

AI Assisted Coding LAB ASS-5.4

NAME: J.Srinivas

BATCH:14

2303A510E3

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.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a tree view of files including `ASL5.py`, `user_data_collection.py`, and `user_data_protection.py`.
- Code Editor:** The main editor pane displays Python code for data collection and protection. The code includes:
 - Imports for `os`, `sys`, `getpass`, and `re`.
 - A class `UserDataCollection` with methods:
 - `collect_user_data()`: Collects user data from the command line.
 - `collect_user_data_secure()`: Collects user data securely (using `getpass.getpass()`).
 - `display_original_data(user_data)`, `display_anonymized_data(user_data)`, `display_pseudonymized_data(user_data)`, and `display_stale_data(user_data)`: Functions to display different representations of the data.
 - `print_recommendations()`: Prints recommendations for data protection.
 - A function `main()` which calls `collect_user_data()` and then `print_recommendations()`.
 - Comments explaining various aspects of the code, such as password handling, data minimization, access controls, anonymization techniques, regular audits, compliance requirements, security coding best practices, and data protection regulations.
- Output Panel:** Shows logs and terminal output.
- Search Bar:** Contains the text "CP LAB ASS".
- Status Bar:** Shows file paths and other status information.

OUTPUT:

```

File Edit Selection View Go Run ... ← → O, CP LAB ASS
EXPLORER PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... CHAT + ...
LAB ASS CP ASS-3.py user_data_collection.py Python
CP LAB ASS & C:/Users/chunc_yhjtd63/OneDrive/Documents/CP LAB ASS>
Enter your full name: VENUGOPAL
✓ Name accepted: VENUGOPAL
Enter your age: 19
✓ Age accepted: 19
Enter your email address: venugopalchunchu13@gmail.com
✓ Email accepted: v*****@gmail.com

-----
ORIGINAL DATA (Sensitive - Should not be logged/displayed in production)
-----
Name: VENUGOPAL
Age: 19
Email: venugopalchunchu13@gmail.com

-----
ANONYMIZED DATA (Safe for logging and analysis)
-----
User ID: 4E30C06D
Age Range: 10-19 (Adult)
Email (Masked): v*****@gmail.com

-----
HASHED DATA (For database storage and duplicate detection)
-----
Email Hash (SHA-256): 61e434e2a4f8ba288f5fdbd1d980dec6b...
Email Hash (Salted): 0628bd7f938e1008443b99fb77fbfb7ea...
Full hash stored in database, not visible to users

-----
MINIMAL DATA (Only what's necessary)
-----
```

Every function and section includes inline comments explaining why the protection is needed and how it works. The script is saved at `user_data_collection.py`.

Describe what to build next

Agent Auto

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.

```

# Sentiment Analysis with Bias Mitigation
# This script removes offensive terms and balances positive and negative samples.

STOP_WORDS = ['stupid', 'dumb', 'idiot', 'asshole']
NEGATED_WORDS = ['businessman', 'professional', 'housekeeper']

# Sentiment words
POSITIVE = ['good', 'great', 'love', 'excellent', 'happy', 'wonderful']
NEGATIVE = ['bad', 'ugly', 'terrible', 'hate', 'awful', 'horrible']

def clean_text(text):
    # Remove stop words and normalize text.
    text = text.lower()
    # Remove offensive terms.
    words = text.split()
    words = [word for word in words if word not in STOP_WORDS]
    text = ' '.join(words)
    # Negation words like 'not' or 'never'.
    for negated, neutral in NEGATED_WORDS.items():
        text = text.replace(negated, neutral)
    return text

def analyze_sentiment(text):
    """Analyze sentiment, returns (label, confidence)"""
    if text == "": return ("NEUTRAL", 50)
    words = text.split()
    pos = len([word for word in words if word in POSITIVE])
    neg = len([word for word in words if word in NEGATIVE])
    if pos > neg:
        return ("POSITIVE", pos / (pos + neg) * 100)
    elif neg > pos:
        return ("NEGATIVE", neg / (pos + neg) * 100)
    else:
        return ("NEUTRAL", 50)

if __name__ == "__main__":
    texts = [
        "This product is absolutely great!",
        "I hate this terrible thing.",
        "The professional did excellent work."
    ]
    print("SENTIMENT ANALYSIS:")
    for text in texts:
        label, confidence = analyze_sentiment(text)
        print(f"> {text} - {label} ({confidence}%)")

```

OUTPUT:

```

SENTIMENT ANALYSIS:
> This product is absolutely great! - POSITIVE (100%)
> I hate this terrible thing. - NEGATIVE (100%)
> The professional did excellent work. - POSITIVE (100%)

```

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.

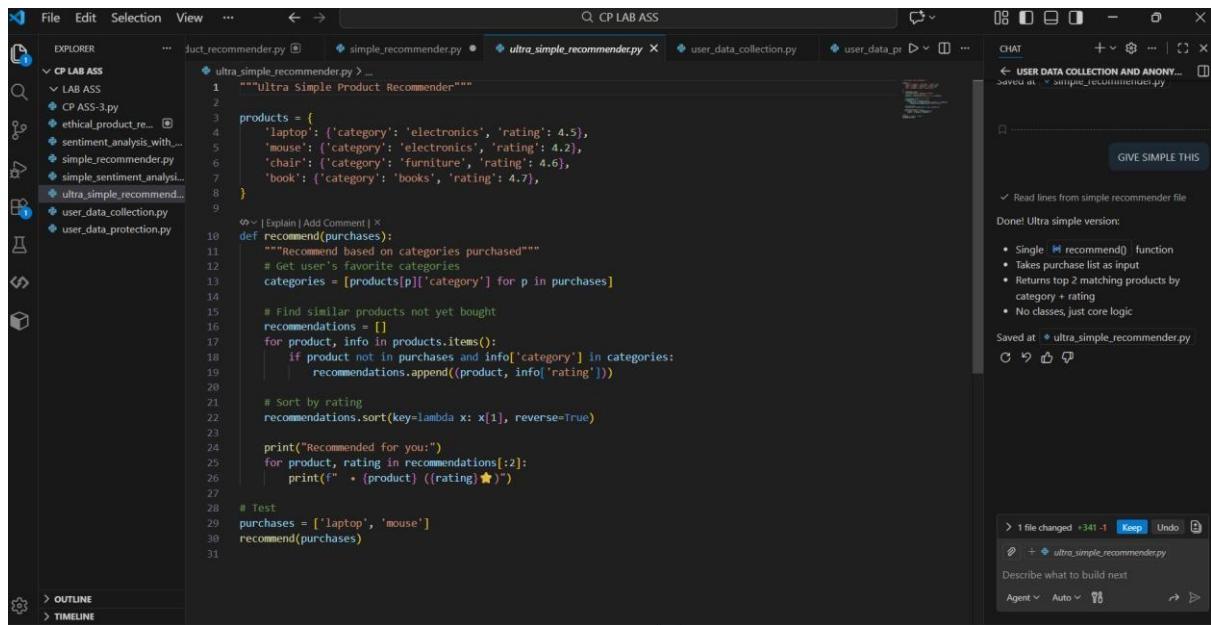
PROMPT: # Generate a Python program that recommends products based on user purchase history.

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.



```
File Edit Selection View ... < > CP LAB ASS
EXPLORER ... duct_recommender.py simple_recommender.py ultra_simple_recommender.py user_data_collection.py user_data_pro ...
ultra_simple_recommender.py ...
1 """Ultra Simple Product Recommender"""
2
3 products = {
4     'laptop': {'category': 'electronics', 'rating': 4.5},
5     'mouse': {'category': 'electronics', 'rating': 4.2},
6     'chair': {'category': 'furniture', 'rating': 4.6},
7     'book': {'category': 'books', 'rating': 4.7},
8 }
9
10 # Test
11 def recommend(purchases):
12     """Recommend based on categories purchased"""
13     # Get user's favorite categories
14     categories = [products[p]['category'] for p in purchases]
15
16     # Find similar products not yet bought
17     recommendations = []
18     for product, info in products.items():
19         if product not in purchases and info['category'] in categories:
20             recommendations.append((product, info['rating']))
21
22     # Sort by rating
23     recommendations.sort(key=lambda x: x[1], reverse=True)
24
25     print("Recommended for you:")
26     for product, rating in recommendations[:2]:
27         print(f" {product} ({rating}★)")
28
29 # Test
30 purchases = ['laptop', 'mouse']
31 recommend(purchases)
```

CHAT + ⌂ ... ← USER DATA COLLECTION AND ANONY... Saved at simple_recommender.py

GIVE SIMPLE THIS

✓ Read lines from simple recommender file

Done! Ultra simple version:

- Single `recommend()` function
- Takes purchase list as input
- Returns top 2 matching products by category + rating
- No classes, just core logic

Saved at ultra_simple_recommender.py

< > ⌂ ⌂

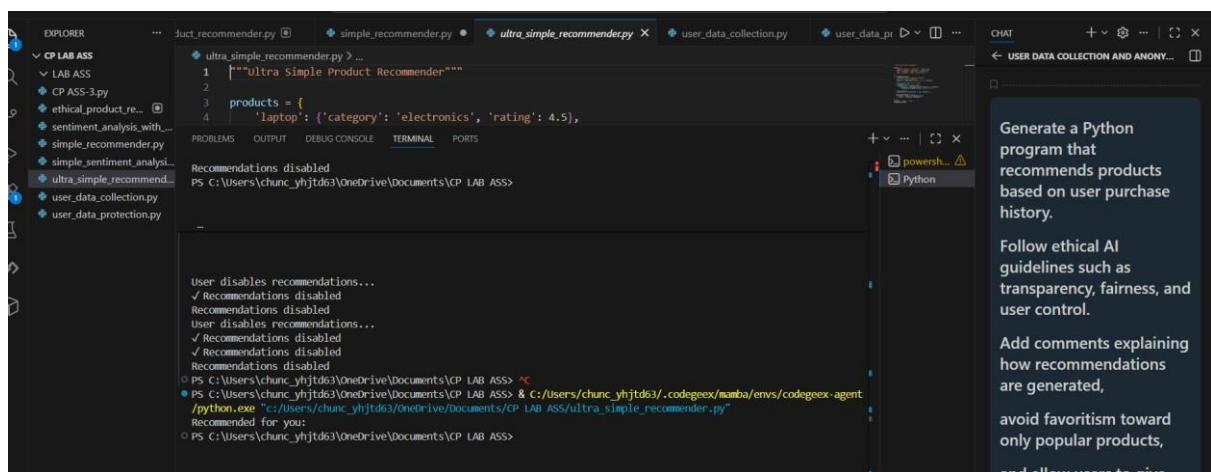
> 1 file changed +341 -1 Keep Undo

PS | + ultra_simple_recommender.py

Describe what to build next

Agent Auto

OUTPUT:



```
File Edit Selection View ... < > CP LAB ASS
EXPLORER ... duct_recommender.py simple_recommender.py ultra_simple_recommender.py user_data_collection.py user_data_pro ...
ultra_simple_recommender.py ...
1 """Ultra Simple Product Recommender"""
2
3 products = {
4     'laptop': {'category': 'electronics', 'rating': 4.5},
5     'mouse': {'category': 'electronics', 'rating': 4.2},
6     'chair': {'category': 'furniture', 'rating': 4.6},
7     'book': {'category': 'books', 'rating': 4.7},
8 }
9
10 # Test
11 def recommend(purchases):
12     """Recommend based on categories purchased"""
13     # Get user's favorite categories
14     categories = [products[p]['category'] for p in purchases]
15
16     # Find similar products not yet bought
17     recommendations = []
18     for product, info in products.items():
19         if product not in purchases and info['category'] in categories:
20             recommendations.append((product, info['rating']))
21
22     # Sort by rating
23     recommendations.sort(key=lambda x: x[1], reverse=True)
24
25     print("Recommended for you:")
26     for product, rating in recommendations[:2]:
27         print(f" {product} ({rating}★)")
28
29 # Test
30 purchases = ['laptop', 'mouse']
31 recommend(purchases)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Recommendations disabled

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS>

User disables recommendations...
 ✓ Recommendations disabled
 Recommendations disabled
 User disables recommendations...
 ✓ Recommendations disabled
 ✓ Recommendations disabled
 Recommendations disabled
 Recommendations disabled

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> & c:/Users/chunc_yhjtd63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjtd63/OneDrive/Documents/CP LAB ASS/ultra_simple_recommender.py"

Recommended for you:

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS>

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

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.

OUTPUT:

The screenshot shows a code editor interface with several files open in the Explorer pane. One file, `simple_secure_logging.py`, is active in the editor. The code contains a function for secure logging that redacts sensitive information like email addresses and passwords. A Copilot-generated documentation sidebar is visible on the right, providing instructions on ethical logging practices.

```

    #!/usr/bin/python
    """
    Simple Secure Logging - No Sensitive Data
    """

    import logging
    import re

    def log_message(message):
        """Logs message to app.log, redacting sensitive data"""
        raw_message = message
        redacted_message = re.sub(r'([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+)', '[REDACTED_EMAIL]', message)
        redacted_password = re.sub(r'[a-zA-Z0-9]{8,}', 'secret123', redacted_message)

        print(f"Raw: {raw_message}")
        print(f"Raw: {redacted_message}")
        print(f"Redacted: User [{REDACTED_EMAIL}] logged in with password secret123")
        print(f"Redacted: User [{REDACTED_EMAIL}] logged in with password secret123")

        with open('app.log', 'a') as f:
            f.write(redacted_message + '\n')

    if __name__ == '__main__':
        log_message("Testing secure logging...")

```

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 simple machine learning model in Python.

Add a README-style or inline documentation explaining how to use the model responsibly,

including explainability, accuracy limitations, fairness considerations,

and the importance of human oversight.

OUTPUT: