AI ASSISTED CODING:

ASSIGNMENT(5.4)

2403A51245

Shashikanth G

Batch-11

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.

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:

```python
# This script collects user data: name, age, and email.
# Copilot inline comments guide how to anonymize and protect user information.

import hashlib

def hash_email(email):
    # Hash the email using SHA-256 to anonymize it before storage.
    # Never store raw emails in logs or databases if privacy is required.
    return hashlib.sha256(email.encode()).hexdigest()

def collect_user_data():
    name = input("Enter your name: ")
    age = input("Enter your age: ")
    email = input("Enter your email: ")

    # Protect user data:
    # - Avoid storing plain text name/email if possible.
    # - Use hashed email for identification, not for direct contact.
    # - Only collect necessary fields (minimize exposure).
    # - Encrypt data at rest if storing in files/databases.
    anonymized_email = hash_email(email)

    user_data = {
        "name": name,  # For full anonymity, do not store real names.
        "age": age,    # Consider storing age ranges instead of exact value.
        "email_hash": anonymized_email  # Store only the hashed email.
    }

    print("Anonymized user data:", user_data)
    # If saving to disk, use file encryption libraries (e.g., cryptography, Fernet).
    # Always comply with privacy laws (e.g., GDPR) when handling personal info.

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

Copilot protection notes (inline):

- Hash emails before storage to anonymize.
- Encrypt files/databases containing user data.
- Avoid storing raw names/emails.
- When possible, store minimum necessary info (e.g., age range instead of exact age).
- Always follow privacy best practices and legal requirements.

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. Expected Output #2:

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

Copilot code:

```python
# This script provides a sentiment analysis function.
# Copilot inline comments explain ways to identify and handle bias in the data.
from textblob import TextBlob
def analyze_sentiment(text):
    # Analyzes the sentiment polarity of the input text.
    # Returns 'positive', 'neutral', or 'negative'.
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    if polarity > 0.1:
        return "positive"
    elif polarity < -0.1:
        return "negative"
    else:
        return "neutral"
# Bias mitigation strategies:
# - Balance the dataset: Ensure training data contains equal samples of positive, neutral, and negative sentiments.
# - Remove or flag offensive terms: Preprocess input to detect and mask hate speech or slurs.
# - Audit for demographic bias: Check if model accuracy varies for different groups and retrain if needed.
# - Regularly review and update labeled data to reflect evolving language and social norms.
def preprocess_text(text):
    # Example: Remove offensive terms using a predefined list.
    offensive_terms = ['term1', 'term2']  # Populate with actual terms
    for term in offensive_terms:
        text = text.replace(term, '[REDACTED]')
    return text
def safe_analyze_sentiment(text):
    # Preprocess to mitigate bias and offensive content.
    clean_text = preprocess_text(text)
    return analyze_sentiment(clean_text)

# For best results, retrain models on balanced and representative data.
# Use explainable AI tools to identify sources of bias in predictions.

if __name__ == "__main__":
    user_input = input("Enter text for sentiment analysis: ")
    result = safe_analyze_sentiment(user_input)
    print("Sentiment:", result)
```

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:

```python
# Short ethical product recommender with fairness, transparency, and user feedback.

from collections import Counter

def recommend_products(user_history, all_products):
    # Recommend top product plus one new for fairness.
    top = Counter(user_history).most_common(1)
    new_items = [p for p in all_products if p not in user_history]
    result = [top[0][0]] if top else []
    if new_items:
        result.append(new_items[0])
    # Explain choices
    reasons = [f"'{result[0]}' matches your history" if result else "",
               f"'{result[1]}' adds diversity" if len(result) > 1 else ""]
    return result, reasons

def feedback(recommendations):
    print("Recommended:", recommendations)
    resp = input("Feedback (yes/no/suggest): ")
    if resp == "suggest":
        alt = input("Suggest a product: ")
        print(f"Thanks! '{alt}' noted.")

if __name__ == "__main__":
    history = ['A', 'B', 'A']
    products = ['A', 'B', 'C']
    recs, why = recommend_products(history, products)
    print("\n".join(why))
    feedback(recs)
```

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:

```python
# Logging functionality for a Python web app.
# Copilot inline comments explain how to avoid logging sensitive info (e.g., passwords, emails).

import logging

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

def log_event(event, details):
    """
    Log an event with details.
    Ethical logging practice:
    - NEVER log sensitive info (passwords, emails, personal data).
    - Scrub details before logging.
    """
    # Remove sensitive keys if present
    sensitive_keys = ['password', 'email', 'token']
    filtered_details = {k: v for k, v in details.items() if k not in sensitive_keys}
    logging.info(f"{event}: {filtered_details}")

# Example usage
def user_login(username, password):
    # NEVER log raw passwords or emails.
    log_event('user_login_attempt', {'username': username})  # Only log non-sensitive info

def password_reset(email):
    # Do NOT log the email address.
    log_event('password_reset_attempt', {})  # Log only the event, not the identifier

# Ethical notes:
# - Avoid logging personal identifiers.
# - Regularly review logs for accidental sensitive data.
# - Use log rotation and secure storage to prevent leaks.
# - Comply with privacy regulations (e.g., GDPR).
```

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:

```python
# Simple machine learning model with responsible usage documentation.
# Copilot inline comments explain explainability, accuracy limits, and fairness.

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

def train_model():
    # Load dataset (Iris for demonstration; review for bias before production use)
    data = load_iris()
    X_train, X_test, y_train, y_test = train_test_split(
        data.data, data.target, test_size=0.2, random_state=42)
    model = RandomForestClassifier()
    model.fit(X_train, y_train)
    accuracy = model.score(X_test, y_test)
    print(f"Test Accuracy: {accuracy:.2f}")
    return model

if __name__ == "__main__":
    model = train_model()
    # Responsible usage notes:
    # - Explainability: Use model.feature_importances_ to understand which features matter most.
    # - Accuracy limits: Always validate accuracy on representative data; don't overstate performance.
    # - Fairness: Check for bias in training data; don't deploy without fairness review.
    # - Never use predictions for critical decisions without human oversight.
    # - Document intended uses and known limitations.
```