

Name:A.Shashidhar

H.No:2303A51798

Batch:26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju	
CourseCode	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	Week3 –	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches
Assignment Number: 5.4(Present assignment number)/24(Total number of assignments)			
Q.No.	Question		Expected Time to complete
1	Lab 5: Ethical Foundations – Responsible AI Coding Practices		Week3 -

	<p>Lab Objectives:</p> <ul style="list-style-type: none">• To explore the ethical risks associated with AI-generated code.• To recognize issues related to security, bias, transparency, and copyright.• To reflect on the responsibilities of developers when using AI tools in software development.• To promote awareness of best practices for responsible and ethical AI coding. <p>Lab Outcomes (LOs):</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none">• Identify and avoid insecure coding patterns generated by AI tools.• Detect and analyze potential bias or discriminatory logic in AI-generated outputs.• Evaluate originality and licensing concerns in reused AI-generated code.• Understand the importance of explainability and transparency in AI-assisted programming.• Reflect on accountability and the human role in ethical AI coding practices. <p>Task Description #1:</p> <ul style="list-style-type: none">• 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. <p>Expected Output #1:</p> <ul style="list-style-type: none">• 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).

```
import hashlib
import uuid

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

    user_id = str(uuid.uuid4())
    hashed_email = hashlib.sha256(email.encode()).hexdigest()

    user_record = {
        "user_id": user_id,
        "age": age,
        "email_hash": hashed_email
    }

    print(f"Data Collected Securely: {user_record}")

collect_user_data()

*** Enter your name: A.Shashidhar
Enter your age: 20
Enter your email: shashidharashadapu348@sru.edu.in
Data Collected Securely: {'user_id': 'b6fc95ee-8bd6-4632-be94-85878f54eb16', 'age': '20', 'email_hash': '85857269428d9275a364f0bb82'}
```

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).

```
| import re  
|  
| def get_sentiment_score(text):  
|     positive_words = {"happy", "excellent", "great", "love", "good", "best", "positive"}  
|     negative_words = {"bad", "awful", "terrible", "hate", "worst", "negative", "disappointed"}  
|  
|     words = re.findall(r'\b\w+\b', text.lower())  
|     score = 0  
|  
|     for word in words:  
|         if word in positive_words:  
|             score += 1  
|         elif word in negative_words:  
|             score -= 1  
|  
|     if score > 0:  
|         return "Positive"  
|     elif score < 0:  
|         return "Negative"  
|     else:  
|         return "Neutral"  
|  
| def mitigated_sentiment_analysis(text):  
|     neutral_terms = {"he", "she", "they", "man", "woman", "old", "young"}  
|     words = re.findall(r'\b\w+\b', text.lower())  
|     clean_words = [w for w in words if w not in neutral_terms]  
|     clean_text = " ".join(clean_words)  
|  
|     return get_sentiment_score(clean_text)  
|  
| # Example usage  
| print(mitigated_sentiment_analysis("The old man was excellent at his craft."))  
|  
| ... Positive
```

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.

```

❶ import random

class EthicalRecommender:
    def __init__(self, product_catalog):
        self.catalog = product_catalog
        self.user_history = {}
        self.feedback_log = []

    def recommend(self, user_id, history):
        self.user_history[user_id] = history
        interests = set(history)

        personalized = [p for p in self.catalog if p['category'] in interests]
        discovery = [p for p in self.catalog if p['category'] not in interests]

        recommendations = random.sample(personalized, min(len(personalized), 3))
        recommendations += random.sample(discovery, min(len(discovery), 2))

        return recommendations

    def get_explanation(self, product, user_id):
        is_personalized = product['category'] in self.user_history.get(user_id, [])
        if is_personalized:
            return f"Because you showed interest in {product['category']}."
        return "Broadening your horizons with a new category."

    def log_feedback(self, user_id, product_name, liked):
        self.feedback_log.append({"user": user_id, "item": product_name, "liked": liked})
        return "System updated with user preference."

catalog = [
    {"name": "Smartphone", "category": "Tech"},
    {"name": "Tablet", "category": "Tech"},
    {"name": "Dumbbells", "category": "Fitness"},
    {"name": "Running Shoes", "category": "Fitness"},
    {"name": "Novel", "category": "Books"}
]

```

```

catalog = [
    {"name": "Smartphone", "category": "Tech"},
    {"name": "Tablet", "category": "Tech"},
    {"name": "Dumbbells", "category": "Fitness"},
    {"name": "Running Shoes", "category": "Fitness"},
    {"name": "Novel", "category": "Books"}
]

engine = EthicalRecommender(catalog)
user_id = "User_01"
history = ["Tech"]

results = engine.recommend(user_id, history)

for item in results:
    reason = engine.get_explanation(item, user_id)
    print(f"Item: {item['name']} | Explanation: {reason}")

print(engine.log_feedback(user_id, "Novel", True))

*** Item: Tablet | Explanation: Because you showed interest in Tech.
Item: Smartphone | Explanation: Because you showed interest in Tech.
Item: Novel | Explanation: Broadening your horizons with a new category.
Item: Running Shoes | Explanation: Broadening your horizons with a new category.
System updated with user preference.

```

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.

```
  import logging
  import re

  class SensitiveDataFilter(logging.Filter):
      def filter(self, record):
          message = str(record.msg)

          email_pattern = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
          password_pattern = r'(password|passwd|pwd|secret|token)\s*[:=]\s*["\']?.*["\']?'

          message = re.sub(email_pattern, "[EMAIL_REDACTED]", message)
          message = re.sub(password_pattern, r'\1: [REDACTED]", message, flags=re.IGNORECASE)

          record.msg = message
          return True

  logger = logging.getLogger("SecureApp")
  logger.setLevel(logging.INFO)

  handler = logging.FileHandler("app_activity.log")
  handler.addFilter(SensitiveDataFilter())

  formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s")
  handler.setFormatter(formatter)
  logger.addHandler(handler)

  def log_user_action(action, user_data):
      log_message = f"Action: {action} | Data: {user_data}"
      logger.info(log_message)

  # Example usage
  log_user_action("Login Attempt", "user: admin@example.com, password: SuperSecret123")
  log_user_action("Profile Update", "Updating email to test@domain.org")
  ...
  INFO:SecureApp:Action: Login Attempt | Data: user: [EMAIL_REDACTED], password: [REDACTED]
  INFO:SecureApp:Action: Profile Update | Data: Updating email to [EMAIL_REDACTED]
```

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.

```
[19] ✓ 0s
● from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import pandas as pd
import joblib

def train_responsible_model(data_path, target_column):
    df = pd.read_csv(data_path)

    X = df.drop(columns=[target_column])
    y = df[target_column]

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)

    predictions = model.predict(X_test)
    report = classification_report(y_test, predictions, output_dict=True)

    joblib.dump(model, 'responsible_model.pkl')

    return model, report

def get_feature_importance(model, feature_names):
    importance = model.coef_[0]
    feature_importance = dict(zip(feature_names, importance))
    return sorted(feature_importance.items(), key=lambda x: abs(x[1]), reverse=True)
```

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots