

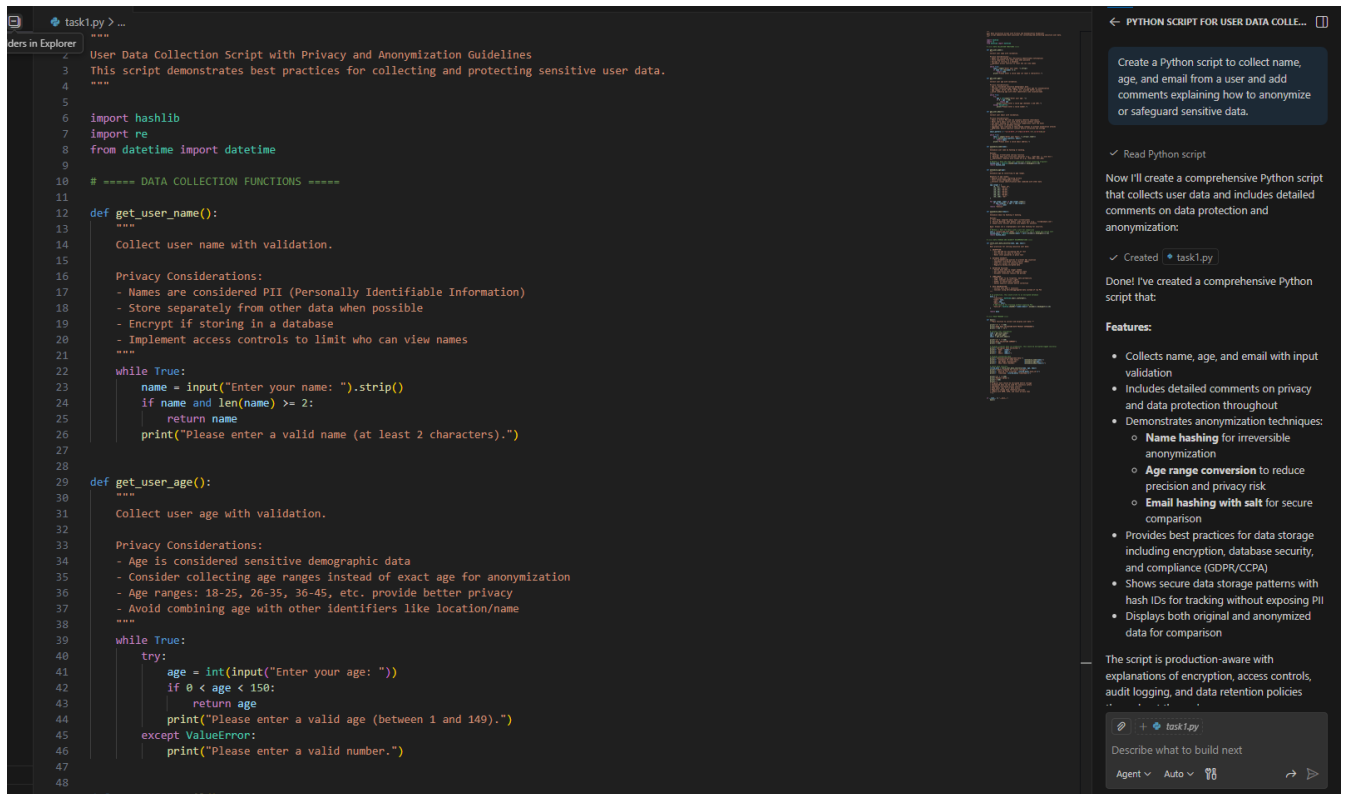
Assignment: 5.4

Name: Ch.Adarsh

Hno: 2303A51044

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.



```
task1.py > ...
User Data Collection Script with Privacy and Anonymization Guidelines
This script demonstrates best practices for collecting and protecting sensitive user data.
import hashlib
import re
from datetime import datetime

# ===== DATA COLLECTION FUNCTIONS =====

def get_user_name():
    """
    Collect user name with validation.

    Privacy Considerations:
    - Names are considered PII (Personally Identifiable Information)
    - Store separately from other data when possible
    - Encrypt if storing in a database
    - Implement access controls to limit who can view names
    """
    while True:
        name = input("Enter your name: ").strip()
        if name and len(name) >= 2:
            return name
        print("Please enter a valid name (at least 2 characters).")

def get_user_age():
    """
    Collect user age with validation.

    Privacy Considerations:
    - Age is considered sensitive demographic data
    - Consider collecting age ranges instead of exact age for anonymization
    - Age ranges: 18-25, 26-35, 36-45, etc. provide better privacy
    - Avoid combining age with other identifiers like location/name
    """
    while True:
        try:
            age = int(input("Enter your age: "))
            if 0 < age < 150:
                return age
            print("Please enter a valid age (between 1 and 149).")
        except ValueError:
            print("Please enter a valid number.")

def get_user_email():
    """
    Collect user email with validation.

    Privacy Considerations:
    - Email addresses are PII
    - Use hashed email for tracking without exposing PII
    - Implement access controls
    """
    while True:
        email = input("Enter your email: ").strip()
        if re.match(r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}", email):
            return email
        print("Please enter a valid email address.")
```

PYTHON SCRIPT FOR USER DATA COLLE...

Create a Python script to collect name, age, and email from a user and add comments explaining how to anonymize or safeguard sensitive data.

✓ Read Python script

Now I'll create a comprehensive Python script that collects user data and includes detailed comments on data protection and anonymization:

✓ Created [task1.py](#)

Done! I've created a comprehensive Python script that:

Features:

- Collects name, age, and email with input validation
- Includes detailed comments on privacy and data protection throughout
- Demonstrates anonymization techniques:
 - **Name hashing** for irreversible anonymization
 - **Age range conversion** to reduce precision and privacy risk
 - **Email hashing with salt** for secure comparison
- Provides best practices for data storage including encryption, database security, and compliance (GDPR/CCPA)
- Shows secure data storage patterns with hash IDs for tracking without exposing PII
- Displays both original and anonymized data for comparison

The script is production-aware with explanations of encryption, access controls, audit logging, and data retention policies

task1.py

Describe what to build next

Agent Auto

```
=====
USER DATA COLLECTION WITH PRIVACY SAFEGUARDS
=====
```

```
Enter your name: Yashwanth
Enter your age: 23
Enter your email: pateruyashwanth6671@gmail.com
```

```
=====
DATA COLLECTION SUMMARY
=====
```

```
Original Data Collected:
```

```
  Name: Yashwanth
  Age: 23
  Email: pateruyashwanth6671@gmail.com
```

```
Anonymized/Safeguarded Data:
```

```
  Anonymous ID (name hash): f49876d80754ccfc
  Age Range (anonymized): 18-25
  Email Hash (hashed): 72503dcb95c46f52
```

```
Data prepared for secure storage:
```

```
  Hash ID (for tracking): 8014ff3e137fc3c6
  Timestamp: 2026-01-22T13:39:24.414973
```

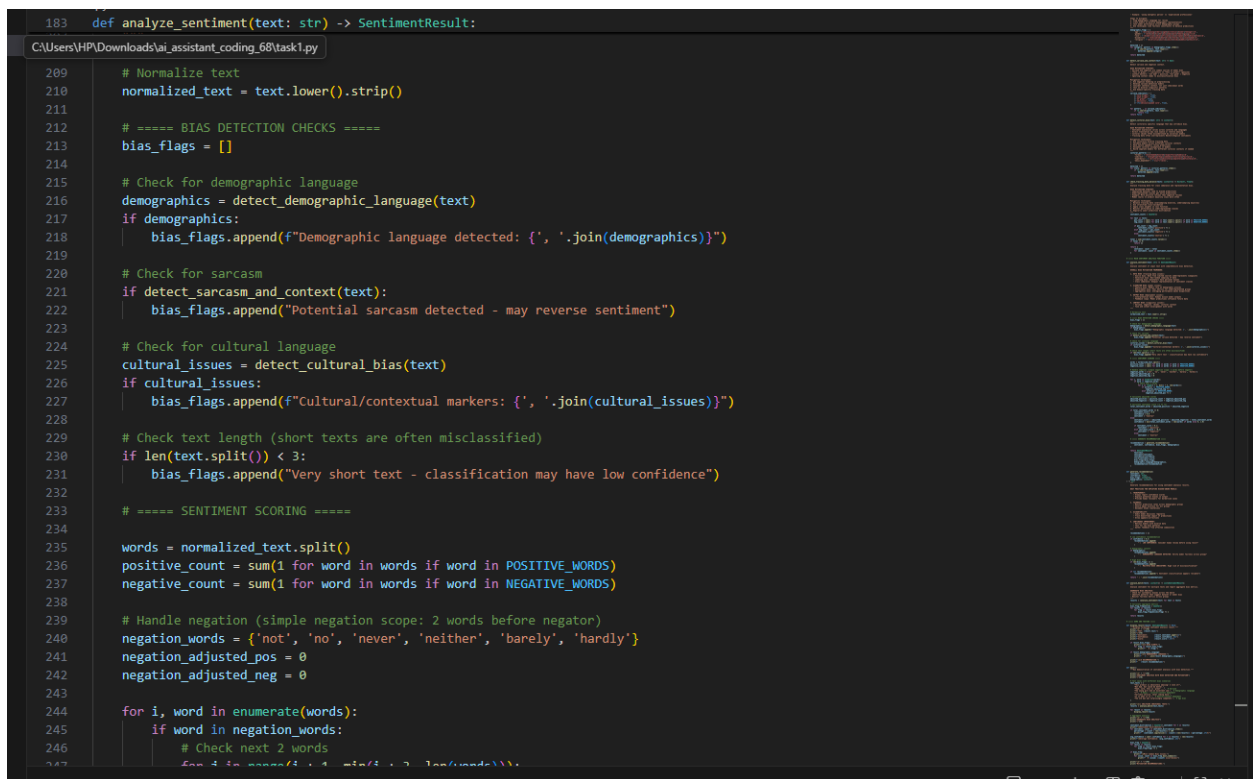
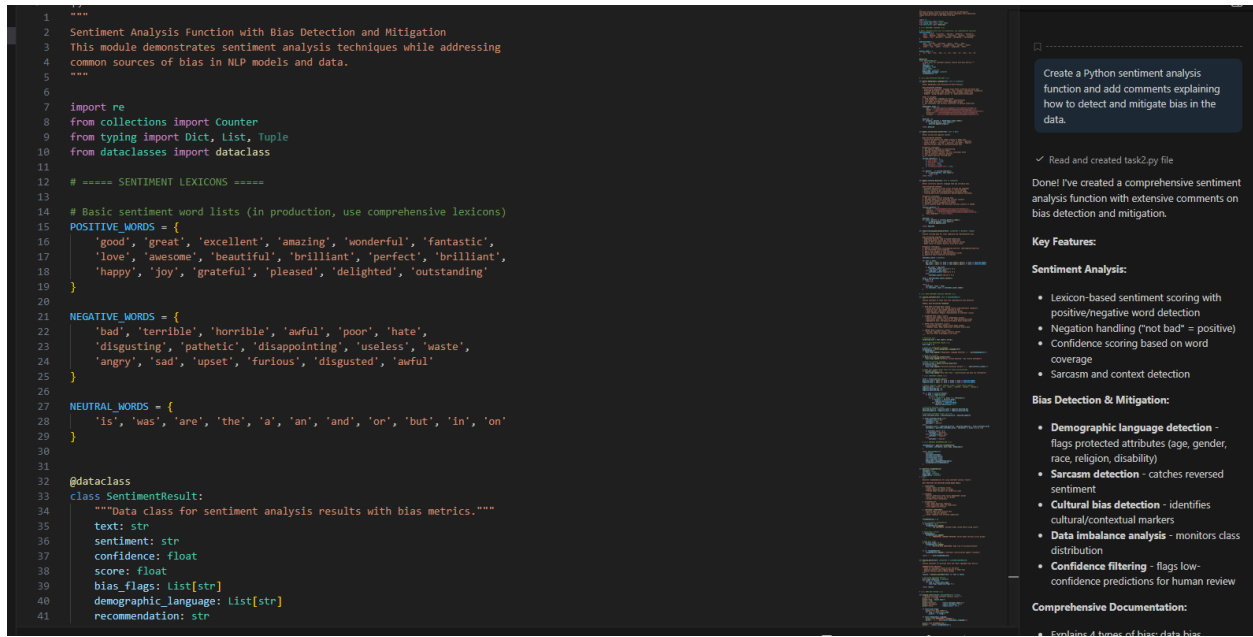
```
=====
PRIVACY NOTES:
=====
```

- ✓ Original data should be encrypted before storage
- ✓ Anonymized data can be used for analytics safely
- ✓ Implement role-based access controls
- ✓ Keep audit logs of all data access
- ✓ Regularly review data retention policies
- ✓ Comply with GDPR, CCPA, and local privacy laws

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.



```

RECOMMENDATION:
  ⚠️ LOW CONFIDENCE: Consider human review before using result | ⚠️ DEMOGRAPHIC LANGUAGE DETECTED: Verify model fairness across groups

=====
AGGREGATE BIAS ANALYSIS
=====

Sentiment Distribution:
  Positive: 2/8 (25.0%)
  Neutral: 5/8 (62.5%)
  Negative: 1/8 (12.5%)

Average Confidence: 5.85%

Most Common Bias Issues:
  • Demographic language detected: age, gender (2 occurrences)
  • Cultural/contextual markers: hyperbolic (1 occurrences)
  • Cultural/contextual markers: emoji_dependent (1 occurrences)

=====
MITIGATION RECOMMENDATIONS:
=====

1. COLLECT DIVERSE DATA:
  - Include multiple languages, cultures, demographics
  - Balance sentiment classes
  - Ensure representation of all user groups

2. IMPROVE PREPROCESSING:
  - Better sarcasm and negation detection
  - Handle emojis and modern language
  - Normalize cultural variations

3. ROBUST EVALUATION:
  - Test across demographic groups
  - Use fairness metrics (group calibration, equalized odds)
  - Conduct user studies with diverse participants

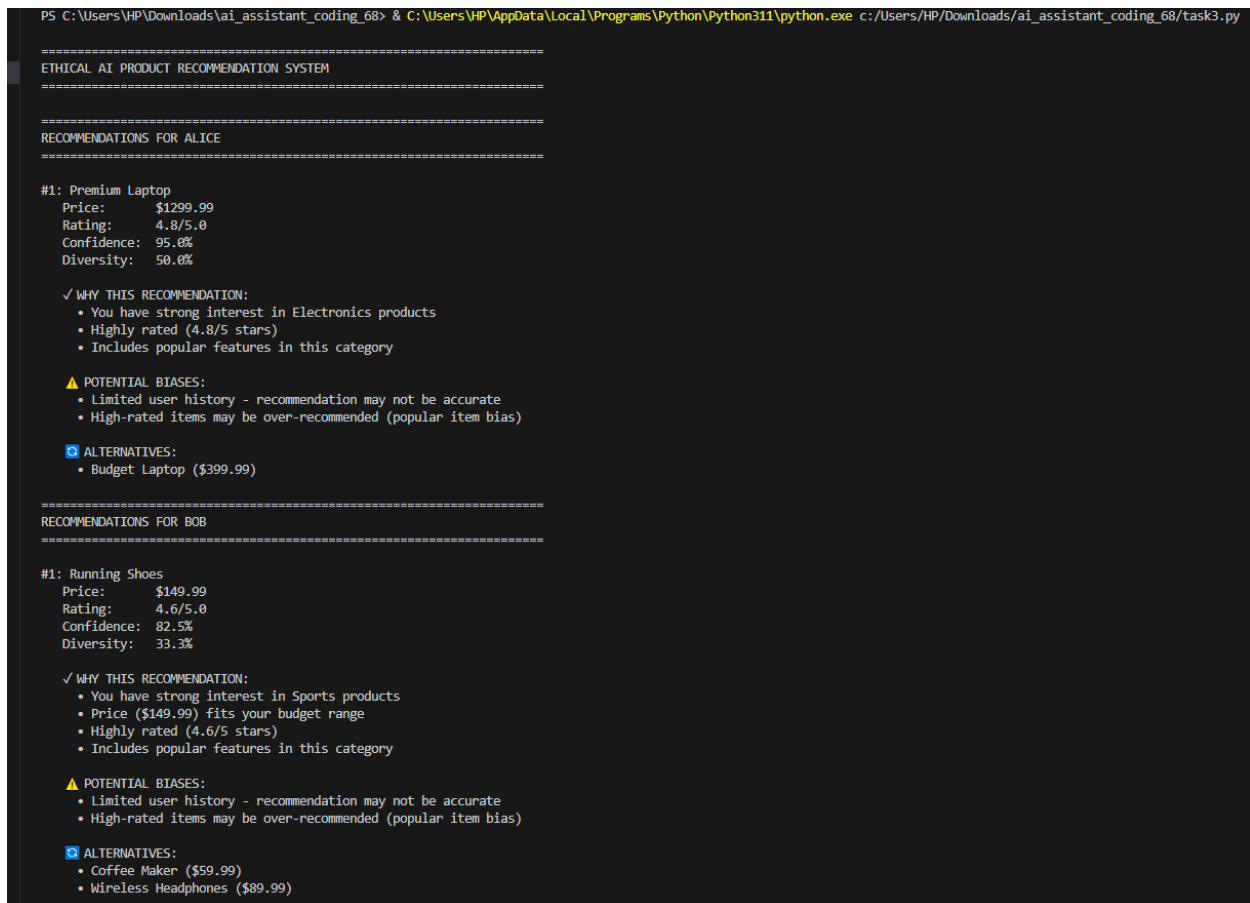
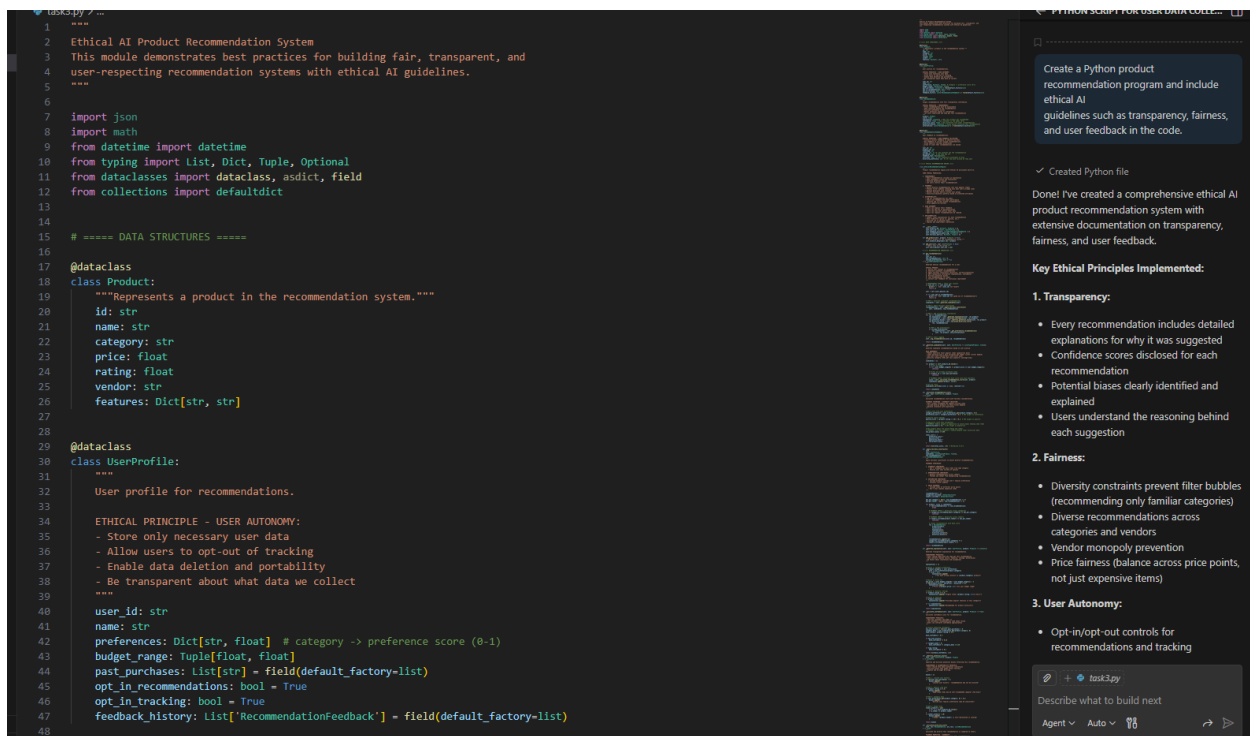
4. ONGOING MONITORING:
  - Track prediction distribution over time
  - Detect feedback loops
  - Audit decisions regularly

5. TRANSPARENCY:
  - Report confidence scores
  - Explain model limitations
  - Allow human review for important decisions

```

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.



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.

```
1  """
2  Secure Logging Functionality for Python Web Applications
3  This module demonstrates best practices for logging in web applications while
4  protecting sensitive user information and maintaining privacy compliance.
5  """
6
7  import logging
8  import logging.handlers
9  import re
10 import json
11 from typing import Dict, Any, Optional
12 from datetime import datetime
13 from functools import wraps
14 import hashlib
15
16
17 # ===== SENSITIVE DATA PATTERNS =====
18
19 # Patterns to detect and redact sensitive information
20 SENSITIVE_PATTERNS = {
21     # Credit card patterns (simplified)
22     'credit_card': r'\b\d{4}[\s-]?d{4}[\s-]?d{4}[\s-]?d{4}\b',
23
24     # Email addresses
25     'email': r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b',
26
27     # Phone numbers
28     'phone': r'\b(?:\+21[-.\s]?|(?[0-9]{3})\s?(?[-.\s]?[0-9]{3})[-.\s]?[0-9]{4}\b',
29
30     # Social Security Numbers (US)
31     'ssn': r'\b(?:\d{3})[\s-]?[0-9]{3}-?(?!\d{2})[0-9]{2}-?(?!\d{4})[0-9]{4}\b',
32
33     # API keys and tokens
34     'api_key': r'[Aa]pi[_-]?[Kk]ey["\']?s*[:-]\s*["\']?[A-Za-z0-9]{20,}\b',
35
36     # Passwords in common formats
37     'password': r'(?i)(password|passwd|pwd)["\']?s*[:-]\s*["\']?[^\s"\']+',
38
39     # Bearer tokens
40     'bearer_token': r'[Bb]earer\s+[A-Za-z0-9._-]+',
41
42     # Database connection strings
43     'db_connection': r'(?i)(user|password|host)=[^\s&]+',
44
45     # IPv4 addresses (less sensitive but can be PII)
46     'ipv4': r'\b(?:25[0-5]|2[0-4][0-9]|01[0-9][0-9]?|\.)?3(?:25[0-5]|2[0-4][0-9]|01[0-9][0-9]?|\b',
47 }
```

Create logging functionality in a Python web app and ensure logs do not capture sensitive user information.

✓ Created Python file

Done! I've created a comprehensive secure logging system for Python web applications that automatically protects sensitive user information.

Key Features:

Sensitive Data Detection & Redaction:

- Pattern matching for credit cards, emails, phone numbers, SSNs, API keys, passwords, tokens, etc.
- Dictionary field detection (redacts values of fields named "password", "email", etc.)
- Automatic hashing of PII for tracking without exposing identity
- Consistent [REDACTED] placeholders

Logging Configuration:

- Console handler for development/monitoring
- Rotating file handler (10MB auto-rotation, keeps 10 backups)
- Optional syslog handler for centralized logging
- Secure formatter that automatically redacts all logs
- Configurable log levels and retention

Privacy-Aware Logging Functions:

- `log_user_login()` - Track logins without

Describe what to build next

```

PS C:\Users\HP\Downloads\ai_assistant_coding_68> & C:\Users\HP\AppData\Local\Programs\Python\Python311\python.exe c:/Users/HP/Downloads/ai_assistant_coding_68/task4.py

=====
SECURE LOGGING FOR PYTHON WEB APPLICATIONS
=====

📄 LOGGING SCENARIOS:
=====

1 USER LOGIN LOGGING:
[2026-01-22 14:00:08,862] INFO - web_app - User HASH:f9e8e37d2e825eb0 logged in successfully
[2026-01-22 14:00:08,864] WARNING - web_app - Failed login attempt for user HASH:f9e8e37d2e825eb0
✓ Logged (sensitive email hashed)

2 API REQUEST LOGGING:
[2026-01-22 14:00:08,865] INFO - web_app - API GET /api/users/profile by HASH:f9e8e37d2e825eb0
✓ Logged (user ID hashed)

3 DATA ACCESS LOGGING:
[2026-01-22 14:00:08,866] INFO - web_app - User HASH:f9e8e37d2e825eb0 performed READ on payment_records
✓ Logged (sensitive access tracked)

4 ERROR LOGGING WITH CONTEXT:
[2026-01-22 14:00:08,867] ERROR - web_app - Error for user HASH:4e920dc577a96695: Payment processing failed
✓ Logged (sensitive fields automatically redacted)

5 SECURITY EVENT LOGGING:
[2026-01-22 14:00:08,868] ERROR - web_app - SECURITY EVENT [BRUTE_FORCE_ATTEMPT]: Multiple failed login attempts from IP [REDACTED]
✓ Logged (security incident tracked)

6 SENSITIVE DATA REDACTION EXAMPLES:

Original: User payment card 4532-1234-5678-9010 was processed
Redacted: User payment card [REDACTED] was processed

Original: API key: sk_live_51234567890abcdef was used
Redacted: API key: sk_live_51234567890abcdef was used

Original: Password reset for user@example.com successful
Redacted: Password reset for [REDACTED] successful

Original: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 authenticated
Redacted: [REDACTED] authenticated

Original: SSN 123-45-6789 verified
Redacted: SSN [REDACTED] verified

Original: Contact: +1-800-555-0123 or john.doe@company.com
Redacted: Contact: +[REDACTED] or [REDACTED]

```

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).

task5.py > ...

225

C:\Users\HPD\Downloads\ai_assistant_coding_68\secure_app.log

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

```
@dataclass
class ApplicantProfile:
    """Loan applicant profile for demonstration."""
    applicant_id: str
    age: float
    income: float
    credit_score: float
    employment_years: float
    protected_group: str # For fairness analysis

@dataclass
class PredictionResult:
    """
    ML prediction result with full transparency.

    TRANSPARENCY PRINCIPLE:
    - Every prediction includes explanation
    - Show confidence/probability
    - Disclose factors contributing to decision
    - Include fairness assessment
    - Document model limitations
    """
    applicant_id: str
    prediction: int # 0: Denied, 1: Approved
    probability: float # Confidence score 0-1
    explanation: List[str] # Why this decision
    fairness_flag: bool # Potential fairness concern
    fairness_reason: Optional[str]
    appeal_available: bool # Can applicant appeal
    model_version: str

@dataclass
class FairnessMetrics:
    """Comprehensive fairness metrics."""
    overall_accuracy: float
    group_accuracies: Dict[str, float]
    approval_rates: Dict[str, float] # Demographic parity
    false_negative_rates: Dict[str, float] # Equalized odds
    false_positive_rates: Dict[str, float] # Equalized odds
    demographic_parity_difference: float # Max difference in approval rates
    equalized_odds_difference: float # Max difference in error rates
```

PYTHON SCRIPT FOR USER DATA COLLE...

Create a machine learning model in Python and document how to use it responsibly, including limitations and fairness considerations.

✓ Created Python file

Done! I've created a comprehensive responsible ML model with extensive documentation on fairness and limitations.

Key Features:

Model Card (Comprehensive Documentation):

- Clear intended use and limitations
- Critical warnings about fairness concerns
- Known biases and mitigation strategies
- Regulatory considerations (Fair Housing Act, GDPR, CCPA, etc)
- Deployment requirements and monitoring

Fairness Metrics:

- Demographic Parity (approval rates across groups)
- Equalized Odds (false positive/negative rates)
- Disparate Impact analysis
- Group-level accuracy tracking
- Automatic fairness alert flags

Transparency Features:

- Human-readable explanations for every prediction
- Confidence scores disclosed
- Fairness concerns flagged for human review
- Rights information (appeals, transparency,

task5.py

Describe what to build next


```
PS C:\Users\HP\Downloads\ai_assistant_coding_68> ^C
PS C:\Users\HP\Downloads\ai_assistant_coding_68> C:/Users/HP/Downloads/ai_assistant_coding_68/.venv/Scripts/python.exe C:\Users\HP\Downloads\ai_assistant_coding_68\task5.py
```

=====

RESPONSIBLE MACHINE LEARNING MODEL

=====

LOAN ELIGIBILITY MODEL CARD

MODEL OVERVIEW:

Name: Loan Eligibility Classifier v1.0
Type: Binary Classification (RandomForestClassifier)
Training Date: 2026-01-22
Purpose: Predict loan eligibility for demonstration purposes
Intended Use: DEMONSTRATION ONLY - Not for production lending decisions

INTENDED USE:

✓ DO USE FOR:

- Educational demonstrations
- Understanding ML fairness concepts
- Testing and validation workflows
- Fairness auditing techniques

✗ DO NOT USE FOR:

- Actual lending decisions
- Production financial services
- High-stakes decisions affecting individuals
- Autonomous decision-making without human review

CRITICAL LIMITATIONS:

1. BIASED DATA:

- Training data contains historical lending patterns
- Reflects past discrimination and biases
- May perpetuate unfair decisions

2. INCOMPLETE INFORMATION:

- Only uses demographic and income features
- Missing important factors (credit history, employment stability)
- Cannot account for life circumstances

3. MODEL LIMITATIONS:

- Assumes historical patterns predict future outcomes
- Cannot capture economic changes or individual circumstances
- Oversimplifies complex financial decisions

4. FAIRNESS CONCERNS:

- Model may have disparate impact on protected groups