WEEK 5.4 ASSIGNMENT

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.

```
[3]
54s
def collect_user_data():
    """
    Collects user's name, age, and email address.

    Returns:
        tuple: A tuple containing the name (str), age (str), and email (str).
    """
    name = input("Please enter your name: ")
    age = input("Please enter your age: ")
    email = input("Please enter your email address: ")
    return name, age, email

print("User data collection function 'collect_user_data' created successfully.")

# Call the function and print the collected data
user_name, user_age, user_email = collect_user_data()
print(f"\nCollected User Data:")
print(f"Name: {user_name}")
print(f"Age: {user_age}")
print(f"Email: {user_email}")
```

```
User data collection function 'collect_user_data' created successfully.
Please enter your name: 2303a513@sru.edu.in
Please enter your age: 20
Please enter your email address: 2303A513@sru.edu.in

Collected User Data:
Name: 2303a513@sru.edu.in
Age: 20
Email: 2303A513@sru.edu.in
```

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.

```python
def analyze_sentiment(text_input):
    """
    Analyzes the sentiment of a given text input.

    Args:
        text_input (str): The text string to analyze.

    Returns:
        str: 'Positive', 'Negative', or 'Neutral' sentiment classification.
    """
    analysis = TextBlob(text_input)
    # Set thresholds for classification
    if analysis.sentiment.polarity > 0:
        return 'Positive'
    elif analysis.sentiment.polarity < 0:
        return 'Negative'
    else:
        return 'Neutral'

print("Sentiment analysis function 'analyze_sentiment' created successfully.")

# Example usage:
text1 = "This is an amazing product! I love it."
text2 = "I am very disappointed with the service."
text3 = "The weather today is neutral."

print(f"'{text1}' -> Sentiment: {analyze_sentiment(text1)}")
print(f"'{text2}' -> Sentiment: {analyze_sentiment(text2)}")
print(f"'{text3}' -> Sentiment: {analyze_sentiment(text3)}")
```

OUTPUT:

```
Sentiment analysis function 'analyze_sentiment' created successfully.
'This is an amazing product! I love it.' -> Sentiment: Positive
'I am very disappointed with the service.' -> Sentiment: Negative
'The weather today is neutral.' -> Sentiment: Neutral
```

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.

```python
import random
user_history = {
    "user1": [],
    "user2": [],
    "user3": []
}
all_products = {
    "prodA": "Laptop",
    "prodB": "Mouse",
    "prodC": "Keyboard",
    "prodD": "Monitor",
    "prodE": "Webcam",
    "prodF": "Headphones",
    "prodG": "Speaker",
    "prodH": "Printer"
}

def record_interaction(user_id, product_id):

    if user_id in user_history:
        if product_id not in user_history[user_id]: # Avoid duplicate entries for simplicity
            user_history[user_id].append(product_id)
            print(f"Interaction recorded: User {user_id} interacted with product {product_id}.")
        else:
            print(f"User {user_id} already interacted with product {product_id}.")
    else:
        user_history[user_id] = [product_id]
        print(f"New user {user_id} created and interaction with product {product_id} recorded.")

def get recommendations(user id, num recommendations=3):
```

OUT PUT:

```
Product recommendation system initialized with ethical considerations.

--- Example Usage ---
Interaction recorded: User user1 interacted with product prodA.
Interaction recorded: User user1 interacted with product prodB.
User user1 already interacted with product prodA.
Interaction recorded: User user2 interacted with product prodC.
Interaction recorded: User user2 interacted with product prodD.

Recommendations for user1: ['Speaker', 'Headphones', 'Keyboard']
Explanation: Recommendations are based on products you have not interacted with before, chosen randomly from available options.
Recommendations for user2: ['Printer', 'Laptop', 'Mouse']
Explanation: Recommendations are based on products you have not interacted with before, chosen randomly from available options.
User user4 not found in history.
Recommendations for user4: ['Webcam', 'Speaker', 'Printer']
Explanation: As a new user, recommendations are random selections from our entire product catalog.
Interaction recorded: User user1 interacted with product prodE.

Recommendations for user1 after new interaction: ['Keyboard', 'Printer']
Explanation: Recommendations are based on products you have not interacted with before, chosen randomly from available options.

--- User Feedback Examples ---
Feedback received for user user1 on product prodF: helpful
Feedback received for user user2 on product prodA: not helpful
Feedback received for user user4 on product prodG: irrelevant
```

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.

```
import logging
logger = logging.getLogger('web_app_logger')
logger.setLevel(logging.DEBUG)
if not logger.handlers:
    file_handler = logging.FileHandler('web_app.log')
    file_handler.setLevel(logging.DEBUG) # Set level for file output
    console_handler = logging.StreamHandler()
    console_handler.setLevel(logging.INFO) # Set level for console output, e.g., INFO and above
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    file_handler.setFormatter(formatter);
    console_handler.setFormatter(formatter);
    logger.addHandler(file_handler)
    logger.addHandler(console_handler)
print("Logger 'web_app_logger' configured successfully.")
logger.debug("This is a debug message. (Only visible in file, not console)")
logger.info("This is an info message. (Visible in file and console)")
logger.warning("This is a warning message. (Visible in file and console)")
logger.error("This is an error message. (Visible in file and console)")
logger.critical("This is a critical message. (Visible in file and console)")
```

```
DEBUG:web_app_logger:This is a debug message. (Only visible in file, not console)
2026-01-22 04:54:24,266 - web_app_logger - INFO - This is an info message. (Visible in file and console)
INFO:web_app_logger:This is an info message. (Visible in file and console)
2026-01-22 04:54:24,273 - web_app_logger - WARNING - This is a warning message. (Visible in file and console)
WARNING:web_app_logger:This is a warning message. (Visible in file and console)
2026-01-22 04:54:24,279 - web_app_logger - ERROR - This is an error message. (Visible in file and console)
ERROR:web_app_logger:This is an error message. (Visible in file and console)
2026-01-22 04:54:24,284 - web_app_logger - CRITICAL - This is a critical message. (Visible in file and console)
CRITICAL:web_app_logger:This is a critical message. (Visible in file and console)
Logger 'web_app_logger' configured successfully.
```

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 limit

```
import warnings
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=ConvergenceWarning)
print("Libraries imported successfully: datasets, train_test_split, LogisticRegression, accuracy_score.")
iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target # Target variable
print(f"\nIris dataset loaded. Features shape: {X.shape}, Target shape: {y.shape}")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
print(f"Data split into training and testing sets.")
print(f"Training set size: {len(X_train)} samples")
print(f"Testing set size: {len(X_test)} samples")
model = LogisticRegression(solver='lbfgs', max_iter=1000, random_state=42)
print("\nLogistic Regression model initialized.")
model.fit(X_train, y_train)
print("Model trained successfully on the training data.")
y_pred = model.predict(X_test)
print("Predictions made on the test set.")
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Evaluation:")
print(f"Accuracy on the test set: {accuracy:.4f}")
```

```
Libraries imported successfully: datasets, train_test_split, LogisticRegression, accuracy_score.

Iris dataset loaded. Features shape: (150, 4), Target shape: (150,)
Data split into training and testing sets.
Training set size: 112 samples
Testing set size: 38 samples

Logistic Regression model initialized.
Model trained successfully on the training data.
Predictions made on the test set.

Model Evaluation:
Accuracy on the test set: 1.0000
```