

# Assignment-5.4

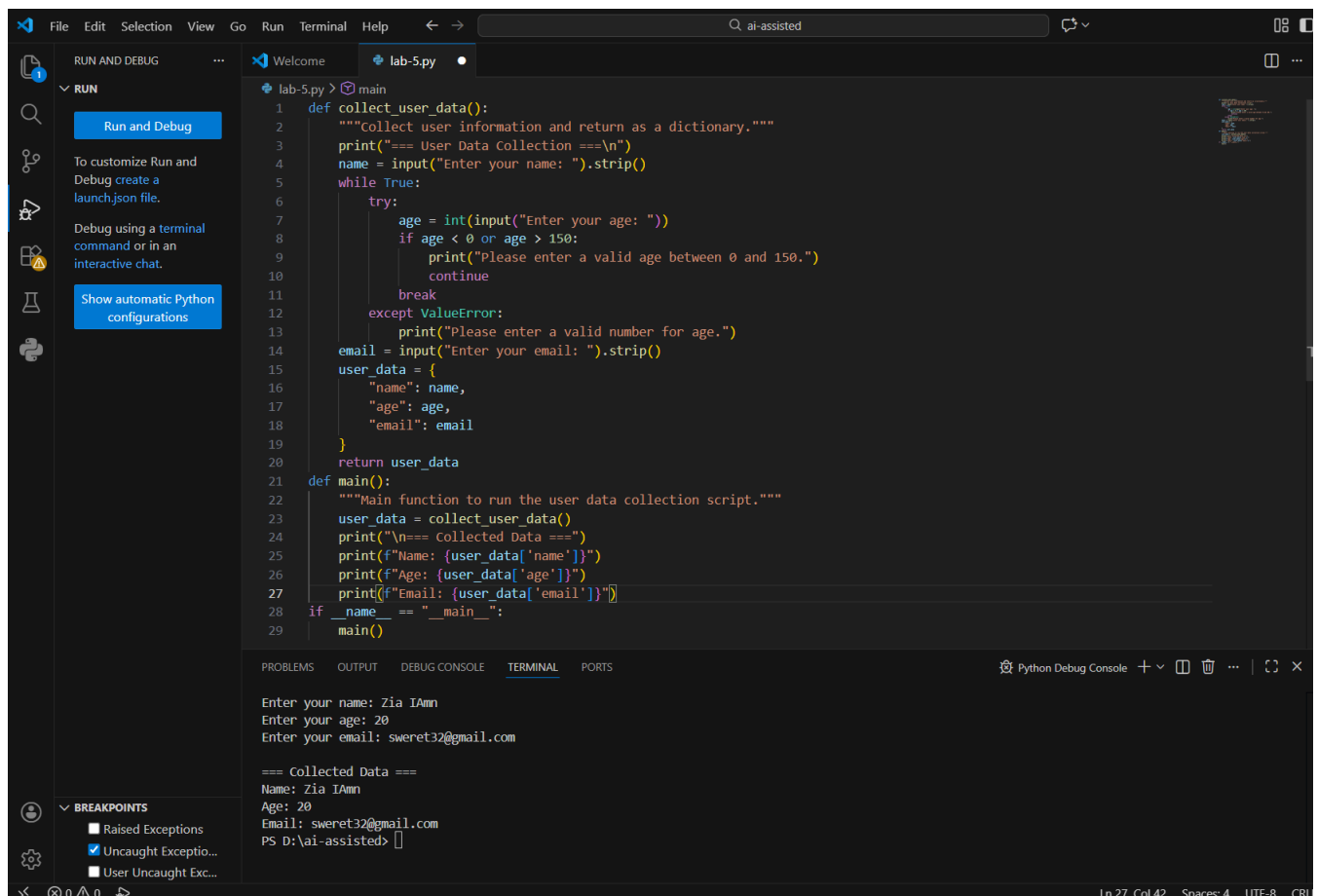
Name: Zobiya Fatima

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.



The screenshot displays the Visual Studio Code interface. The editor window shows a Python file named `lab-5.py` with the following code:

```
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
22 def main():
23     """Main function to run the user data collection script."""
24     user_data = collect_user_data()
25     print("\n== Collected Data ==")
26     print(f"Name: {user_data['name']}")
27     print(f"Age: {user_data['age']}")
28     print(f"Email: {user_data['email']}")
29
30 if __name__ == "__main__":
31     main()
```

The terminal window at the bottom shows the execution of the script with the following output:

```
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>
```

## 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.

```

1 import uuid
2 def collect_user_data():
3     """Collect user information and return as a dictionary."""
4     print("== User Data Collection ==\n")
5     name = input("Enter your name: ").strip()
6     while True:
7         try:
8             age = int(input("Enter your age: "))
9             if age < 0 or age > 150:
10                print("Please enter a valid age between 0 and 150.")
11                continue
12                break
13            except ValueError:
14                print("Please enter a valid number for age.")
15        email = input("Enter your email: ").strip()
16        # Generate a unique ID instead of storing personal identifiers
17        user_id = str(uuid.uuid4())
18        user_data = {
19            "user_id": user_id, # Use ID instead of name
20            "age_group": categorize_age(age), # Store age ranges, not exact age
21            # "email": email # Don't store email; use hashed version if needed
22        }
23        return user_data
24    def categorize_age(age):
25        """Categorize age into ranges for anonymization."""
26        if age < 18:
27            return "under_18"
28        elif age < 65:
29            return "18_64"
30        else:
31            return "65_plus"

```

## 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.

```

1 from textblob import TextBlob
2 def sentiment_analysis(text):
3     """
4     Analyze the sentiment of a given text.
5     Args:
6         text (str): The input text to analyze.
7     Returns:
8         dict: A dictionary with sentiment label and score.
9     """
10    blob = TextBlob(text)
11    polarity = blob.sentiment.polarity
12    if polarity > 0.1:
13        sentiment = "positive"
14    elif polarity < -0.1:
15        sentiment = "negative"
16    else:
17        sentiment = "neutral"
18    return {
19        "sentiment": sentiment,
20        "score": polarity
21    }
22 if __name__ == "__main__":
23     user_text = input("Enter text to analyze: ")
24     result = sentiment_analysis(user_text)
25     print(result)

```

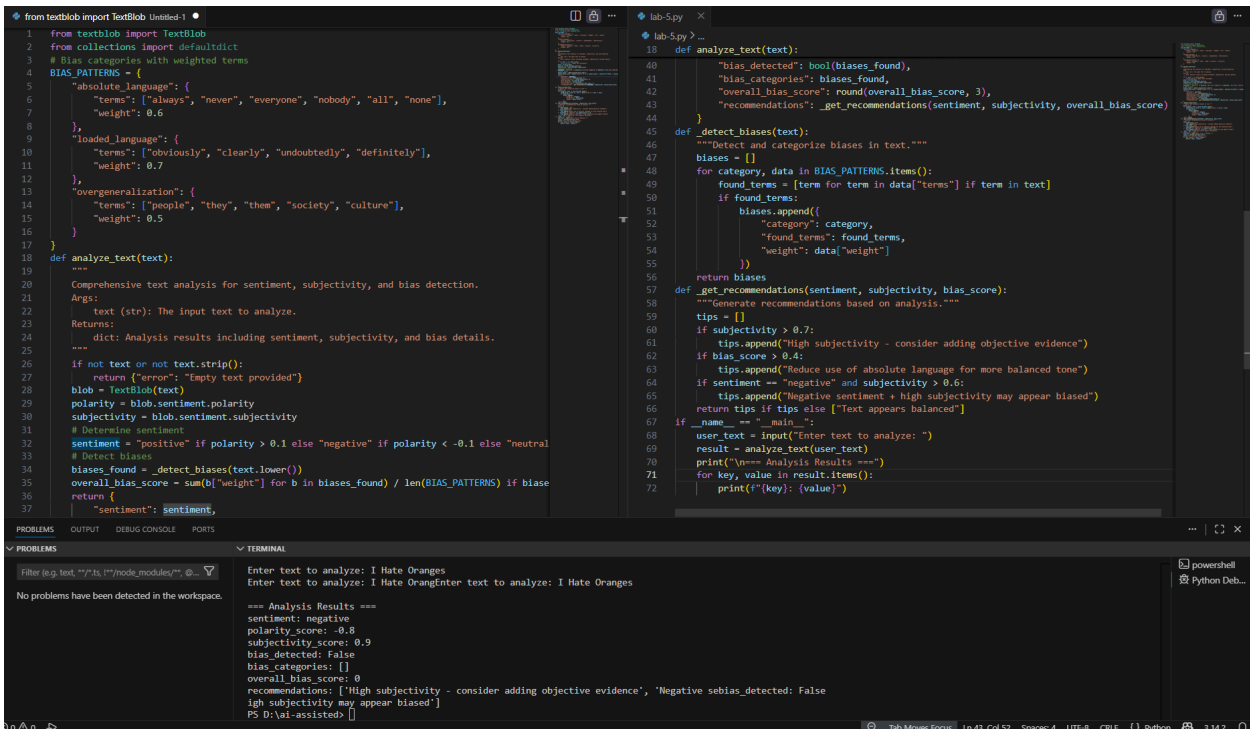
Terminal Output:

```

-5.py'
Enter text to analyze: I Hate Oranges
{'sentiment': 'negative', 'score': -0.8}
PS D:\ai-assisted> ^C
PS D:\ai-assisted>
PS D:\ai-assisted> ^C
PS D:\ai-assisted> ^C
PS D:\ai-assisted>
PS D:\ai-assisted> d;; cd 'd:\ai-assisted'; & 'c:\Users\TAMAMNA\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\TAMAMNA\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '52891' '-' 'D:\ai-assisted\lab-5.py'
Enter text to analyze: I love Mangoes
{'sentiment': 'positive', 'score': 0.5}
PS D:\ai-assisted>

```

- Identifying and handling potential biases in the data



```
1 from textblob import TextBlob
2 from collections import defaultdict
3 # Bias categories with weighted terms
4 BIAS_PATTERNS = {
5     "absolute language": {
6         "terms": ["always", "never", "everyone", "nobody", "all", "none"],
7         "weight": 0.6
8     },
9     "loaded language": {
10        "terms": ["obviously", "clearly", "undoubtedly", "definitely"],
11        "weight": 0.7
12    },
13    "overgeneralization": {
14        "terms": ["people", "they", "them", "society", "culture"],
15        "weight": 0.5
16    }
17 }
18 def analyze_text(text):
19     """
20     Comprehensive text analysis for sentiment, subjectivity, and bias detection.
21     Args:
22         text (str): The input text to analyze.
23     Returns:
24         dict: Analysis results including sentiment, subjectivity, and bias details.
25     """
26     if not text or not text.strip():
27         return {"error": "Empty text provided"}
28     blob = TextBlob(text)
29     polarity = blob.sentiment.polarity
30     subjectivity = blob.sentiment.subjectivity
31     # Determine sentiment
32     sentiment = "positive" if polarity > 0.1 else "negative" if polarity < -0.1 else "neutral"
33     # Detect biases
34     biases_found = _detect_biases(text.lower())
35     overall_bias_score = sum(b["weight"] for b in biases_found) / len(BIAS_PATTERNS) if biases_found else 0
36     return {
37         "sentiment": sentiment,
38         "polarity_score": polarity,
39         "subjectivity_score": subjectivity,
40         "bias_detected": bool(biases_found),
41         "bias_categories": biases_found,
42         "overall_bias_score": round(overall_bias_score, 3),
43         "recommendations": _get_recommendations(sentiment, subjectivity, overall_bias_score)
44     }
45
46 def _detect_biases(text):
47     """Detect and categorize biases in text."""
48     biases = []
49     for category, data in BIAS_PATTERNS.items():
50         found_terms = [term for term in data["terms"] if term in text]
51         if found_terms:
52             biases.append({
53                 "category": category,
54                 "found_terms": found_terms,
55                 "weight": data["weight"]
56             })
57     return biases
58
59 def _get_recommendations(sentiment, subjectivity, bias_score):
60     """Generate recommendations based on analysis."""
61     tips = []
62     if subjectivity > 0.7:
63         tips.append("High subjectivity - consider adding objective evidence")
64     if bias_score > 0.4:
65         tips.append("Reduce use of absolute language for more balanced tone")
66     if sentiment == "negative" and subjectivity > 0.6:
67         tips.append("Negative sentiment + high subjectivity may appear biased")
68     return tips if tips else ["Text appears balanced"]
69
70 if __name__ == "__main__":
71     user_text = input("Enter text to analyze: ")
72     result = analyze_text(user_text)
73     print("\n== Analysis Results ==")
74     for key, value in result.items():
75         print(f"{key}: {value}")
```

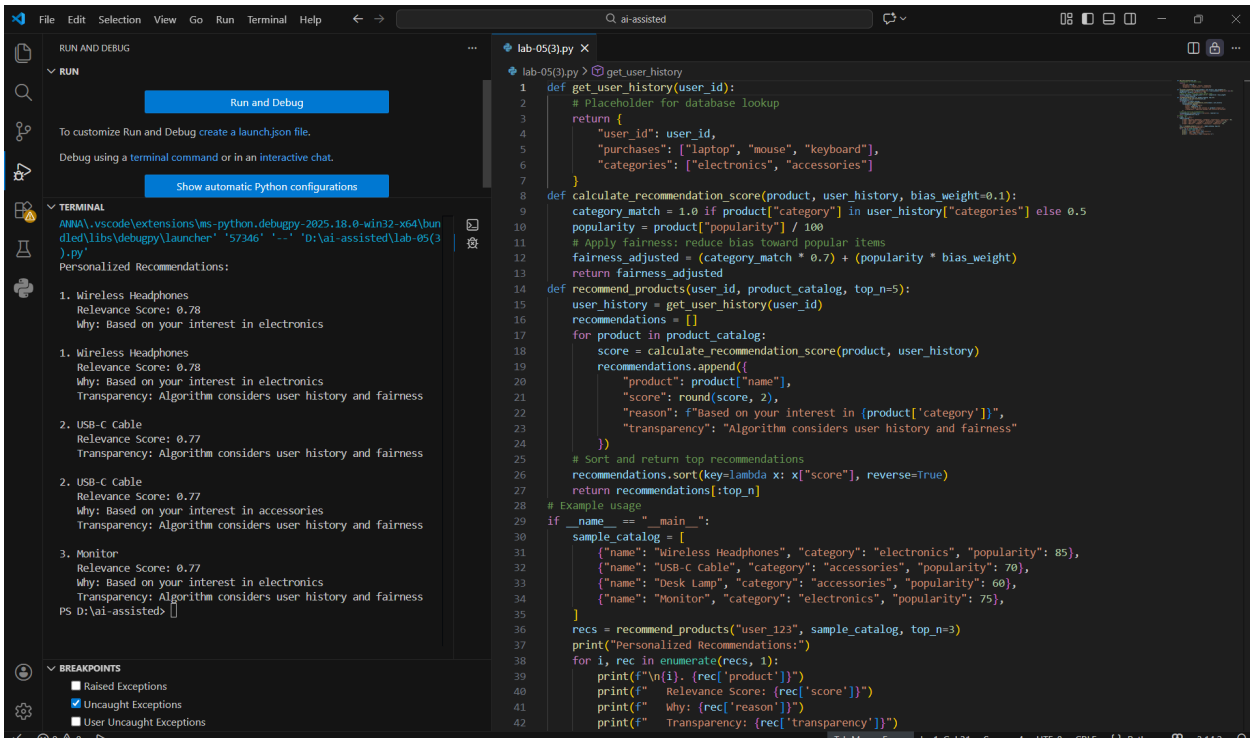
Enter text to analyze: I Hate Oranges  
Enter text to analyze: I Hate Oranges

```
== Analysis Results ==
sentiment: negative
polarity_score: -0.8
subjectivity_score: 0.9
bias_detected: False
bias_categories: []
overall_bias_score: 0
recommendations: ['High subjectivity - consider adding objective evidence', 'Negative sentiment + high subjectivity may appear biased']
PS D:\ai-assisted> []
```

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.



The screenshot shows a VS Code editor with a Python file named `lab-05(3).py`. The code is a recommendation system that takes a user ID and a product catalog as input and returns a list of recommended products. The code includes functions for getting user history, calculating recommendation scores, and recommending products. It also includes a sample catalog and an example usage section. The code is written in a dark theme and includes comments explaining the logic and ethical considerations like transparency and fairness.

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

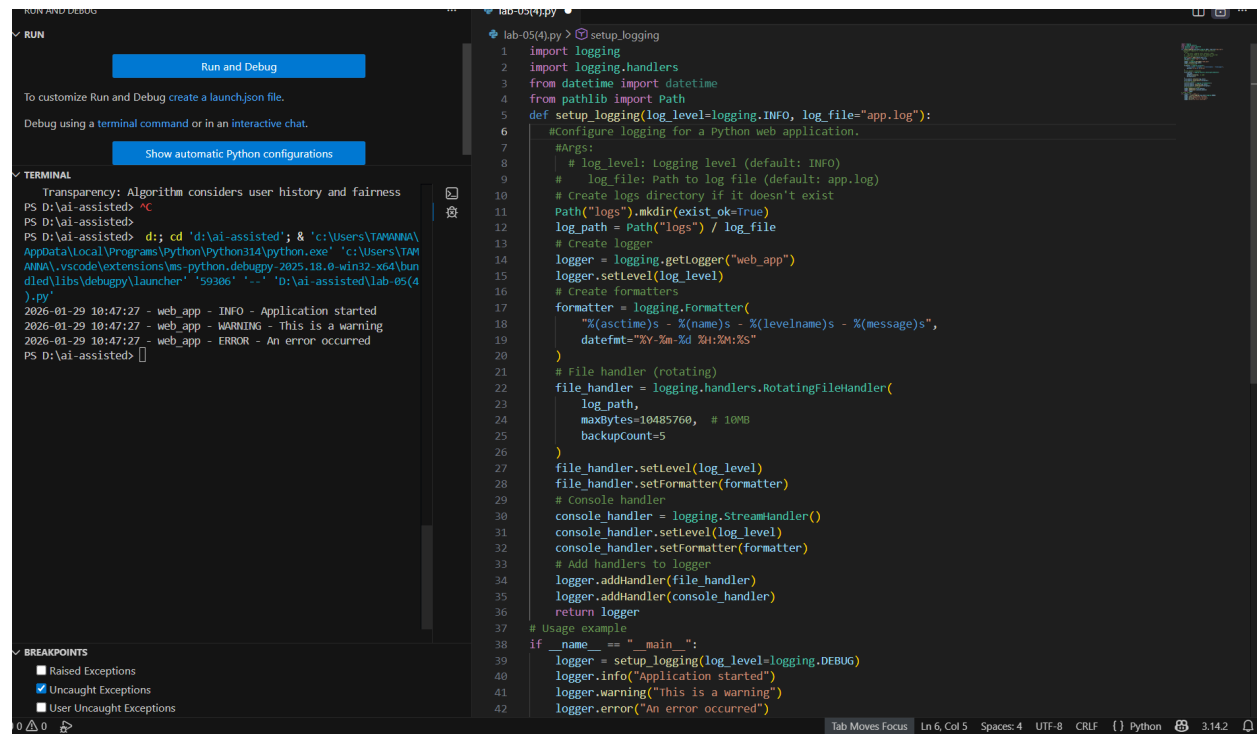
The terminal output shows the results of the recommendation system:

```
Personalized Recommendations:
1. Wireless Headphones
   Relevance Score: 0.78
   Why: Based on your interest in electronics
   Transparency: Algorithm considers user history and fairness
2. USB-C Cable
   Relevance Score: 0.77
   Transparency: Algorithm considers user history and fairness
3. Monitor
   Relevance Score: 0.77
   Why: Based on your interest in electronics
   Transparency: Algorithm considers user history and fairness
```

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 Python IDE with a file named `lab-05(4).py`. The code defines a `setup_logging` function that configures a logging system. It uses a rotating file handler for `app.log` with a maximum size of 10MB and 5 backup files. It also adds a console handler. The script includes a usage example that demonstrates logging at different levels (INFO, WARNING, ERROR).

```
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("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")
```

The terminal output shows the execution of the script:

```
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
```

## Summary:

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

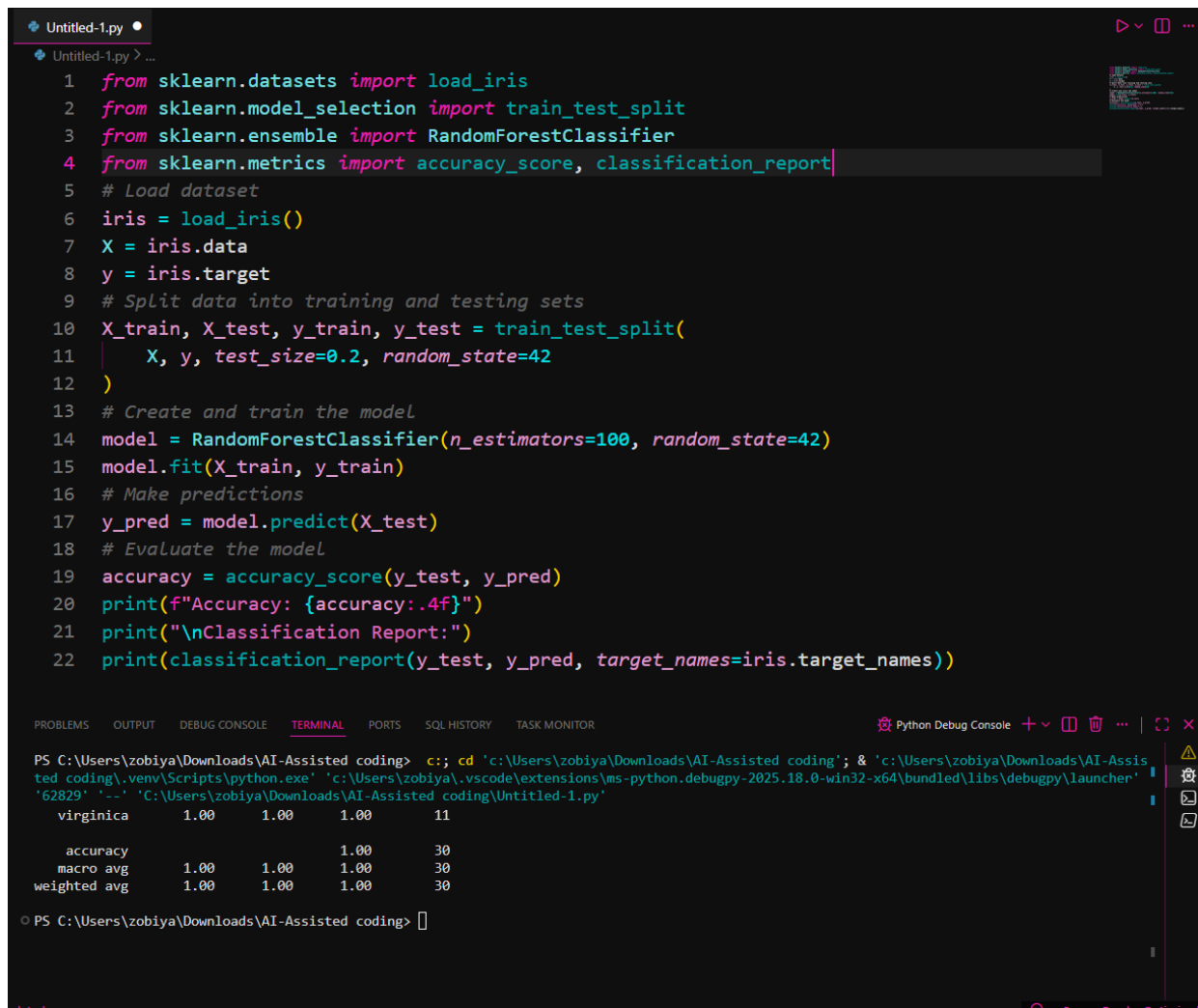
- Logging Configuration:** Configures a rotating file handler that logs to `'app.log'` with a max size of 10MB and keeps 5 backup files.
- SensitiveDataFilter Class:** A custom logging filter that detects and redacts sensitive information including:
  - Passwords
  - Email addresses
  - Phone numbers
  - Credit card numbers
  - Social Security numbers
  - API keys/tokens
- 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).



```
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))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR Python Debug Console

```
PS C:\Users\zobiya\Downloads\AI-Assisted coding> c::; cd 'c:\Users\zobiya\Downloads\AI-Assisted coding'; & 'c:\Users\zobiya\Downloads\AI-Assisted coding\.venv\Scripts\python.exe' 'c:\Users\zobiya\.vscode\extensions\ms-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

PS C:\Users\zobiya\Downloads\AI-Assisted coding>
```

- Documentation on how to use it properly:

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

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 understand the model's decisions.
- Accuracy limits: The reported accuracy is specific to the Iris dataset and the chosen model.
- 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.