

# AI Assisted Coding

## Assignment (5.4)

---

2403A51271(M.Ziauddin)

Batch-12(CSE)

---

### 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.

### 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)

### Github copilot Code

```
20 lines - 446 bytes

1  def collect_user_data():
2      print("Please enter the following information:")
3      name = input("Name: ")
4      age = input("Age: ")
5      email = input("Email: ")
6
7      user_data = {
8          "name": name,
9          "age": age,
10         "email": email
11     }
12
13     print("\nCollected User Data:")
14     for key, value in user_data.items():
15         print(f"{key.capitalize()}: {value}")
16
17     return user_data
18
19 if __name__ == "__main__":
20     collect_user_data()
```

Copilot code with comments on how to nonymize or protect this data.:

Python

 Copy

```
def collect_user_data():
    # Prompt user for personal information

    print("Please enter the following information:")
    name = input("Name: ")      # Consider hashing or pseudonymizing this if
    age = input("Age: ")        # Avoid collecting if not necessary
    email = input("Email: ")    # Mask or encrypt before storing/transmittin

    # Store user data in a dictionary

    user_data = {
        "name": name,
        "age": age,
        "email": email
    }

    # Display collected data (avoid this in production if sensitive)

    print("\nCollected User Data:")
    for key, value in user_data.items():
        print(f"{key.capitalize()}: {value}")

    # In a real-world scenario:

    # - Use encryption for storage (e.g., AES)
```

```
    # - Avoid printing sensitive data to console/logs
    # - Apply access controls and audit logging
    # - Consider anonymizing or pseudonymizing data for analytics

    return user_data

if __name__ == "__main__":
    collect_user_data()
```

## Task Description 0-2:

- Ask Copilot to generate a Python function for sentiment analysis. Then prompt Copilot to

identify and handle potential biases in the data—

Copilot code:

```
Python Copy

from textblob import TextBlob

def analyze_sentiment(text):
    """
    Analyzes the sentiment of the given text using TextBlob.
    Returns polarity and sentiment classification.
    """
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity # Range: [-1.0, 1.0]

    if polarity > 0:
        sentiment = "Positive"
    elif polarity < 0:
        sentiment = "Negative"
    else:
        sentiment = "Neutral"

    return {
        "text": text,
        "polarity": polarity,
        "sentiment": sentiment
    }

# Example usage

if __name__ == "__main__":
    sample_text = input("Enter text to analyze: ")
    result = analyze_sentiment(sample_text)
    print(f"\nSentiment: {result['sentiment']}")
    print(f"Polarity Score: {result['polarity']}")
```

identify and handle potential biases in the data :

## ✓ Updated Code with Bias Awareness

Python

Copy

```
from textblob import TextBlob

def analyze_sentiment(text):
    """
    Analyzes the sentiment of the given text using TextBlob.
    Returns polarity and sentiment classification.

    ⚠ Bias Considerations:
    - TextBlob uses a predefined lexicon that may not reflect cultural or contextual nuances.
    - Slang, sarcasm, or dialects may be misclassified.
    - Sentiment scores may skew toward certain writing styles or topics.
    """

    blob = TextBlob(text)
    polarity = blob.sentiment.polarity # Range: [-1.0, 1.0]

    # Basic sentiment classification
    if polarity > 0:
        sentiment = "Positive"
    elif polarity < 0:
        sentiment = "Negative"
    else:
        sentiment = "Neutral"

    return {
        "text": text,
        "polarity": polarity,
        "sentiment": sentiment
    }

# Example usage
if __name__ == "__main__":
    sample_text = input("Enter text to analyze: ")

    # ⚠ Bias Handling Tip:
    # Consider preprocessing input to normalize slang, emojis, or abbreviations.
    # You could also log or flag low-confidence or ambiguous results for review.

    result = analyze_sentiment(sample_text)
    print(f"\nSentiment: {result['sentiment']}")
    print(f"Polarity Score: {result['polarity']}")
```

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.

Expected Output #3:

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

## Copilot Code:

```

60 lines · 2 KB
1  import random
2
3  # Example product database
4  PRODUCTS = [
5      {"id": 1, "name": "Wireless Mouse", "category": "Electronics"},
6      {"id": 2, "name": "Yoga Mat", "category": "Fitness"},
7      {"id": 3, "name": "Water Bottle", "category": "Fitness"},
8      {"id": 4, "name": "Bluetooth Headphones", "category": "Electronics"},
9      {"id": 5, "name": "Notebook", "category": "Stationery"},
10     {"id": 6, "name": "Ballpoint Pen", "category": "Stationery"},
11 ]
12
13 def recommend_products(user_history):
14     """
15     Recommend products based on user history.
16     Ethical Guidelines:
17         - Transparency: Explain why recommendations are made.
18         - Fairness: Avoid biased or discriminatory suggestions.
19     """
20     # Gather categories from user history
21     categories = [product["category"] for product in user_history]
22     if not categories:
23         print("No user history available. Showing popular products:")
24         recommendations = random.sample(PRODUCTS, k=min(3, len(PRODUCTS)))
25         explanation = "Recommended popular products as no user history was found."
26     else:
27         # Recommend products from the user's preferred categories
28         preferred_categories = set(categories)
29         recommendations = [
30             product for product in PRODUCTS

```

```
60 lines · 2 KB

30     product for product in PRODUCTS
31     if product["category"] in preferred_categories and product not in user_history
32     ]
33     # If not enough recommendations, randomly fill
34     if len(recommendations) < 3:
35         additional = [
36             product for product in PRODUCTS
37             if product not in recommendations and product not in user_history
38         ]
39         recommendations += random.sample(additional, k=min(3 - len(recommendations), len(addi
40     recommendations = recommendations[:3])
41     explanation = (
42         "Recommended based on your interest in: "
43         + ", ".join(preferred_categories)
44     )
45
46     # Transparency: show the reason for recommendations
47     print("Ethical Notice: Recommendations are generated based on your previous product interacti
48     print(f"Explanation: {explanation}\n")
49     print("Recommended Products:")
50     for product in recommendations:
51         print(f"- {product['name']} ({product['category']})")
52
53 if __name__ == "__main__":
54     # Example usage
55     # Simulate user history
56     user_history = [
57         {"id": 2, "name": "Yoga Mat", "category": "Fitness"},
58         {"id": 3, "name": "Water Bottle", "category": "Fitness"}
59     ]
60     recommend_products(user_history)

product_recommender.py
```

#### 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.

Expected Output #4:

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



48 lines · 1 KB

```
1  import logging
2  from flask import Flask, request
3
4  app = Flask(__name__)
5
6  # Configure logging
7  logging.basicConfig(
8      filename='app.log',
9      level=logging.INFO,
10     format='%(asctime)s %(levelname)s %(message)s'
11 )
12
13 def filter_sensitive_data(data):
14     """
15     Redact sensitive fields from data dictionaries before logging.
16     Add more field names if needed.
17     """
18     SENSITIVE_KEYS = {'password', 'secret', 'token', 'email'}
19     redacted = {}
20     for key, value in data.items():
21         if key.lower() in SENSITIVE_KEYS:
22             redacted[key] = '[REDACTED]'
23         else:
24             redacted[key] = value
25     return redacted
26
27 @app.before_request
28 def log_request_info():
29     # Copy request form data and filter sensitive information
30     if request.method == 'POST':
31         log_data = filter_sensitive_data(request.form.to_dict())
32         logging.info(f"POST {request.path} - Data: {log_data}")
33     else:
34         logging.info(f"{request.method} {request.path}")
35
36 @app.route('/login', methods=['POST'])
37 def login():
38     # Example login endpoint
39     # Do NOT log sensitive authentication data
40     return "Login attempt logged (without sensitive info)."
41
42 @app.route('/')
43 def home():
44     return "Welcome to the home page."
45
46 if __name__ == "__main__":
47     print("Ethical Notice: Logging is configured to avoid recording sensitive informati")
48     app.run(debug=True)
```

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).

Expected Output #5:

- Copilot-generated model code with a README or inline documentation suggesting responsible usage, limitations, and fairness considerations [Copilot Code](#):

```

1  import pandas as pd
2  from sklearn.model_selection import train_test_split
3  from sklearn.ensemble import RandomForestClassifier
4  import joblib
5
6  def train_model(data_path, model_path="trained_model.pkl"):
7      """
8          Trains a RandomForestClassifier on the provided dataset.
9
10         Parameters:
11             data_path (str): Path to the CSV file containing the data.
12             model_path (str): Path to save the trained model.
13
14         The CSV should have the target variable in the last column.
15         """
16         # Load data
17         data = pd.read_csv(data_path)
18         X = data.iloc[:, :-1]
19         y = data.iloc[:, -1]
20
21         # Split data
22         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24         # Train model
25         clf = RandomForestClassifier(n_estimators=100, random_state=42)
26         clf.fit(X_train, y_train)
27
28         # Save model
29         joblib.dump(clf, model_path)
30         print(f"Model trained and saved to {model_path}")
31
32         # Evaluate model
33         accuracy = clf.score(X_test, y_test)
34         print(f"Test Accuracy: {accuracy:.2f}")
35
36     if __name__ == "__main__":
37         # Example usage: replace 'your_data.csv' with your actual data file
38         train_model('your_data.csv')

```