# AI ASSISTED CODING

**2303A52008**

**BATCH 43**

**ASSESS WEEK 5.4.**

## TASK 1:

```python
import hashlib

def collect_user_data():
    name = input("Enter your name: ")
    age = input("Enter your age: ")
    email = input("Enter your email: ")
    hashed_email = hashlib.sha256(email.encode()).hexdigest()
    user_data = {
        "name": name,
        "age": age,
        "email_hash": hashed_email
    }
    return user_data


if __name__ == "__main__":
    data = collect_user_data()
    print("User data collected securely:")
    print(data)
```

## Output:

```
Enter your name: vikas
Enter your age: 19
Enter your email: 2303a52281@sru.edu.in
User data collected securely:
{'name': 'vikas', 'age': '19', 'email_hash': '5ff49d8220cd11086eaf452375af9c9e678e200f1df8bd9f351b508df5dfa0ab'}
PS C:\Users\vikas\Downloads\AI Assist Coding>
```

## Summary:

This Python script demonstrates how user data such as name, age, and email can be collected securely while following basic data protection principles. Instead of storing sensitive information like email addresses in plain text, the script anonymizes the data by applying SHA-256 hashing, which helps protect user identity in case of data exposure. Inline comments explain important security practices, including encrypting stored data, limiting access based on permissions, and avoiding unnecessary data collection. The script also highlights the importance of secure data transmission and proper data disposal when the information is no longer needed. Overall, it reflects how GitHub Copilot can assist in generating both functional code and meaningful guidance for safeguarding user data.

## TASK 2: handle potential biases in the data.

```python
    print(data)
"""
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

def train_sentiment_model(texts, labels):
    model = Pipeline([
        ("vectorizer", CountVectorizer(
            stop_words="english",
            lowercase=True
        )),
        ("classifier", MultinomialNB())
    ])
    model.fit(texts, labels)
    return model
def predict_sentiment(model, new_text):
    return model.predict([new_text])[0]
if __name__ == "__main__":
    sample_texts = [
        "The product is excellent",
        "Worst experience ever",
        "It is okay, nothing special"
    ]
    sample_labels = ["Positive", "Negative", "Neutral"]
    sentiment_model = train_sentiment_model(sample_texts, sample_labels)
    result = predict_sentiment(sentiment_model, "The service was good")
    print("Predicted Sentiment:", result)
```

## Output:

```
Predicted Sentiment: Negative
PS C:\Users\vikas\Downloads\AI Assist Coding>
```

## Summary:

This sentiment analysis function shows how GitHub Copilot can generate a basic machine-learning pipeline while also addressing potential bias in the data. Inline comments highlight strategies such as balancing sentiment classes, removing offensive or biased terms, and reviewing training data for cultural or demographic skew. The use of stop-word removal helps reduce irrelevant language that may distort predictions. Additionally, the code encourages monitoring model outputs to detect consistently biased predictions. Overall, it demonstrates how bias awareness can be integrated directly into AI-assisted code generation.

## Task 3:

```python
def recommend_products(user_history, product_catalog):
    recommendations = []
    explanation = []
    category_count = {}
    for item in user_history:
        category = item["category"]
        category_count[category] = category_count.get(category, 0) + 1
    sorted_categories = sorted(
        category_count.items(), key=lambda x: x[1], reverse=True
    )
    max_per_category = 2

    category_recommend_count = {}

    for product in product_catalog:
        category = product["category"]

        # Recommend only relevant but diverse products
        if category in dict(sorted_categories):
            count = category_recommend_count.get(category, 0)

            if count < max_per_category:
                recommendations.append(product)
                category_recommend_count[category] = count + 1
                explanation.append(
                    f"Recommended because you showed interest in {category} products."
                )
    return recommendations, explanation
def collect_user_feedback(product_id, feedback):
    print(f"Feedback received for product {product_id}: {feedback}")
if __name__ == "__main__":
    user_history = [
        {"product": "Running Shoes", "category": "Sports"},
        {"product": "Football", "category": "Sports"},
        {"product": "Yoga Mat", "category": "Fitness"}
    ]

    product_catalog = [
        {"id": 1, "name": "Cricket Bat", "category": "Sports"},
        {"id": 2, "name": "Dumbbells", "category": "Fitness"},
        {"id": 3, "name": "Tennis Racket", "category": "Sports"},
        {"id": 4, "name": "Resistance Bands", "category": "Fitness"}
    ]

    recs, reasons = recommend_products(user_history, product_catalog)

    for r, reason in zip(recs, reasons):
        print(f"{r['name']} → {reason}")

    collect_user_feedback(1, "Not interested")
```

**Output:**

```
ython.exe" "c:/Users/vikas/Downloads/AI Assist Coding/week 5.4.py"
Could not find platform independent libraries <prefix>
Cricket Bat → Recommended because you showed interest in Sports products.
Dumbbells → Recommended because you showed interest in Fitness products.
Tennis Racket → Recommended because you showed interest in Sports products.
Resistance Bands → Recommended because you showed interest in Fitness products.
Resistance Bands → Recommended because you showed interest in Fitness products.
Feedback received for product 1: Not interested
PS C:\Users\vikas\Downloads\AI Assist Coding>

Feedback received for product 1: Not interested
PS C:\Users\vikas\Downloads\AI Assist Coding>
Feedback received for product 1: Not interested
Feedback received for product 1: Not interested
PS C:\Users\vikas\Downloads\AI Assist Coding>
```

## Summary:

This product recommendation program demonstrates how GitHub Copilot can generate ethical AI-driven solutions by incorporating transparency, fairness, and user feedback mechanisms. The code explains why each product is recommended, helping users understand the logic behind suggestions. Fairness checks limit over-representation of any single category, preventing favoritism and filter bubbles. A feedback function allows users to influence future recommendations without exposing personal identity. Overall, the program aligns recommendation logic with responsible and ethical AI guidelines.

## TASK 4:

```python
import logging

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler("app.log"),
        logging.StreamHandler()  # shows output in terminal
    ]
)

def log_user_action(action, user_id=None):
    if user_id:
        anonymized_user = f"user_{hash(user_id) % 10000}"
    else:
        anonymized_user = "anonymous_user"

    logging.info("Action performed: %s | User: %s", action, anonymized_user)

if __name__ == "__main__":
    log_user_action("User login attempt", user_id="vikas123")
    log_user_action("Viewed product page")
```

## Output:

```
Could not find platform independent libraries <prefix>
2026-01-29 14:04:23,156 - INFO - Action performed: User login attempt | User: user_2278
2026-01-29 14:04:23,156 - INFO - Action performed: Viewed product page | User: anonymous_user
```

## Summary:

This logging code illustrates how GitHub Copilot can help implement responsible logging practices in a Python web application. Instead of recording sensitive personal information, the script uses anonymized user identifiers and logs only high-level user actions. Inline comments clearly emphasize ethical logging principles such as avoiding passwords and emails, restricting log access, and rotating log files regularly. By focusing on transparency and minimal data collection, the code ensures logs are useful for debugging while respecting user privacy. This approach aligns logging functionality with ethical and legal data protection standards.

## TASK 5:

```
    data = load_iris()
    X, y = data.data, data.target
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )
    model = LogisticRegression(max_iter=200)
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    acc = accuracy_score(y_test, predictions)
    print("Model Accuracy:", acc)
    return model
if __name__ == "__main__":
    trained_model = train_model()
```

## Output:

```
Could not find platform independent libraries <prefix>
Model Accuracy: 1.0
PS C:\Users\vikas\Downloads\AI Assist Coding>
```

## Summary:

This machine learning example demonstrates how GitHub Copilot can generate a predictive model while also embedding responsible AI documentation directly within the code. The comments explain the importance of model interpretability, highlight that accuracy has limitations, and caution against using the model for high-stakes decisions. Fairness considerations emphasize balanced datasets and group-wise evaluation to reduce bias. By documenting ethical constraints and data responsibility, the code promotes transparent and accountable use of machine learning systems.