

Assignment: 5.4

Name: N.AKHIL

Hno: 2303A51065

BT.NO :16

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 displays the Visual Studio Code editor with a Python script named `secure_user_input.py`. The script is designed for secure data collection, using `hashlib` to hash email addresses with SHA-256. It includes a `collect_user_data()` function that prompts for name, age, and email, then stores the data in a dictionary with the hashed email. A `__main__` block calls this function. The terminal shows the script being executed, with user input: name 'Akhil', age '20', and email '2303a51065@sru.edu.in'. The output shows the stored data with the email hashed.

```
1 # Secure User Input - Privacy-focused data collection
2 # Email is hashed to protect user privacy; never store sensitive data in plain text
3 import hashlib
4 def collect_user_data():
5     """Collect and securely store user information."""
6     name = input("Enter your name: ")
7     age = input("Enter your age: ")
8     email = input("Enter your email: ")
9
10    # Hash email using SHA-256 for privacy protection
11    hashed_email = hashlib.sha256(email.encode()).hexdigest()
12
13    user_data = {"name": name, "age": age, "email_hash": hashed_email}
14    print(f"\nStored securely: {user_data}")
15    return user_data
16
17 if __name__ == "__main__":
18     collect_user_data()
19
```

The GitHub Copilot chat window on the right shows the prompt: "Write a short and simple Python script (under 20 lines) that collects user name, age, and email." The requirements listed are: hash or anonymize the email before storing, do not store sensitive data in plain text, add brief comments about privacy and security, include a main block (if `name == '__main__'`), and keep the code minimal and beginner-friendly. The chat also shows the script was created and provides key features: email hashing using SHA-256 via `hashlib`, brief security comments, a main block, and minimal/beginner-friendly code using only standard libraries.

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', 'on', '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
```

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                 next_word = words[i + j] if i + j < len(words) else None
224                 if next_word:
225                     if next_word in POSITIVE_WORDS:
226                         negation_adjusted_pos += 1
227                     elif next_word in NEGATIVE_WORDS:
228                         negation_adjusted_neg += 1

```

```

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

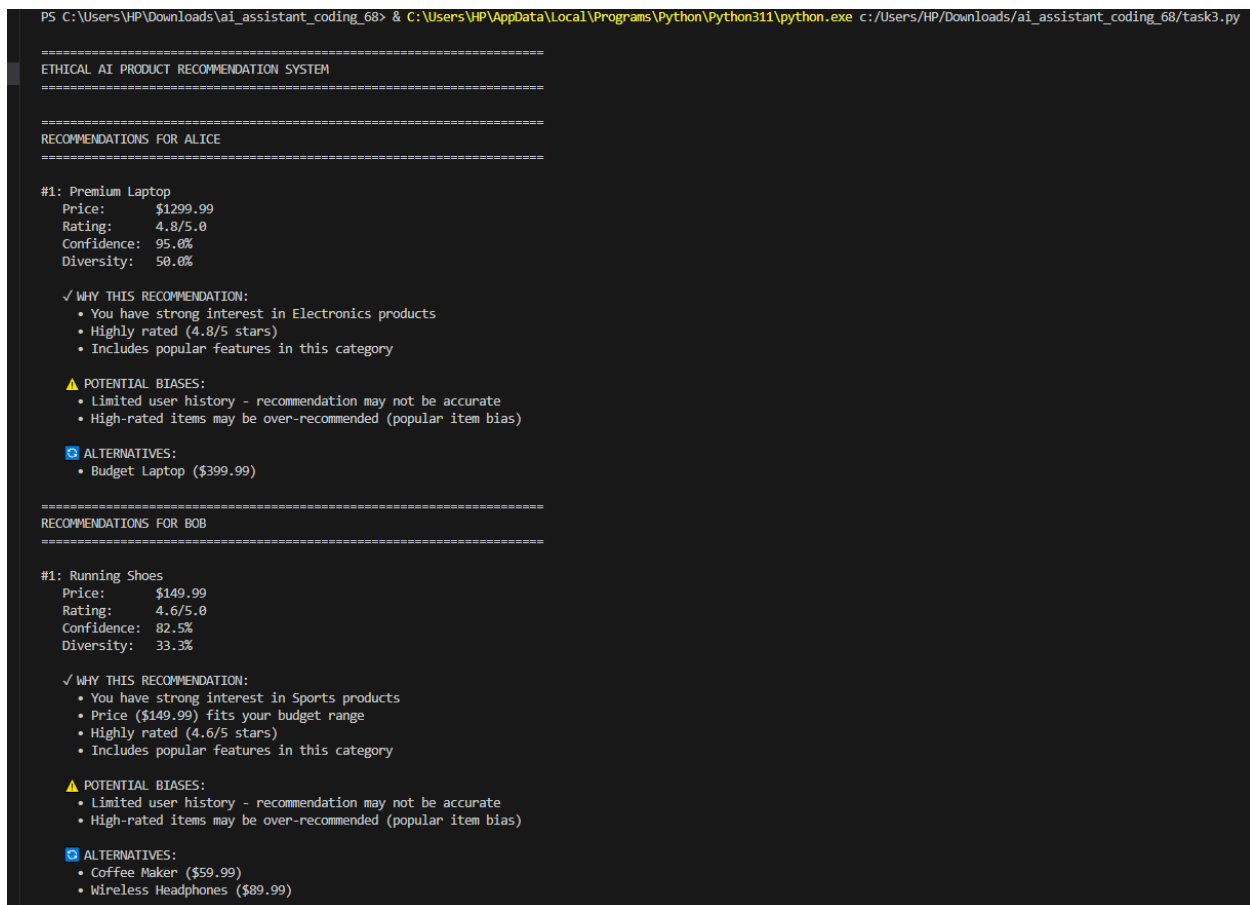
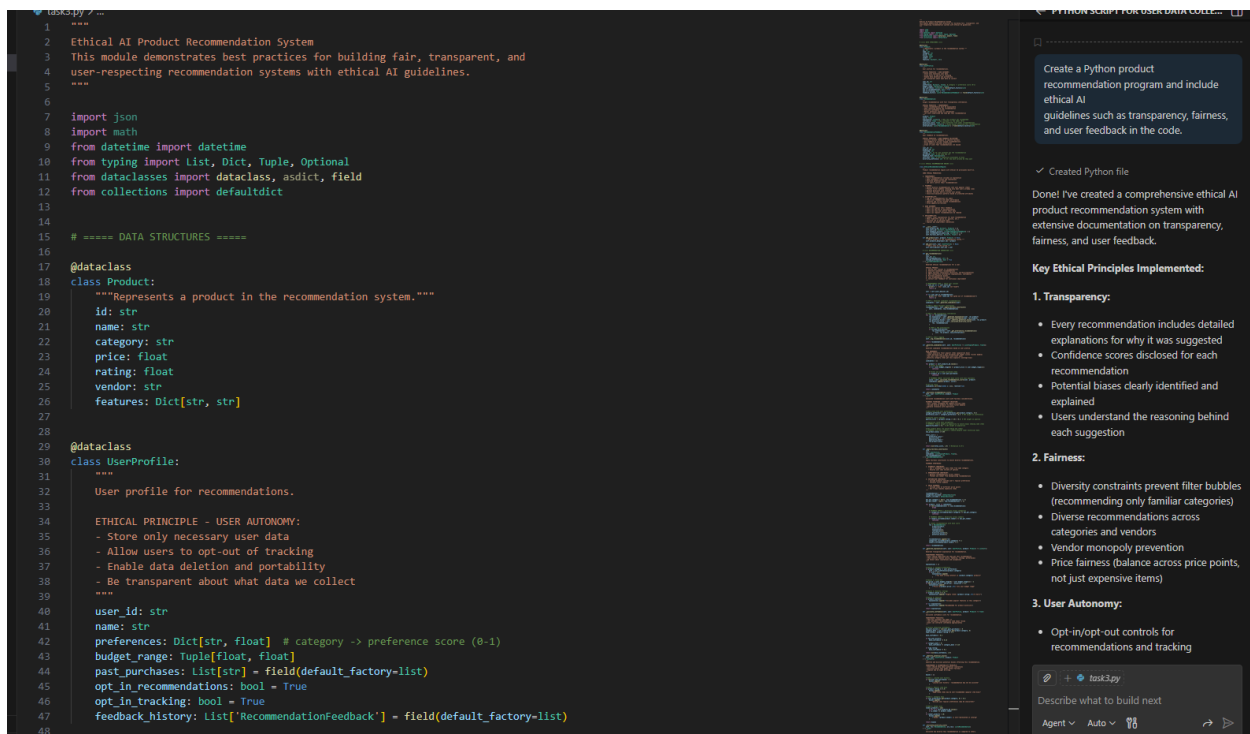
4. ONGOING MONITORING:
  - Track prediction distribution over time
  - Detect feedback loops
  - Audit decisions regularly

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]?)(?:[0-9]{3})?[-.\s]?[0-9]{3}[-.\s]?[0-9]{4}\b',
29
30     # Social Security Numbers (US)
31     'ssn': r'\b(?:[0-9]{3}-?[0-9]{2}-?[0-9]{4})\b',
32
33     # API keys and tokens
34     'api_key': r'[Aa]pi[_-]?[k]ey["\']?s*[:-]\s*["\']?[A-Za-z0-9]{20,}\b',
35
36     # Passwords in common formats
37     'password': r'^(?!(password|passwd|pwd))["\']?s*[:-]\s*["\']?[^\s"\']+$',
38
39     # Bearer tokens
40     'bearer_token': r'[Bb]earer\s+[A-Za-z0-9._-]*',
41
42     # Database connection strings
43     'db_connection': r'^(?!(user|password|host)=)[^\s&]+$',
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]{3}\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

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).

task5.py > ...

225

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

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