

## **Assignment-5.4**

Name: Zobiya Fatima  
Hall Ticket No: 2303A51879  
Batch No: 14

## 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.

File Edit Selection View Go Run Terminal Help ← → 🔍 ai-assisted

RUN AND DEBUG ...

RUN

Run and Debug

To customize Run and Debug create a launch.json file.

Debug using a terminal command or in an interactive chat.

Show automatic Python configurations

>Welcome lab-5.py

```
1 def collect_user_data():
2     """Collect user information and return as a dictionary."""
3     print("==> User Data Collection ==\n")
4     name = input("Enter your name: ").strip()
5     while True:
6         try:
7             age = int(input("Enter your age: "))
8             if age < 0 or age > 150:
9                 print("Please enter a valid age between 0 and 150.")
10                continue
11            break
12        except ValueError:
13            print("Please enter a valid number for age.")
14    email = input("Enter your email: ").strip()
15    user_data = {
16        "name": name,
17        "age": age,
18        "email": email
19    }
20    return user_data
21 def main():
22     """Main function to run the user data collection script."""
23     user_data = collect_user_data()
24     print("\n==> Collected Data ==")
25     print(f"Name: {user_data['name']}")
26     print(f"Age: {user_data['age']}")
27     print(f"Email: {user_data['email']}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python Debug Console

```
Enter your name: Zia IAmn
Enter your age: 20
Enter your email: sweret32@gmail.com

==> Collected Data ==
Name: Zia IAmn
Age: 20
Email: sweret32@gmail.com
PS D:\ai-assisted> 
```

BREAKPOINTS

- Raised Exceptions
- Uncaught Exception...
- User Uncaught Exc...

In 27. Col 42 Spaces: 4 UTF-R CRL

## Summary:

The code given was 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).

- Comments in how to anonymize or protect this data.

The screenshot shows a Python development environment with the following details:

- File Explorer:** Shows a project structure with files like `lab-5.py`, `__init__.py`, and `test.py`.
- Run Configuration:** Set to "Python" and "Python 3.11".
- Terminal:** Displays the command `PS D:\ai-assisted> python lab-5.py`.
- Code Editor:** The `lab-5.py` file is open, containing a function `collect_user_data` that collects user information (name, age, email) and returns it as a dictionary. It includes exception handling for invalid input.
- Breakpoints:** Several breakpoints are set in the code, with one currently active at line 19.
- Output:** Shows the execution of the script, with the output of the `print` statements visible.
- Help:** A search bar at the top right is set to "ai-assisted".

## 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.

The screenshot shows the VS Code interface with the following details:

- Left Sidebar:** RUN section with "Run and Debug" button highlighted.
- Right Sidebar:** BREAKPOINTS section with three checkboxes: "Raised Exceptions", "Uncaught Exception...", and "User Uncaught Exc...".
- Code Editor:** A Python script named `lab-5.py` containing a function for sentiment analysis. The code uses `TextBlob` to analyze text polarity and returns a dictionary with sentiment and score. It also includes a main block to prompt the user for text and print the result.
- Terminal:** Shows the execution of the script. The user enters text "I Hate Oranges", and the script outputs: `{'sentiment': 'negative', 'score': -0.8}`. Then, the user enters text "I love Mangoes", and the script outputs: `{'sentiment': 'positive', 'score': 0.5}`.
- Bottom Right:** Taskbar with multiple tabs for "Python Deb..." and "powershell".

- Identifying and handling potential biases in the data

```

from textblob import TextBlob
# Bias categories with weighted terms
BIAS_PATTERNS = [
    {"category": "absolute_language", "terms": ["always", "never", "everyone", "nobody", "all", "none"], "weight": 0.6},
    {"category": "overgeneralization", "terms": ["people", "they", "them", "society", "culture"], "weight": 0.5}
]
def analyze_text(text):
    """
    Comprehensive text analysis for sentiment, subjectivity, and bias detection.
    Args:
        text (str): The input text to analyze.
    Returns:
        dict: Analysis results including sentiment, subjectivity, and bias details.
    """
    if not text or not text.strip():
        return {"error": "Empty text provided"}
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    subjectivity = blob.sentiment.subjectivity
    # Detect bias
    biases_found = _detect_bias(blob.lower())
    overall_bias_score = sum(b["weight"] for b in biases_found) / len(BIAS_PATTERNS) if biases_found else 0
    return {
        "sentiment": sentiment,
        "subjectivity": subjectivity,
        "bias_detected": bool(biases_found),
        "bias_categories": biases_found,
        "overall_bias_score": round(overall_bias_score, 3),
        "recommendations": get_recommendations(sentiment, subjectivity, overall_bias_score)
    }
def _detect_bias(text):
    """
    Detect and categorize biases in text.
    """
    biases = []
    for category, data in BIAS_PATTERNS.items():
        found_terms = [term for term in data["terms"] if term in text]
        if found_terms:
            biases.append({
                "category": category,
                "found_terms": found_terms,
                "weight": data["weight"]
            })
    return biases
def get_recommendations(sentiment, subjectivity, bias_score):
    """
    Generate recommendations based on analysis.
    """
    tips = []
    if subjectivity > 0.7:
        tips.append("High subjectivity - consider adding objective evidence")
    if bias_score > 0.4:
        tips.append("Reduce use of absolute language for more balanced tone")
    if sentiment == "negative" and subjectivity > 0.6:
        tips.append("Negative sentiment + high subjectivity may appear biased")
    return tips if tips else ["Text appears balanced"]
if __name__ == "__main__":
    user_text = input("Enter text to analyze: ")
    result = analyze_text(user_text)
    print("\n--- Analysis Results ---")
    for key, value in result.items():
        print(f"{key}: {value}")

```

The terminal shows the output of running the script with the input "I Hate Oranges". The output includes the analysis results: sentiment: negative, polarity\_score: -0.9, subjectivity: 0.9, bias\_detected: False, bias\_categories: [], overall\_bias\_score: 0, and recommendations: ['High subjectivity - consider adding objective evidence', 'Negative sentiment + high subjectivity may appear biased'].

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

## 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.

```
lab-05(3).py > get_user_history
1 def get_user_history(user_id):
2     # Placeholder for database lookup
3     return {
4         "user_id": user_id,
5         "purchases": ["laptop", "mouse", "keyboard"],
6         "categories": ["electronics", "accessories"]
7     }
8 def calculate_recommendation_score(product, user_history, bias_weight=0.1):
9     category_match = 1.0 if product["category"] in user_history["categories"] else 0.5
10    popularity = product["popularity"] / 100
11    # Apply fairness: reduce bias toward popular items
12    fairness_adjusted = (category_match * 0.7) + (popularity * bias_weight)
13    return fairness_adjusted
14 def recommend_products(user_id, product_catalog, top_n=5):
15     user_history = get_user_history(user_id)
16     recommendations = []
17     for product in product_catalog:
18         score = calculate_recommendation_score(product, user_history)
19         recommendations.append({
20             "product": product["name"],
21             "score": round(score, 2),
22             "reason": f"Based on your interest in {product['category']}",
23             "transparency": "Algorithm considers user history and fairness"
24         })
25     # Sort and return top recommendations
26     recommendations.sort(key=lambda x: x["score"], reverse=True)
27     return recommendations[:top_n]
28 # Example usage
29 if __name__ == "__main__":
30     sample_catalog = [
31         {"name": "Wireless Headphones", "category": "electronics", "popularity": 85},
32         {"name": "USB-C Cable", "category": "accessories", "popularity": 70},
33         {"name": "Desk Lamp", "category": "accessories", "popularity": 60},
34         {"name": "Monitor", "category": "electronics", "popularity": 75},
35     ]
36     recs = recommend_products("user_123", sample_catalog, top_n=3)
37     print("Personalized Recommendations:")
38     for i, rec in enumerate(recs, 1):
39         print(f"\n{i}: {rec['product']}")
40         print(f"  Relevance Score: {rec['score']}")
41         print(f"  Why: {rec['reason']}")
42         print(f"  Transparency: {rec['transparency']}")
```

Summary: The code written by Github-Copilot includes explanations, fairness checks (e.g., avoiding favoritism), and user feedback options in the code.

## 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.

The screenshot shows a code editor interface with two main panes. The left pane is titled 'RUN AND DEBUG' and contains sections for 'RUN', 'TERMINAL', and 'BREAKPOINTS'. The 'TERMINAL' section shows command-line history and log messages. The right pane displays the source code for 'lab-05(4).py'.

```
lab-05(4).py
=====
# lab-05(4).py > setup_logging
1 import logging
2 import logging.handlers
3 from datetime import datetime
4 from pathlib import Path
5 def setup_logging(log_level=logging.INFO, log_file="app.log"):
6     #configure logging for a Python web application.
7     #Args:
8     #    log_level: Logging level (default: INFO)
9     #    log_file: Path to log file (default: app.log)
10    # Create logs directory if it doesn't exist
11    path = Path("logs").mkdir(exist_ok=True)
12    log_path = Path("logs") / log_file
13    # Create logger
14    logger = logging.getLogger("web_app")
15    logger.setLevel(log_level)
16    # Create formatters
17    formatter = logging.Formatter(
18        "%(asctime)s - %(name)s - %(levelname)s - %(message)s",
19        datefmt="%Y-%m-%d %H:%M:%S"
20    )
21    # File handler (rotating)
22    file_handler = logging.handlers.RotatingFileHandler(
23        log_path,
24        maxBytes=10485760, # 10MB
25        backupCount=5
26    )
27    file_handler.setLevel(log_level)
28    file_handler.setFormatter(formatter)
29    # Console handler
30    console_handler = logging.StreamHandler()
31    console_handler.setLevel(log_level)
32    console_handler.setFormatter(formatter)
33    # Add handlers to logger
34    logger.addHandler(file_handler)
35    logger.addHandler(console_handler)
36    return logger
37
38 # Usage example
39 if __name__ == "__main__":
40     logger = setup_logging(log_level=logging.DEBUG)
41     logger.info("Application started")
42     logger.warning("This is a warning")
43     logger.error("An error occurred")
```

Terminal Output:

```
Transparency: Algorithm considers user history and fairness
PS D:\ai-assisted> C
PS D:\ai-assisted>
PS D:\ai-assisted> dt; cd 'd:\ai-assisted'; & 'c:\Users\TAMANWA\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\TAMANWA\vscode\extensisons\ms-python.python_debugpy-2025.18.0-win32-x64\bundles\libs\debgui\launcher' '59360' -- "D:\ai-assisted\lab-05(4).py"
2026-01-29 10:47:27 - web_app - INFO - Application started
2026-01-29 10:47:27 - web_app - WARNING - This is a warning
2026-01-29 10:47:27 - web_app - ERROR - An error occurred
PS D:\ai-assisted> [REDACTED]
```

Bottom status bar: Tab Moves Focus | Lr 6 Col 5 | Spaces: 4 | UTF-8 | CRLF | {} Python | 3.14.2 | Q

### Summary:

This code sets up a logging system with sensitive data filtering. Here's the summary:

- 1. Logging Configuration:** Configures a rotating file handler that logs to `app.log` with a max size of 10MB and keeps 5 backup files.
- 2. SensitiveDataFilter Class:** A custom logging filter that detects and redacts sensitive information including:
  - Passwords
  - Email addresses
  - Phone numbers
  - Credit card numbers
  - Social Security numbers
  - API keys/tokens
- 3. Pattern Matching:** Uses regex patterns to find sensitive data in log messages and replaces them with `[REDACTED\_TYPE]` placeholders.

**4.Example Usage:** Demonstrates logging various messages containing sensitive data, which are automatically filtered before being written to the log file.

The purpose is to prevent accidentally logging confidential information while maintaining a complete audit trail.

### 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).

A screenshot of the Visual Studio Code interface. The top bar shows the title 'Untitled-1.py'. Below the editor, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected), 'PORTS', 'SQL HISTORY', and 'TASK MONITOR'. The terminal window displays the following command and output:

```
PS C:\Users\zobiya\Downloads\AI-Assisted coding> c:; cd 'c:\Users\zobiya\Downloads\AI-Assisted coding'; & 'c:\Users\zobiya\vscode\extensions\ms-python.python\debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' 62829 ... 'C:\Users\zobiya\Downloads\AI-Assisted coding\Untitled-1.py'
virginica      1.00      1.00      1.00      11
               accuracy      1.00      30
               macro avg     1.00      1.00      1.00      30
               weighted avg  1.00      1.00      1.00      30
```

The code in the editor is:

```
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.metrics import accuracy_score, classification_report
5 # Load dataset
6 iris = load_iris()
7 X = iris.data
8 y = iris.target
9 # Split data into training and testing sets
10 X_train, X_test, y_train, y_test = train_test_split(
11     X, y, test_size=0.2, random_state=42
12 )
13 # Create and train the model
14 model = RandomForestClassifier(n_estimators=100, random_state=42)
15 model.fit(X_train, y_train)
16 # Make predictions
17 y_pred = model.predict(X_test)
18 # Evaluate the model
19 accuracy = accuracy_score(y_test, y_pred)
20 print(f"Accuracy: {accuracy:.4f}")
21 print("\nClassification Report:")
22 print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

- Documentation on how to use it properly:

```
"""
This script demonstrates how to train and evaluate a Random Forest classifier on the Iris dataset.

Steps performed:
1. Loads the Iris dataset.
2. Splits the data into training and testing sets.
3. Trains a Random Forest classifier.
4. Makes predictions on the test set.
5. Evaluates the model's accuracy and prints a classification report.

Usage Notes:
- This example is intended for educational and demonstration purposes.
- The Iris dataset is balanced and well-understood, so results may not generalize to other datasets.
- Model explainability: Random Forests provide feature importance scores, which can help interpret the model.
- Accuracy limits: The reported accuracy is specific to the Iris dataset and the chosen model configuration.
- Responsible use: Always assess model fairness, potential biases, and applicability to your specific use case.

"""

```

**Summary:** This script demonstrates how to train and evaluate a Random Forest classifier on the Iris dataset using scikit-learn.

Steps performed:

1. Loads the Iris dataset.
2. Splits the data into training and testing sets.
3. Trains a Random Forest classifier.
4. Makes predictions on the test set.
5. Evaluates the model's accuracy and prints a classification report.