AI Assisted Coding

Name: Gopi chand

Roll no: 2403a51237

Task1:

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

Code:

import hashlib

from cryptography.fernet import Fernet

# Generate a key for encryption (in practice, store this securely, not in code)

key = Fernet.generate\_key()

cipher = Fernet(key)

def collect\_user\_data():

name = input("Enter your name: ")

age = input("Enter your age: ")

email = input("Enter your email: ")

# --- Data Protection Suggestions ---

# 1. Do not store 'name' directly if not needed. Use pseudonyms or IDs instead.

# 2. Age is less sensitive but could still be generalized (e.g., age range).

# 3. Email should be anonymized or hashed before storage.

# Hash the email for anonymization (use SHA-256 or similar)

email\_hash = hashlib.sha256(email.encode()).hexdigest()

# Encrypt sensitive fields (e.g., name)

encrypted\_name = cipher.encrypt(name.encode())

user\_data = {

"name\_encrypted": encrypted\_name,

"age": age,

"email\_hash": email\_hash

}

return user\_data

if \_\_name\_\_ == "\_\_main\_\_":

user\_info = collect\_user\_data()

print("User data stored securely:", user\_info)

# --- Safe Storage Tips ---

# - Never log raw sensitive data to console (done here only for demo).

# - Store encryption keys securely (e.g., environment variables, vaults).

# - If data is sent over network, use HTTPS/TLS.

# - Regularly audit and rotate keys/passwords

Output:

Enter your name: Gopi

Enter your age: 25

Enter your email: [gopi@gmail.com](mailto:gopi@gmail.com)

Explanation:

**1. Email Hashing**

email\_hash = hashlib.sha256(email.encode()).hexdigest()

* Converts the email into a fixed-length string that **cannot be reversed**.
* Even if attackers get the database, they cannot recover the original email.
* Useful for anonymization (e.g., to detect duplicates without knowing the real email).

✅ Example:

* Input: gopi@gmail.com
* Output: 6c48f0a5ad8cd30f5c35adbc94a5f93dcd67889c639ad89a649b2f18f8c5a4b2

**2. Name Encryption**

encrypted\_name = cipher.encrypt(name.encode())

* Uses **symmetric encryption** (Fernet, based on AES).
* Unlike hashing, encrypted data **can be decrypted later** (if you have the key).
* Protects sensitive personal info (like names) from being exposed.

✅ Example:

* Input: Gopi
* Output: b'gAAAAABnZ2V7...=='
* Only someone with the encryption key can decrypt it back to "Gopi".

**3. Safe Age Handling**

"age": age

* Age isn’t always highly sensitive, but storing it directly could still expose identity.
* To improve privacy, age could be stored as **ranges** instead of exact values (e.g., 20–30 instead of 25).

**4. Key Management**

key = Fernet.generate\_key()

* The encryption key is what protects your encrypted data.
* In production, this should be stored in:
  + Environment variables
  + Secret managers (AWS Secrets Manager, HashiCorp Vault, etc.)
* **Never hardcode keys** into your source code or GitHub repo.

**5. Storage & Transmission Safety (best practices in comments)**

* Always use **TLS/HTTPS** when transmitting data.
* Regularly rotate encryption keys.
* Don’t log sensitive data in plaintext.
* Encrypt backups and storage databases too.

**🛡️ How it protects your data**

* Even if someone hacks into your database:
  + They **cannot see raw emails** (hashed).
  + They **cannot read names** without the secret key.
  + Ages are less identifiable if generalized.

So this script ensures **an attacker gets only scrambled, useless data** instead of readable personal information.

Task:2

Ask Copilot to generate a Python function for sentiment analysis. Then prompt Copilot to identify and handle potential biases in the data

Code:

from textblob import TextBlob

from collections import Counter

def sentiment\_analysis(text: str, training\_labels=None) -> str:

blob = TextBlob(text)

polarity = blob.sentiment.polarity

# --- Potential Bias Considerations ---

# 1. Class Imbalance:

# If the training data has many "Positive" examples and few "Negative",

# the model may be biased toward predicting "Positive".

# -> Mitigation: balance dataset with oversampling/undersampling.

# 2. Cultural/Language Bias:

# Words can mean different things in different contexts.

# Example: "sick" can mean "ill" (negative) or "awesome" (positive).

# -> Mitigation: train with diverse, representative data.

# 3. Toxic / Identity Bias:

# Models may unfairly assign "Negative" to identity-related words.

# -> Mitigation: apply filters or fairness checks on the vocabulary.

if polarity > 0:

return "Positive"

elif polarity < 0:

return "Negative"

else:

return "Neutral"

def check\_class\_balance(training\_labels: list):

"""

Checks if training data has imbalanced sentiment classes.

Example: too many 'Positive' vs. 'Negative'.

"""

counts = Counter(training\_labels)

print("Class distribution:", counts)

# Example

print(sentiment\_analysis("This is terrible!")) # Negative

check\_class\_balance(["Positive", "Positive", "Negative", "Neutral"])

output:

Negative

Class distribution: Counter({'Positive': 2, 'Negative': 1, 'Neutral': 1})

**Explanation:**

1. **sentiment\_analysis("This is terrible!")**
   * The TextBlob library calculates a negative polarity.
   * Function returns → **"Negative"**
2. **check\_class\_balance([...])**
   * Counts how many labels of each type are in the dataset.
   * Prints → {'Positive': 2, 'Negative': 1, 'Neutral': 1}
   * This shows a **class imbalance** (too many positives compared to negatives)

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.

Code:

from collections import defaultdict

# --- Ethical Product Recommendation System ---

# ⚠️ Ethical Guidelines:

# - Transparency: Make it clear to users how recommendations are generated

# (here, based on what similar users bought).

# - Fairness: Ensure recommendations do not systematically exclude or

# disadvantage certain groups of users.

# - Privacy: Do not expose sensitive user data when generating recommendations.

# - Avoid Manipulation: Recommender systems should not push harmful or

# addictive products without user consent.

# - Limitations: This is a toy example using purchase history only.

# Real systems should consider diversity, fairness, and explainability.

purchase\_history = {

"user1": ["Laptop", "Mouse", "Keyboard"],

"user2": ["Laptop", "Headphones"],

"user3": ["Mouse", "Keyboard", "Monitor"]

}

def recommend\_products(user, history, top\_n=2):

"""

Recommend products for a given user.

Parameters:

- user (str): The user ID to recommend for.

- history (dict): Mapping of users to product lists.

- top\_n (int): Number of recommendations to return.

Returns:

- List of recommended products.

Responsible Use:

- Explain to the user that recommendations are based on collaborative filtering.

- Ensure the dataset is diverse to avoid biased recommendations.

- Regularly audit the model for fairness.

"""

user\_items = set(history.get(user, []))

recommendations = defaultdict(int)

for other\_user, items in history.items():

if other\_user == user:

continue

for item in items:

if item not in user\_items:

recommendations[item] += 1

sorted\_recs = sorted(recommendations.items(), key=lambda x: x[1], reverse=True)

return [item for item, \_ in sorted\_recs[:top\_n]]

# Example

print("Recommendations for user1:", recommend\_products("user1", purchase\_history))

output:

Recommendations for user1: ['Headphones', 'Monitor']

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

Code:

import logging

# Configure logging

logging.basicConfig(

filename="app.log",

level=logging.INFO,

format="%(asctime)s - %(levelname)s - %(message)s"

)

def login(user\_id, email, password):

# SECURITY: Do NOT log sensitive data like email or password

logging.info(f"User login attempt: user\_id={user\_id}") # Safe to log

# Simulate login check (placeholder)

if password == "securepassword": # Don't do this in real apps!

logging.info(f"User {user\_id} logged in successfully.")

else:

logging.warning(f"User {user\_id} failed to log in.")

# Example usage

login("user\_001", "user@example.com", "securepassword")

output:

2025-08-28 15:00:00,001 - INFO - User login attempt: user\_id=user\_001

2025-08-28 15:00:00,002 - INFO - User user\_001 logged in successfully.

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).

Code:

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.datasets import load\_breast\_cancer

from sklearn.metrics import classification\_report

import pandas as pd

# Load dataset (example: breast cancer)

data = load\_breast\_cancer()

X = pd.DataFrame(data.data, columns=data.feature\_names)

y = pd.Series(data.target)

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train model

model = LogisticRegression(max\_iter=10000)

model.fit(X\_train, y\_train)

# Evaluate

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

# Print model coefficients for interpretability

print("\nFeature Importances (coefficients):")

for feature, coef in zip(X.columns, model.coef\_[0]):

print(f"{feature}: {coef:.4f}")

output:

precision recall f1-score support

0 0.95 0.94 0.95 43

1 0.96 0.97 0.96 71

accuracy 0.95 114

macro avg 0.95 0.95 0.95 114

weighted avg 0.95 0.95 0.95 114

Feature Importances (coefficients):

mean radius: 0.4563

mean texture: -0.1224