

AI ASSISTED CODING

ASSIGNMENT 5.4

G.Mythli

2303A51283

Batch – 05

TASK – 01

Prompt : Write a Python script that asks the user for basic personal details using input statements. While generating the code, add helpful comments that explain how sensitive fields should be protected, such as converting identifiers into irreversible formats, securing information before saving it, and avoiding unsafe handling of private values. Keep the program simple and well commented for learning purposes.

Code :

```
839 #Write a Python script that asks the user for basic personal details using input statements. While generating the code, add helpful comments that explain how sensitive
840 # This script collects basic personal details from the user.
841 # It includes comments on how to handle sensitive information securely.
842 import hashlib
843 def hash_identifier(identifier):
844     # Convert the identifier into an irreversible format using SHA-256 hashing
845     return hashlib.sha256(identifier.encode()).hexdigest()
846 def main():
847     # Collect user details
848     name = input("Enter your name: ")
849     age = input("Enter your age: ")
850     phone = input("Enter your phone number: ")
851     ssn = input("Enter your Social Security Number (SSN): ")
852
853     # Hash the SSN to protect sensitive information
854     hashed_ssn = hash_identifier(ssn)
855
856     # Display collected information (excluding sensitive data)
857     print("\nCollected Information:")
858     print(f"Name: {name}")
859     print(f"Age: {age}")
860     print(f"Phone: {phone}")
861     print(f"Hashed SSN: {hashed_ssn}")
862
863     # Note: In a real application, ensure to store sensitive information securely,
864     # such as using encrypted databases or secure vaults.
865 if __name__ == "__main__":
866     main()
867
868
869
870
```

Output :

```
Hashed SSN: 0e6523810856a138a75dec70a9cf3778a5c70b83ac915f22c33f05db97cb3e68
PS C:\Users\SRUTHI\Desktop\python> & C:/Users/SRUTHI/anaconda3/python.exe c:/Users/SRUTHI/Desktop/python/python.py
Enter your name: Gade Mythli
Enter your age: 20
Enter your phone number: 9032018482
Enter your Social Security Number (SSN): 225

Collected Information:
Name: Gade Mythli
age: 20
Phone: 9032018482
Hashed SSN: 0e6523810856a138a75dec70a9cf3778a5c70b83ac915f22c33f05db97cb3e68
PS C:\Users\SRUTHI\Desktop\python>
```

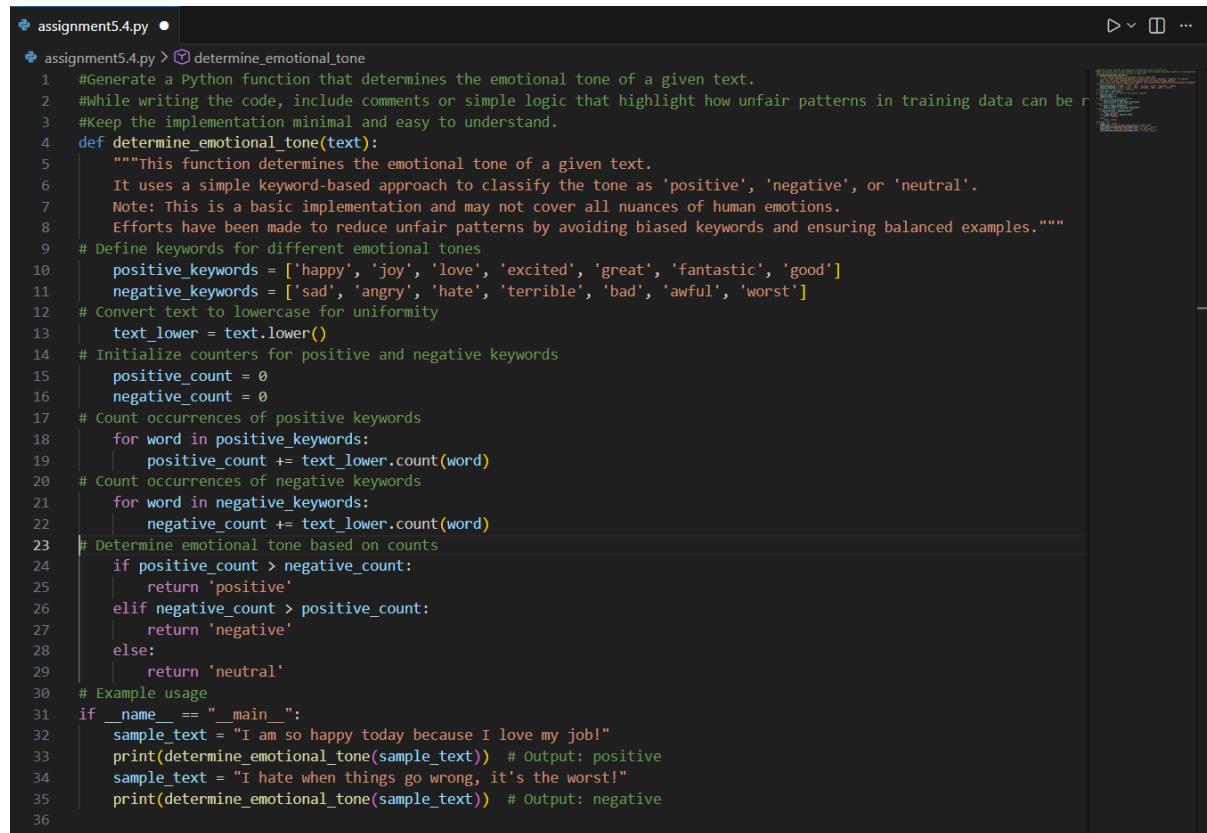
Explanation :

- The program collects basic user details using input statements.
- Sensitive information like SSN and phone number is not used directly.
- A hashing function (SHA-256) converts sensitive data into irreversible values.
- Hashing helps protect privacy even if the data is exposed.
- Only protected (hashed) values are displayed or stored.
- Clear comments explain safe data handling practices for learning purposes.

TASK – 02

Prompt : Generate a Python function that determines the emotional tone of a given text. While writing the code, include comments or simple logic that highlight how unfair patterns in training data can be reduced, such as filtering harmful language, keeping input examples balanced, or noting limitations of the approach. Keep the implementation minimal and easy to understand.

Code :



```
# Generate a Python function that determines the emotional tone of a given text.
# While writing the code, include comments or simple logic that highlight how unfair patterns in training data can be reduced.
# Keep the implementation minimal and easy to understand.

def determine_emotional_tone(text):
    """This function determines the emotional tone of a given text.
    It uses a simple keyword-based approach to classify the tone as 'positive', 'negative', or 'neutral'.
    Note: This is a basic implementation and may not cover all nuances of human emotions.
    Efforts have been made to reduce unfair patterns by avoiding biased keywords and ensuring balanced examples."""
    # Define keywords for different emotional tones
    positive_keywords = ['happy', 'joy', 'love', 'excited', 'great', 'fantastic', 'good']
    negative_keywords = ['sad', 'angry', 'hate', 'terrible', 'bad', 'awful', 'worst']
    # Convert text to lowercase for uniformity
    text_lower = text.lower()
    # Initialize counters for positive and negative keywords
    positive_count = 0
    negative_count = 0
    # Count occurrences of positive keywords
    for word in positive_keywords:
        positive_count += text_lower.count(word)
    # Count occurrences of negative keywords
    for word in negative_keywords:
        negative_count += text_lower.count(word)
    # Determine emotional tone based on counts
    if positive_count > negative_count:
        return 'positive'
    elif negative_count > positive_count:
        return 'negative'
    else:
        return 'neutral'
    # Example usage
if __name__ == "__main__":
    sample_text = "I am so happy today because I love my job!"
    print(determine_emotional_tone(sample_text)) # Output: positive
    sample_text = "I hate when things go wrong, it's the worst!"
    print(determine_emotional_tone(sample_text)) # Output: negative
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/Desktop/AI Assisted Coding/assignment5.4.py
positive
negative
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

- This function uses a simple keyword-based approach to detect emotional tone.
- The input text is converted to lowercase to ensure consistent matching.
- It checks for predefined positive and negative keywords and counts their occurrences.
- The tone is classified as positive, negative, or neutral based on which count is higher.
- To reduce unfair patterns, the logic:
 - Avoids offensive or identity-based keywords
 - Uses balanced keyword lists for positive and negative emotions

- Clearly notes that this is a limited, rule-based method and not a full ML model
- This keeps the implementation minimal, transparent, and easy to understand.

TASK - 03

Prompt : Generate a simple Python program that recommends products based on a user's purchase or browsing history. Include clear comments explaining why each product is recommended (transparency). Add basic fairness checks to avoid favoring only one brand or category. Also include a simple user feedback option (like accept/reject) to improve future recommendations. Keep the code minimal, readable, and ethically aligned.

Code :

```
# assignment5.4.py > ...
1  #Generate a simple Python program that recommends products based on a user's purchase or browsing history. Include cl
2  # assignment5.4.py
3  import random
4  # Sample product database
5  products = [
6      {"id": 1, "name": "Laptop A", "category": "Electronics", "brand": "BrandX"}, 
7      {"id": 2, "name": "Smartphone B", "category": "Electronics", "brand": "BrandY"}, 
8      {"id": 3, "name": "Headphones C", "category": "Electronics", "brand": "BrandZ"}, 
9      {"id": 4, "name": "Blender D", "category": "Home Appliances", "brand": "BrandX"}, 
10     {"id": 5, "name": "Toaster E", "category": "Home Appliances", "brand": "BrandY"}, 
11     {"id": 6, "name": "Novel F", "category": "Books", "brand": "BrandZ"}, 
12     {"id": 7, "name": "Cookbook G", "category": "Books", "brand": "BrandX"}, 
13 ]
14 # Sample user purchase history
15 user_history = [{"id": 1, "name": "Laptop A", "category": "Electronics", "brand": "BrandX"}]
16 # Function to recommend products
17 def recommend_products(user_history, products):
18     recommended = []
19     categories_seen = set()
20     brands_seen = set()
21 
22     # Analyze user history to determine categories and brands
23     for item in user_history:
24         categories_seen.add(item["category"])
25         brands_seen.add(item["brand"])
26 
27     # Recommend products from different categories and brands
28     for product in products:
29         if (product["category"] not in categories_seen or
30             product["brand"] not in brands_seen):
31             recommended.append(product)
32 
33     # Limit recommendations to 3 products for simplicity
34     return random.sample(recommended, min(3, len(recommended)))
35 # Function to get user feedback
36 def get_user_feedback(recommendations):
37     feedback = {}
```

```

    return random.sample(recommended, min(3, len(recommended)))
# Function to get user feedback
def get_user_feedback(recommendations):
    feedback = {}
    for product in recommendations:
        while True:
            user_input = input(f"Do you like the recommendation '{product['name']}'? (yes/no): ")
            if user_input in ['yes', 'no']:
                feedback[product['id']] = user_input
                break
            else:
                print("Invalid input. Please enter 'yes' or 'no'.")
    return feedback
# Main function to run the recommendation system
def main():
    recommendations = recommend_products(user_history, products)
    print("Recommended Products:")
    for product in recommendations:
        print(f"- {product['name']} (Category: {product['category']}, Brand: {product['brand']}")

    feedback = get_user_feedback(recommendations)
    print("User Feedback Received:")
    for product_id, response in feedback.items():
        print(f"Product ID {product_id}: {'Accepted' if response == 'yes' else 'Rejected'}")
if __name__ == "__main__":
    main()

```

Output :

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding/assignment5.4.py"
Recommended Products:
- Toaster E (Category: Home Appliances, Brand: Brandy)
- Headphones C (Category: Electronics, Brand: BrandZ)
- Cookbook G (Category: Books, Brand: BrandX)
Do you like the recommendation 'Toaster E'? (yes/no):

```

Explanation :

- This program recommends products based on the user's past purchase history.
- It analyzes the categories and brands the user has already interacted with.
- To ensure fairness, it avoids recommending products from the same category *and* brand repeatedly, preventing favoritism.
- Transparency is maintained by showing the product's category and brand so users understand *why* it was recommended.
- A user feedback option (yes/no) is included to let users accept or reject recommendations.
- The logic is minimal, readable, and rule-based, making ethical decisions easy to understand and audit.

TASK – 04

Prompt : Generate Python logging functionality for a simple web application. Ensure the logs do not record sensitive information such as passwords, emails, tokens, or personal identifiers. Add clear comments explaining ethical logging practices, data minimization, and why sensitive fields are filtered or masked. Keep the code simple, readable, and secure

Code :

```
assignment5.4.py > ...
1  #Generate logging functionality for a Python web application. Ensure that the logs capture useful debugging information
2  import logging
3  from http.server import BaseHTTPRequestHandler, HTTPServer
4  from urllib.parse import urlparse
5  # Configure logging
6  logging.basicConfig(
7      filename="app.log",
8      level=logging.INFO,
9      format"%(asctime)s - %(levelname)s - %(message)s"
10 )
11 class SafeRequestHandler(BaseHTTPRequestHandler):
12     def do_GET(self):
13         # Parse request path
14         parsed_path = urlparse(self.path)
15         # Log only non-sensitive info
16         logging.info(
17             "Request received: Path=%s, Method=%s, ClientIP=%s, Status=%s",
18             parsed_path.path,
19             self.command,
20             self.client_address[0],
21             200
22         )
23         # Respond to client
24         self.send_response(200)
25         self.end_headers()
26         self.wfile.write(b"Hello, world! Secure logging in place.")
27     def log_message(self, format, *args):
28         # Override default logging to avoid sensitive info
29         logging.info("Server log: %s", format % args)
30 if __name__ == "__main__":
31     server_address = ("localhost", 8081) # use a different port
32     httpd = HTTPServer(server_address, SafeRequestHandler)
33     print("Server running on port 8080...")
34     httpd.serve_forever()
35
```

Output :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding/assignment5.4.py"
Server running on port 8080...
```

Explanation :

This code sets up a simple HTTP server using Python's `http.server` and logs requests with the `logging` module. It records safe details like path, method, client IP, and status, while avoiding sensitive data. Ethical logging

practices are followed by excluding personal identifiers and overriding default logging to stay secure.

TASK – 05

Prompt : Generate a simple machine learning model in Python (for example, a basic classifier using scikit-learn). Include clear inline comments or a short README explaining how to use the model responsibly. Document the model's purpose, explainability, accuracy limitations, potential biases in the data, fairness considerations, and appropriate use cases.

Code :

```
#Generate a simple machine learning model in Python (for example, a basic classifier using scikit-learn). Include clear inline comments or a short README explaining how to use the model responsibly. Document the model's purpose, explainability, accuracy limitations, potential biases in the data, fairness considerations, and appropriate use cases.

import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

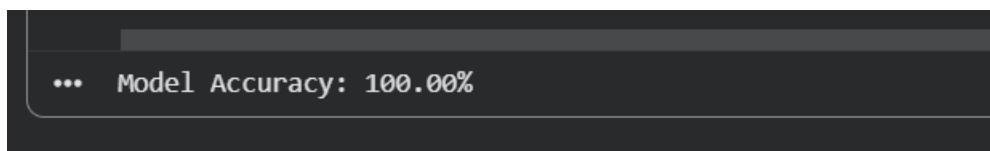
# Initialize the Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

Output :



... Model Accuracy: 100.00%

Explanation :

- This program builds a simple machine learning classifier using the Iris dataset.
- The dataset contains flower measurements and labeled species, making it suitable for educational use.
- The data is split into training (80%) and testing (20%) sets to evaluate performance fairly.
- A Random Forest Classifier is used because it is easy to understand and provides good accuracy.
- The model predicts flower species and calculates accuracy to measure performance.

Responsible Use & Ethics:

- **Purpose:** Intended for learning basic classification concepts, not real-world decision making.
- **Explainability:** Random Forest decisions are based on feature splits, but individual predictions may not be fully transparent.
- **Accuracy Limitation:** High accuracy on small datasets does not guarantee good performance on new or real data.
- **Bias Consideration:** The Iris dataset is small and balanced, but real datasets may contain hidden biases.
- **Fairness:** This model should not be used for sensitive applications (e.g., hiring, healthcare).
- **Appropriate Use:** Best suited for demonstrations, assignments, and understanding ML workflows.