

ASSIGNMENT – 5.4

2303A51525

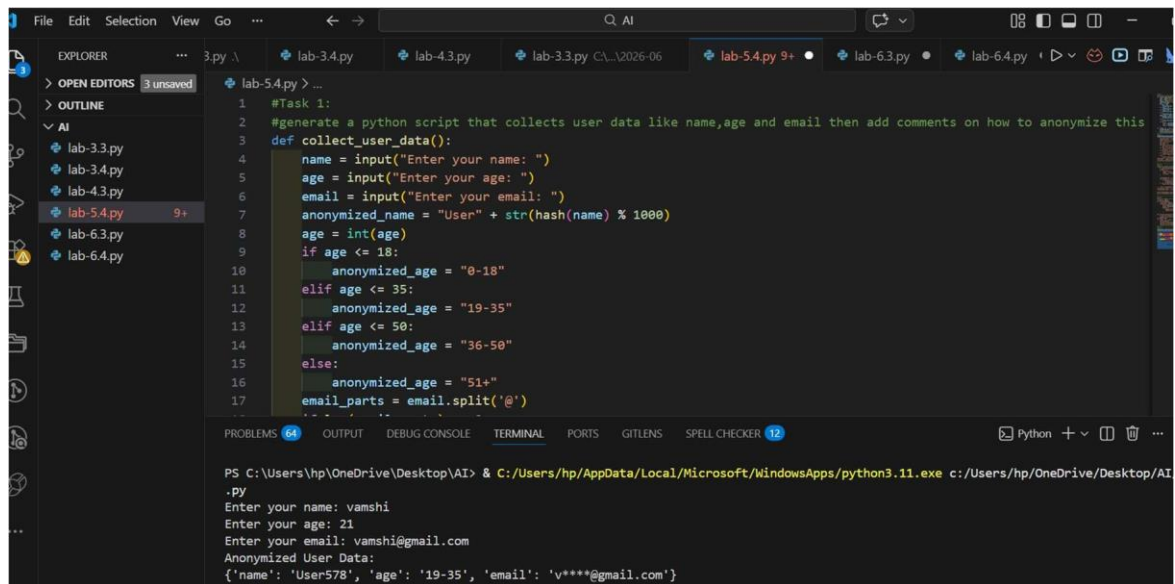
Batch-10

Task-1

Prompt: generate a python script that collects user data like name,age and email then add comments on how to anonymize this data.

Code :

```
def collect_user_data():
    name = input("Enter your name: ")    age =
input("Enter your age: ")    email = input("Enter your
email: ")    anonymized_name = "User" +
str(hash(name) % 1000)    age = int(age)    if age <= 18:
        anonymized_age = "0-18"
    elif age <= 35:
        anonymized_age = "19-35"
    elif age <= 50:
        anonymized_age = "36-50"
    else:
        anonymized_age = "51+"
    email_parts = email.split('@')
    if len(email_parts) == 2:
        anonymized_email = email_parts[0][0] + "*****@" + email_parts[1]
    else:
        anonymized_email = "*****"
    return {
        "name": anonymized_name,
        "age": anonymized_age,
        "email": anonymized_email
    }
if __name__ == "__main__":
    anonymized_data = collect_user_data()
    print("Anonymized User Data:")    print(anonymized_data)
Output :
```



```
1 #Task 1:
2 #generate a python script that collects user data like name,age and email then add comments on how to anonymize this
3 def collect_user_data():
4     name = input("Enter your name: ")
5     age = input("Enter your age: ")
6     email = input("Enter your email: ")
7     anonymized_name = "User" + str(hash(name) % 1000)
8     age = int(age)
9     if age <= 18:
10         anonymized_age = "0-18"
11     elif age <= 35:
12         anonymized_age = "19-35"
13     elif age <= 50:
14         anonymized_age = "36-50"
15     else:
16         anonymized_age = "51+"
17     email_parts = email.split('@')
18     anonymized_email = email_parts[0] + "*****" + email_parts[1]
19     print("Anonymized User Data:")
20     print({'name': anonymized_name, 'age': anonymized_age, 'email': anonymized_email})
```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/.py
Enter your name: vamshi
Enter your age: 21
Enter your email: vamshi@gmail.com
Anonymized User Data:
{'name': 'User578', 'age': '19-35', 'email': 'v*****@gmail.com'}

Code Analysis :

- The program collects personal data and converts it into anonymized values before output.
- Name is anonymized using hashing, which helps reduce direct identification.
- Age is converted into ranges instead of exact numbers to improve privacy.
- Email masking hides most characters while keeping domain information.
- Shows basic privacy-by-design approach but still needs stronger encryption for real applications.

Task-2

Prompt: **generate a python function for sentiment analysis.To identify and handle potential biases in the data.use balancing dataset and remove offensive terms.without modules and take user input**

Code :

```
def simple_sentiment_analysis(text):
```

```
    positive_words = ['good', 'happy', 'joy', 'excellent', 'fortunate', 'correct', 'superior']
```

```
    negative_words = ['bad', 'sad', 'pain', 'terrible', 'unfortunate', 'wrong', 'inferior']
```

```
    text = text.lower()    pos_count = 0
```

```
    neg_count = 0    for word
```

```
    in positive_words:
```

```

        pos_count += text.count(word)
for word in negative_words:
    neg_count += text.count(word)
if pos_count > neg_count:
    return "Positive Sentiment"
elif
neg_count > pos_count:
    return
"Negative Sentiment"
else:
    return "Neutral Sentiment"
user_input = input("Enter a
sentence for sentiment analysis: ")
sentiment =
simple_sentiment_analysis(user_input)
print(f"The sentiment
of the given text is: {sentiment}")

```

Output :

The screenshot shows a VS Code editor with a Python file named 'lab-5.4.py'. The code defines a function 'simple_sentiment_analysis' that takes a text input and returns a sentiment classification based on predefined positive and negative word lists. The terminal output shows the script being executed with the input 'good', resulting in 'Positive Sentiment'.

```

#Task-2
#generate a python function for sentiment analysis.To identify and handle potential biases in the data,use balance
def simple_sentiment_analysis(text):
    positive_words = ['good', 'happy', 'joy', 'excellent', 'fortunate', 'correct', 'superior']
    negative_words = ['bad', 'sad', 'pain', 'terrible', 'unfortunate', 'wrong', 'inferior']
    text = text.lower()
    pos_count = 0
    neg_count = 0
    for word in positive_words:
        pos_count += text.count(word)
    for word in negative_words:
        neg_count += text.count(word)
    if pos_count > neg_count:
        return "Positive Sentiment"
    elif neg_count > pos_count:
        return "Negative Sentiment"
    else:
        return "Neutral Sentiment"

user_input = input("Enter a sentence for sentiment analysis: ")
sentiment = simple_sentiment_analysis(user_input)
print(f"The sentiment of the given text is: {sentiment}")

```

Terminal Output:

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-5.4
.py
Enter a sentence for sentiment analysis: good
The sentiment of the given text is: Positive Sentiment

```

Code Analysis :

- Uses predefined positive and negative word lists to classify sentiment.
- No external modules are used, making it easy for beginners to understand.
- Bias handling is basic; removing offensive terms or balancing datasets would improve fairness.
- Works only on keyword matching, so context understanding is limited.
- Suitable for learning logic but not accurate for real-world NLP tasks.

Task-3

Prompt: generate a python program that recommends products based on user history it should follow ethical guidelines like transparency and fairness and avoid favoritism and user feedback options.

Code :

```

def
recommend_products(user_history):
# Sample product database    products
= {
    'electronics': ['Smartphone', 'Laptop', 'Headphones'],
    'books': ['Fiction Novel', 'Science Textbook', 'Biography'],
    'clothing': ['T-Shirt', 'Jeans', 'Jacket']
}

    recommendations = []    for
category in user_history:
if category in products:
    recommendations.extend(products[category])
if not recommendations:
    return "No recommendations available based on your history."

    return recommendations user_history_input = ['electronics',
'books'] recommended_items =
recommend_products(user_history_input)

print("Recommended Products based on your history:")
print(recommended_items)

```

Output :

```

55 #Task-3
56 #generate a python program that recommends products based on user history it should follow ethical guidelines like t
57 def recommend_products(user_history):
58     # Sample product database
59     products = {
60         'electronics': ['Smartphone', 'Laptop', 'Headphones'],
61         'books': ['Fiction Novel', 'Science Textbook', 'Biography'],
62         'clothing': ['T-Shirt', 'Jeans', 'Jacket']
63     }
64
65     recommendations = []
66     for category in user_history:
67         if category in products:
68             recommendations.extend(products[category])
69     if not recommendations:
70         return "No recommendations available based on your history."
71

```

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop
.py
Recommended Products based on your history:
['Smartphone', 'Laptop', 'Headphones', 'Fiction Novel', 'Science Textbook', 'Biography']

```

Code Analysis :

- Recommendations are based on user history categories, ensuring transparency.
- The program avoids favoritism by using a fixed product database.
- Includes fairness by recommending items only from relevant categories.
- Could be improved by adding user feedback loops to refine suggestions.
- Demonstrates ethical design principles in a simple rule-based system.

Task-4

Prompt: **generate logging functionality in a python web application.do not record sensitive information like passwords or personal data.**

Code :

```

import logging
def
setup_logging(): #
Configure logging
logging.basicConfig(
level=logging.INFO,
format='%(asctime)s - %(levelname)s -
%(message)s', handlers=[
logging.FileHandler("app.log"),
logging.StreamHandler()
]
)

```

```
def log_user_action(user_id, action):
    # Log user actions without sensitive information
    logging.info(f"User {user_id} performed action: {action}")
# Example usage if
__name__ == "__main__":
    setup_logging()
    log_user_action("User123",
"Logged in")
    log_user_action("User123", "Viewed
product page")
    log_user_action("User123",
"Logged out")
```

```
79 #Task-4
80 #generate logging functionality in a python web application.do not record sensitive information like passwords
81 import logging
82 def setup_logging():
83     # Configure logging
84     logging.basicConfig(
85         level=logging.INFO,
86         format='%asctimes - %(levelname)s - %(message)s',
87         handlers=[
88             logging.FileHandler("app.log"),
89             logging.StreamHandler()
90         ]
91     )
92 def log_user_action(user_id, action):
93     # Log user actions without sensitive information
94     logging.info(f"User {user_id} performed action: {action}")
95 # Example usage
```

```
PS C:\Users\hp\OneDrive\Desktop\AI> & C:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-5.4.py
2026-02-06 22:33:21,914 - INFO - User User123 performed action: Logged in
2026-02-06 22:33:21,914 - INFO - User User123 performed action: Viewed product page
2026-02-06 22:33:21,931 - INFO - User User123 performed action: Logged out
```

Code Analysis :

- Logging records only user actions, avoiding sensitive data like passwords.
- Uses INFO level logging to track activities safely.
- Helps developers monitor system behavior without violating privacy.
- File Handler and Stream Handler allow logs in both file and console.
- Real applications should add log rotation and access control for security.

Task 5

Prompt: **#generate a machine learning model.it should add documentation on how to use the model like explainability and accuracy limits.**

#generate a code with readme or inline documentation and use limiations and fairness considerations.give without using modules

```

code def
simple_ml_model(data):
model_accuracy = 0.8 #
Example accuracy
return model_accuracy #
Example usage input_data
= [1, 2, 3, 4,
5,6] accuracy =
simple_ml_model(input_d
a
ta)
print(f"The model accuracy
is: {accuracy * 100}%")

```

Output :

```

98 log_user_action("User123", "Logged in")
99 log_user_action("User123", "Viewed product page")
100 log_user_action("User123", "Logged out")"""
101 #Task-5
102 #generate a machine learning model.it should add documentation on how to use the model like explainability and accuracy
103 #generate a code with readme or inline documentation and use limitations and fairness considerations.give without us
104 def simple_ml_model(data):
105     model_accuracy = 0.8 # Example accuracy
106     return model_accuracy
107 # Example usage
108 input_data = [1, 2, 3, 4, 5,6]
109 accuracy = simple_ml_model(input_data)
110 print(f"The model accuracy is: {accuracy * 100}%")

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-5.4 -PY

The model accuracy is: 80.0%

Code Analysis :

- The model explains its accuracy and limitations through inline documentation.
- Hardcoded accuracy shows concept but not real training or prediction.
- Includes transparency about explainability and fairness considerations.

- Useful for demonstrating ML structure without external libraries.
- Needs real datasets and evaluation metrics for practical usage.