe fields"""
85     msg = f'{action} | {user_id}'
86     if safe_details:
87         msg += f' | {safe_details}'
88     logger.info(msg)
89
90
91 # Example Usage
92 if __name__ == "__main__":
93     print("... Simple ethical Logging demo ...\\n")
94
95     logger = setup_logger("app", log_file="app.log")
96
97     print("Test 1: Password Masking")
98     logger.info("Login with password=SecurePass123")
99
100    print("Test 2: Email Masking")
101    logger.info("Send email to user@example.com")
102
103    print("Test 3: API Key Masking")
104    logger.info("API key st_live_3243434343434343")
105
106    print("Test 4: Credit Card Masking")
107    logger.info("Payment with card 4532-1234-5678-9999")
108
109    print("Test 5: User Action Logging")
110    log_user_action(logger, "purchase", "user_123", status="success", amount=99.99)
111
112    print("...n" + " " * 80)
113    print("PRACTICE: Only log necessary information")
114    print("PRACTICE: Set permissions to 'info' (audit only)")
115    print("PRACTICE: Actions should validate and sanitize")
116    print("PRACTICE: Never store sensitive data in logs")
117
118
```

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.

```
EXPLORER
> HTML TUTORIALS
> PROJECTS
> RUN CONFIGURATION
> JAVA PROJECTS
> > HTML Tutorials

As-454.py ethical_recommendation_system.py ethical_logging.py responsible_ml_model.py

C:\> Users \Grechma > Documents > AI ASS > responsible_ml_model.py >

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("Thank you for opting out of recommendations. Your preferences will be respected.")
76     else:
77         print("Thank you for your feedback: {feedback}")
78
79 # --- Ethics & 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     """Analyze the sentiment of the input text.
103     Returns polarity (-1 to 1) and subjectivity (0 to 1).
104     """
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.

