LAB ASSIGNMENT-5.4

2303A51337

BATCH-10

➢ TASK-1:

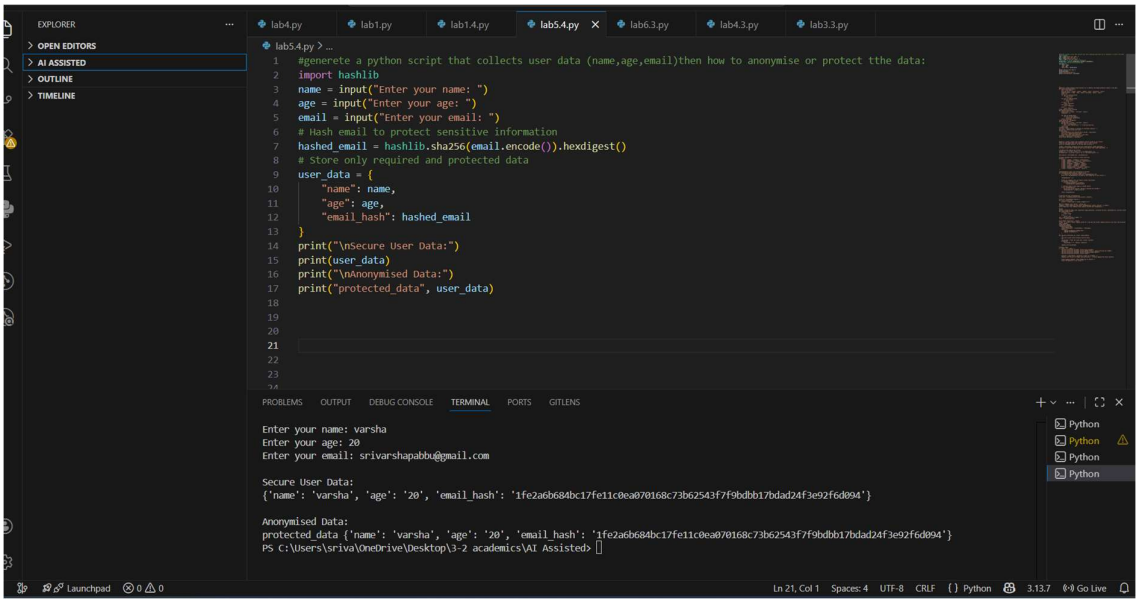
PROMPT:

Generete a python script that collects user data (name,age,email)then how to anonymise or protect tthe data:

CODE:

```
import hashlib

name = input("Enter your name: ")

age = input("Enter your age: ")

email = input("Enter your email: ")

# Hash email to protect sensitive information

hashed_email = hashlib.sha256(email.encode()).hexdigest()

# Store only required and protected data

user_data = {

  "name": name,

  "age": age,

  "email_hash": hashed_email

}

print("\nSecure User Data:")

print(user_data)

print("\nAnonymised Data:")

print("protected_data", user_data)
```

OUTPUT:



EXPLANATION:

- This program collects a user's name, age, and email as input.
- To protect sensitive information, the email is converted into a hashed value using the algorithm from the hashlib library, so the real email is not stored directly.
- The program then stores only the necessary and protected data in a dictionary, replacing the email with its hash.
- Finally, it prints the secure user data and shows the anonymised protected version, demonstrating how personal data can be safely stored.

❖ TASK-2

PROMPT:

Generate a python program using function for to identify and handle potential biases in the data

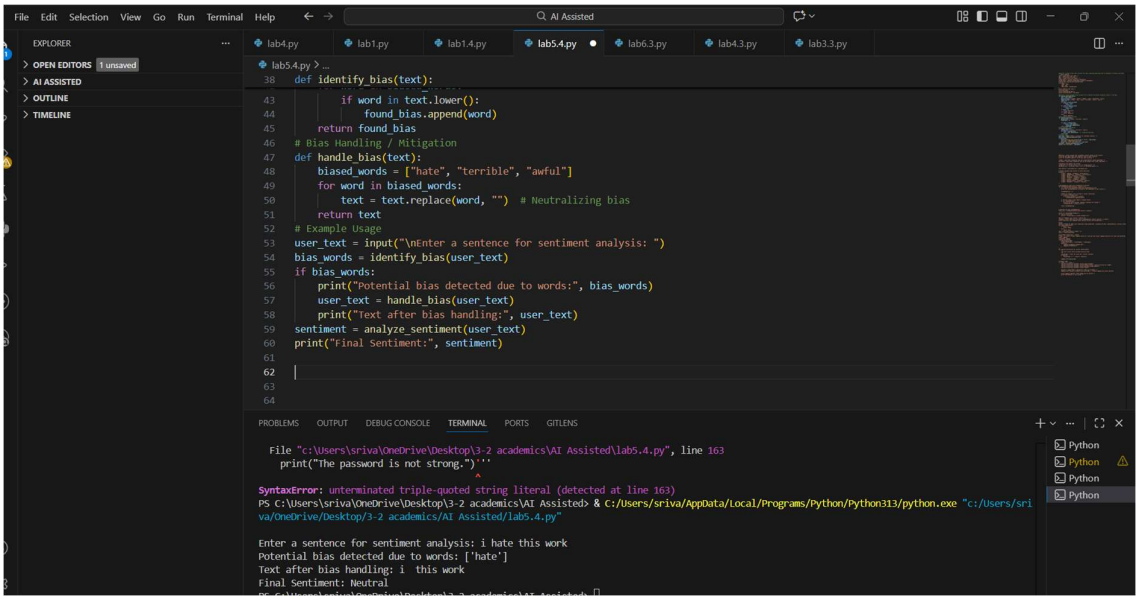
CODE:

```
def analyze_sentiment(text):
    text = text.lower()
    positive_words = ["good", "great", "happy", "love", "excellent", "nice"]
    negative_words = ["bad", "sad", "hate", "terrible", "awful", "poor"]
    score = 0
    for word in positive_words:
```

```
        if word in text:
            score += 1
    for word in negative_words:
        if word in text:
            score -= 1
    if score > 0:
        return "Positive"
    elif score < 0:
        return "Negative"
    else:
        return "Neutral"
# Bias Identification Function
def identify_bias(text):
    biased_words = ["hate", "terrible", "awful"]
    found_bias = []
    for word in biased_words:
        if word in text.lower():
            found_bias.append(word)
    return found_bias
# Bias Handling / Mitigation
def handle_bias(text):
    biased_words = ["hate", "terrible", "awful"]
    for word in biased_words:
        text = text.replace(word, "")  # Neutralizing bias
    return text
# Example Usage
user_text = input("\nEnter a sentence for sentiment analysis: ")
bias_words = identify_bias(user_text)
```

```
if bias_words:
    print("Potential bias detected due to words:", bias_words)
    user_text = handle_bias(user_text)
    print("Text after bias handling:", user_text)
sentiment = analyze_sentiment(user_text)
print("Final Sentiment:", sentiment)
```

OUTPUT:



EXPLANATION:

- This program performs sentiment analysis and bias detection on a given sentence.
- First, it checks the text for biased or strongly negative words like *hate, terrible*. If such words are found, it reports them and removes them from the sentence to reduce bias.
- Then, the modified text is analyzed to determine whether its sentiment is Positive, Negative, or Neutral based on the presence of predefined positive and negative words.
- Finally, the program prints the cleaned text if bias was found and the final sentiment result.

❖ TASK-3:

PROMPT:

Generate a python program that recommends products based on user history

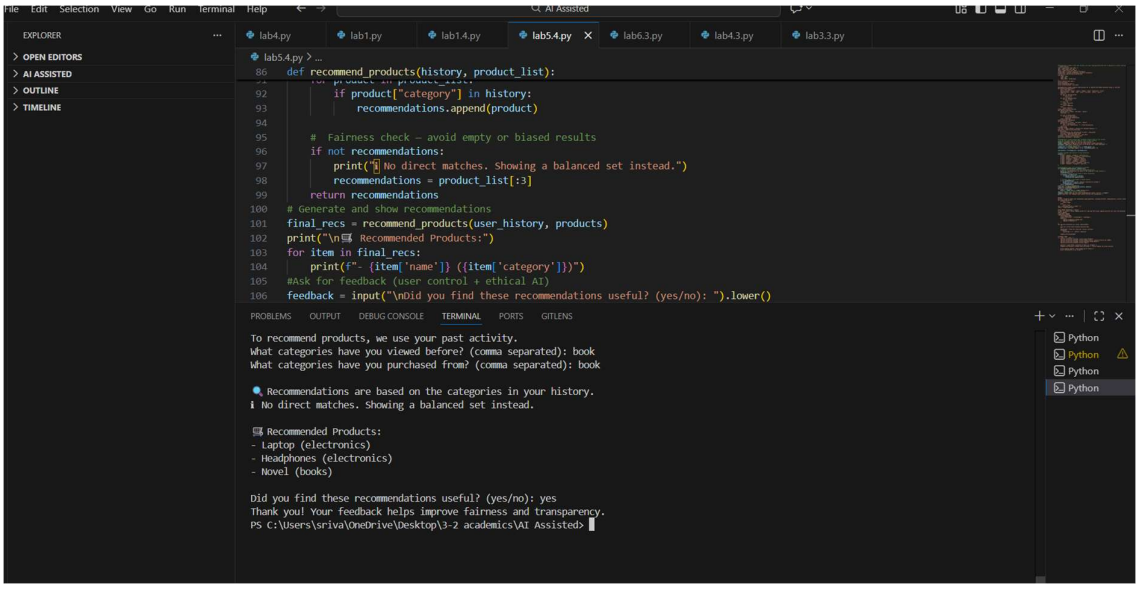
 # Ask the user about their past behavior (this is their history)

CODE:

```
print("To recommend products, we use your past activity.")
viewed = input("What categories have you viewed before? (comma separated): ")
purchased = input("What categories have you purchased from? (comma separated): ")
# Normalize and combine user history
viewed_list = [v.strip().lower() for v in viewed.split(",")]
purchased_list = [p.strip().lower() for p in purchased.split(",")]
user_history = set(viewed_list + purchased_list)
# Product database kept diverse to avoid favoritism
products = [
  {"name": "Laptop", "category": "electronics"},
  {"name": "Headphones", "category": "electronics"},
  {"name": "Novel", "category": "books"},
  {"name": "Notebook", "category": "books"},
  {"name": "T-shirt", "category": "fashion"},
  {"name": "Jacket", "category": "fashion"},
  {"name": "Running Shoes", "category": "sports"},
  {"name": "Football", "category": "sports"},
]
# Recommendation logic with transparency & fairness
def recommend_products(history, product_list):
  # Copilot: Be transparent about how recommendations work
  print("\n Recommendations are based on the categories in your history.")
  recommendations = []
  # Match by category only (no brand or seller favoritism)
  for product in product_list:
    if product["category"] in history:
```

```
        recommendations.append(product)
    #  Fairness check — avoid empty or biased results
    if not recommendations:
        print("ℹ No direct matches. Showing a balanced set instead.")
        recommendations = product_list[:3]
    return recommendations
# Generate and show recommendations
final_recs = recommend_products(user_history, products)
print("\n🛒 Recommended Products:")
for item in final_recs:
    print(f"- {item['name']} ({item['category']})")
#Ask for feedback (user control + ethical AI)
feedback = input("\nDid you find these recommendations useful? (yes/no): ").lower()
print("Thank you! Your feedback helps improve fairness and transparency.")
```

OUTPUT:



EXPLANATION:

❖ TASK-4:

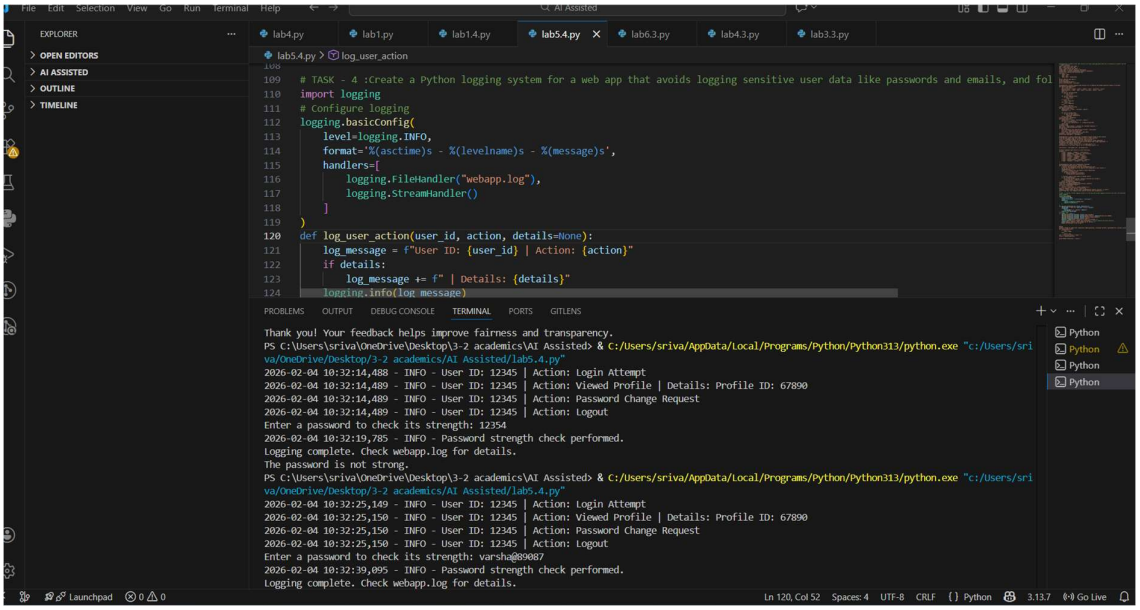
PROMPT:

Create a Python logging system for a web app that avoids logging sensitive user data like passwords and emails, and follows ethical logging practices.

CODE:

```
import logging

# Configure logging

logging.basicConfig(

  level=logging.INFO,

  format='%(asctime)s - %(levelname)s - %(message)s',

  handlers=[

    logging.FileHandler("webapp.log"),

    logging.StreamHandler()

  ]

)

def log_user_action(user_id, action, details=None):

  log_message = f"User ID: {user_id} | Action: {action}"

  if details:

    log_message += f" | Details: {details}"

  logging.info(log_message)

# Example usage

if __name__ == "__main__":

  log_user_action(user_id=12345, action="Login Attempt")

  log_user_action(user_id=12345, action="Viewed Profile", details="Profile ID: 67890")

  log_user_action(user_id=12345, action="Password Change Request")

  log_user_action(user_id=12345, action="Logout")

  password = input("Enter a password to check its strength: ")

  logging.info("Password strength check performed.")  # Avoid logging the actual password

  print("Logging complete. Check webapp.log for details.")
```

OUTPUT:



EXPLANATION:

- This program uses Python's logging module to record user actions in a safe and organized way. It is configured to write logs both to a file (webapp.log) and to the console, including the time and type of each message.
- The function log_user_action() creates a log entry for actions such as login, viewing a profile, changing a password, and logout. Sensitive information like the actual password is not logged, which helps protect user privacy.
- Finally, the program prints a message telling the user that logging is complete and where the log file can be checked.

❖ TASK-5:

PROMPT:

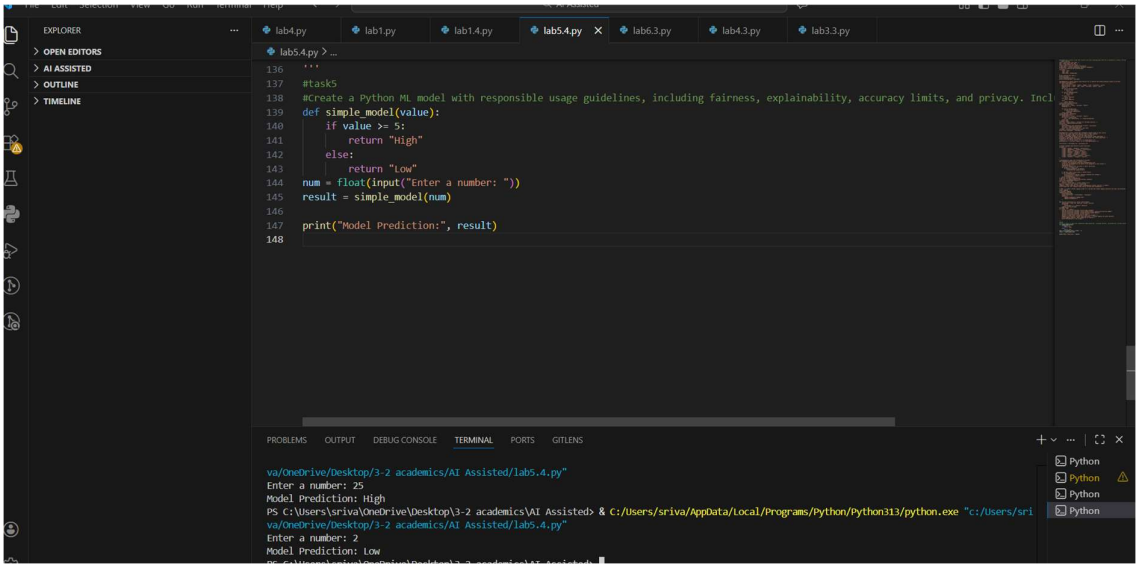
Create a Python ML model with responsible usage guidelines, including fairness, explainability, accuracy limits, and privacy. Include a simple example prediction.

CODE:

```
def simple_model(value):
  if value >= 5:
    return "High"
```

```
    else:
        return "Low"
num = float(input("Enter a number: "))
result = simple_model(num)
print("Model Prediction:", result)
```

OUTPUT:



EXPLANATION:

- This program acts like a very simple prediction model. It takes a number as input from the user and passes it to the `simple_model()` function.
- The function checks the value and returns high if the number is 5 or greater, otherwise it returns low.
- The result is then printed as the model's prediction. This shows how a basic decision rule can be used to make a simple classification.