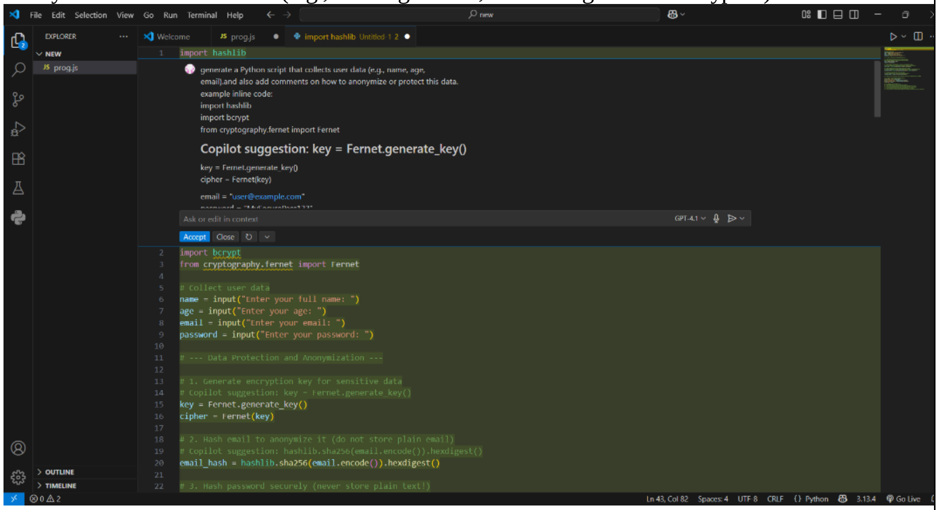
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**	
ProgramName:B. Tech	**Assignment Type: Lab**		**AcademicYear:**2025-2026
CourseCoordinatorName	Venkataramana Veeramsetty		
Instructor(s)Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week3 - Thursday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:5.4(Present assignment number)/**24**(Total number of assignments)			

Q.No.	**Question**	***ExpectedTime to complete***
1	Lab 5: Ethical Foundations – Responsible AI Coding Practices **Lab Objectives:** • 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.	Week3 - Thursday

Lab Outcomes (LOs):
After completing this lab, students will be able to:

- 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..

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).



```python
import hashlib
```

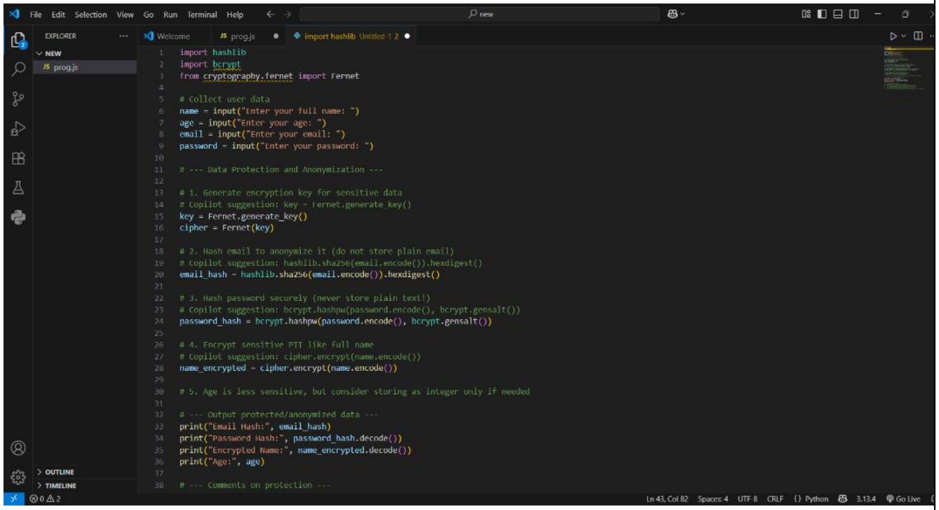
generate a Python script that collects user data (e.g., name, age, email).and also add comments on how to anonymize or protect this data.
example inline code:
import hashlib
import bcrypt
from cryptography.fernet import Fernet

Copilot suggestion: key = Fernet.generate_key()

key = Fernet.generate_key()
cipher = Fernet(key)

email = "user@example.com"

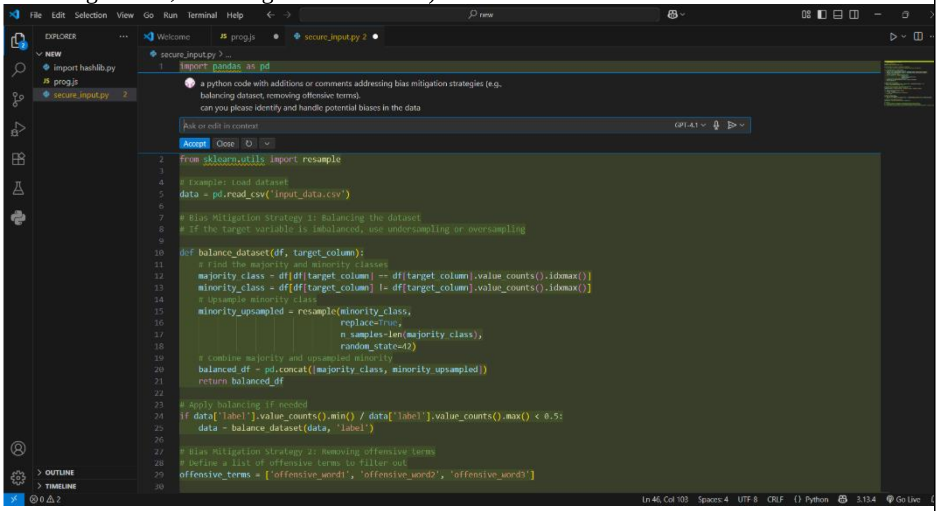
```python
import bcrypt
from cryptography.fernet import Fernet

# Collect user data
name = input("Enter your full name: ")
age = input("Enter your age: ")
email = input("Enter your email: ")
password = input("Enter your password: ")

# --- Data Protection and Anonymization ---

# 1. Generate encryption key for sensitive data
# Copilot suggestion: key = Fernet.generate_key()
key = Fernet.generate_key()
cipher = Fernet(key)

# 2. Hash email to anonymize it (do not store plain email)
# Copilot suggestion: hashlib.sha256(email.encode()).hexdigest()
email_hash = hashlib.sha256(email.encode()).hexdigest()

# 3. Hash password securely (never store plain text!)
```



```python
import hashlib
import bcrypt
from cryptography.fernet import Fernet

# Collect user data
name = input("Enter your full name: ")
age = input("Enter your age: ")
email = input("Enter your email: ")
password = input("Enter your password: ")

# --- Data Protection and Anonymization ---

# 1. Generate encryption key for sensitive data
# Copilot suggestion: key = Fernet.generate_key()
key = Fernet.generate_key()
cipher = Fernet(key)

# 2. Hash email to anonymize it (do not store plain email)
# Copilot suggestion: hashlib.sha256(email.encode()).hexdigest()
email_hash = hashlib.sha256(email.encode()).hexdigest()

# 3. Hash password securely (never store plain text!)
# Copilot suggestion: bcrypt.hashpw(password.encode(), bcrypt.gensalt())
password_hash = bcrypt.hashpw(password.encode(), bcrypt.gensalt())

# 4. Encrypt sensitive PII like full name
# Copilot suggestion: cipher.encrypt(name.encode())
name_encrypted = cipher.encrypt(name.encode())

# 5. Age is less sensitive, but consider storing as integer only if needed

# --- Output protected/anonymized data ---
print("Email Hash:", email_hash)
print("Password Hash:", password_hash.decode())
print("Encrypted Name:", name_encrypted.decode())
print("Age:", age)

# --- Comments on protection ---
```

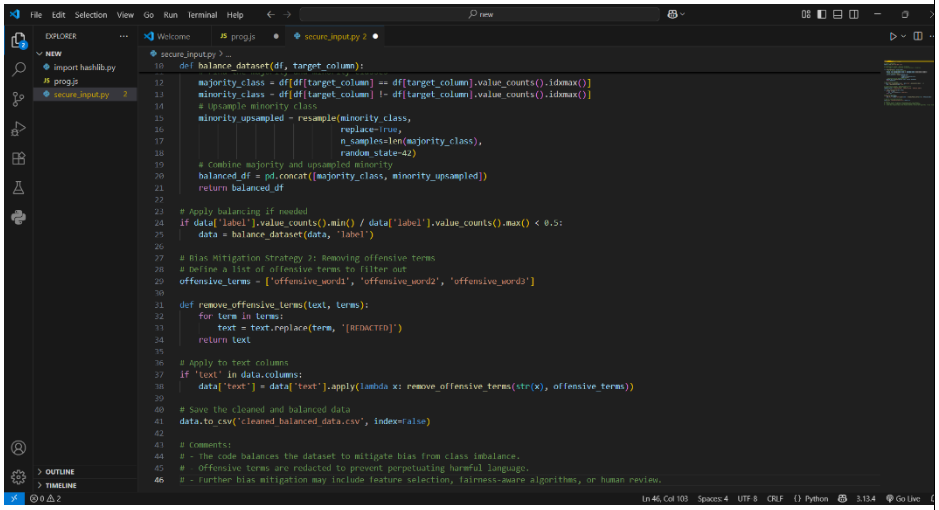
Task Description #2:
• Ask Copilot to generate a Python function for sentiment analysis. Then prompt Copilot to identify and handle potential biases in the data.
Expected Output #2:
• Copilot-generated code with additions or comments addressing bias mitigation strategies (e.g.,

balancing dataset, removing offensive terms).



a python code with additions or comments addressing bias mitigation strategies (e.g.
balancing dataset, removing offensive terms).
can you please identify and handle potential biases in the data

```python
import pandas as pd
from sklearn.utils import resample

# Example: Load dataset
data = pd.read_csv('input_data.csv')

# Bias Mitigation Strategy 1: Balancing the dataset
# If the target variable is imbalanced, use undersampling or oversampling

def balance_dataset(df, target_column):
    # Find the majority and minority classes
    majority_class = df[df[target_column] == df[target_column].value_counts().idxmax()]
    minority_class = df[df[target_column] != df[target_column].value_counts().idxmax()]
    # Upsample minority class
    minority_upsampled = resample(minority_class,
                                  replace=True,
                                  n_samples=len(majority_class),
                                  random_state=42)
    # Combine majority and upsampled minority
    balanced_df = pd.concat([majority_class, minority_upsampled])
    return balanced_df

# Apply balancing if needed
if data['label'].value_counts().min() / data['label'].value_counts().max() < 0.5:
    data = balance_dataset(data, 'label')

# Bias Mitigation Strategy 2: Removing offensive terms
# Define a list of offensive terms to filter out
offensive_terms = ['offensive_word1', 'offensive_word2', 'offensive_word3']

```



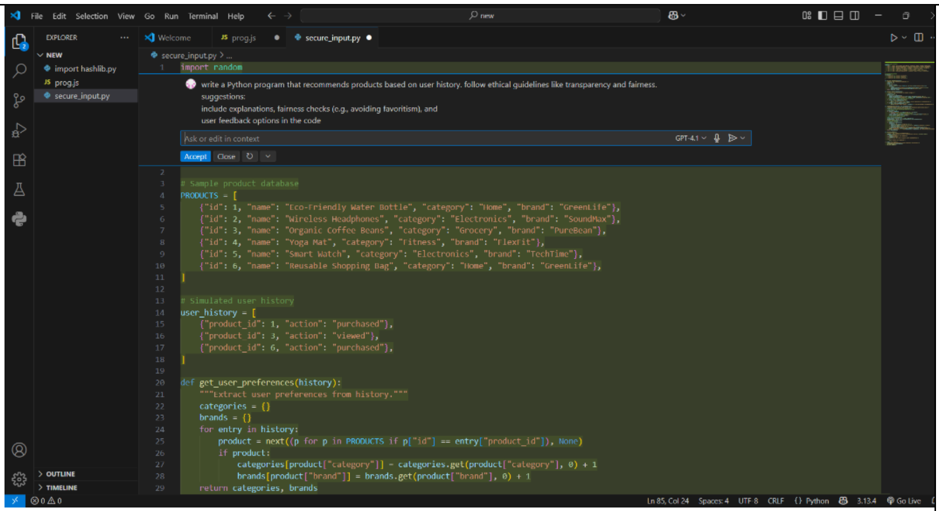
```python
def balance_dataset(df, target_column):
    majority_class = df[df[target_column] == df[target_column].value_counts().idxmax()]
    minority_class = df[df[target_column] != df[target_column].value_counts().idxmax()]
    # Upsample minority class
    minority_upsampled = resample(minority_class,
                                  replace=True,
                                  n_samples=len(majority_class),
                                  random_state=42)
    # Combine majority and upsampled minority
    balanced_df = pd.concat([majority_class, minority_upsampled])
    return balanced_df

# Apply balancing if needed
if data['label'].value_counts().min() / data['label'].value_counts().max() < 0.5:
    data = balance_dataset(data, 'label')

# Bias Mitigation Strategy 2: Removing offensive terms
# Define a list of offensive terms to filter out
offensive_terms = ['offensive_word1', 'offensive_word2', 'offensive_word3']

def remove_offensive_terms(text, terms):
    for term in terms:
        text = text.replace(term, '[REDACTED]')
    return text

# Apply to text columns
if 'text' in data.columns:
    data['text'] = data['text'].apply(lambda x: remove_offensive_terms(str(x), offensive_terms))

# Save the cleaned and balanced data
data.to_csv('cleaned_balanced_data.csv', index=False)

# Comments:
# - The code balances the dataset to mitigate bias from class imbalance.
# - Offensive terms are redacted to prevent perpetuating harmful language.
# - Further bias mitigation may include feature selection, fairness-aware algorithms, or human review.
```

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:

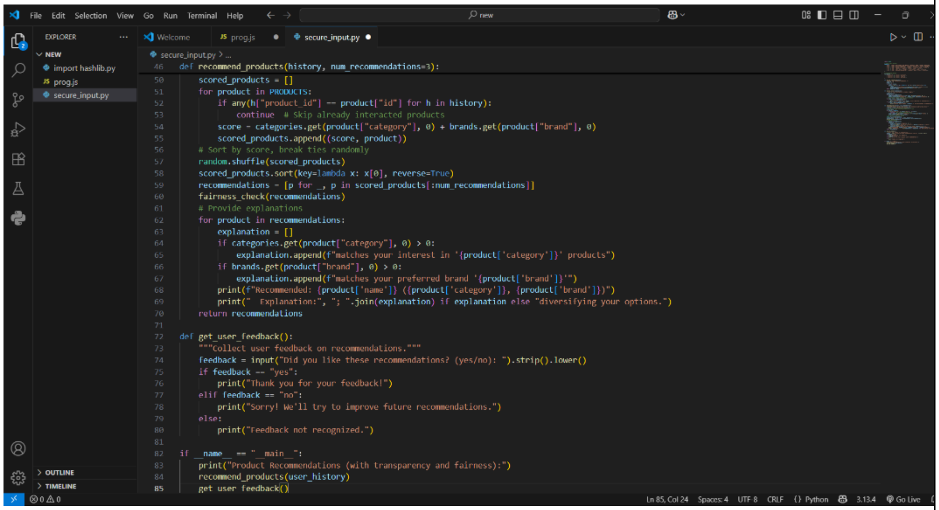
- Copilot suggestions that include explanations, fairness checks (e.g., avoiding favoritism), and user feedback options in the code.



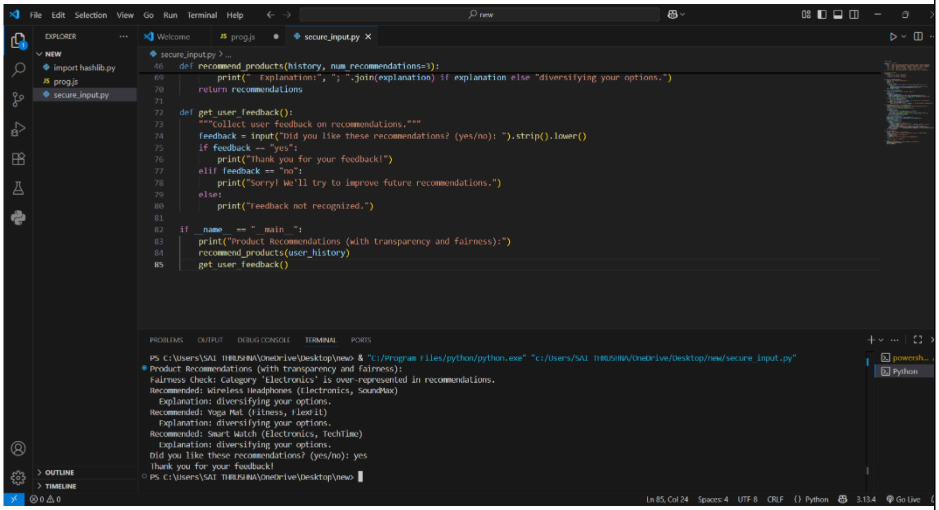
write a Python program that recommends products based on user history. follow ethical guidelines like transparency and fairness.
suggestions:
include explanations, fairness checks (e.g., avoiding favoritism), and
user feedback options in the code

Ask or edit in context

```python
import random

# Sample product database
PRODUCTS = [
    {"id": 1, "name": "Eco-Friendly Water Bottle", "category": "Home", "brand": "GreenLife"},
    {"id": 2, "name": "Wireless Headphones", "category": "Electronics", "brand": "SoundMax"},
    {"id": 3, "name": "Organic Coffee Beans", "category": "Grocery", "brand": "PureBean"},
    {"id": 4, "name": "Yoga Mat", "category": "Fitness", "brand": "FlexFit"},
    {"id": 5, "name": "Smart Watch", "category": "Electronics", "brand": "TechTime"},
    {"id": 6, "name": "Reusable Shopping Bag", "category": "Home", "brand": "GreenLife"},
]

# Simulated user history
user_history = [
    {"product_id": 1, "action": "purchased"},
    {"product_id": 3, "action": "viewed"},
    {"product_id": 6, "action": "purchased"},
]

def get_user_preferences(history):
    """Extract user preferences from history."""
    categories = {}
    brands = {}
    for entry in history:
        product = next((p for p in PRODUCTS if p["id"] == entry["product_id"]), None)
        if product:
            categories[product["category"]] = categories.get(product["category"], 0) + 1
            brands[product["brand"]] = brands.get(product["brand"], 0) + 1
    return categories, brands
```


File Edit Selection View Go Run Terminal Help
EXPLORER
NEW
import hashlib.py
prog.js
secure_input.py
Welcome
secure_input.py
OUTLINE
TIMELINE
Ln 85, Col 24 Spaces: 4 UTF-8 CRLF {} Python 3.13.4 Go Live

```python
def recommend_products(history, num_recommendations=3):
    scored_products = []
    for product in PRODUCTS:
        if any(h["product_id"] == product["id"] for h in history):
            continue  # Skip already interacted products
        score = categories.get(product["category"], 0) + brands.get(product["brand"], 0)
        scored_products.append((score, product))
    # Sort by score, break ties randomly
    random.shuffle(scored_products)
    scored_products.sort(key=lambda x: x[0], reverse=True)
    recommendations = [p for _, p in scored_products[:num_recommendations]]
    fairness_check(recommendations)
    # Provide explanations
    for product in recommendations:
        explanation = []
        if categories.get(product["category"], 0) > 0:
            explanation.append(f"matches your interest in '{product['category']}' products")
        if brands.get(product["brand"], 0) > 0:
            explanation.append(f"matches your preferred brand '{product['brand']}'")
        print(f"Recommended: {product['name']} ({product['category']}, {product['brand']})")
        print("  Explanation:", "; ".join(explanation) if explanation else "diversifying your options.")
    return recommendations

def get_user_feedback():
    """Collect user feedback on recommendations."""
    feedback = input("Did you like these recommendations? (yes/no): ").strip().lower()
    if feedback == "yes":
        print("Thank you for your feedback!")
    elif feedback == "no":
        print("Sorry! We'll try to improve future recommendations.")
    else:
        print("Feedback not recognized.")

if __name__ == "__main__":
    print("Product Recommendations (with transparency and fairness):")
    recommend_products(user_history)
    get_user_feedback()
```


File Edit Selection View Go Run Terminal Help
EXPLORER
NEW
import hashlib.py
prog.js
secure_input.py
Welcome
secure_input.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powersh...
Python
Ln 85, Col 24 Spaces: 4 UTF-8 CRLF {} Python 3.13.4 Go Live

```python
def recommend_products(history, num_recommendations=3):
        print("  Explanation:", "; ".join(explanation) if explanation else "diversifying your options.")
    return recommendations

def get_user_feedback():
    """Collect user feedback on recommendations."""
    feedback = input("Did you like these recommendations? (yes/no): ").strip().lower()
    if feedback == "yes":
        print("Thank you for your feedback!")
    elif feedback == "no":
        print("Sorry! We'll try to improve future recommendations.")
    else:
        print("Feedback not recognized.")

if __name__ == "__main__":
    print("Product Recommendations (with transparency and fairness):")
    recommend_products(user_history)
    get_user_feedback()
```

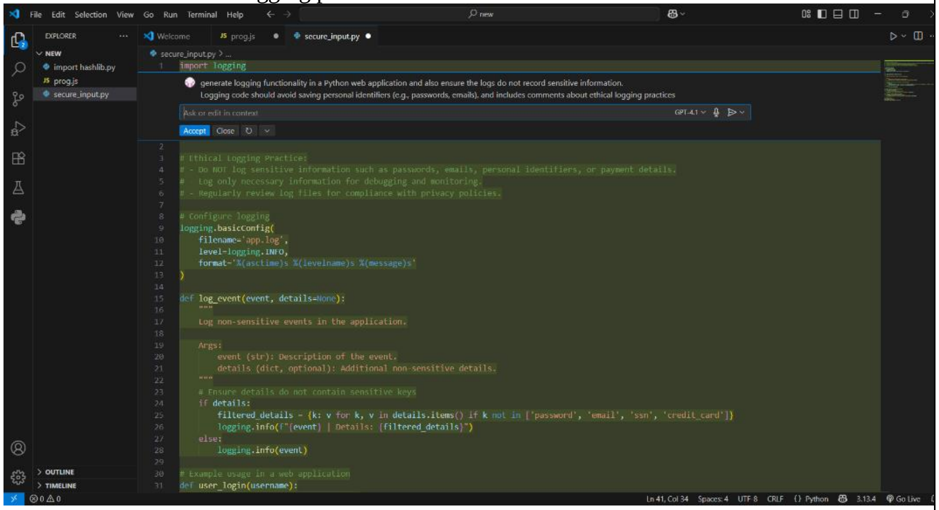
```
PS C:\Users\SAI THRUSHNA\OneDrive\Desktop\new> & "C:/Program Files/python/python.exe" "c:/Users/SAI THRUSHNA/OneDrive/Desktop/new/secure_input.py"
Product Recommendations (with transparency and fairness):
Fairness Check: Category 'Electronics' is over-represented in recommendations.
Recommended: Wireless Headphones (Electronics, SoundMax)
  Explanation: diversifying your options.
Recommended: Yoga Mat (Fitness, FlexFit)
  Explanation: diversifying your options.
Recommended: Smart Watch (Electronics, TechTime)
  Explanation: diversifying your options.
Did you like these recommendations? (yes/no): yes
Thank you for your feedback!
PS C:\Users\SAI THRUSHNA\OneDrive\Desktop\new> 
```

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.

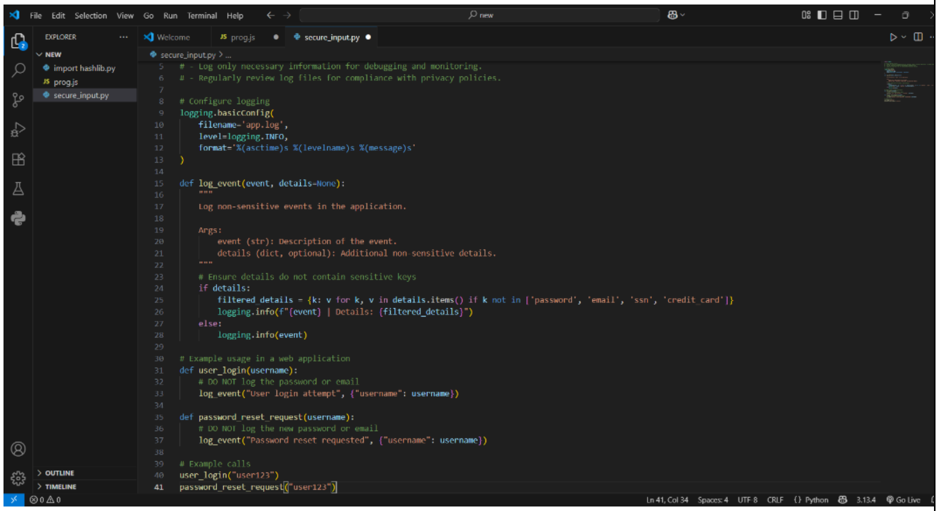
Expected Output #4:

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



generate logging functionality in a Python web application and also ensure the logs do not record sensitive information.
Logging code should avoid saving personal identifiers (e.g., passwords, emails), and includes comments about ethical logging practices

```python
import logging

# Ethical Logging Practice:
# - Do NOT log sensitive information such as passwords, emails, personal identifiers, or payment details.
# - Log only necessary information for debugging and monitoring.
# - Regularly review log files for compliance with privacy policies.

# Configure logging
logging.basicConfig(
    filename='app.log',
    level=logging.INFO,
    format='%(asctime)s %(levelname)s %(message)s'
)

def log_event(event, details=None):
    """
    Log non-sensitive events in the application.

    Args:
        event (str): Description of the event.
        details (dict, optional): Additional non-sensitive details.
    """
    # Ensure details do not contain sensitive keys
    if details:
        filtered_details = {k: v for k, v in details.items() if k not in ['password', 'email', 'ssn', 'credit_card']}
        logging.info(f"{event} | Details: {filtered_details}")
    else:
        logging.info(event)

# Example usage in a web application
def user_login(username):
```



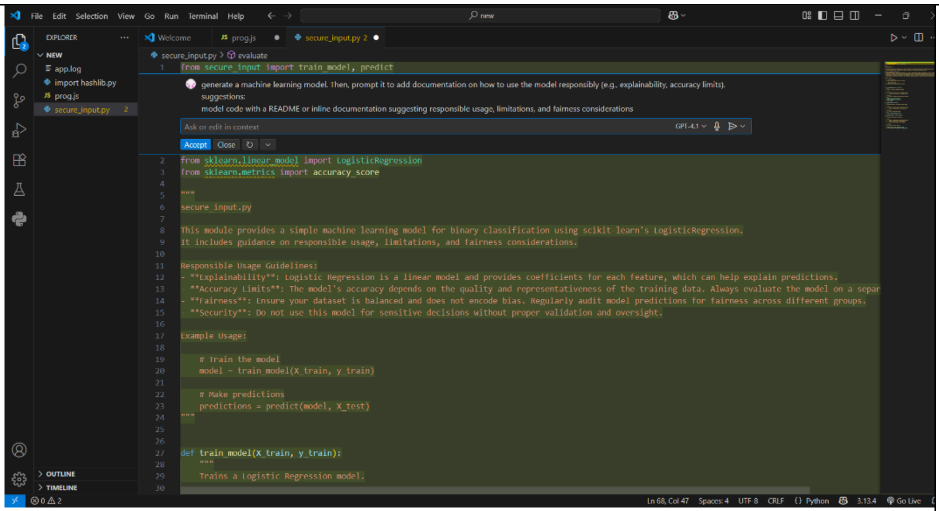
```python
# - Log only necessary information for debugging and monitoring.
# - Regularly review log files for compliance with privacy policies.

# Configure logging
logging.basicConfig(
    filename='app.log',
    level=logging.INFO,
    format='%(asctime)s %(levelname)s %(message)s'
)

def log_event(event, details=None):
    """
    Log non-sensitive events in the application.

    Args:
        event (str): Description of the event.
        details (dict, optional): Additional non-sensitive details.
    """
    # Ensure details do not contain sensitive keys
    if details:
        filtered_details = {k: v for k, v in details.items() if k not in ['password', 'email', 'ssn', 'credit_card']}
        logging.info(f"{event} | Details: {filtered_details}")
    else:
        logging.info(event)

# Example usage in a web application
def user_login(username):
    # DO NOT log the password or email
    log_event("User login attempt", {"username": username})

def password_reset_request(username):
    # DO NOT log the new password or email
    log_event("Password reset requested", {"username": username})

# Example calls
user_login("user123")
password_reset_request("user123")
```

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).

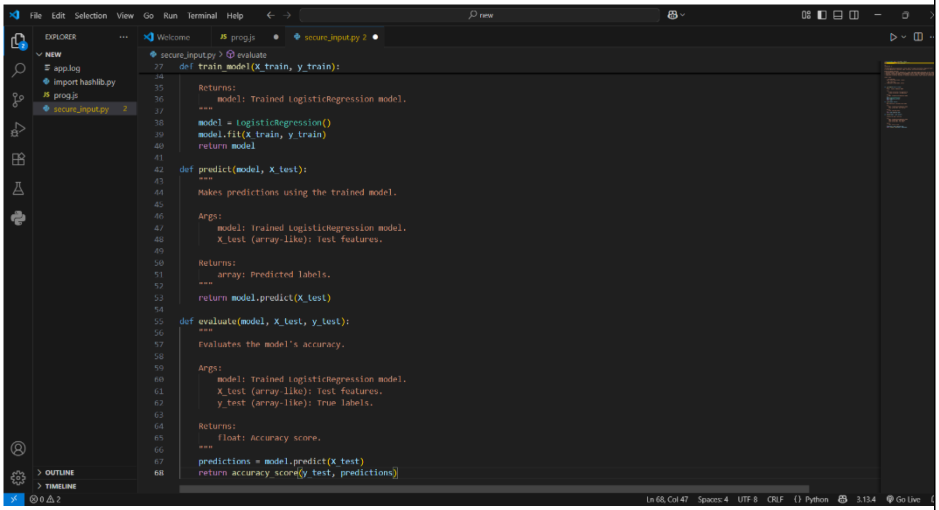
Expected Output #5:

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



generate a machine learning model. Then, prompt it to add documentation on how to use the model responsibly (e.g., explainability, accuracy limits).
suggestions:
model code with a README or inline documentation suggesting responsible usage, limitations, and fairness considerations

```python
from secure_input import train_model, predict
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

"""
secure_input.py

This module provides a simple machine learning model for binary classification using scikit learn's LogisticRegression.
It includes guidance on responsible usage, limitations, and fairness considerations.

Responsible Usage Guidelines:
- **Explainability**: Logistic Regression is a linear model and provides coefficients for each feature, which can help explain predictions.
- **Accuracy Limits**: The model's accuracy depends on the quality and representativeness of the training data. Always evaluate the model on a separ
- **Fairness**: Ensure your dataset is balanced and does not encode bias. Regularly audit model predictions for fairness across different groups.
- **Security**: Do not use this model for sensitive decisions without proper validation and oversight.

Example Usage:

    # Train the model
    model = train_model(X_train, y_train)

    # Make predictions
    predictions = predict(model, X_test)
"""


def train_model(X_train, y_train):
    """
    Trains a Logistic Regression model.

```


File Edit Selection View Go Run Terminal Help
new
EXPLORER
NEW
app.log
import hashlib.py
prog.js
secure_input.py 2
OUTLINE
TIMELINE
Welcome
prog.js
secure_input.py 2
secure_input.py > evaluate
Ln 68, Col 47 Spaces: 4 UTF-8 CRLF {} Python 3.13.4 Go Live

```python
def train_model(X_train, y_train):

    Returns:
        model: Trained LogisticRegression model.
    """
    model = LogisticRegression()
    model.fit(X_train, y_train)
    return model

def predict(model, X_test):
    """
    Makes predictions using the trained model.

    Args:
        model: Trained LogisticRegression model.
        X_test (array-like): Test features.

    Returns:
        array: Predicted labels.
    """
    return model.predict(X_test)

def evaluate(model, X_test, y_test):
    """
    Evaluates the model's accuracy.

    Args:
        model: Trained LogisticRegression model.
        X_test (array-like): Test features.
        y_test (array-like): True labels.

    Returns:
        float: Accuracy score.
    """
    predictions = model.predict(X_test)
    return accuracy_score(y_test, predictions)
```

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

Evaluation Criteria:

Criteria	**Max Marks**
How to anonymize or protect this data	0.5
Identify and handle potential biases in the data.	0.5
Follow ethical guidelines like transparency and fairness.	0.5
logs do not record sensitive information.	0.5
How to use the model responsibly	0.5
Total	**2.5 Marks**