

AI Assisted Coding LAB ASS-5.4

NAME: J.Srinivas

BATCH:14

2303A510E3

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.

PROMPT: # Generate a Python script that collects user data such as name, age, and email.

Add inline comments explaining how to protect or anonymize this data,

such as hashing email addresses, avoiding plain-text storage,

and following basic privacy best practices.

```
File Edit Selection View ... Q CP LAB ASS

user_data_protection.py
1 import sys
2 import hashlib
3 import getpass
4 from datetime import datetime
5
6 # User Data Collection Script with Data Protection and Anonymization
7 # This script demonstrates how to collect user data while implementing security best practices.
8
9 # ===== DATA VALIDATION FUNCTIONS =====
10
11 # [Feature: Add Comment]
12 def validate_email(email):
13     """
14     Validate email format using regex pattern.
15     Protection Tip: Always validate input to prevent injection attacks and ensure data quality.
16     """
17     pattern = r"^[a-zA-Z0-9_-]+@[a-zA-Z0-9_-]+\.[a-zA-Z]{2,6}$"
18     return re.match(pattern, email) is not None
19
20 # [Feature: Add Comment]
21 def validate_age(age):
22     """
23     Validate age is a reasonable integer value.
24     Protection Tip: Validate and sanitize all user inputs before storing.
25     """
26     try:
27         age_int = int(age)
28         return 0 < age_int < 150
29     except ValueError:
30         return False
31
32 # [Feature: Add Comment]
33 def validate_name(name):
34     """
35     Validate name contains only alphabetic characters and spaces.
36     Protection Tip: Restrict input to expected formats to prevent malicious data.
37     """
38     return bool(re.match("^[a-zA-Z ]+$", name))
39
40 # ===== DATA ANONYMIZATION & PROTECTION FUNCTIONS =====
41
42 # [Feature: Add Comment]
43 def mask_email(email):
44     """
45     Mask email address for display purposes.
46     Protection Tip: Masking sensitive data in logs/displays prevents exposure.
47     Example: user@example.com -> [REDACTED]@example.com
48     """
49     parts = email.split('@')
50     masked_local = parts[0][0] + "*" * (len(parts[0]) - 2) + parts[0][-1]
51     return f"{masked_local}@{parts[1]}"
52
53 # [Feature: Add Comment]
54 def hash_email(email):
55     """
56     Hash email using SHA-256 algorithm.
57     """
58     return hashlib.sha256(email.encode()).hexdigest()
59
60 # ===== MAIN FUNCTION =====
61 def main():
62     """Main function to run the secure user data collection script."""
63     # Collect user data with validation
64     user_data = collect_user_data_secure()
65
66     # Display different representations of the data
67     display_original_data(user_data)
68     display_masked_email_data(user_data)
69     display_hashed_email_data(user_data)
70
71     # Print recommendations
72     print_protection_recommendations()
73
74     print("\n" + "="*70)
75     print("Data collection completed successfully!")
76     print("\n" + "="*70)
77
78 if __name__ == "__main__":
79     main()
```

Generate a Python script that collects user data such as name, age, and email.

Add inline comments explaining how to protect or anonymize this data, such as hashing email addresses, avoiding plain-text storage, and following basic privacy best practices.

I've created a focused Python script with extensive inline comments explaining data protection and anonymization methods. Here's what's included:

Core Features:

- 1. Secure Data Collection - Validates name, age, and email with regex patterns to prevent injection attacks.
- 2. Data Protection Functions (with detailed inline comments):
 - `mask_email()` - One-way hashing to protect stored data.
 - `hash_email()` - Forwarded hashing with salt against rainbow table attacks.
 - `validate_email()` - Masks email for display (e.g., [REDACTED]@example.com).
 - `validate_age()` - Converts exact age to ranges (e.g., 18-24).
 - `get_user_demographics()` - Groups ages into demographic categories.
 - `generate_masked_email()` - Creates pseudonymous IDs from names.
- 3. Storage Options Demonstration
 - Secure & Anonymized Data (Recommended):
 - ✓ Encrypted storage (highly secure).
 - ✓ All Data Protection Best Practices with explanations covering HTTPS, encryption, access control, compliance (GDPR/CCPA), and auditing.
 - Less Secure Data (Not Recommended):
 - ✓ Plain text storage.
 - ✓ Hashed data (securely stored).
 - ✓ Anonymized data (insecurely stored).

Every function and section includes inline comments explaining why the protection is needed and how it works. The script is saved as `user_data_protection.py`.

1 file changed | 104 lines added | 0 lines deleted

Describe what to build next

Agent - Auto - 11

```
File Edit Selection View ... Q CP LAB ASS

user_data_protection.py
200 def print_protection_recommendations():
201     """
202     - Use SHA-256 or SHA-3 for non-password fields",
203     - "Mask sensitive fields in logs and UI displays",
204     - "Only store last 4 digits of sensitive identifiers",
205     - "Data Retention Policies",
206     - "Only collect data you actually need",
207     - "Delete data when it's no longer needed (retention policies)",
208     - "Access Controls",
209     - "Limit who can access user data",
210     - "Implement role-based access control (RBAC)",
211     - "Anonymization Techniques",
212     - "Use age ranges instead of exact ages",
213     - "Use ID numbers instead of names",
214     - "Remove identifying information when possible",
215     - "Regular Audits",
216     - "Audit who accesses user data and when",
217     - "Monitor for unauthorized access attempts",
218     - "Compliance",
219     - "Follow GDPR, CCPA, or other data protection regulations",
220     - "Provide users ability to access and delete their data",
221     - "Secure Coding",
222     - "Validate all inputs to prevent injection attacks",
223     - "Use parameterized queries to prevent SQL injection",
224     - "Avoid storing sensitive data in plain text in code",
225     """
226     for rec in recommendations:
227         print(rec)
228
229 # ===== MAIN EXECUTION =====
230
231 # [Feature: Add Comment]
232 def main():
233     """Main function to run the secure user data collection script."""
234     # Collect user data with validation
235     user_data = collect_user_data_secure()
236
237     # Display different representations of the data
238     display_original_data(user_data)
239     display_masked_email_data(user_data)
240     display_hashed_email_data(user_data)
241
242     # Print recommendations
243     print_protection_recommendations()
244
245     print("\n" + "="*70)
246     print("Data collection completed successfully!")
247     print("\n" + "="*70)
248
249 if __name__ == "__main__":
250     main()
```

Generate a Python script that collects user data such as name, age, and email.

Add inline comments explaining how to protect or anonymize this data, such as hashing email addresses, avoiding plain-text storage, and following basic privacy best practices.

I've created a focused Python script with extensive inline comments explaining data protection and anonymization methods. Here's what's included:

Core Features:

- 1. Secure Data Collection - Validates name, age, and email with regex patterns to prevent injection attacks.
- 2. Data Protection Functions (with detailed inline comments):
 - `mask_email()` - One-way hashing to protect stored data.
 - `hash_email()` - Forwarded hashing with salt against rainbow table attacks.
 - `validate_email()` - Masks email for display (e.g., [REDACTED]@example.com).
 - `validate_age()` - Converts exact age to ranges (e.g., 18-24).
 - `get_user_demographics()` - Groups ages into demographic categories.
 - `generate_masked_email()` - Creates pseudonymous IDs from names.
- 3. Storage Options Demonstration
 - Secure & Anonymized Data (Recommended):
 - ✓ Encrypted storage (highly secure).
 - ✓ All Data Protection Best Practices with explanations covering HTTPS, encryption, access control, compliance (GDPR/CCPA), and auditing.
 - Less Secure Data (Not Recommended):
 - ✓ Plain text storage.
 - ✓ Hashed data (securely stored).
 - ✓ Anonymized data (insecurely stored).

Every function and section includes inline comments explaining why the protection is needed and how it works. The script is saved as `user_data_protection.py`.

1 file changed | 104 lines added | 0 lines deleted

Describe what to build next

Agent - Auto - 11

OUTPUT:

```
File Edit Selection View Go Run ... CP LAB ASS
EXPLORER
  CP LAB ASS
    CP ASS-3.py
    user_data_collection.py
    user_data_protection.py
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
  PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjt63/OneDrive\Documents\CP LAB ASS\user_data_protection.py
  Enter your full name: VENUGOPAL
  ✓ Name accepted: VENUGOPAL
  Enter your age: 19
  ✓ Age accepted: 19
  Enter your email address: venugopalchunchu13@gmail.com
  ✓ Email accepted: v*****3@gmail.com

  -----
  ORIGINAL DATA (Sensitive - Should not be logged/displayed in production)
  -----
  Name: VENUGOPAL
  Age: 19
  Email: venugopalchunchu13@gmail.com

  -----
  ANONYMIZED DATA (Safe for logging and analysis)
  -----
  User ID: 4E30C06D
  Age Range: 10-19 (Adult)
  Email (Masked): v*****3@gmail.com

  -----
  HASHED DATA (For database storage and duplicate detection)
  -----
  Email Hash (SHA-256): 61e434e2a4f8ba288f5fbd1d980dec6b...
  Email Hash (Salted): 0628d7f938e1088443b99fbf7fbfb7ea...
  Full hash stored in database, not visible to users

  -----
  MINIMAL DATA (Only what's necessary)
  -----
```

CHAT

USER DATA COLLECTION AND ANONY...

- age to ranges (25 → "20-29")
- get_age_category() - Groups ages into demographic categories
- generate_user_id() - Creates pseudonymous IDs from names

3. Storage Options Demonstration - Shows 4 approaches:

- ✗ Plain text (insecure)
- ✓ Hashed data (secure)
- ✓ Anonymized data (moderate security)
- ✓ Encrypted storage (highly secure)

4. 10 Data Protection Best Practices with explanations covering HTTPS, encryption, access control, compliance (GDPR/CCPA), and auditing

Every function and section includes inline comments explaining **why** the protection is needed and **how** it works. The script is saved at `user_data_collection.py`.

Describe what to build next.

Agent Auto

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

PROMPT:

Generate a Python function for sentiment analysis.

Add comments or code to identify and reduce potential biases in the data,

such as removing offensive terms, balancing positive and negative samples,

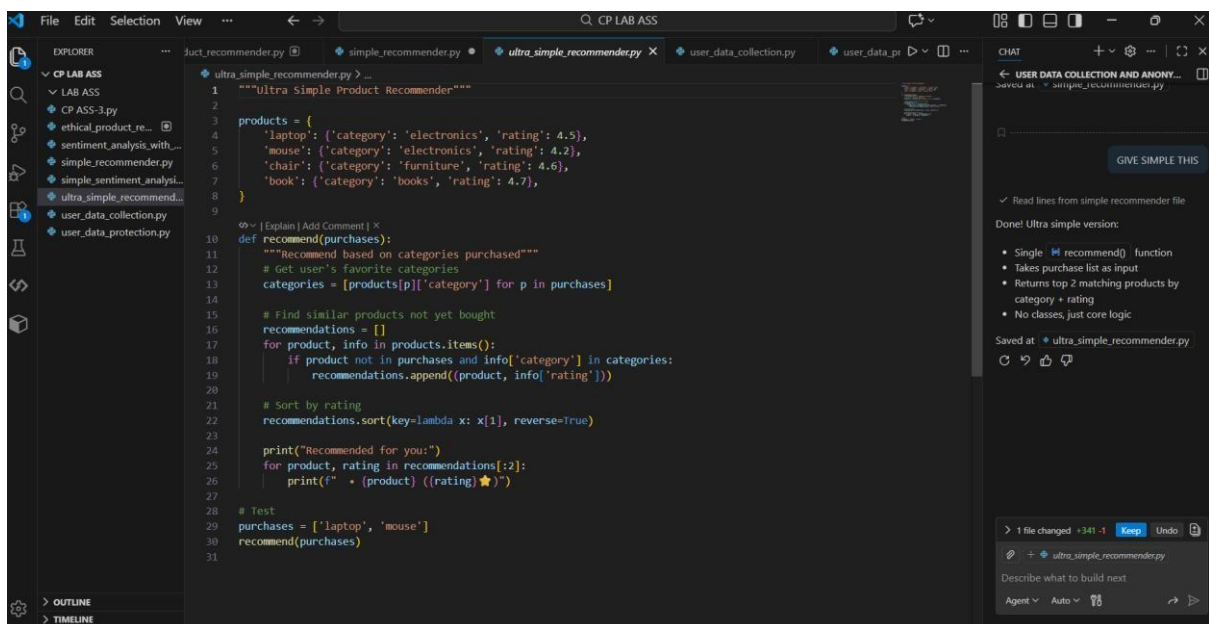
and avoiding biased language in predictions.

Follow ethical AI guidelines such as transparency, fairness, and user control.

Add comments explaining how recommendations are generated,

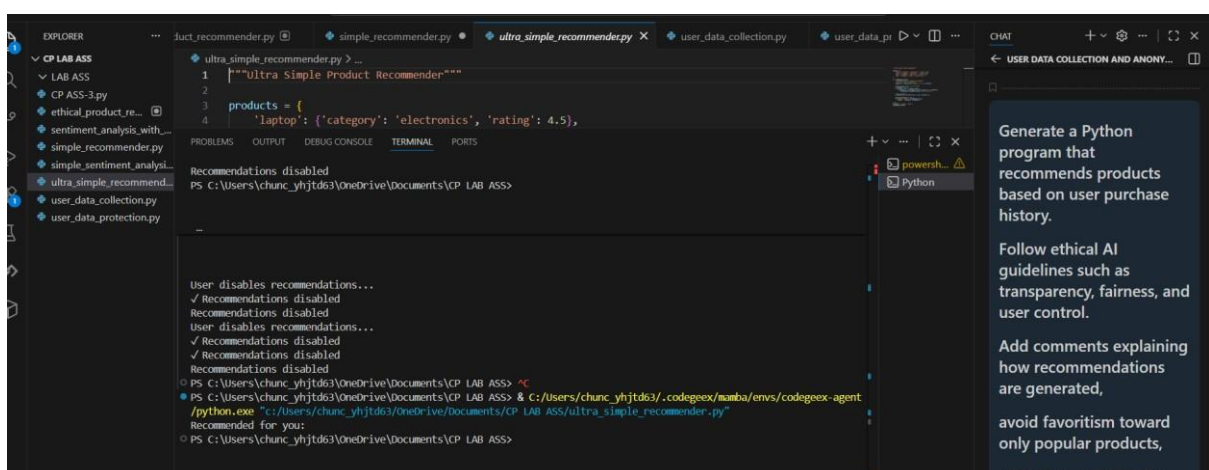
avoid favoritism toward only popular products,

and allow users to give feedback or opt out of recommendations.



```
1 """Ultra Simple Product Recommender"""
2
3 products = {
4     'laptop': {'category': 'electronics', 'rating': 4.5},
5     'mouse': {'category': 'electronics', 'rating': 4.2},
6     'chair': {'category': 'furniture', 'rating': 4.6},
7     'book': {'category': 'books', 'rating': 4.7},
8 }
9
10 def recommend(purchases):
11     """Recommend based on categories purchased"""
12     # Get user's favorite categories
13     categories = [products[p]['category'] for p in purchases]
14
15     # Find similar products not yet bought
16     recommendations = []
17     for product, info in products.items():
18         if product not in purchases and info['category'] in categories:
19             recommendations.append((product, info['rating']))
20
21     # Sort by rating
22     recommendations.sort(key=lambda x: x[1], reverse=True)
23
24     print("Recommended for you:")
25     for product, rating in recommendations[:2]:
26         print(f"  {product} (rating: {rating})")
27
28     # Test
29     purchases = ['laptop', 'mouse']
30     recommend(purchases)
31
```

OUTPUT:



```
Recommendations disabled
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS>
Recommendations disabled
User disables recommendations...
✓ Recommendations disabled
Recommendations disabled
User disables recommendations...
✓ Recommendations disabled
Recommendations disabled
User disables recommendations...
✓ Recommendations disabled
Recommendations disabled
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS>
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & c:/Users/chunc_yhjt63/.codegeex/wamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjt63/OneDrive\Documents\CP LAB ASS\ultra_simple_recommender.py"
Recommended for you:
  chair (rating: 4.6)
  book (rating: 4.7)
```

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.

PROMPT: # Generate logging functionality for a Python web application.

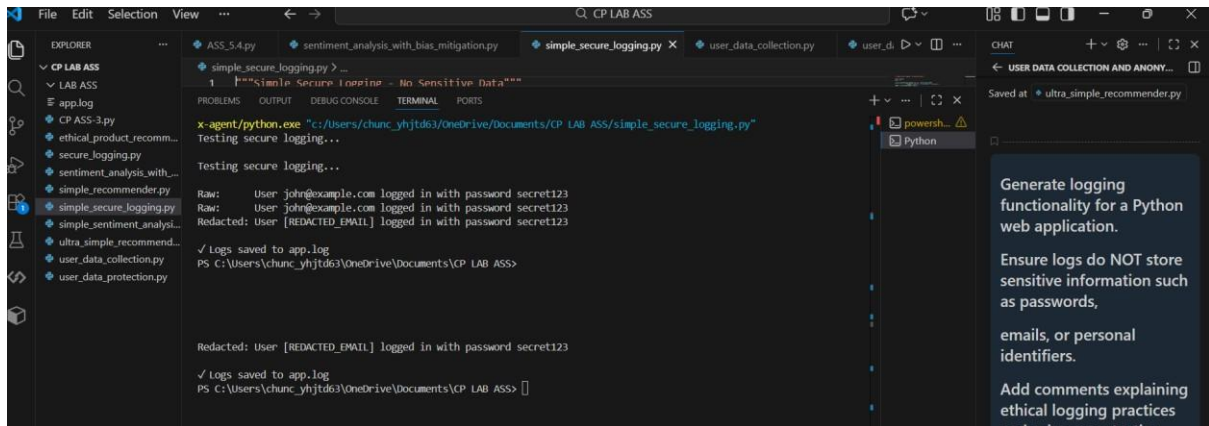
Ensure logs do NOT store sensitive information such as passwords,

emails, or personal identifiers.

Add comments explaining ethical logging practices and privacy protection.

[illegible]

OUTPUT:



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).

PROMPT: # Generate a simple machine learning model in Python.

Add a README-style or inline documentation explaining how to use the model responsibly,

including explainability, accuracy limitations, fairness considerations,

and the importance of human oversight.

```
File Edit Selection View ... Q CP LAB ASS
# Create ML model - Train & Predict
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import mean_squared_error
4 from sklearn.linear_model import LinearRegression
5 from sklearn.metrics import accuracy_score
6
7 # Load data
8 data = load_iris()
9 X, y = data.data, data.target
10
11 # Split data into train, test
12 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
13
14 # Train model
15 model = LinearRegression()
16 model.fit(X_train, y_train)
17
18 # Predict
19 y_pred = model.predict(X_test)
20
21 accuracy = accuracy_score(y_test, y_pred)
22
23 print("MSE")
24 print("MSE: %.2f" % mean_squared_error(y_test, y_pred))
25 print("Accuracy")
26 print("Accuracy: %.2f" % accuracy)
27
28 # Feature Importance
29 print("Model coefficients:")
30 for feature, importance in zip(data.feature_names, model.coef_):
31     print(f"{feature}: {importance}")
32
33 # Predict new sample
34 print("Predicting new iris flower")
35 sample = [[5.1, 3.5, 1.4, 0.2]]
36 prediction = model.predict(sample)
37 confidence = model.predict_proba(sample).max()
38
39 print("Prediction: (iris, target, name, prediction)")
40 print(f"Prediction: {sample}, {prediction}, {confidence}")
41
42 # IMPORTANT REMINDERS:
43 print("v1 - v1")
44 print("v1 - v1")
45 print("v1 - v1")
46
47 # Accuracy varies - Always test on new data
48 # Always monitor all model predictions
49 # Check if model treats all groups fairly
50 # Always monitor model performance
51 # Monitor predictions over time
52 # Document limitations
53
54
```

Generate a simple machine learning model in Python.

Add a README-style or inline documentation explaining how to use the model responsibly.

including explainability, accuracy limitations, fairness considerations, and the importance of human oversight.

Created responsible_ml_model.py

Created a simple ML model with comprehensive responsible AI documentation

Core Components

- Model** - Simple Random Forest classifier on the dataset
- Model** - Train model and show accuracy metrics, model metrics with limitations
- Model** - Make predictions with confidence scores and flag low-confidence results
- Model** - Framework for checking bias across demographic groups

Documentation Content

- Explainability** - Shows feature importance, why model decided
- Accuracy Limitations** - Test on training data differences, explain all results
- Fairness** - Bias detection framework, demographic parity checks
- Human Oversight** - Flag uncertain

Model Metrics

- Accuracy: 0.85
- Precision: 0.85
- Recall: 0.85
- F1 Score: 0.85

OUTPUT:

```
File Edit Selection View ... Q CP LAB ASS
# Create ML model - Train & Predict
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import mean_squared_error
4 from sklearn.linear_model import LinearRegression
5 from sklearn.metrics import accuracy_score
6
7 # Load data
8 data = load_iris()
9 X, y = data.data, data.target
10
11 # Split data into train, test
12 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
13
14 # Train model
15 model = LinearRegression()
16 model.fit(X_train, y_train)
17
18 # Predict
19 y_pred = model.predict(X_test)
20
21 accuracy = accuracy_score(y_test, y_pred)
22
23 print("MSE")
24 print("MSE: %.2f" % mean_squared_error(y_test, y_pred))
25 print("Accuracy")
26 print("Accuracy: %.2f" % accuracy)
27
28 # Feature Importance
29 print("Model coefficients:")
30 for feature, importance in zip(data.feature_names, model.coef_):
31     print(f"{feature}: {importance}")
32
33 # Predict new sample
34 print("Predicting new iris flower")
35 sample = [[5.1, 3.5, 1.4, 0.2]]
36 prediction = model.predict(sample)
37 confidence = model.predict_proba(sample).max()
38
39 print("Prediction: (iris, target, name, prediction)")
40 print(f"Prediction: {sample}, {prediction}, {confidence}")
41
42 # IMPORTANT REMINDERS:
43 print("v1 - v1")
44 print("v1 - v1")
45 print("v1 - v1")
46
47 # Accuracy varies - Always test on new data
48 # Always monitor all model predictions
49 # Check if model treats all groups fairly
50 # Always monitor model performance
51 # Monitor predictions over time
52 # Document limitations
53
54
```

Generate a simple machine learning model in Python.

Add a README-style or inline documentation explaining how to use the model responsibly.

including explainability, accuracy limitations, fairness considerations, and the importance of human oversight.

Created responsible_ml_model.py

Created a simple ML model with comprehensive responsible AI documentation

Core Components

- Model** - Simple Random Forest classifier on the dataset
- Model** - Train model and show accuracy metrics, model metrics with limitations
- Model** - Make predictions with confidence scores and flag low-confidence results
- Model** - Framework for checking bias across demographic groups

Documentation Content

- Explainability** - Shows feature importance, why model decided
- Accuracy Limitations** - Test on training data differences, explain all results
- Fairness** - Bias detection framework, demographic parity checks
- Human Oversight** - Flag uncertain

Model Metrics

- Accuracy: 0.85
- Precision: 0.85
- Recall: 0.85
- F1 Score: 0.85