Assignment – 5.4

S.sathwik

2403A51273

24BTCAICSB12

Task Descrip on #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.

Output #1:

```python
import hashlib

def collect_user_data():
    # Politely prompt the user for their information
    print("Hi there! We'd like to get to know you a little better.")
    name = input("May I have your name? ")
    age = input("How old are you? ")
    email = input("What is your email address? ")
    # Store the collected information
    user_data = {"name": name,"age": age,"email": email}
    return user_data

def anonymize_user_data(user_data):
    # To respect your privacy, we can take steps to protect or anonymize your data:
    # - Hashing the email means we keep your identity private and can't reverse it to find your actual email.
    # - We avoid keeping your name unless it's necessary.
    # - Sensitive data should never be stored in plain text or without protection.
    hashed_email = hashlib.sha256(user_data["email"].encode()).hexdigest()
    anonymized_data = {
        "name": "Anonymous",  # Replace name with a generic label for anonymity
        "age": user_data["age"],
        "email_hash": hashed_email  # Store only the hashed value of the email
    }
    return anonymized_data

if __name__ == "__main__":
    print("Welcome! We value your privacy.")
    data = collect_user_data()
    print("\nThank you! Here's what we collected:")
    print(data)
    print("\nTo better protect your privacy, here's how we can anonymize your information:")
    anonymized = anonymize_user_data(data)
    print(anonymized)
    print("\nNote: When saving any sensitive information, always consider encrypting it and restricting access.")
```

Comments by Copilot (inline):

- Use hashing (e.g., SHA-256) to anonymize sensi ve informa on like emails.
- Avoid storing user names or emails in plain text.
- Use "REDACTED" or pseudonyms for names if full anonymiza on is required.
- Always use encryp on and access controls when storing sensi ve data persistently

Task Descrip on #2:

• Ask Copilot to generate a Python func on for sen ment analysis. Then prompt Copilot to iden fy and handle poten al biases in the data.

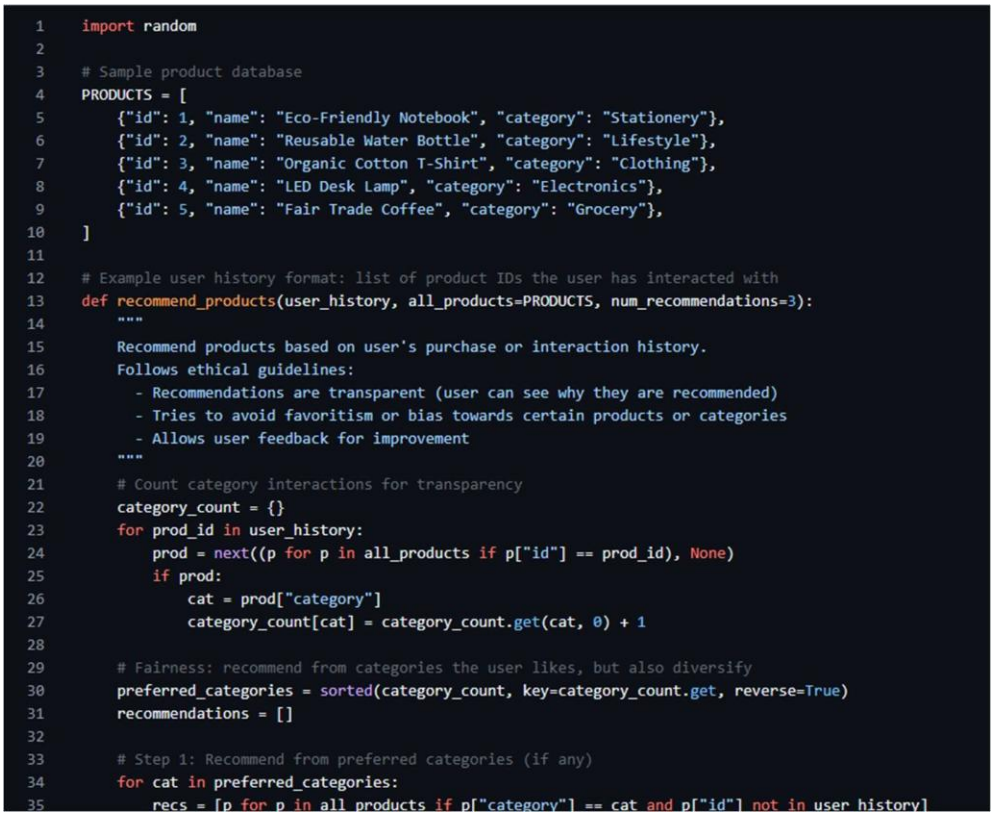
Output #2:

```python
from textblob import TextBlob

def analyze_sentiment(text):
    """
    Analyze the sentiment of input text using TextBlob.
    Returns polarity and subjectivity scores.
    """
    # Perform sentiment analysis
    blob = TextBlob(text)
    sentiment = blob.sentiment
    return {
        "polarity": sentiment.polarity,     # [-1.0, 1.0], negative to positive sentiment
        "subjectivity": sentiment.subjectivity  # [0.0, 1.0], objective to subjective
    }

# --- Bias Mitigation Strategies ---
# 1. Data Balancing: Ensure your dataset is balanced with respect to sentiment classes (positive, neutral, negative).
#    Imbalanced datasets can cause the model to be biased toward the majority class.
# 2. Remove Offensive or Sensitive Terms: Filter out or flag text containing hate speech, slurs, or offensive language to avoid perpetuating bias.
# 3. Language & Demographic Diversity: Include diverse language styles, dialects, and demographics to avoid overfitting to a specific group.
# 4. Regular Auditing: Periodically audit model predictions and outputs for signs of bias, and retrain with corrected datasets if necessary.
# 5. Manual Review: For critical applications, use human-in-the-loop to review flagged or ambiguous sentiment predictions.

# Example usage:
if __name__ == "__main__":
    # Example input
    user_text = input("Enter a sentence to analyze its sentiment: ")
    result = analyze_sentiment(user_text)
    print("Sentiment analysis result:", result)
    print("\nNote: For best results, ensure your training and test data are diverse, balanced, and free from offensive content.")
```

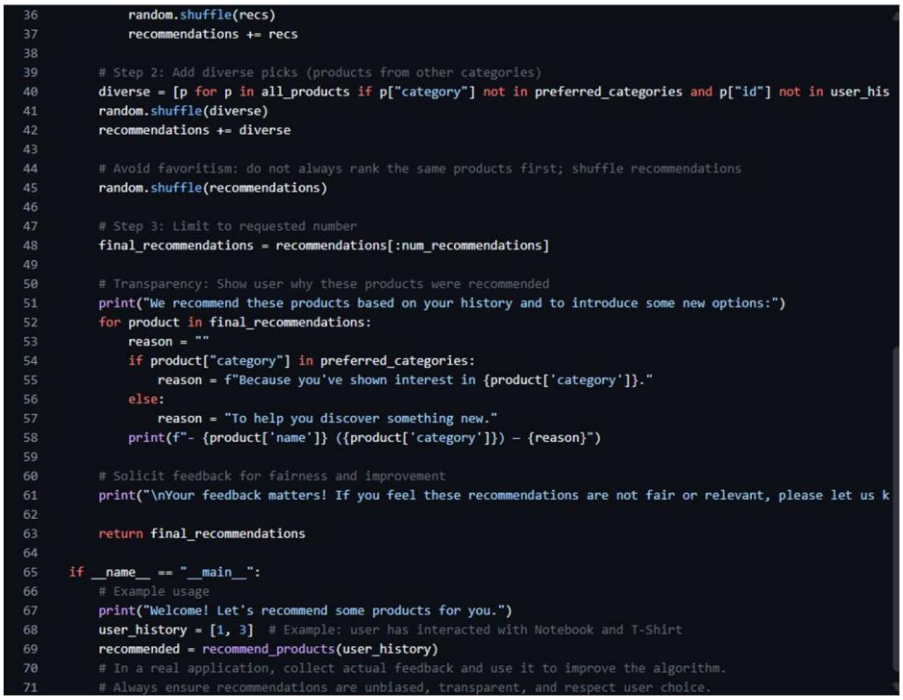
Task Descrip on #3:

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

Output #3:



```python
import random

# Sample product database
PRODUCTS = [
    {"id": 1, "name": "Eco-Friendly Notebook", "category": "Stationery"},
    {"id": 2, "name": "Reusable Water Bottle", "category": "Lifestyle"},
    {"id": 3, "name": "Organic Cotton T-Shirt", "category": "Clothing"},
    {"id": 4, "name": "LED Desk Lamp", "category": "Electronics"},
    {"id": 5, "name": "Fair Trade Coffee", "category": "Grocery"},
]

# Example user history format: list of product IDs the user has interacted with
def recommend_products(user_history, all_products=PRODUCTS, num_recommendations=3):
    """
    Recommend products based on user's purchase or interaction history.
    Follows ethical guidelines:
      - Recommendations are transparent (user can see why they are recommended)
      - Tries to avoid favoritism or bias towards certain products or categories
      - Allows user feedback for improvement
    """
    # Count category interactions for transparency
    category_count = {}
    for prod_id in user_history:
        prod = next((p for p in all_products if p["id"] == prod_id), None)
        if prod:
            cat = prod["category"]
            category_count[cat] = category_count.get(cat, 0) + 1

    # Fairness: recommend from categories the user likes, but also diversify
    preferred_categories = sorted(category_count, key=category_count.get, reverse=True)
    recommendations = []

    # Step 1: Recommend from preferred categories (if any)
    for cat in preferred_categories:
        recs = [p for p in all_products if p["category"] == cat and p["id"] not in user_history]
```



```
        random.shuffle(recs)
        recommendations += recs

    # Step 2: Add diverse picks (products from other categories)
    diverse = [p for p in all_products if p["category"] not in preferred_categories and p["id"] not in user_his
    random.shuffle(diverse)
    recommendations += diverse

    # Avoid favoritism: do not always rank the same products first; shuffle recommendations
    random.shuffle(recommendations)

    # Step 3: Limit to requested number
    final_recommendations = recommendations[:num_recommendations]

    # Transparency: Show user why these products were recommended
    print("We recommend these products based on your history and to introduce some new options:")
    for product in final_recommendations:
        reason = ""
        if product["category"] in preferred_categories:
            reason = f"Because you've shown interest in {product['category']}."
        else:
            reason = "To help you discover something new."
        print(f"- {product['name']} ({product['category']}) – {reason}")

    # Solicit feedback for fairness and improvement
    print("\nYour feedback matters! If you feel these recommendations are not fair or relevant, please let us k

    return final_recommendations

if __name__ == "__main__":
    # Example usage
    print("Welcome! Let's recommend some products for you.")
    user_history = [1, 3]  # Example: user has interacted with Notebook and T-Shirt
    recommended = recommend_products(user_history)
    # In a real application, collect actual feedback and use it to improve the algorithm.
    # Always ensure recommendations are unbiased, transparent, and respect user choice.
```

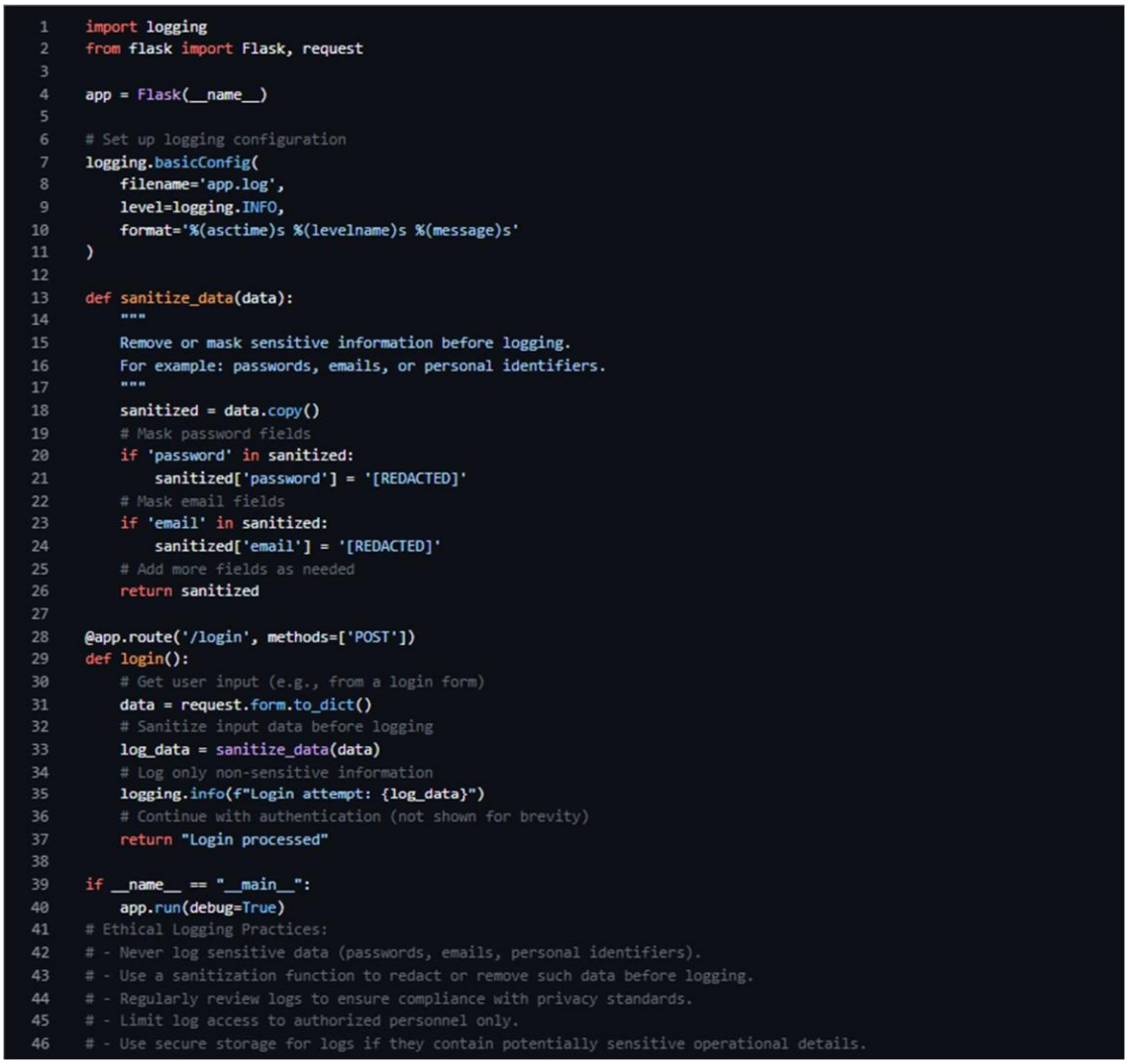
Ethical Guidelines in Code:

- Recommenda ons are explained to the user for transparency.
- A empts to balance between user's interests and diversity to avoid bias or favori sm.
- Recommenda ons are shuffled to avoid always promo ng the same products.
- Users are invited to provide feedback on fairness and relevance.

Task Descrip on #4:

- Prompt Copilot to generate logging func onality in a Python web applica on. Then, ask it to ensure the logs do not record sensi ve informa on.

Output #4:



```python
import logging
from flask import Flask, request

app = Flask(__name__)

# Set up logging configuration
logging.basicConfig(
    filename='app.log',
    level=logging.INFO,
    format='%(asctime)s %(levelname)s %(message)s'
)

def sanitize_data(data):
    """
    Remove or mask sensitive information before logging.
    For example: passwords, emails, or personal identifiers.
    """
    sanitized = data.copy()
    # Mask password fields
    if 'password' in sanitized:
        sanitized['password'] = '[REDACTED]'
    # Mask email fields
    if 'email' in sanitized:
        sanitized['email'] = '[REDACTED]'
    # Add more fields as needed
    return sanitized

@app.route('/login', methods=['POST'])
def login():
    # Get user input (e.g., from a login form)
    data = request.form.to_dict()
    # Sanitize input data before logging
    log_data = sanitize_data(data)
    # Log only non-sensitive information
    logging.info(f"Login attempt: {log_data}")
    # Continue with authentication (not shown for brevity)
    return "Login processed"

if __name__ == "__main__":
    app.run(debug=True)
# Ethical Logging Practices:
# - Never log sensitive data (passwords, emails, personal identifiers).
# - Use a sanitization function to redact or remove such data before logging.
# - Regularly review logs to ensure compliance with privacy standards.
# - Limit log access to authorized personnel only.
# - Use secure storage for logs if they contain potentially sensitive operational details.
```

Key points:

- Sensi ve fields (like 'password' and 'email') are redacted before logging.
- Comments explain why and how to avoid logging personal iden fiers.
- Ethical logging prac ces are summarized at the end of the file.

Task Descrip on #5:

• Ask Copilot to generate a machine learning model. Then, prompt it to add documenta on on how to use the model responsibly (e.g., explainability, accuracy limits).

Output #5:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

def load_data(path):
    """
    Load data from a CSV file.
    """
    return pd.read_csv(path)

def train_model(X, y):
    """
    Train a simple logistic regression model.
    """
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = LogisticRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("Classification Report:\n", classification_report(y_test, y_pred))
    return model

def predict(model, X_new):
    """
    Predict using the trained model.
    """
    return model.predict(X_new)

if __name__ == "__main__":
    # Example usage:
    # Assume data.csv has columns: "feature1", "feature2", ..., "label"
    data = load_data("data.csv")
    X = data.drop("label", axis=1)
    y = data["label"]
    model = train_model(X, y)
    # To predict on new data:
    # predictions = predict(model, X_new)
    print("Model training complete. Refer to README for responsible usage guidelines.")
```