

AI ASSISTED CODING:

ASSIGNMENT(5.4)

2403A51246

B praneeth

Batch-11

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.

Expected Output #1:

- A script with inline Copilot-suggested code and comments explaining how to safeguard or anonymize user information (e.g., hashing emails, not storing data unencrypted)

Github copilot code:

```
1  # This script collects user data: name, age, and email.
2  # Copilot inline comments guide how to anonymize and protect user information.
3
4  import hashlib
5
6  def hash_email(email):
7      # Hash the email using SHA-256 to anonymize it before storage.
8      # Never store raw emails in logs or databases if privacy is required.
9      return hashlib.sha256(email.encode()).hexdigest()
10
11  def collect_user_data():
12      name = input("Enter your name: ")
13      age = input("Enter your age: ")
14      email = input("Enter your email: ")
15
16      # Protect user data:
17      # - Avoid storing plain text name/email if possible.
18      # - Use hashed email for identification, not for direct contact.
19      # - Only collect necessary fields (minimize exposure).
20      # - Encrypt data at rest if storing in files/databases.
21      anonymized_email = hash_email(email)
22
23      user_data = {
24          "name": name, # For full anonymity, do not store real names.
25          "age": age,   # Consider storing age ranges instead of exact value.
26          "email_hash": anonymized_email # Store only the hashed email.
27      }
28
29      print("Anonymized user data:", user_data)
30      # If saving to disk, use file encryption libraries (e.g., cryptography, Fernet).
31      # Always comply with privacy laws (e.g., GDPR) when handling personal info.
32
33  if __name__ == "__main__":
34      collect_user_data()
```

Copilot protection notes (inline):

- Hash emails before storage to anonymize.
- Encrypt files/databases containing user data.
- Avoid storing raw names/emails.
- When possible, store minimum necessary info (e.g., age range instead of exact age).
- Always follow privacy best practices and legal requirements.

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. Expected Output #2:
- Copilot-generated code with additions or comments addressing bias mitigation strategies (e.g., balancing dataset, removing offensive terms):

Copilot code:

```
1  # This script provides a sentiment analysis function.
2  # Copilot inline comments explain ways to identify and handle bias in the data.
3  from textblob import TextBlob
4  def analyze_sentiment(text):
5      # Analyzes the sentiment polarity of the input text.
6      # Returns 'positive', 'neutral', or 'negative'.
7      blob = TextBlob(text)
8      polarity = blob.sentiment.polarity
9      if polarity > 0.1:
10         return "positive"
11     elif polarity < -0.1:
12         return "negative"
13     else:
14         return "neutral"
15     # Bias mitigation strategies:
16     # - Balance the dataset: Ensure training data contains equal samples of positive, neutral, and negative sentiments.
17     # - Remove or flag offensive terms: Preprocess input to detect and mask hate speech or slurs.
18     # - Audit for demographic bias: Check if model accuracy varies for different groups and retrain if needed.
19     # - Regularly review and update labeled data to reflect evolving language and social norms.
20     def preprocess_text(text):
21         # Example: Remove offensive terms using a predefined list.
22         offensive_terms = ['term1', 'term2'] # Populate with actual terms
23         for term in offensive_terms:
24             text = text.replace(term, '[REDACTED]')
25         return text
26     def safe_analyze_sentiment(text):
27         # Preprocess to mitigate bias and offensive content.
28         clean_text = preprocess_text(text)
29         return analyze_sentiment(clean_text)
30
31     # For best results, retrain models on balanced and representative data.
32     # Use explainable AI tools to identify sources of bias in predictions.
33
34     if __name__ == "__main__":
35         user_input = input("Enter text for sentiment analysis: ")
36         result = safe_analyze_sentiment(user_input)
37         print("Sentiment:", result)
```

Task Description 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.

Expected Output #3:

- Copilot suggestions that include explanations, fairness checks (e.g., avoiding favoritism), and user feedback options in the code:

Copilot Code:

```
1  # Short ethical product recommender with fairness, transparency, and user feedback.
2
3  from collections import Counter
4
5  def recommend_products(user_history, all_products):
6      # Recommend top product plus one new for fairness.
7      top = Counter(user_history).most_common(1)
8      new_items = [p for p in all_products if p not in user_history]
9      result = [top[0][0]] if top else []
10     if new_items:
11         result.append(new_items[0])
12     # Explain choices
13     reasons = [f'{result[0]}' matches your history" if result else "",
14               f'{result[1]}' adds diversity" if len(result) > 1 else ""]
15     return result, reasons
16
17 def feedback(recommendations):
18     print("Recommended:", recommendations)
19     resp = input("Feedback (yes/no/suggest): ")
20     if resp == "suggest":
21         alt = input("Suggest a product: ")
22         print(f"Thanks! '{alt}' noted.")
23
24 if __name__ == "__main__":
25     history = ['A', 'B', 'A']
26     products = ['A', 'B', 'C']
27     recs, why = recommend_products(history, products)
28     print("\n".join(why))
29     feedback(recs)
```

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.

Expected Output #4:

- Logging code that avoids saving personal identifiers (e.g., passwords, emails), and includes comments about ethical logging practices:

Copilot Code:

```
1  # Logging functionality for a Python web app.
2  # Copilot inline comments explain how to avoid logging sensitive info (e.g., passwords, emails).
3
4  import logging
5
6  # Configure logging
7  logging.basicConfig(
8      filename='app.log',
9      level=logging.INFO,
10     format='%(asctime)s %(levelname)s %(message)s'
11 )
12
13 def log_event(event, details):
14     """
15     Log an event with details.
16     Ethical logging practice:
17     - NEVER log sensitive info (passwords, emails, personal data).
18     - Scrub details before logging.
19     """
20     # Remove sensitive keys if present
21     sensitive_keys = ['password', 'email', 'token']
22     filtered_details = {k: v for k, v in details.items() if k not in sensitive_keys}
23     logging.info(f"{event}: {filtered_details}")
24
25 # Example usage
26 def user_login(username, password):
27     # NEVER log raw passwords or emails.
28     log_event('user_login_attempt', {'username': username}) # Only log non-sensitive info
29
30 def password_reset(email):
31     # Do NOT log the email address.
32     log_event('password_reset_attempt', {}) # Log only the event, not the identifier
33
34 # Ethical notes:
35 # - Avoid logging personal identifiers.
36 # - Regularly review logs for accidental sensitive data.
37 # - Use log rotation and secure storage to prevent leaks.
38 # - Comply with privacy regulations (e.g., GDPR).
```

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).

Expected Output #5:

- Copilot-generated model code with a README or inline documentation suggesting responsible usage, limitations, and fairness considerations:

Copilot Code:

```
1  # Simple machine learning model with responsible usage documentation.
2  # Copilot inline comments explain explainability, accuracy limits, and fairness.
3
4  from sklearn.datasets import load_iris
5  from sklearn.model_selection import train_test_split
6  from sklearn.ensemble import RandomForestClassifier
7
8  def train_model():
9      # Load dataset (Iris for demonstration; review for bias before production use)
10     data = load_iris()
11     X_train, X_test, y_train, y_test = train_test_split(
12         data.data, data.target, test_size=0.2, random_state=42)
13     model = RandomForestClassifier()
14     model.fit(X_train, y_train)
15     accuracy = model.score(X_test, y_test)
16     print(f"Test Accuracy: {accuracy:.2f}")
17     return model
18
19 if __name__ == "__main__":
20     model = train_model()
21     # Responsible usage notes:
22     # - Explainability: Use model.feature_importances_ to understand which features matter most.
23     # - Accuracy limits: Always validate accuracy on representative data; don't overstate performance.
24     # - Fairness: Check for bias in training data; don't deploy without fairness review.
25     # - Never use predictions for critical decisions without human oversight.
26     # - Document intended uses and known limitations.
```