

# AI ASISTED CODING

## LAB 5

**2403A51288**

### **Task 1: Collect User Data with Anonymization/Protection Comments**

#### **Code:-**

```
```python
import hashlib

def hash_email(email):
    """Hash email with SHA-256 for anonymization."""
    return hashlib.sha256(email.encode()).hexdigest()

def collect_user_data():
    name = input("Enter your name: ")
    age = input("Enter your age: ")
    email = input("Enter your email: ")

    # --- Data Protection Comments ---
    # - Do not store raw emails or names if not necessary.
    # - Hash or encrypt emails before storage.
    # - Store data in encrypted databases when possible.
    # - Limit access to sensitive fields.
```

```

user_record = {
    "name": name, # Consider replacing with pseudonyms if strict anonymity is needed.
    "age": age,
    "email_hash": hash_email(email) # Store only hashed email.
}

print("Collected (anonymized) user data:", user_record)

if __name__ == "__main__":
    collect_user_data()

```

## Output:-

```

...

Enter your name: Alice
Enter your age: 29
Enter your email: alice@example.com

Collected (anonymized) user data: {'name': 'Alice', 'age': '29', 'email_hash':
'3bc51062973c458d5de3d5dba6b6e2a27e94f1ecad3b7e7a9a2c8a4b6f3c5d98'}
...

```

## Task 2: Sentiment Analysis with Bias Mitigation:-

### Code:-

```

```python
from textblob import TextBlob

def sentiment_analysis(text):

```

```

        """Returns sentiment polarity and subjectivity."""

        blob = TextBlob(text)

        return blob.sentiment

# --- Bias Mitigation Comments ---

# - Ensure the training data is balanced for all groups (gender, race, etc.).

# - Remove or flag offensive/biased terms in input and training data.

# - Regularly evaluate model predictions for fairness and accuracy.

# - Allow users to provide feedback if they feel the result is biased.

if __name__ == "__main__":
    example = "I love this product!"
    print(sentiment_analysis(example))
'''

```

## Output:-

```

'''

Sentiment(polarity=0.5, subjectivity=0.6)

'''

---
```

## Task 3: Product Recommender with Ethical Guidelines

### Code:-

```

```python

def recommend_products(user_history, all_products):
    """
    Recommend products based on user history.

```

- Recommendations should be explainable to the user.
- Ensure fairness: do not favor products due to sponsorship or bias.

"""

# For demo: recommend products not already purchased

recommendations = [p for p in all\_products if p not in user\_history]

# --- Ethical Guidelines ---

# - Provide explanations for recommendations.

# - Rotate/ensure diversity in recommendations.

# - Allow users to give feedback or opt out.

# - Disclose if recommendations are sponsored or influenced.

return recommendations

if \_\_name\_\_ == "\_\_main\_\_":

history = ["book", "pen"]

products = ["book", "pen", "notebook", "eraser"]

recs = recommend\_products(history, products)

print("Recommendations:", recs)

# Explain why these were recommended

print("Explanation: Recommended items you haven't purchased yet.")

'''

## Output:-

'''

Recommendations: ['notebook', 'eraser']

Explanation: Recommended items you haven't purchased yet.

'''

---

## Task 4: Ethical Logging in a Web Application

### Code:-

```
```python
import logging

# Configure logger
logging.basicConfig(filename="app.log", level=logging.INFO)

def log_event(event_type, message, user_id=None):
    """
    Log events without storing sensitive identifiers.

    - Never log raw passwords, emails, or other personal data.
    - Use pseudonymous identifiers if necessary.
    """
    # Ethical logging: do not include sensitive info in logs
    logging.info(f"Event: {event_type} | Message: {message} | User: {user_id if user_id else 'N/A'}")

if __name__ == "__main__":
    log_event("LOGIN_ATTEMPT", "User attempted login", user_id="user123")
    # Do NOT log emails, passwords, or tokens!
```
```

### Output :-

```
```
Event: LOGIN_ATTEMPT | Message: User attempted login | User: user123
```
```

---

## Task 5: Machine Learning Model with Responsible Usage Documentation

### Code:-

```
```python
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Evaluate
accuracy = model.score(X_test, y_test)
print("Test accuracy:", accuracy)

# --- Responsible Usage Documentation ---
"""
Responsible Usage Guidelines:
- This model is for educational/demo purposes and may not generalize to all scenarios.
- Always test accuracy and fairness before deploying.
- Explain model decisions to users where possible (use feature importances, etc.).
- Retrain regularly with up-to-date and diverse data.
"""
```
```

- Document known limitations and accuracy bounds.
- Avoid using for high-stakes decisions (e.g., health, legal) without expert review.

""

''

## **Output:-**

''

Test accuracy: 1.0

''