

Assignment – 5.4

B.Shiva Sai

2403A51257

24BTCAICSB11

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.

Output #1:

```
import hashlib

def collect_user_data():
    # Politely prompt the user for their information
    print("Hi there! We'd like to get to know you a little better.")
    name = input("May I have your name? ")
    age = input("How old are you? ")
    email = input("What is your email address? ")
    # Store the collected information
    user_data = {"name": name, "age": age, "email": email}
    return user_data

def anonymize_user_data(user_data):
    # To respect your privacy, we can take steps to protect or anonymize your data:
    # - Hashing the email means we keep your identity private and can't reverse it to find your actual email.
    # - We avoid keeping your name unless it's necessary.
    # - Sensitive data should never be stored in plain text or without protection.
    hashed_email = hashlib.sha256(user_data["email"].encode()).hexdigest()
    anonymized_data = {
        "name": "Anonymous", # Replace name with a generic label for anonymity
        "age": user_data["age"],
        "email_hash": hashed_email # Store only the hashed value of the email
    }
    return anonymized_data

if __name__ == "__main__":
    print("Welcome! We value your privacy.")
    data = collect_user_data()
    print("\nThank you! Here's what we collected:")
    print(data)
    print("\nTo better protect your privacy, here's how we can anonymize your information:")
    anonymized = anonymize_user_data(data)
    print(anonymized)
    print("\nNote: When saving any sensitive information, always consider encrypting it and restricting access.")
```

Comments by Copilot (inline):

- Use hashing (e.g., SHA-256) to anonymize sensitive information like emails.
- Avoid storing user names or emails in plain text.
- Use "REDACTED" or pseudonyms for names if full anonymization is required.
- Always use encryption and access controls when storing sensitive data persistently

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.

Output #2:

```
1  from textblob import TextBlob
2
3  def analyze_sentiment(text):
4      """
5      Analyze the sentiment of input text using TextBlob.
6      Returns polarity and subjectivity scores.
7      """
8      # Perform sentiment analysis
9      blob = TextBlob(text)
10     sentiment = blob.sentiment
11     return {
12         "polarity": sentiment.polarity,      # [-1.0, 1.0], negative to positive sentiment
13         "subjectivity": sentiment.subjectivity # [0.0, 1.0], objective to subjective
14     }
15
16     # --- Bias Mitigation Strategies ---
17     # 1. Data Balancing: Ensure your dataset is balanced with respect to sentiment classes (positive, neutral, negative).
18     #    Imbalanced datasets can cause the model to be biased toward the majority class.
19     # 2. Remove Offensive or Sensitive Terms: Filter out or flag text containing hate speech, slurs, or offensive language to avoid perpetuating bias.
20     # 3. Language & Demographic Diversity: Include diverse language styles, dialects, and demographics to avoid overfitting to a specific group.
21     # 4. Regular Auditing: Periodically audit model predictions and outputs for signs of bias, and retrain with corrected datasets if necessary.
22     # 5. Manual Review: For critical applications, use human-in-the-loop to review flagged or ambiguous sentiment predictions.
23
24     # Example usage:
25     if __name__ == "__main__":
26         # Example input
27         user_text = input("Enter a sentence to analyze its sentiment: ")
28         result = analyze_sentiment(user_text)
29         print("Sentiment analysis result:", result)
30         print("\nNote: For best results, ensure your training and test data are diverse, balanced, and free from offensive content.")
```

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.

Output #3:

```
1  import random
2
3  # Sample product database
4  PRODUCTS = [
5      {"id": 1, "name": "Eco-Friendly Notebook", "category": "Stationery"},
6      {"id": 2, "name": "Reusable Water Bottle", "category": "Lifestyle"},
7      {"id": 3, "name": "Organic Cotton T-Shirt", "category": "Clothing"},
8      {"id": 4, "name": "LED Desk Lamp", "category": "Electronics"},
9      {"id": 5, "name": "Fair Trade Coffee", "category": "Grocery"},
10 ]
11
12 # Example user history format: list of product IDs the user has interacted with
13 def recommend_products(user_history, all_products=PRODUCTS, num_recommendations=3):
14     """
15     Recommend products based on user's purchase or interaction history.
16     Follows ethical guidelines:
17     - Recommendations are transparent (user can see why they are recommended)
18     - Tries to avoid favoritism or bias towards certain products or categories
19     - Allows user feedback for improvement
20     """
21     # Count category interactions for transparency
22     category_count = {}
23     for prod_id in user_history:
24         prod = next((p for p in all_products if p["id"] == prod_id), None)
25         if prod:
26             cat = prod["category"]
27             category_count[cat] = category_count.get(cat, 0) + 1
28
29     # Fairness: recommend from categories the user likes, but also diversify
30     preferred_categories = sorted(category_count, key=category_count.get, reverse=True)
31     recommendations = []
32
33     # Step 1: Recommend from preferred categories (if any)
34     for cat in preferred_categories:
35         recs = [p for p in all_products if p["category"] == cat and p["id"] not in user_history]
```

```

36     random.shuffle(recs)
37     recommendations += recs
38
39     # Step 2: Add diverse picks (products from other categories)
40     diverse = [p for p in all_products if p["category"] not in preferred_categories and p["id"] not in user_history]
41     random.shuffle(diverse)
42     recommendations += diverse
43
44     # Avoid favoritism: do not always rank the same products first; shuffle recommendations
45     random.shuffle(recommendations)
46
47     # Step 3: Limit to requested number
48     final_recommendations = recommendations[:num_recommendations]
49
50     # Transparency: Show user why these products were recommended
51     print("We recommend these products based on your history and to introduce some new options:")
52     for product in final_recommendations:
53         reason = ""
54         if product["category"] in preferred_categories:
55             reason = f"Because you've shown interest in {product['category']}."
56         else:
57             reason = "To help you discover something new."
58         print(f"- {product['name']} ({product['category']}) - {reason}")
59
60     # Solicit feedback for fairness and improvement
61     print("\nYour feedback matters! If you feel these recommendations are not fair or relevant, please let us know.")
62
63     return final_recommendations
64
65 if __name__ == "__main__":
66     # Example usage
67     print("Welcome! Let's recommend some products for you.")
68     user_history = [1, 3] # Example: user has interacted with Notebook and T-Shirt
69     recommended = recommend_products(user_history)
70     # In a real application, collect actual feedback and use it to improve the algorithm.
71     # Always ensure recommendations are unbiased, transparent, and respect user choice.

```

Ethical Guidelines in Code:

- Recommendations are explained to the user for transparency.
- Attempts to balance between user's interests and diversity to avoid bias or favoritism.
- Recommendations are shuffled to avoid always promoting the same products.
- Users are invited to provide feedback on fairness and relevance.

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.

Output #4:

```
1  import logging
2  from flask import Flask, request
3
4  app = Flask(__name__)
5
6  # Set up logging configuration
7  logging.basicConfig(
8      filename='app.log',
9      level=logging.INFO,
10     format='%(asctime)s %(levelname)s %(message)s'
11 )
12
13 def sanitize_data(data):
14     """
15     Remove or mask sensitive information before logging.
16     For example: passwords, emails, or personal identifiers.
17     """
18     sanitized = data.copy()
19     # Mask password fields
20     if 'password' in sanitized:
21         sanitized['password'] = '[REDACTED]'
22     # Mask email fields
23     if 'email' in sanitized:
24         sanitized['email'] = '[REDACTED]'
25     # Add more fields as needed
26     return sanitized
27
28 @app.route('/login', methods=['POST'])
29 def login():
30     # Get user input (e.g., from a login form)
31     data = request.form.to_dict()
32     # Sanitize input data before logging
33     log_data = sanitize_data(data)
34     # Log only non-sensitive information
35     logging.info(f"Login attempt: {log_data}")
36     # Continue with authentication (not shown for brevity)
37     return "Login processed"
38
39 if __name__ == "__main__":
40     app.run(debug=True)
41
42 # Ethical Logging Practices:
43 # - Never log sensitive data (passwords, emails, personal identifiers).
44 # - Use a sanitization function to redact or remove such data before logging.
45 # - Regularly review logs to ensure compliance with privacy standards.
46 # - Limit log access to authorized personnel only.
47 # - Use secure storage for logs if they contain potentially sensitive operational details.
```

Key points:

- Sensitive fields (like 'password' and 'email') are redacted before logging.
- Comments explain why and how to avoid logging personal identifiers.
- Ethical logging practices are summarized at the end of the file.

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).

Output #5:

```
1  import pandas as pd
2  from sklearn.model_selection import train_test_split
3  from sklearn.linear_model import LogisticRegression
4  from sklearn.metrics import classification_report
5
6  def load_data(path):
7      """
8      Load data from a CSV file.
9      """
10     return pd.read_csv(path)
11
12  def train_model(X, y):
13      """
14      Train a simple logistic regression model.
15      """
16     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
17     model = LogisticRegression()
18     model.fit(X_train, y_train)
19     y_pred = model.predict(X_test)
20     print("Classification Report:\n", classification_report(y_test, y_pred))
21     return model
22
23  def predict(model, X_new):
24      """
25      Predict using the trained model.
26      """
27     return model.predict(X_new)
28
29  if __name__ == "__main__":
30     # Example usage:
31     # Assume data.csv has columns: "feature1", "feature2", ..., "label"
32     data = load_data("data.csv")
33     X = data.drop("label", axis=1)
34     y = data["label"]
35     model = train_model(X, y)
36     # To predict on new data:
37     # predictions = predict(model, X_new)
38     print("Model training complete. Refer to README for responsible usage guidelines.")
```

```
# Simple Machine Learning Model

## Overview
This project provides a basic template for training and evaluating a
logistic regression model using scikit-learn.

## Responsible Usage Guidelines

- **Explainability:**
  This model uses logistic regression, which is relatively interpretable.
  You can check `model.coef_` and `model.intercept_` to understand feature
  importance and decision boundaries.

- **Accuracy Limits:**
  - The model's performance depends on the quality, balance, and
  representativeness of the provided dataset.
  - Always validate metrics (precision, recall, F1, etc.) using the
  provided classification report.
  - Do not use the model in critical applications without thorough
  evaluation.

- **Fairness Considerations:**
  - Ensure your dataset does not contain biases (e.g., demographic or
  gender imbalance) that might be learned by the model and reflected in its
  predictions.
  - Regularly audit predictions for disparate impact and retrain with a
  balanced dataset if necessary.

- **Transparency:**
  - Document the data sources and preprocessing steps.
  - Clearly communicate the model's intended use cases and limitations to
  stakeholders.

- **Privacy:**
  - Do not include personally identifiable information (PII) in training
  data without appropriate consent and anonymization.

## Limitations

- This model is a simple baseline and may not capture complex
  relationships in data.
- It may perform poorly on unbalanced datasets or in the presence of
  outliers.
- The model is not robust to adversarial examples or novel, out-of-
  distribution data.

## Usage

1. Place your CSV data in the project directory as `data.csv`.
2. Run `python simple_ml_model.py` to train and evaluate the model.
3. Adapt the code for your own datasets, being mindful of responsible ML
  practices.
```