

Assignment: 5.4

Name: N.NITHISH

Hno: 2303A51066

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 Python script titled "task1.py" in a code editor. The script includes comments explaining data protection and anonymization techniques. It defines functions for getting user name, age, and email, each with validation logic. A sidebar on the right provides a summary of the script's purpose and features.

```
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 sensitive personal information
    - Implement strict validation rules for email format
    - Consider using a masked or hashed version for storage
    """

    while True:
        email = input("Enter your email: ").strip()
        if re.match(r"^\w+@\w+\.\w+$", email):
            return email
        print("Please enter a valid email address.")


if __name__ == "__main__":
    print("Data Collection Script Starting...")
    # Collect user data
    name = get_user_name()
    age = get_user_age()
    email = get_user_email()

    # Hash the collected data
    hashed_data = hashlib.sha256((name + str(age) + email).encode()).hexdigest()

    # Log the hashed data to a file
    with open("user_data.log", "w") as log_file:
        log_file.write(f"Hashed Data: {hashed_data}\n")
        log_file.write(f"Name: {name}\n")
        log_file.write(f"Age: {age}\n")
        log_file.write(f"Email: {email}\n")

    print("Data collection and hashing completed successfully!")
```

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.

```

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
32 @dataclass
33 class SentimentResult:
34     """Data class for sentiment analysis results with bias metrics."""
35     text: str
36     sentiment: str
37     confidence: float
38     score: float
39     bias_flags: List[str]
40     demographic_language: List[str]
41     recommendation: str

```

Annotations and comments:

- Top right: "Create a Python sentiment analysis function and add comments explaining how to detect and mitigate bias in the data."
- Bottom right: "Done! I've created a comprehensive sentiment analysis function with extensive comments on bias detection and mitigation."
- Key Features: "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:
    C:\Users\HP\Downloads\ai_assistant_coding_68\task1.py
184
185     # Normalize text
186     normalized_text = text.lower().strip()
187
188     # ===== BIAS DETECTION CHECKS =====
189     bias_flags = []
190
191     # Check for demographic language
192     demographics = detect_demographic_language(text)
193     if demographics:
194         bias_flags.append(f"Demographic language detected: {', '.join(demographics)}")
195
196     # Check for sarcasm
197     if detect_sarcasm_and_context(text):
198         bias_flags.append("Potential sarcasm detected - may reverse sentiment")
199
200     # Check for cultural language
201     cultural_issues = detect_cultural_bias(text)
202     if cultural_issues:
203         bias_flags.append(f"Cultural/contextual markers: {', '.join(cultural_issues)}")
204
205     # Check text length (short texts are often misclassified)
206     if len(text.split()) < 3:
207         bias_flags.append("Very short text - classification may have low confidence")
208
209     # ===== SENTIMENT SCORING =====
210
211     words = normalized_text.split()
212     positive_count = sum(1 for word in words if word in POSITIVE_WORDS)
213     negative_count = sum(1 for word in words if word in NEGATIVE_WORDS)
214
215     # Handle negation (simple negation scope: 2 words before negator)
216     negation_words = {'not', 'no', 'never', 'neither', 'barely', 'hardly'}
217     negation_adjusted_pos = 0
218     negation_adjusted_neg = 0
219
220     for i, word in enumerate(words):
221         if word in negation_words:
222             # Check next 2 words
223             for j in range(i+1, min(i+3, len(words)-1), 1):
224                 if words[j] in negation_words:
225                     negation_adjusted_pos += 1
226                     break
227
228             negation_adjusted_pos -= 1
229
230             if words[i+1] in negation_words:
231                 negation_adjusted_neg += 1
232
233             if words[i+2] in negation_words:
234                 negation_adjusted_neg -= 1
235
236     # Calculate overall sentiment score
237     score = (positive_count - negative_count) / (positive_count + negative_count)
238
239     # Determine sentiment based on score
240     if score > 0.5:
241         sentiment = "Positive"
242     elif score < -0.5:
243         sentiment = "Negative"
244     else:
245         sentiment = "Neutral"
246
247     # Create and return sentiment result object
248     return SentimentResult(text, sentiment, confidence=score, bias_flags=bias_flags,
249                           demographic_language=demographics, recommendation="Review flagged bias areas")

```

```
■ 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.

```

1  """
2  Ethical AI Product Recommendation System
3  This module demonstrates best practices for building fair, transparent, and
4  user-respecting recommendation systems with ethical AI guidelines.
5  """
6
7  import json
8  import math
9  from datetime import datetime
10 from typing import List, Dict, Tuple, Optional
11 from dataclasses import dataclass, field
12 from collections import defaultdict
13
14
15 # ===== DATA STRUCTURES =====
16
17 @dataclass
18 class Product:
19     """Represents a product in the recommendation system."""
20     id: str
21     name: str
22     category: str
23     price: float
24     rating: float
25     vendor: str
26     features: Dict[str, str]
27
28
29 @dataclass
30 class UserProfile:
31     """
32     User profile for recommendations.
33
34     ETHICAL PRINCIPLE - USER AUTONOMY:
35     - Store only necessary user data
36     - Allow users to opt-out of tracking
37     - Enable data deletion and portability
38     - Be transparent about what data we collect
39     """
40     user_id: str
41     name: str
42     preferences: Dict[str, float] # category -> preference score (0-1)
43     budget_range: Tuple[float, float]
44     past_purchases: List[str] = field(default_factory=list)
45     opt_in_recommendations: bool = True
46     opt_in_tracking: bool = True
47     feedback_history: List[RecommendationFeedback] = field(default_factory=list)
48

```

```

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/task3.py
=====
ETHICAL AI PRODUCT RECOMMENDATION SYSTEM
=====

=====

RECOMMENDATIONS FOR ALICE
=====

#1: Premium Laptop
Price:      $1299.99
Rating:     4.8/5.0
Confidence: 95.0%
Diversity:  50.0%

✓ WHY THIS RECOMMENDATION:
• You have strong interest in Electronics products
• Highly rated (4.8/5 stars)
• Includes popular features in this category

⚠ POTENTIAL BIASES:
• Limited user history - recommendation may not be accurate
• High-rated items may be over-recommended (popular item bias)

▢ ALTERNATIVES:
• Budget Laptop ($399.99)

=====

RECOMMENDATIONS FOR BOB
=====

#1: Running Shoes
Price:      $149.99
Rating:     4.6/5.0
Confidence: 82.5%
Diversity:  33.3%

✓ WHY THIS RECOMMENDATION:
• You have strong interest in Sports products
• Price ($149.99) fits your budget range
• Highly rated (4.6/5 stars)
• Includes popular features in this category

⚠ POTENTIAL BIASES:
• Limited user history - recommendation may not be accurate
• High-rated items may be over-recommended (popular item bias)

▢ ALTERNATIVES:
• Coffee Maker ($59.99)
• Wireless Headphones ($89.99)

```

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.

```
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
=====

1 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
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

```

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 CONCERN:
 - Model may have disparate impact on protected groups