

Assignment-5.4

2303A52292

Batch-43

Task 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.

```
import hashlib
import os

def collect_user_data():
    """Collects user data and demonstrates data protection techniques."""
    name = input("Enter your name: ")
    age = input("Enter your age: ")
    email = input("Enter your email: ")

    user_data = {
        'name': name,
        'age': age,
        'email': email
    }

    print("\n--- Raw Data Collected ---")
    print(user_data)

    hashed_email = hashlib.sha256(email.encode()).hexdigest()
    print(f"Unhashed Email (for anonymization): {email}")
    print(f"Hashed Email (for anonymization): {hashed_email}")

    user_id = os.urandom(8).hex()
    print(f"Pseudonym (User ID): {user_id}")

    print("\nNote: For actual storage, 'name', 'age', 'email' would ideally be encrypted.")
    print("e.g., using libraries like 'cryptography' for symmetric/asymmetric encryption.")

    protected_data = {
        'user_id': user_id,
        'age': age,
        'hashed_email': hashed_email
    }

    print("\n--- Protected/Anonymized Data (for storage) ---")
    print(protected_data)

    print("\nCopilot Alert: Avoid storing raw sensitive data like 'name' and 'email' unencrypted.")
    print("Always apply appropriate protection measures before persistence.")
```

```
print("\nNote: For actual storage, 'name', 'age', 'email' would ideally be encrypted.")
print("e.g., using libraries like 'cryptography' for symmetric/asymmetric encryption.")

protected_data = {
    'user_id': user_id,
    'age': age,
    'hashed_email': hashed_email
}

print("\n--- Protected/Anonymized Data (for storage) ---")
print(protected_data)

print("\nCopilot Alert: Avoid storing raw sensitive data like 'name' and 'email' unencrypted.")
print("Always apply appropriate protection measures before persistence.")

collect_user_data()

--- User Data Collection ---
Enter your name: sanjana
Enter your age: 20
Enter your email: sanjanamaydan@gmail.com

--- Raw Data Collected ---
{'name': 'sanjana', 'age': '20', 'email': 'sanjanamaydan@gmail.com'}

Hashed Email (for anonymization): 14861fb5500287c509da32548778c0fe59c8088a43036c2811f4493da84300
Pseudonym (User ID): 863568c866162d4

Note: For actual storage, 'name', 'age', 'email' would ideally be encrypted.
e.g., using libraries like 'cryptography' for symmetric/asymmetric encryption.

--- Protected/Anonymized Data (for storage) ---
{'user_id': '863568c866162d4', 'age': '20', 'hashed_email': '14861fb5500287c509da32548778c0fe59c8088a43036c2811f4493da84300'}

Copilot Alert: Avoid storing raw sensitive data like 'name' and 'email' unencrypted.
Always apply appropriate protection measures before persistence.
```

Task 2:

- Ask Copilot to generate a Python function for sentiment analysis.

Then prompt Copilot to identify and handle potential biases in the data.

```

def sentiment_analysis(text):
    text = text.lower().strip()

    offensive_terms = ["stupid", "idiot", "dumb", "hate", "racist"]
    for term in offensive_terms:
        text = text.replace(term, "")

    positive_words = ["good", "happy", "excellent", "love", "great"]
    negative_words = ["bad", "sad", "terrible", "angry", "worst"]

    positive_score = sum(word in text for word in positive_words)
    negative_score = sum(word in text for word in negative_words)

    if abs(positive_score - negative_score) <= 1:
        return "Neutral"
    elif positive_score > negative_score:
        return "Positive"
    else:
        return "Negative"

if __name__ == "__main__":
    sample_text = "I love this product, it's excellent!"
    result = sentiment_analysis(sample_text)
    print(f"Sentiment: {result}")

Sentiment: Positive

```

Release notes

Please follow our [blog](#) to see more information about new features, tips and tricks, and featured notebooks such as [Analyzing a Bank Failure with Colab](#).

2026-01-20

- Launched Data Explorer - a new feature that lets you search Kaggle datasets, models, and competitions directly from a Colab notebook!
- Gemini 3 is now available in the Colab UI. Gemini 3 is also available in VS Code. Colab is also available for use in Antigravity, Cursor, and Windurst via the Open VSX Registry!
- H10 is being rolled out for more users.
- Launched a new modern design for the Colab UI.

Python package upgrades

- accelerate 1.1.0 → 1.12.0
- astropy 7.1.0 → 7.2.0
- fastapi 0.88.0 → 2.31.0
- cachetools 5.5.2 → 6.2.4
- dataproc-spark-connect 0.8.3 → 1.0.1
- diffusers 0.35.2 → 0.36.0
- flexx 0.10.0 → 0.10.1
- google 2.0.3 → 3.0.0
- google-ads 1.17.0 → 1.21.0
- google-auth 2.38.0 → 2.43.0
- google-generative-models 1.49.0 → 1.55.0
- h5py 5.4.0 → 5.50.0
- holiday 0.84 → 0.88
- humane 4.14.0 → 4.15.0
- langchain 0.3.27 → 1.2.4
- lxml 4.9.1 → 4.9.2 → 0.6.4
- lxml 5.4.0 → 5.50.0
- mlp 1.21.0 → 1.25.0
- marshalls 2.11.0 → 2.15.0
- networf 3.5 → 3.6.1

Task 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.

```

result = sentiment_analysis(sample_text)
print(f"Sentiment: {result}")

Sentiment: Positive

def recommend_products(user_history):
    recommendations = []

    # Fairness check: do not over-promote a single brand
    for product in user_history:
        if product not in recommendations:
            recommendations.append(product)

    # Transparency message
    print("Recommendations are based on your browsing history.")
    print("You can reset or customize recommendations anytime.")

    return recommendations

# Example usage
history = ["Laptop", "Headphones", "Laptop"]
print(recommend_products(history))

...
Recommendations are based on your browsing history.
You can reset or customize recommendations anytime.
['Laptop', 'Headphones']

```

Release notes

Please follow our [blog](#) to see more information about new features, tips and tricks, and featured notebooks such as [Analyzing a Bank Failure with Colab](#).

2026-01-20

- Launched Data Explorer - a new feature that lets you search Kaggle datasets, models, and competitions directly from a Colab notebook!
- Gemini 3 is now available in the Colab UI. Gemini 3 is also available in VS Code. Colab is also available for use in Antigravity, Cursor, and Windurst via the Open VSX Registry!
- H10 is being rolled out for more users.
- Launched a new modern design for the Colab UI.

Python package upgrades

- accelerate 1.1.0 → 1.12.0
- astropy 7.1.0 → 7.2.0
- fastapi 0.88.0 → 2.31.0
- cachetools 5.5.2 → 6.2.4
- dataproc-spark-connect 0.8.3 → 1.0.1
- diffusers 0.35.2 → 0.36.0
- flexx 0.10.0 → 0.10.1
- google 2.0.3 → 3.0.0
- google-ads 1.17.0 → 1.21.0
- google-auth 2.38.0 → 2.43.0
- google-generative-models 1.49.0 → 1.55.0
- h5py 5.4.0 → 5.50.0
- holiday 0.84 → 0.88
- humane 4.14.0 → 4.15.0
- langchain 0.3.27 → 1.2.4
- lxml 4.9.1 → 4.9.2 → 0.6.4
- lxml 5.4.0 → 5.50.0
- mlp 1.21.0 → 1.25.0
- marshalls 2.11.0 → 2.15.0
- networf 3.5 → 3.6.1

Task 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 Google Colab notebook titled 'oled34.ipynb'. The code in the cell is a Python script that demonstrates basic logging. It imports the 'logging' module, configures it to log to a file named 'app.log' at level INFO, and defines a 'login' function that checks if the password is 'admin123'. If it is, it prints 'Login successful' and returns True; otherwise, it prints 'Login failed' and returns False. The script then shows an example usage where it prompts for a username and password, logs them, and prints a welcome message or error based on the login result.

```
[{"L": 1, "C": "import logging"}, {"L": 2, "C": "# Configure logging"}, {"L": 3, "C": "logging.basicConfig("}, {"L": 4, "C": "filename='app.log',"}, {"L": 5, "C": "level=logging.INFO, "}, {"L": 6, "C": "format='%(asctime)s - %(levelname)s - %(message)s'"}, {"L": 7, "C": ")"}, {"L": 8, "C": "def login(username, password):"}, {"L": 9, "C": "# Copilot warning: "}, {"L": 10, "C": "# NEVER log passwords, emails, or personal identifiers"}, {"L": 11, "C": "    logging.info('Login attempt for user') # Do not log username"}, {"L": 12, "C": "    if password == 'admin123':"}, {"L": 13, "C": "        logging.info('Login successful')"}, {"L": 14, "C": "        return True"}, {"L": 15, "C": "    else: "}, {"L": 16, "C": "        logging.warning('Login failed')"}, {"L": 17, "C": "    return False"}, {"L": 18, "C": "# Example usage"}, {"L": 19, "C": "if __name__ == '__main__':"}, {"L": 20, "C": "    username = input('Enter username: ')"}, {"L": 21, "C": "    password = input('Enter password: ')"}, {"L": 22, "C": "    if login(username, password):"}, {"L": 23, "C": "        print('Welcome!')"}, {"L": 24, "C": "    else: "}, {"L": 25, "C": "        print('Invalid credentials.')"}]
```

Task 5:

- Ask Copilot to generate a machine learning model. Then, prompt

it to add documentation on how to use the model responsibly

The screenshot shows a Google Colab notebook titled 'Untitled34.ipynb'. The code in the cell is a Python script for a simple decision tree classifier. It imports 'DecisionTreeClassifier' from 'sklearn.tree'. It defines training data with features X (study hours) and labels y (Pass/Fail). The model is trained on this data. A user is prompted to enter study hours, and the model predicts whether they will pass or fail based on the input.

```
[{"L": 1, "C": "from sklearn.tree import DecisionTreeClassifier"}, {"L": 2, "C": "# Training data: study_hours"}, {"L": 3, "C": "X = [[1], [2], [3], [4], [5], [6]]"}, {"L": 4, "C": "# Labels: 0 = Fail, 1 = Pass"}, {"L": 5, "C": "y = [0, 0, 0, 1, 1, 1]"}, {"L": 6, "C": "# Create and train model"}, {"L": 7, "C": "model = DecisionTreeClassifier()"}, {"L": 8, "C": "model.fit(X, y)"}, {"L": 9, "C": "# Take user input"}, {"L": 10, "C": "hours = int(input('Enter study hours: '))"}, {"L": 11, "C": "# Predict"}, {"L": 12, "C": "prediction = model.predict([[hours]])"}, {"L": 13, "C": "# Output result"}, {"L": 14, "C": "if prediction[0] == 1: "}, {"L": 15, "C": "    print('Prediction: Pass')"}, {"L": 16, "C": "else: "}, {"L": 17, "C": "    print('Prediction: Fail')"}]
```