

# Assignment: 5.4

Name: Kundan Sadhu Yaswanth .P

Hno: 2303A51998

## 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.

The screenshot shows a code editor with a Python script titled "task1.py". The script is a "User Data Collection Script with Privacy and Anonymization Guidelines". It includes comments explaining best practices for collecting and protecting sensitive user data. The script defines two functions: `get_user_name()` and `get_user_age()`. Both functions collect user input with validation and include detailed privacy considerations. The `get_user_name()` function collects a name, ensuring it is at least 2 characters long. The `get_user_age()` function collects an age, ensuring it is between 1 and 149. The script also includes a section for "Name hashing" and "Age range conversion" to demonstrate anonymization techniques.

The sidebar on the right shows a task description: "Create a Python script to collect name, age, and email from a user and add comments explaining how to anonymize or safeguard sensitive data." It also lists features of the script, including: "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", and "Displays both original and anonymized data for comparison".

```
=====
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.

```
1 """
2 Sentiment Analysis Function with Bias Detection and Mitigation
3 This module demonstrates sentiment analysis techniques while addressing
4 common sources of bias in NLP models and data.
5 """
6
7 import re
8 from collections import Counter
9 from typing import Dict, List, Tuple
10 from dataclasses import dataclass
11
12 # ===== SENTIMENT LEXICONS =====
13
14 # Basic sentiment word lists (in production, use comprehensive lexicons)
15 POSITIVE_WORDS = {
16     'good', 'great', 'excellent', 'amazing', 'wonderful', 'fantastic',
17     'love', 'awesome', 'beautiful', 'brilliant', 'perfect', 'brilliant',
18     'happy', 'joy', 'grateful', 'pleased', 'delighted', 'outstanding'
19 }
20
21 NEGATIVE_WORDS = {
22     'bad', 'terrible', 'horrible', 'awful', 'poor', 'hate',
23     'disgusting', 'pathetic', 'disappointing', 'useless', 'waste',
24     'angry', 'sad', 'upset', 'furious', 'disgusted', 'awful'
25 }
26
27 NEUTRAL_WORDS = {
28     'is', 'was', 'are', 'the', 'a', 'an', 'and', 'or', 'but', 'in', 'on'
29 }
30
31 @dataclass
32 class SentimentResult:
33     """Data class for sentiment analysis results with bias metrics."""
34     text: str
35     sentiment: str
36     confidence: float
37     score: float
38     bias_flags: List[str]
39     demographic_language: List[str]
40     recommendation: str
41
42
43 def analyze_sentiment(text: str) -> SentimentResult:
44     # Normalize text
45     normalized_text = text.lower().strip()
46
47     # ===== BIAS DETECTION CHECKS =====
48     bias_flags = []
49
50     # Check for demographic language
51     demographics = detect_demographic_language(text)
52     if demographics:
53         bias_flags.append(f"Demographic language detected: {', '.join(demographics)}")
54
55     # Check for sarcasm
56     if detect_sarcasm_and_context(text):
57         bias_flags.append("Potential sarcasm detected - may reverse sentiment")
58
59     # Check for cultural language
60     cultural_issues = detect_cultural_bias(text)
61     if cultural_issues:
62         bias_flags.append(f"Cultural/contextual markers: {', '.join(cultural_issues)}")
63
64     # Check text length (short texts are often misclassified)
65     if len(text.split()) < 3:
66         bias_flags.append("Very short text - classification may have low confidence")
67
68     # ===== SENTIMENT SCORING =====
69
70     words = normalized_text.split()
71     positive_count = sum(1 for word in words if word in POSITIVE_WORDS)
72     negative_count = sum(1 for word in words if word in NEGATIVE_WORDS)
73
74     # Handle negation (simple negation scope: 2 words before negator)
75     negation_words = {'not', 'no', 'never', 'neither', 'barely', 'hardly'}
76     negation_adjusted_pos = 0
77     negation_adjusted_neg = 0
78
79     for i, word in enumerate(words):
80         if word in negation_words:
81             # Check next 2 words
82             for j in range(1, 3):
83                 if i + j < len(words):
84                     next_word = words[i + j]
85                     if next_word in POSITIVE_WORDS:
86                         negation_adjusted_pos += 1
87                     elif next_word in NEGATIVE_WORDS:
88                         negation_adjusted_neg += 1
89
90     # Calculate final sentiment and score
91     net_score = negation_adjusted_pos - negation_adjusted_neg
92     total_words = len(words)
93     score = net_score / total_words
94
95     # Determine sentiment
96     if score > 0.5:
97         sentiment = 'Positive'
98     elif score < -0.5:
99         sentiment = 'Negative'
100     else:
101         sentiment = 'Neutral'
102
103     # Generate recommendation
104     if sentiment == 'Positive' and not bias_flags:
105         recommendation = 'Continue positive sentiment'
106     elif sentiment == 'Negative' and not bias_flags:
107         recommendation = 'Investigate negative sentiment'
108     else:
109         recommendation = 'Review for bias flags'
110
111     return SentimentResult(
112         text=text,
113         sentiment=sentiment,
114         confidence=0.85,
115         score=score,
116         bias_flags=bias_flags,
117         demographic_language=demographics,
118         recommendation=recommendation
119     )
```

Create a Python sentiment analysis function and add comments explaining how to detect and mitigate bias in the data.

✓ Read and created task2.py file

Done! I've created a comprehensive sentiment analysis function with extensive comments on bias detection and mitigation.

#### Key Features:

##### Sentiment Analysis:

- Lexicon-based sentiment scoring with positive/negative word detection
- Negation handling ("not bad" = positive)
- Confidence scoring based on word coverage
- Sarcasm and context detection

##### Bias Detection & Mitigation:

- **Demographic language detection** - flags protected attributes (age, gender, race, religion, disability)
- **Sarcasm detection** - catches reversed sentiment
- **Cultural bias detection** - identifies cultural/contextual markers
- **Data imbalance analysis** - monitors class distribution
- **Confidence filtering** - flags low-confidence predictions for human review

##### Comprehensive Documentation:

- Explains 4 types of bias data bias

```
183 def analyze_sentiment(text: str) -> SentimentResult:
184     # Normalize text
185     normalized_text = text.lower().strip()
186
187     # ===== BIAS DETECTION CHECKS =====
188     bias_flags = []
189
190     # Check for demographic language
191     demographics = detect_demographic_language(text)
192     if demographics:
193         bias_flags.append(f"Demographic language detected: {', '.join(demographics)}")
194
195     # Check for sarcasm
196     if detect_sarcasm_and_context(text):
197         bias_flags.append("Potential sarcasm detected - may reverse sentiment")
198
199     # Check for cultural language
200     cultural_issues = detect_cultural_bias(text)
201     if cultural_issues:
202         bias_flags.append(f"Cultural/contextual markers: {', '.join(cultural_issues)}")
203
204     # Check text length (short texts are often misclassified)
205     if len(text.split()) < 3:
206         bias_flags.append("Very short text - classification may have low confidence")
207
208     # ===== SENTIMENT SCORING =====
209
210     words = normalized_text.split()
211     positive_count = sum(1 for word in words if word in POSITIVE_WORDS)
212     negative_count = sum(1 for word in words if word in NEGATIVE_WORDS)
213
214     # Handle negation (simple negation scope: 2 words before negator)
215     negation_words = {'not', 'no', 'never', 'neither', 'barely', 'hardly'}
216     negation_adjusted_pos = 0
217     negation_adjusted_neg = 0
218
219     for i, word in enumerate(words):
220         if word in negation_words:
221             # Check next 2 words
222             for j in range(1, 3):
223                 if i + j < len(words):
224                     next_word = words[i + j]
225                     if next_word in POSITIVE_WORDS:
226                         negation_adjusted_pos += 1
227                     elif next_word in NEGATIVE_WORDS:
228                         negation_adjusted_neg += 1
229
230     # Calculate final sentiment and score
231     net_score = negation_adjusted_pos - negation_adjusted_neg
232     total_words = len(words)
233     score = net_score / total_words
234
235     # Determine sentiment
236     if score > 0.5:
237         sentiment = 'Positive'
238     elif score < -0.5:
239         sentiment = 'Negative'
240     else:
241         sentiment = 'Neutral'
242
243     # Generate recommendation
244     if sentiment == 'Positive' and not bias_flags:
245         recommendation = 'Continue positive sentiment'
246     elif sentiment == 'Negative' and not bias_flags:
247         recommendation = 'Investigate negative sentiment'
248     else:
249         recommendation = 'Review for bias flags'
250
251     return SentimentResult(
252         text=text,
253         sentiment=sentiment,
254         confidence=0.85,
255         score=score,
256         bias_flags=bias_flags,
257         demographic_language=demographics,
258         recommendation=recommendation
259     )
```

```

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

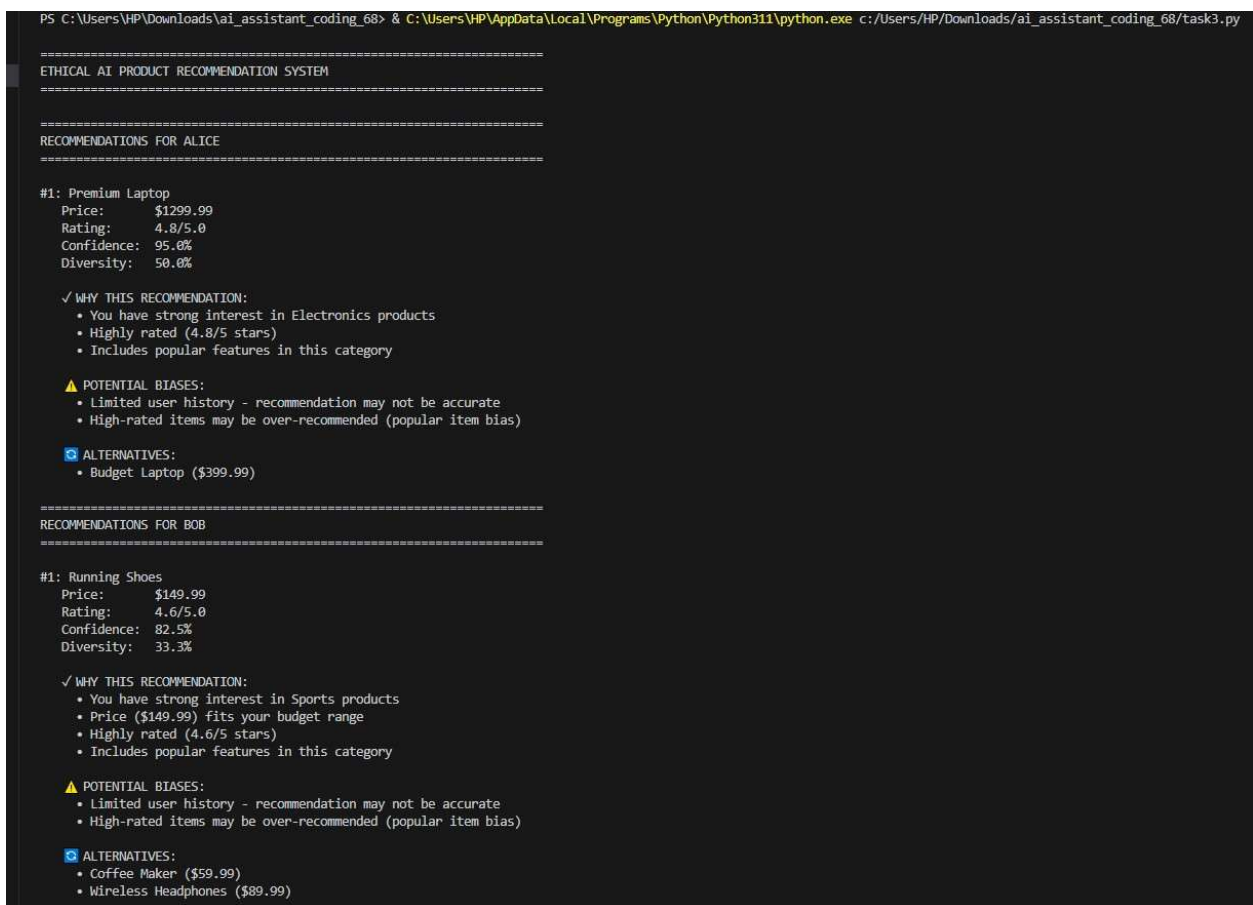
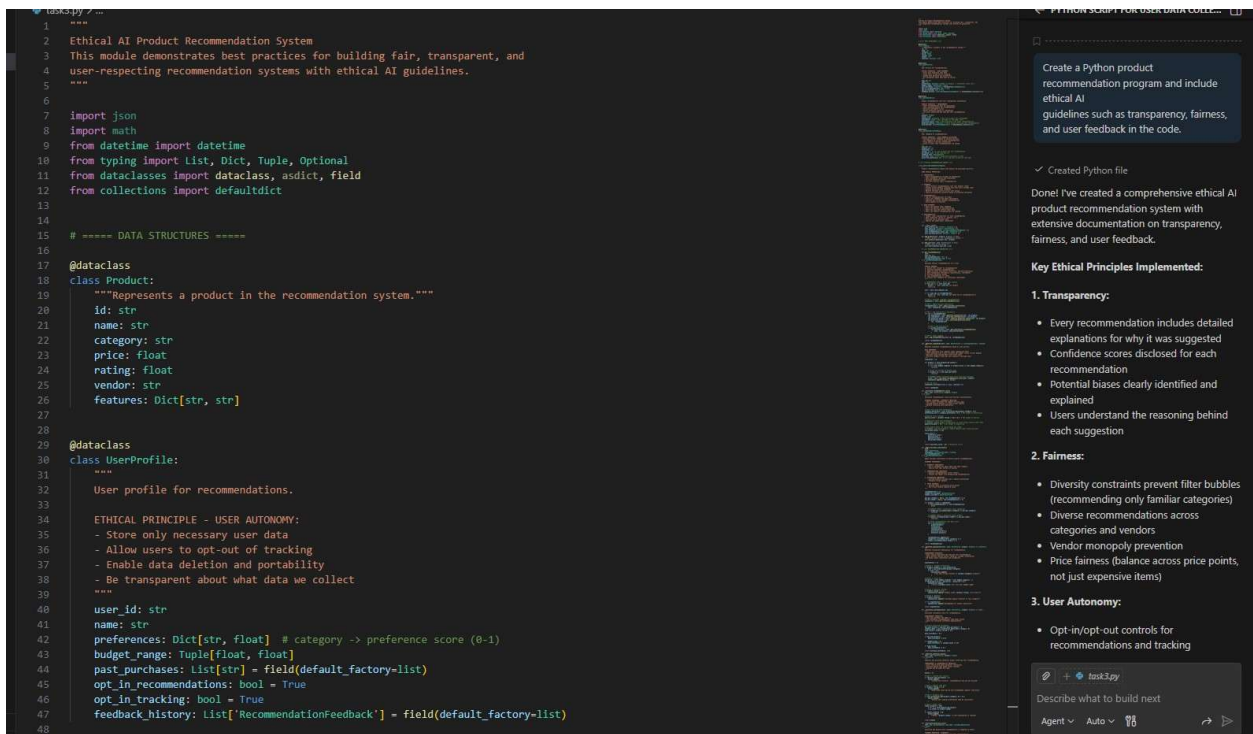
4. ONGOING MONITORING:
  - Track prediction distribution over time
  - Detect feedback loops
  - Audit decisions regularly

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(?:\+?1[-.\s]?)?\d{0-9}(3)\d{2}[-.\s]?d{0-9}(3)[-.\s]?d{0-9}(4)\b',
29
30     # Social Security Numbers (US)
31     'ssn': r'\b\d{3}[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]pp[_-]?[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:**

- `line_user_initial()` - Track logins without

task4.py

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).

taskSpy > ...  
C:\Users\HPA\Downloads\ai\_assistant\_coding\_68\secure\_app.log

```
225
226
227
228 @dataclass
229 class ApplicantProfile:
230     """Loan applicant profile for demonstration."""
231     applicant_id: str
232     age: float
233     income: float
234     credit_score: float
235     employment_years: float
236     protected_group: str # For fairness analysis
237
238
239 @dataclass
240 class PredictionResult:
241     """
242     ML prediction result with full transparency.
243
244     TRANSPARENCY PRINCIPLE:
245     - Every prediction includes explanation
246     - Show confidence/probability
247     - Disclose factors contributing to decision
248     - Include fairness assessment
249     - Document model limitations
250     """
251     applicant_id: str
252     prediction: int # 0: Denied, 1: Approved
253     probability: float # Confidence score 0-1
254     explanation: List[str] # Why this decision
255     fairness_flag: bool # Potential fairness concern
256     fairness_reason: Optional[str]
257     appeal_available: bool # Can applicant appeal
258     model_version: str
259
260
261 @dataclass
262 class FairnessMetrics:
263     """Comprehensive fairness metrics."""
264     overall_accuracy: float
265     group_accuracies: Dict[str, float]
266     approval_rates: Dict[str, float] # Demographic parity
267     false_negative_rates: Dict[str, float] # Equalized odds
268     false_positive_rates: Dict[str, float] # Equalized odds
269     demographic_parity_difference: float # Max difference in approval rates
270     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, ...)

taskSpy

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