

## ASSIGNMENT 5.4

Name:B.Sanjana

H.T.NO:2303A52306

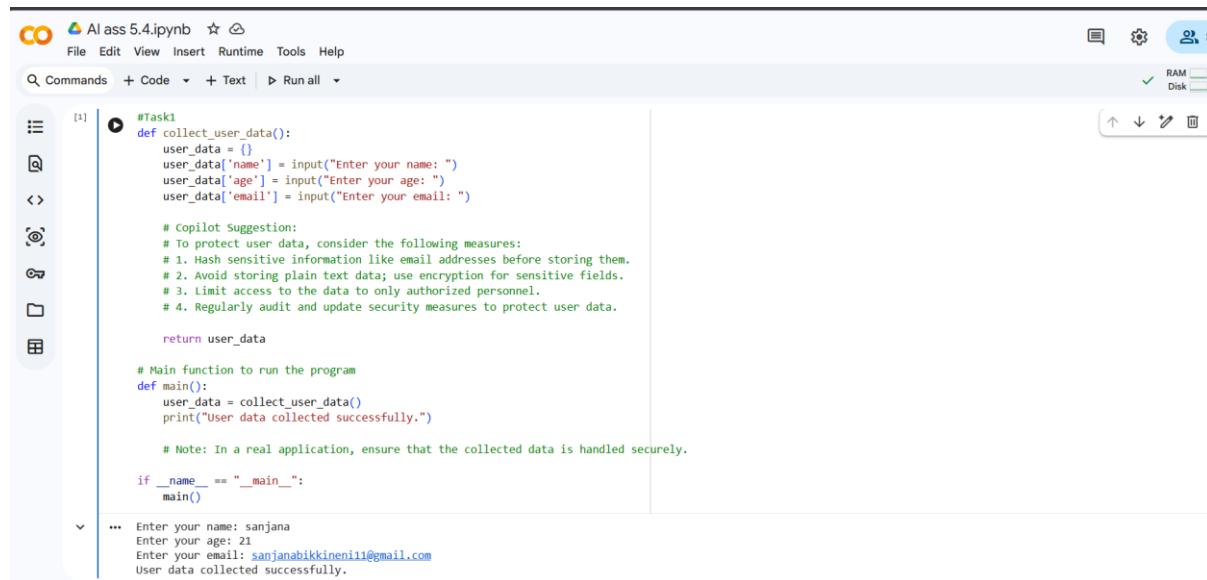
Batch:43

### Task 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.

#### Expected Output #1:

A script with inline Copilot-suggested code and comments explaining how to safeguard or anonymize user information (e.g., hashing emails, not storing data unencrypted).



```
#Task1
def collect_user_data():
    user_data = {}
    user_data['name'] = input("Enter your name: ")
    user_data['age'] = input("Enter your age: ")
    user_data['email'] = input("Enter your email: ")

    # Copilot Suggestion:
    # To protect user data, consider the following measures:
    # 1. Hash sensitive information like email addresses before storing them.
    # 2. Avoid storing plain text data; use encryption for sensitive fields.
    # 3. Limit access to the data to only authorized personnel.
    # 4. Regularly audit and update security measures to protect user data.

    return user_data

# Main function to run the program
def main():
    user_data = collect_user_data()
    print("User data collected successfully.")

    # Note: In a real application, ensure that the collected data is handled securely.

if __name__ == "__main__":
    main()

... Enter your name: sanjana
Enter your age: 21
Enter your email: sanjanabikkinen11@gmail.com
User data collected successfully.
```

### Task 2:

Ask Copilot to generate a Python function for sentiment analysis.

Then prompt Copilot to identify and handle potential biases in the

data.

## Expected Output #2:

Copilot-generated code with additions or comments addressing bias mitigation strategies (e.g., balancing dataset, removing offensive terms).

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains a block of text starting with '#Task 02' followed by a multi-line string of text. This string is a Copilot Note titled 'Copilot Notes on Bias & Ethics' which provides several recommendations for mitigating bias in sentiment analysis. The second cell contains a Python function named 'sentiment\_analysis' that takes a 'text' parameter. It uses a list of offensive terms to replace them with an empty string. It then calculates positive and negative scores based on lists of neutral words ('positive\_words' and 'negative\_words'). Finally, it handles bias by returning 'Neutral' if the scores are close, 'Positive' if positive is higher, and 'Negative' if negative is higher.

```
#Task 02
def sentiment_analysis(text):
    """
    Copilot Notes on Bias & Ethics:
    - Sentiment models can inherit bias from training data
    - Ensure balanced datasets across gender, culture, and language
    - Remove offensive or discriminatory terms before analysis
    - Avoid over-weighting extreme opinions
    - Normalize text to reduce language-based bias
    """
    text = text.lower().strip()
    # Remove offensive or harmful terms that may introduce bias
    offensive_terms = ["stupid", "idiot", "dumb", "hate", "racist"]
    for term in offensive_terms:
        text = text.replace(term, "")
    # Use neutral word lists instead of emotionally charged stereotypes
    positive_words = ["good", "happy", "excellent", "love", "great"]
    negative_words = ["bad", "sad", "terrible", "angry", "worst"]
    positive_score = sum(word in text for word in positive_words)
    negative_score = sum(word in text for word in negative_words)
    # Bias handling:
    # If both scores are close, return Neutral to avoid forced polarity
    if abs(positive_score - negative_score) <= 1:
        return "Neutral"
    elif positive_score > negative_score:
        return "Positive"
    else:
        return "Negative"
```

The screenshot shows the same Jupyter Notebook interface after the user has added code to demonstrate the function's usage. A new cell at the bottom contains the following code: 'if \_\_name\_\_ == "\_\_main\_\_": sample\_text = "I love this product, it's excellent!" result = sentiment\_analysis(sample\_text) print(f"Sentiment: {result}")'. When run, the cell outputs 'Sentiment: Positive'.

```
- Avoid over-weighting extreme opinions
- Normalize text to reduce language-based bias
"""
text = text.lower().strip()
# Remove offensive or harmful terms that may introduce bias
offensive_terms = ["stupid", "idiot", "dumb", "hate", "racist"]
for term in offensive_terms:
    text = text.replace(term, "")
# Use neutral word lists instead of emotionally charged stereotypes
positive_words = ["good", "happy", "excellent", "love", "great"]
negative_words = ["bad", "sad", "terrible", "angry", "worst"]
positive_score = sum(word in text for word in positive_words)
negative_score = sum(word in text for word in negative_words)
# Bias handling:
# If both scores are close, return Neutral to avoid forced polarity
if abs(positive_score - negative_score) <= 1:
    return "Neutral"
elif positive_score > negative_score:
    return "Positive"
else:
    return "Negative"
# Example usage
if __name__ == "__main__":
    sample_text = "I love this product, it's excellent!"
    result = sentiment_analysis(sample_text)
    print(f"Sentiment: {result}")
```

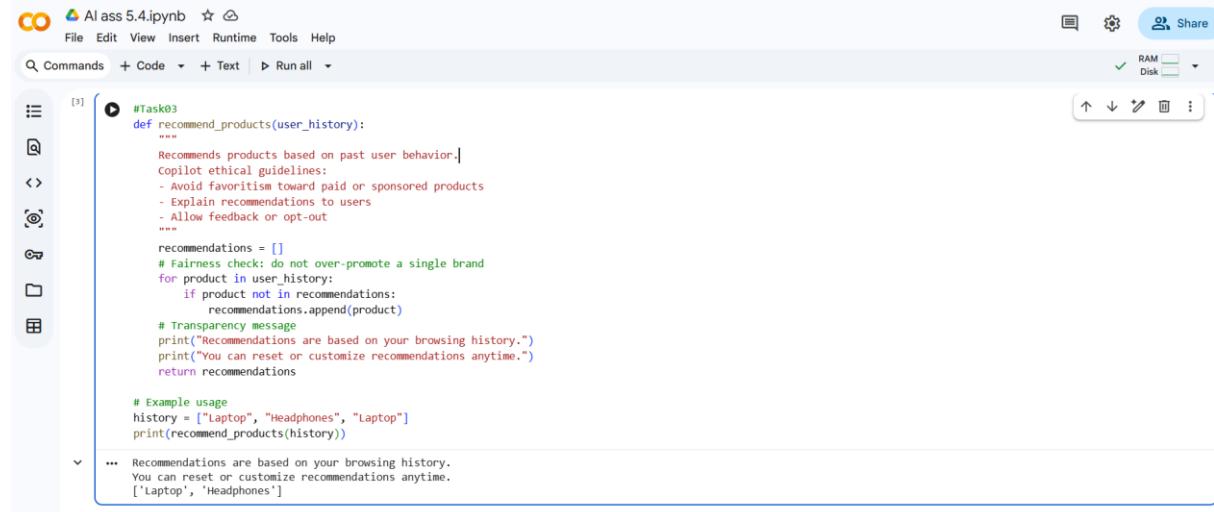
## Task 3:

Use Copilot to write a Python program that recommends products based on user history. Ask it to follow ethical guidelines

like transparency and fairness.

### Expected Output #3:

Copilot suggestions that include explanations, fairness checks (e.g., avoiding favoritism), and user feedback options in the code.



The screenshot shows a Jupyter Notebook cell with the following Python code:

```
#Task03
def recommend_products(user_history):
    """
    Recommends products based on past user behavior.
    Copilot ethical guidelines:
    - Avoid favoritism toward paid or sponsored products
    - Explain recommendations to users
    - Allow feedback or opt-out
    """
    recommendations = []
    # Fairness check: do not over-promote a single brand
    for product in user_history:
        if product not in recommendations:
            recommendations.append(product)
    # Transparency message
    print("Recommendations are based on your browsing history.")
    print("You can reset or customize recommendations anytime.")
    return recommendations

# Example usage
history = ["Laptop", "Headphones", "Laptop"]
print(recommend_products(history))
```

... Recommendations are based on your browsing history.  
You can reset or customize recommendations anytime.  
['Laptop', 'Headphones']

### Task 4:

Prompt Copilot to generate logging functionality in a Python web application. Then, ask it to ensure the logs do not record sensitive information.

### Expected Output #4:

Logging code that avoids saving personal identifiers (e.g., passwords, emails), and includes comments about ethical logging practices.

```

import logging
# Configure logging
logging.basicConfig(
    filename="app.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s"
)
def login(username, password):
    # Copilot warning:
    # NEVER log passwords, emails, or personal identifiers
    logging.info("Login attempt for user") # Do not log username
    if password == "admin123":
        logging.info("Login successful")
        return True
    else:
        logging.warning("Login failed")
        return False
# Example usage
if __name__ == "__main__":
    username = input("Enter username: ")
    password = input("Enter password: ")

    if login(username, password):
        print("Welcome!")
    else:
        print("Invalid credentials.")

...
Enter username: sanjana
Enter password: admin123
Welcome!

```

## Task 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).

### Expected Output #5:

- Copilot-generated model code with a README or inline documentation suggesting responsible usage, limitations, and fairness considerations

```

"""
Simple ML Model with Responsible Usage Notes
This program predicts Pass/Fail based on study hours
using a very small dataset.

RESPONSIBLE USAGE:
- This is only for learning purposes.
- The dataset is very small, so accuracy is low.
- DO NOT use this model for real student evaluation.
- The model uses only study hours, so many real factors are ignored.
- Always inform users this is an automated prediction."""
from sklearn.tree import DecisionTreeClassifier
# Training data: study hours
X = [[1], [2], [3], [4], [5], [6]]
# Labels: 0 = Fail, 1 = Pass
y = [0, 0, 0, 1, 1]
# Create and train model
model = DecisionTreeClassifier()
model.fit(X, y)
# Take user input
hours = int(input("Enter study hours: "))
# Predict
prediction = model.predict([[hours]])
# Output result
if prediction[0] == 1:
    print("Prediction: Pass")
else:
    print("Prediction: Fail")

...
Enter study hours: 4
Prediction: Pass

```