AI ASSSITED CODING

LABTEST-04

NAME: TAFSIL AHMED

ENROLLNO:2403A52071

BATCH:04

SET-02:

Q1 Identify privacy risks in camera-based analytics.

• Task 1: Use AI to inspect code that stores video metadata.

Prompt: AI/Analyst review the database schema and retention policy code for data fields that constitute Personally Identifiable Information (PII) or enable re-identification.

CODE:

TASK1.py > ...

```python
import pandas as pd

def inspect_metadata_schema(schema_df):
    """Simulates AI inspecting a metadata schema for privacy risks."""
    print("## 🔍 Analyzing Metadata Schema for PII Risk")

    # 1. Flag High-Risk PII Fields
    high_risk_fields = ['person_id', 'camera_gps_coords', 'bounding_box_coordinates']

    print("\n--- PII Field Analysis ---")
    for field in schema_df['column_name']:
        if field.lower() in high_risk_fields:
            # Code flags fields that, when combined, allow for tracking individuals (re-identification).
            print(f"🚨 RISK ALERT: Field '{field}' is high-risk PII and enables re-identification.")

    # 2. Inspect Data Retention Policy
    retention_setting = 9999 # Days (Set to an obviously excessive value for the demo)
    if retention_setting > 365:
        print(f"\n--- Retention Policy Analysis ---")
        # Code warns against excessive retention, which increases the risk exposure window.
        print(f"⚠ WARNING: Data retention is set to {retention_setting} days (indefinite/excessive).")
        print("  - Recommended Action: Reduce retention or enforce differential deletion for PII.")
    else:
        print("✅ Retention policy seems reasonable.")

# Mock Schema (Data the AI would analyze)
mock_schema = pd.DataFrame({
    'column_name': ['person_ID', 'time_stamp', 'camera_GPS_coords', 'event_type', 'is_blurred'],
    'data_type': ['INT', 'DATETIME', 'TEXT', 'TEXT', 'BOOL']
})

# Run Inspection
inspect_metadata_schema(mock_schema)
```

OUTPUT:

```
Exception ignored in: <module '_collections_abc' (frozen)>
Traceback (most recent call last):
  File "<string>", line 0, in <module>
KeyboardInterrupt:
<frozen importlib._bootstrap>:488: RuntimeWarning: Cython module failed to patch module with custom type
## 🔍 Analyzing Metadata Schema for PII Risk

--- PII Field Analysis ---
🚨 RISK ALERT: Field 'person_ID' is high-risk PII and enables re-identification.
🚨 RISK ALERT: Field 'camera_GPS_coords' is high-risk PII and enables re-identification.

--- Retention Policy Analysis ---
⚠WARNING: Data retention is set to 9999 days (indefinite/excessive).
  - Recommended Action: Reduce retention or enforce differential deletion for PII.
PS C:\Users\tafse\OneDrive\Desktop\AI_coding.py>
```

OBSERVATION:

After reviewing code that stores video metadata, focus on whether it collects unnecessary personal identifiers, lacks encryption or access controls, and fails to enforce retention or deletion policies. These issues increase risks of unauthorized access, profiling, and long-term surveillance without user consent, violating privacy principles and regulations.

Q1. Task 2: Propose anonymization (blurring, hashing) and implementation.

Prompt: "Implement pseudonymization (hashing) for unique identifiers and ensure that only analytical aggregates are primarily stored, rather than raw individual logs."

Code:

```python
task2.py > ...
import hashlib
import uuid
import time
def pseudonymize_data(original_id, salt):
    """Implements one-way cryptographic hashing for pseudonymization."""
    # Combine the original ID with a secret salt
    data_to_hash = f"{original_id}-{salt}"

    # Use SHA-256 for a robust, one-way hash
    hashed_id = hashlib.sha256(data_to_hash.encode()).hexdigest()

    return hashed_id

# --- Implementation Example ---
SECRET_SALT = str(uuid.uuid4()) # Use a strong, unchanging secret salt for consistency
person_id_original = 458921
location_data = "Public_Square_1"
timestamp = time.time()

# 1. Hashing the ID
# The original PII (person_id_original) is immediately replaced with an irreversible hash.
person_id_hashed = pseudonymize_data(person_id_original, SECRET_SALT)

# 2. Aggregating/Storing the Anonymized Metadata
metadata_record = {
    "session_hash": person_id_hashed, # Stored: Pseudonym
    "location": location_data,
    "hour_of_day": time.strftime('%H', time.localtime(timestamp)),
    "count": 1 # Analysis is done on aggregated counts, not individual records
}

print("\n## 🔒 Pseudonymization Implementation")
print(f"Original ID: {person_id_original}")
print(f"Hashed ID (Stored): {person_id_hashed}")
print(f"Analysis Record:\n{metadata_record}")
```

Output:

```
PS C:\Users\tafse\OneDrive\Desktop\AI_coding.py> & C:\Users\tafse\anaconda3\python.exe c:/Users/tafse/OneDrive/Desktop/AI_coding.py/task2.py

## 🔒 Pseudonymization Implementation
Original ID: 458921
Hashed ID (Stored): 98c731c0d5dfa8d796cc315860fdfc129a8aa2e9f4410144320cc8a1c329c4db
Analysis Record:
{'session_hash': '98c731c0d5dfa8d796cc315860fdfc129a8aa2e9f4410144320cc8a1c329c4db', 'location': 'Public_Square_1', 'hour_of_day': '15', 'count': 1}
PS C:\Users\tafse\OneDrive\Desktop\AI_coding.py>
```

Observation:

To anonymize video analytics metadata, implement blurring on identifiable image regions like faces or license plates using OpenCV in the preprocessing pipeline, ensuring sensitive visual data is obscured before storage. Hash sensitive metadata fields such as face embeddings or IDs using secure cryptographic hash functions with salts to prevent direct identification. Combine with pseudonymization for consistent identity masking and enforce strict access controls, encryption, and automatic data retention policies to enhance privacy and compliance.

Q2: Bias in resource allocation (e.g., safety patrols).

• Task 1: Use AI to detect skews in historical allocation.

Prompt: "Analyze historical resource logs segmented by protected attribute proxies (e.g., neighborhood income level) to calculate and flag the Disparity Ratio in service delivery."

Code:

```python
task3.py > ...
import pandas as pd
def detect_historical_skew(allocation_df):
    """Simulates AI detecting resource allocation skew based on proxy data."""
    print("## 📊 Detecting Bias in Historical Allocation")

    # Calculate Patrol Hours per Capita for each neighborhood
    allocation_df['Hours_Per_Capita'] = (
        allocation_df['patrol_hours'] / allocation_df['population']
    )
    # Calculate average allocation for the two extreme groups
    low_income_data = allocation_df[allocation_df['income_level'] == 'Low']
    high_income_data = allocation_df[allocation_df['income_level'] == 'High']
    avg_low = low_income_data['Hours_Per_Capita'].mean()
    avg_high = high_income_data['Hours_Per_Capita'].mean()
    if avg_low <= 0: return # Avoid division by zero
    # Calculate Disparity Ratio
    disparity_ratio = avg_high / avg_low
    print(f"\nAverage H/C (High Income): {avg_high:.2f}")
    print(f"Average H/C (Low Income): {avg_low:.2f}")
    print(f"Disparity Ratio (High/Low): {disparity_ratio:.2f}")
    if disparity_ratio > 1.5:
        # Code flags a ratio greater than 1.5 as evidence of significant historical bias.
        print("🚨 SKEW ALERT: Disparity Ratio > 1.5.")
        print("  - High-income areas receive significantly (historically) disproportionate resources.")
    else:
        print("✅ Allocation ratio is relatively balanced.")
# Mock Historical Data
mock_allocation_data = pd.DataFrame({
    'neighborhood': ['A', 'B', 'C', 'D'],
    'income_level': ['High', 'Low', 'High', 'Low'],
    'population': [10000, 15000, 5000, 12000],
    'patrol_hours': [350, 150, 200, 100],
    'reported_risk': [0.8, 0.7, 0.5, 0.6]
})
# Run Skew Detection
detect_historical_skew(mock_allocation_data)
```

Ln 34, Col 3   Spaces: 4   UTF-8   CRLF   { } Python

Output:

```
PS C:\Users\tafse\OneDrive\Desktop\AI_coding.py> & C:\Users\tafse\anaconda3\python.exe c:/Users/tafse/OneDrive/Desktop/AI_coding.py/task3.py
## 📊 Detecting Bias in Historical Allocation

Average H/C (High Income): 0.04
Average H/C (Low Income): 0.01
Disparity Ratio (High/Low): 4.09
🚨 SKEW ALERT: Disparity Ratio > 1.5.
  - High-income areas receive significantly (historically) disproportionate resources.
```

Observation: Using AI to detect bias in resource allocation involves analysing historical data for patterns showing unequal distribution of safety patrols across areas. Look for skews where certain neighbourhoods receive disproportionately fewer or more patrols relative to crime rates or population needs. Flag anomalies indicating potential systemic bias or unfair prioritization that may perpetuate inequality or reduce effectiveness.

Task 2: Create fairness-aware allocation algorithm

Prompt: "Design an allocation algorithm that introduces an Equity Constraint (a minimum service floor) to ensure historically underserved groups receive a baseline level of resources, regardless of a potentially skewed risk prediction."

Code:

```python
task4.py > ...
import numpy as np
import pandas as pd
def fairness_aware_allocation(data_df, total_resources, min_equity_floor_pct=0.15):
    """Allocates resources based on risk, constrained by an equity floor.
    Resources are first reserved for underserved areas, then distributed by risk."""
    print("\n## ⚖️ Fairness-Aware Allocation Algorithm")
    # 1. Identify Underserved Groups (Proxy: Low Income)
    low_income_data = data_df[data_df['income_level'] == 'Low']
    other_data = data_df[data_df['income_level'] != 'Low']
    # 2. Calculate Equity Floor
    # Set a portion of resources (e.g., 15%) that must go to the underserved group.
    equity_floor_total = total_resources * min_equity_floor_pct
    if not low_income_data.empty:
        floor_per_area = equity_floor_total / len(low_income_data)
        # Initialize allocation: these areas get the minimum equity floor first.
        allocations = low_income_data['neighborhood'].apply(lambda x: floor_per_area).to_dict()
        remaining_resources = total_resources - equity_floor_total
    else:
        allocations = {}
        remaining_resources = total_resources
    print(f"  - Total Resources: {total_resources:.2f}")
    print(f"  - Equity Floor Reserved: {equity_floor_total:.2f}")
    # Remaining resources are distributed based on predicted risk for all areas not in the floor group.
    total_risk = other_data['reported_risk'].sum()
    if total_risk > 0:
        other_data['risk_weight'] = other_data['reported_risk'] / total_risk
        for index, row in other_data.iterrows():
            allocation = remaining_resources * row['risk_weight']
            # Add to the existing dictionary (updates allocation for high-income areas)
            allocations[row['neighborhood']] = allocation
    # 4. Consolidate and Output
    final_allocation = pd.Series(allocations).sort_index()
    print("\n--- Final Allocation (Hours) ---")
# Use the mock data from Task 1 (create simple mock data if not provided)
mock_allocation_data = pd.DataFrame([
    {"neighborhood": "Neighborhood A", "income_level": "Low",    "reported_risk": 0.9},
    {"neighborhood": "Neighborhood B", "income_level": "High",   "reported_risk": 0.4},
    {"neighborhood": "Neighborhood C", "income_level": "Low",    "reported_risk": 0.7},
    {"neighborhood": "Neighborhood D", "income_level": "Medium", "reported_risk": 0.2},
])
mock_data = mock_allocation_data.copy()
TOTAL_HOURS_TO_ALLOCATE = 600
# Run the fairness algorithm
fairness_aware_allocation(mock_data, TOTAL_HOURS_TO_ALLOCATE)
fairness_aware_allocation(mock_data, TOTAL_HOURS_TO_ALLOCATE)
```

Output:

```
PS C:\Users\tafse\OneDrive\Desktop\AI_coding.py> & C:\Users\tafse\anaconda3\python.exe c:/Users/tafse/OneDrive/Desktop/AI_coding.py/task4.py
Traceback (most recent call last):
  File "c:\Users\tafse\OneDrive\Desktop\AI_coding.py\task4.py", line 2, in <module>
    import pandas as pd
  File "C:\Users\tafse\anaconda3\Lib\site-packages\pandas\__init__.py", line 26, in <module>
632, in find_spec
  File "<frozen importlib._bootstrap_external>", line 152, in _path_stat
KeyboardInterrupt
PS C:\Users\tafse\OneDrive\Desktop\AI_coding.py> (C:\Users\tafse\anaconda3\shell\condabin\conda-hook.ps1) ; (conda activate base)
(base) PS C:\Users\tafse\OneDrive\Desktop\AI_coding.py>
```

Observation:

A fairness-aware allocation algorithm should balance resource distribution by incorporating demographic, geographic, and crime severity factors to avoid bias. It must use equitable metrics ensuring underserved areas receive appropriate patrols, dynamically adjusting allocations based on feedback and changing conditions. Transparent criteria and periodic audits enhance accountability and reduce systemic inequality in safety patrol deployment.