AI Assisted Coding

Assignment (5.4)

2403A51263(Konda Shiva)

Batch-11(CSE)

Description 1:

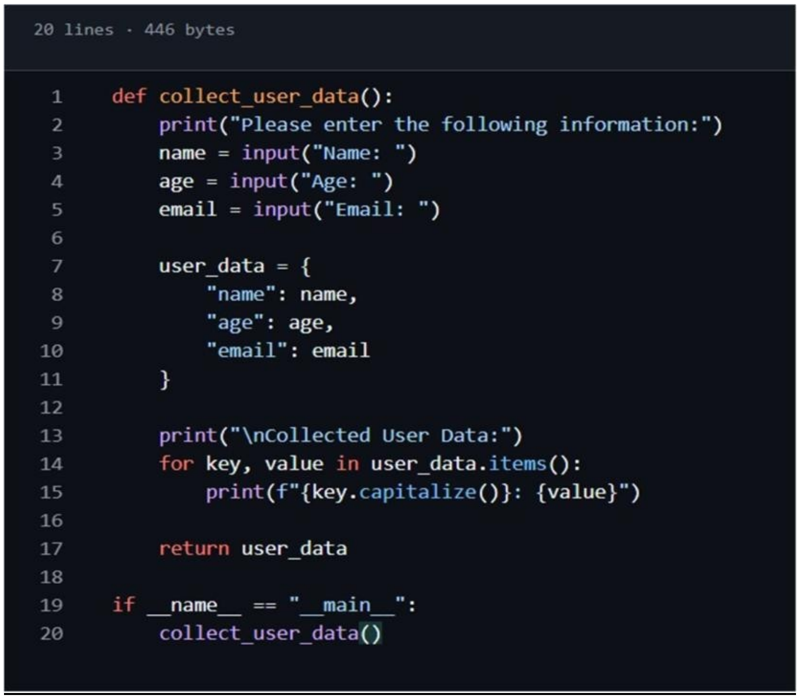
- Prompt GitHub Copilot to generate a Python script that collects user data (e.g., name, age, email). Then, ask Copilot to add comments on how to anonymize or protect this data.

Expected Output #1:

- A script with inline Copilot-suggested code and comments explaining how to safeguard or

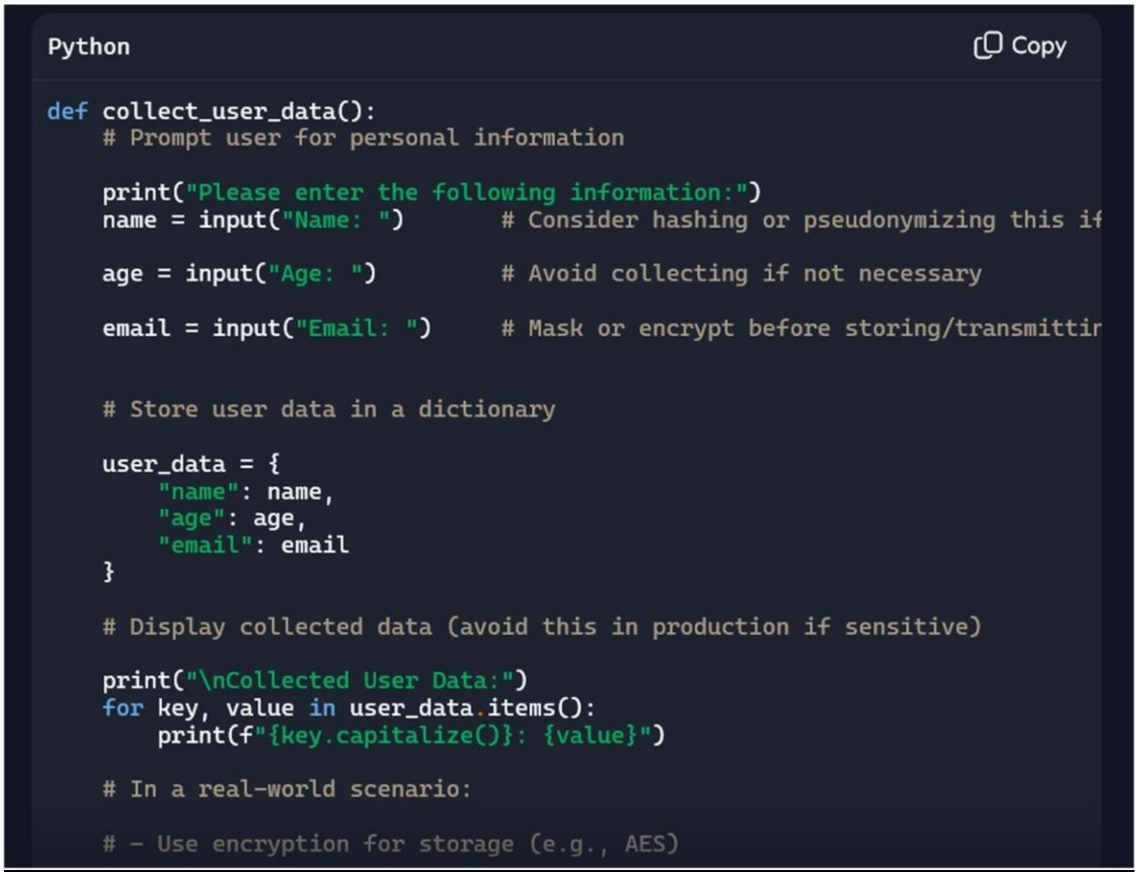
anonymize user information (e.g., hashing emails, not storing data unencrypted)

Github copilot Code

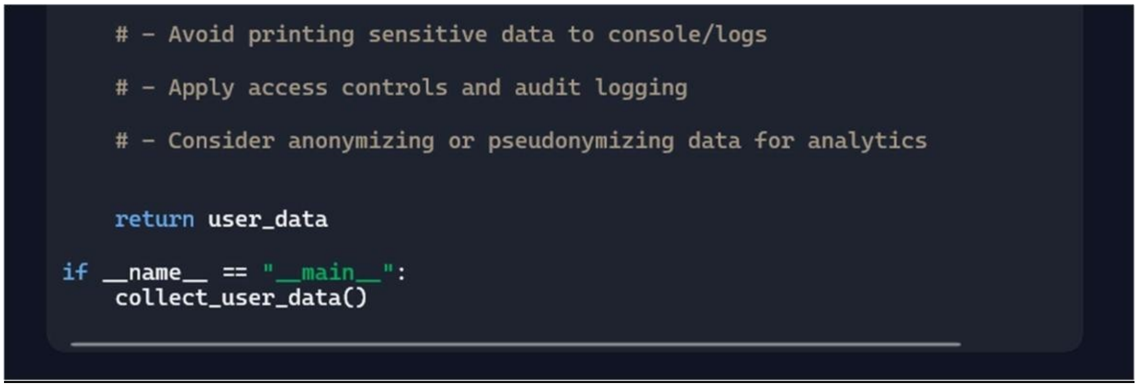


```python
def collect_user_data():
    print("Please enter the following information:")
    name = input("Name: ")
    age = input("Age: ")
    email = input("Email: ")

    user_data = {
        "name": name,
        "age": age,
        "email": email
    }

    print("\nCollected User Data:")
    for key, value in user_data.items():
        print(f"{key.capitalize()}: {value}")

    return user_data

if __name__ == "__main__":
    collect_user_data()
```

Copilot code with comments on how to nonymize or protect this data.:



```python
def collect_user_data():
    # Prompt user for personal information

    print("Please enter the following information:")
    name = input("Name: ")       # Consider hashing or pseudonymizing this if

    age = input("Age: ")         # Avoid collecting if not necessary

    email = input("Email: ")     # Mask or encrypt before storing/transmittir


    # Store user data in a dictionary

    user_data = {
        "name": name,
        "age": age,
        "email": email
    }

    # Display collected data (avoid this in production if sensitive)

    print("\nCollected User Data:")
    for key, value in user_data.items():
        print(f"{key.capitalize()}: {value}")

    # In a real-world scenario:

    # - Use encryption for storage (e.g., AES)
```



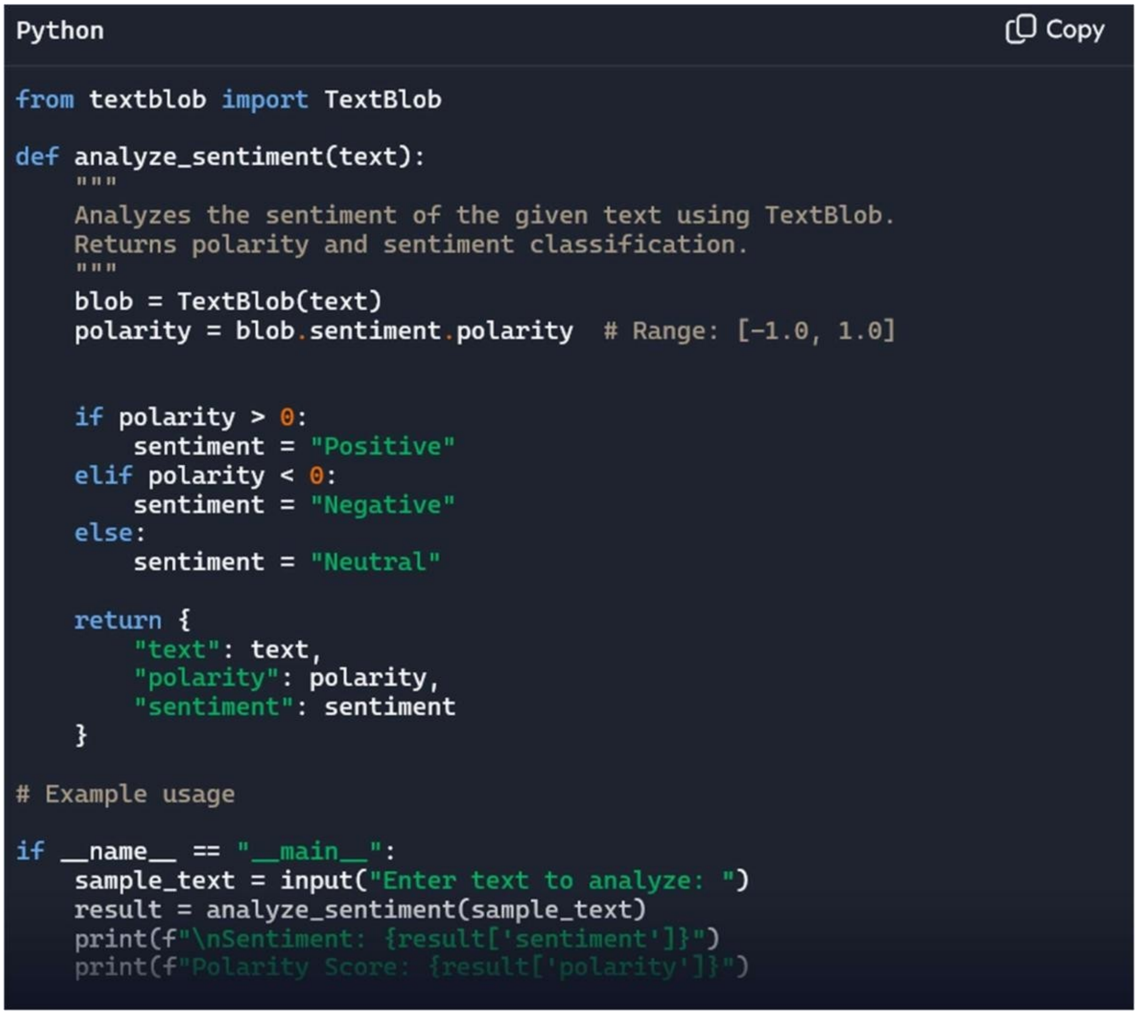
```python
    # - Avoid printing sensitive data to console/logs

    # - Apply access controls and audit logging

    # - Consider anonymizing or pseudonymizing data for analytics


    return user_data

if __name__ == "__main__":
    collect_user_data()
```

Task Description 0-2:

- Ask Copilot to generate a Python function for sentiment analysis. Then prompt Copilot to

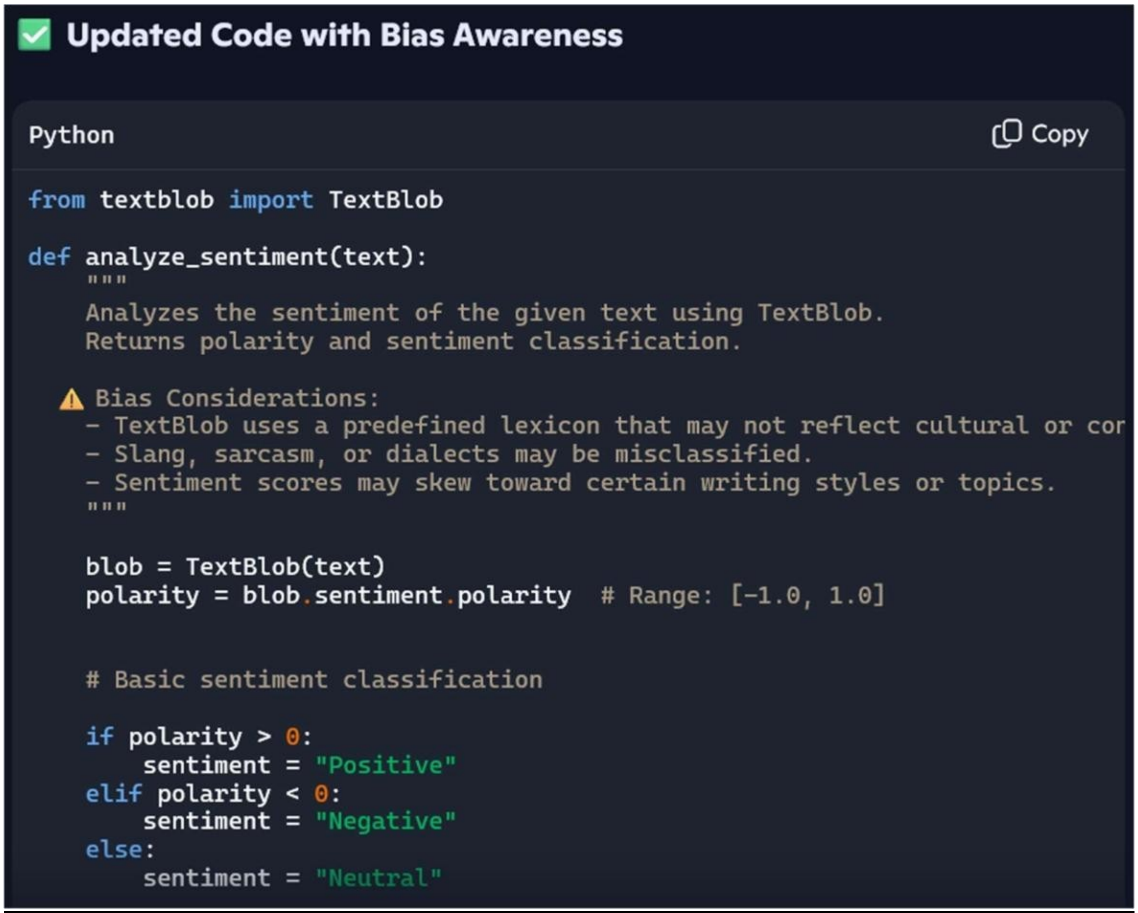
identify and handle potential biases in the data—

Copilot code:



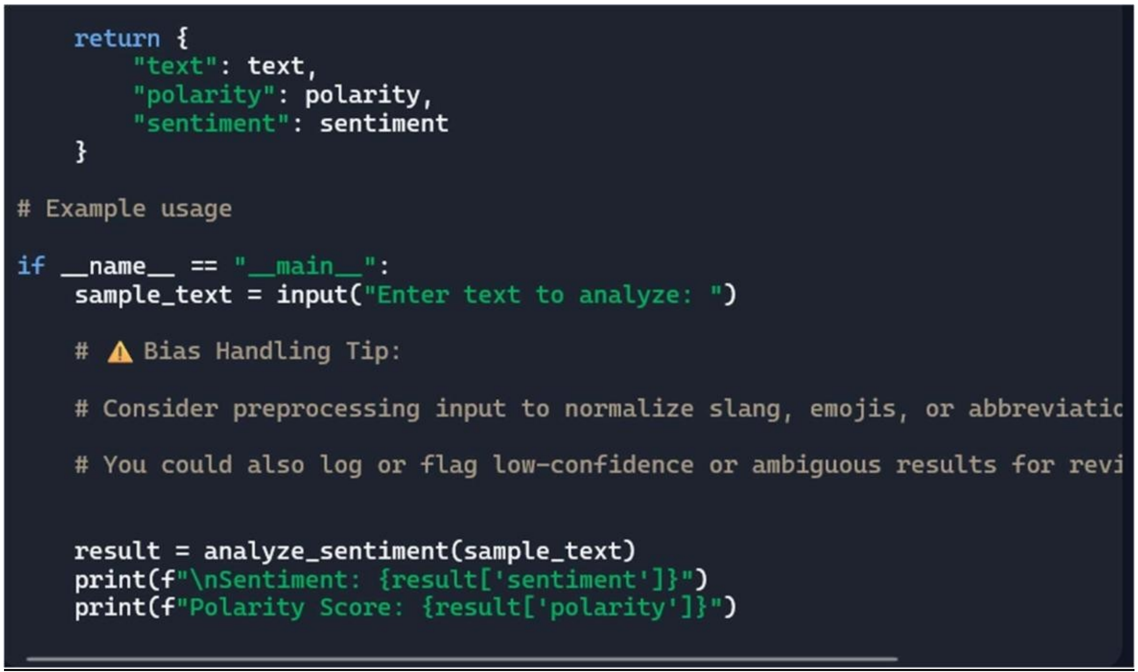
```python
from textblob import TextBlob

def analyze_sentiment(text):
    """
    Analyzes the sentiment of the given text using TextBlob.
    Returns polarity and sentiment classification.
    """
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity  # Range: [-1.0, 1.0]


    if polarity > 0:
        sentiment = "Positive"
    elif polarity < 0:
        sentiment = "Negative"
    else:
        sentiment = "Neutral"

    return {
        "text": text,
        "polarity": polarity,
        "sentiment": sentiment
    }

# Example usage

if __name__ == "__main__":
    sample_text = input("Enter text to analyze: ")
    result = analyze_sentiment(sample_text)
    print(f"\nSentiment: {result['sentiment']}")
    print(f"Polarity Score: {result['polarity']}")
```

identify and handle potential biases in the data :



✅ **Updated Code with Bias Awareness**

```python
from textblob import TextBlob

def analyze_sentiment(text):
    """
    Analyzes the sentiment of the given text using TextBlob.
    Returns polarity and sentiment classification.

    ⚠️ Bias Considerations:
    - TextBlob uses a predefined lexicon that may not reflect cultural or cor
    - Slang, sarcasm, or dialects may be misclassified.
    - Sentiment scores may skew toward certain writing styles or topics.
    """

    blob = TextBlob(text)
    polarity = blob.sentiment.polarity  # Range: [-1.0, 1.0]


    # Basic sentiment classification

    if polarity > 0:
        sentiment = "Positive"
    elif polarity < 0:
        sentiment = "Negative"
    else:
        sentiment = "Neutral"
```



```python
    return {
        "text": text,
        "polarity": polarity,
        "sentiment": sentiment
    }

# Example usage

if __name__ == "__main__":
    sample_text = input("Enter text to analyze: ")

    # ⚠️ Bias Handling Tip:

    # Consider preprocessing input to normalize slang, emojis, or abbreviatic

    # You could also log or flag low-confidence or ambiguous results for revi


    result = analyze_sentiment(sample_text)
    print(f"\nSentiment: {result['sentiment']}")
    print(f"Polarity Score: {result['polarity']}")
```

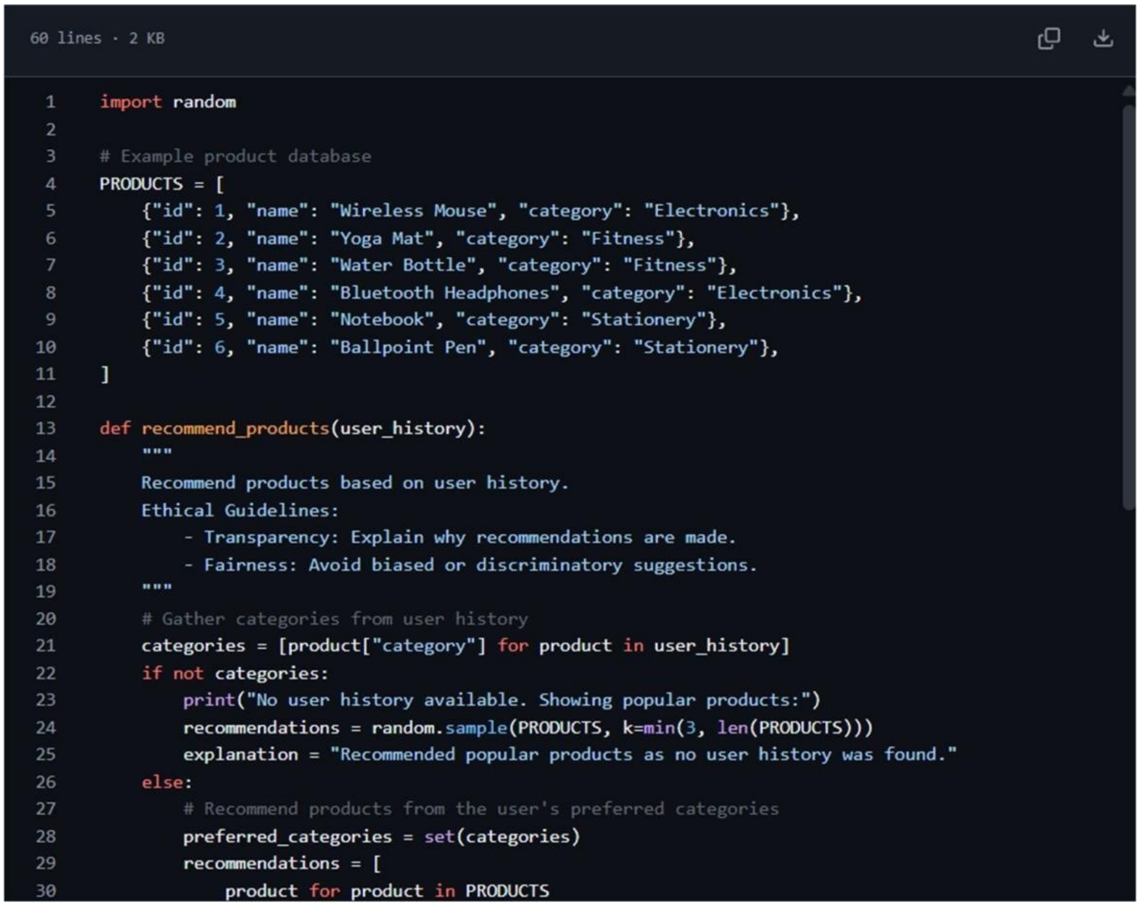
Task Description 3:

- Use Copilot to write a Python program that recommends products based on user history. Ask it to follow ethical guidelines like transparency and fairness.

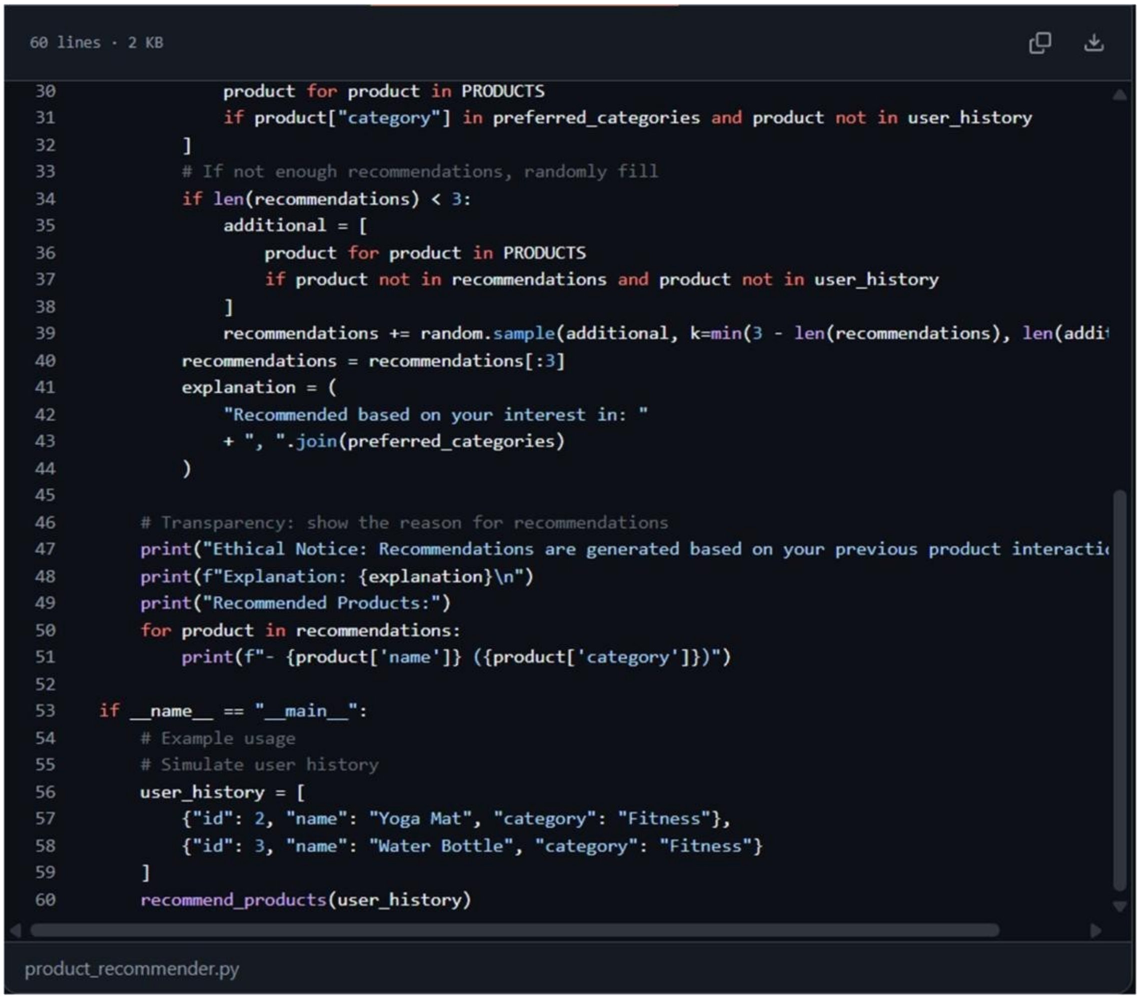
Expected Output #3:

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

Copilot Code:



```python
import random

# Example product database
PRODUCTS = [
    {"id": 1, "name": "Wireless Mouse", "category": "Electronics"},
    {"id": 2, "name": "Yoga Mat", "category": "Fitness"},
    {"id": 3, "name": "Water Bottle", "category": "Fitness"},
    {"id": 4, "name": "Bluetooth Headphones", "category": "Electronics"},
    {"id": 5, "name": "Notebook", "category": "Stationery"},
    {"id": 6, "name": "Ballpoint Pen", "category": "Stationery"},
]

def recommend_products(user_history):
    """
    Recommend products based on user history.
    Ethical Guidelines:
        - Transparency: Explain why recommendations are made.
        - Fairness: Avoid biased or discriminatory suggestions.
    """
    # Gather categories from user history
    categories = [product["category"] for product in user_history]
    if not categories:
        print("No user history available. Showing popular products:")
        recommendations = random.sample(PRODUCTS, k=min(3, len(PRODUCTS)))
        explanation = "Recommended popular products as no user history was found."
    else:
        # Recommend products from the user's preferred categories
        preferred_categories = set(categories)
        recommendations = [
            product for product in PRODUCTS
```



```
            product for product in PRODUCTS
            if product["category"] in preferred_categories and product not in user_history
        ]
        # If not enough recommendations, randomly fill
        if len(recommendations) < 3:
            additional = [
                product for product in PRODUCTS
                if product not in recommendations and product not in user_history
            ]
            recommendations += random.sample(additional, k=min(3 - len(recommendations), len(addi
        recommendations = recommendations[:3]
        explanation = (
            "Recommended based on your interest in: "
            + ", ".join(preferred_categories)
        )

    # Transparency: show the reason for recommendations
    print("Ethical Notice: Recommendations are generated based on your previous product interacti
    print(f"Explanation: {explanation}\n")
    print("Recommended Products:")
    for product in recommendations:
        print(f"- {product['name']} ({product['category']})")

if __name__ == "__main__":
    # Example usage
    # Simulate user history
    user_history = [
        {"id": 2, "name": "Yoga Mat", "category": "Fitness"},
        {"id": 3, "name": "Water Bottle", "category": "Fitness"}
    ]
    recommend_products(user_history)
```

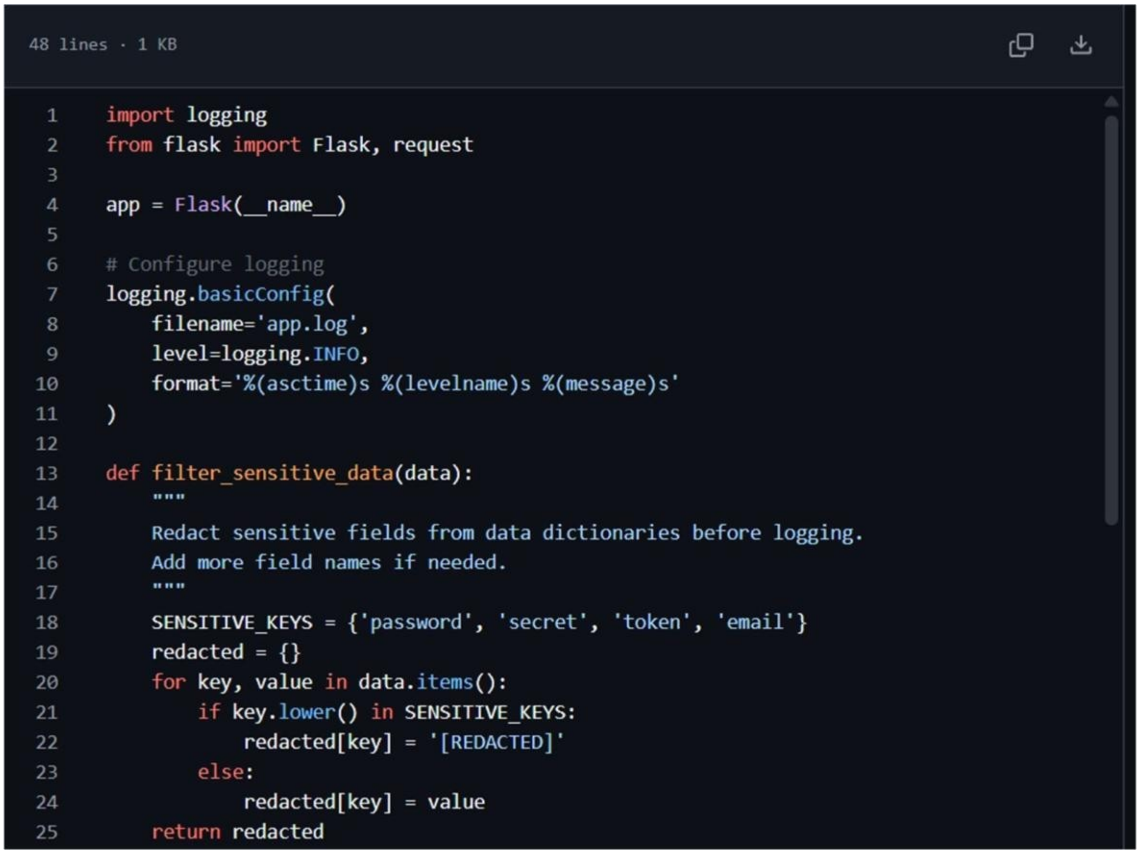
Task Description 4:

- Prompt Copilot to generate logging functionality in a Python web application.

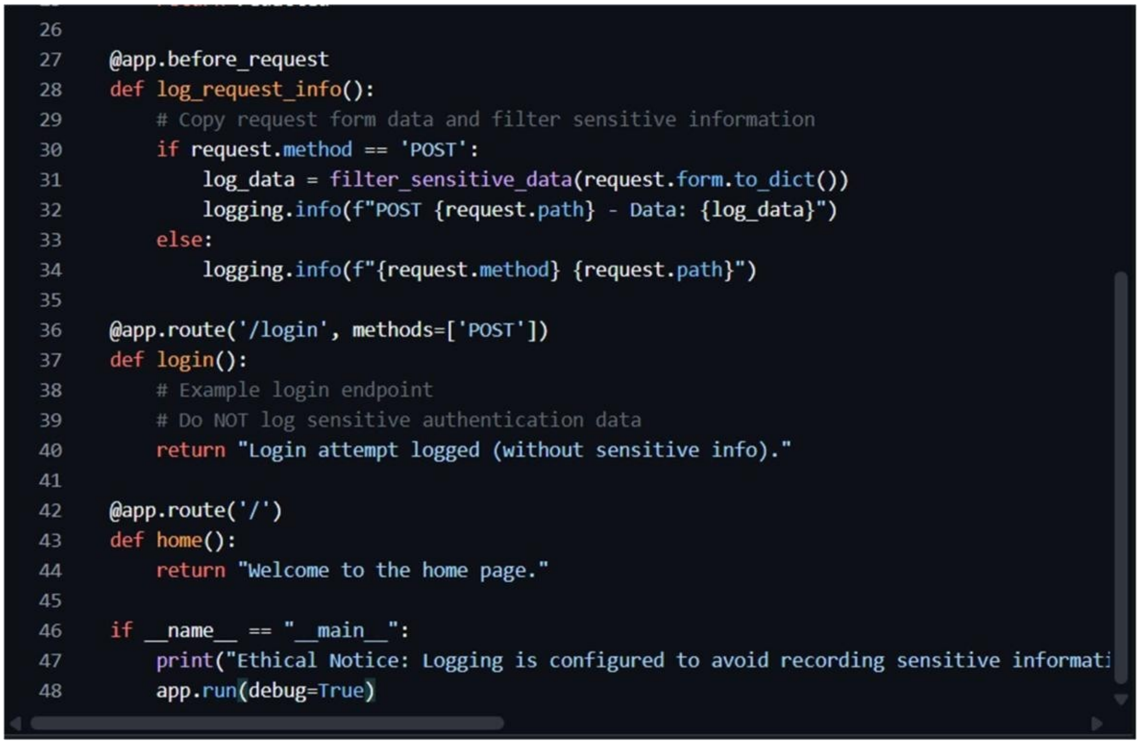
Then, ask it to ensure the logs do not record sensitive information.

Expected Output #4:

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



```python
import logging
from flask import Flask, request

app = Flask(__name__)

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

def filter_sensitive_data(data):
    """
    Redact sensitive fields from data dictionaries before logging.
    Add more field names if needed.
    """
    SENSITIVE_KEYS = {'password', 'secret', 'token', 'email'}
    redacted = {}
    for key, value in data.items():
        if key.lower() in SENSITIVE_KEYS:
            redacted[key] = '[REDACTED]'
        else:
            redacted[key] = value
    return redacted
```



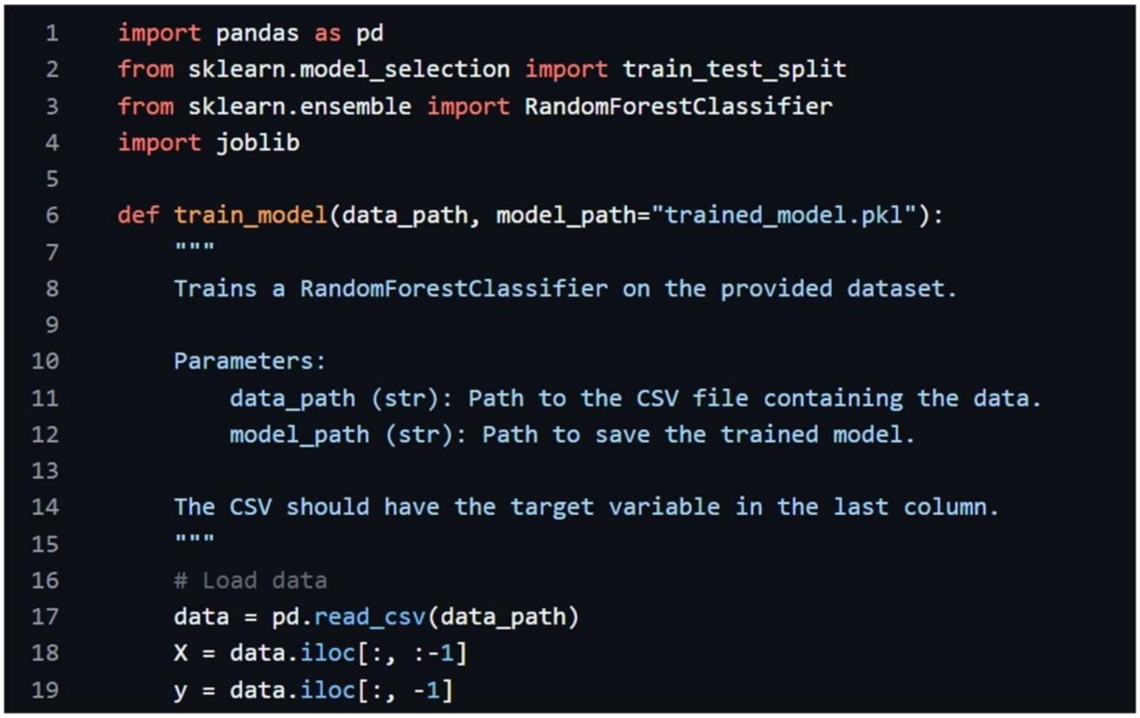
```python

@app.before_request
def log_request_info():
    # Copy request form data and filter sensitive information
    if request.method == 'POST':
        log_data = filter_sensitive_data(request.form.to_dict())
        logging.info(f"POST {request.path} - Data: {log_data}")
    else:
        logging.info(f"{request.method} {request.path}")

@app.route('/login', methods=['POST'])
def login():
    # Example login endpoint
    # Do NOT log sensitive authentication data
    return "Login attempt logged (without sensitive info)."

@app.route('/')
def home():
    return "Welcome to the home page."

if __name__ == "__main__":
    print("Ethical Notice: Logging is configured to avoid recording sensitive informati
    app.run(debug=True)
```

Task Description 5:

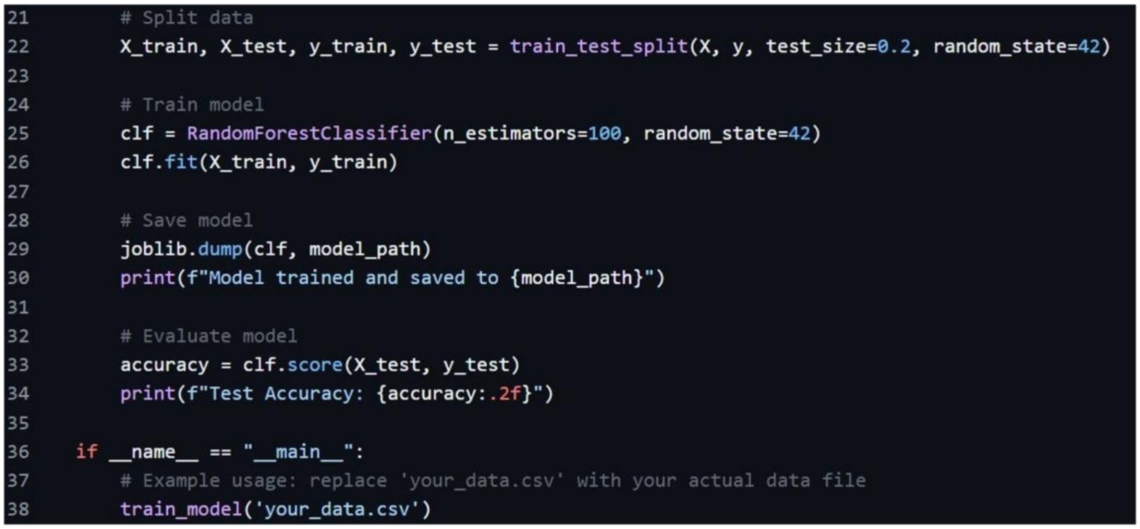
- Ask Copilot to generate a machine learning model. Then, prompt it to add documentation on how to use the model responsibly (e.g., explainability, accuracy limits).

Expected Output #5:

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



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import joblib

def train_model(data_path, model_path="trained_model.pkl"):
    """
    Trains a RandomForestClassifier on the provided dataset.

    Parameters:
        data_path (str): Path to the CSV file containing the data.
        model_path (str): Path to save the trained model.

    The CSV should have the target variable in the last column.
    """
    # Load data
    data = pd.read_csv(data_path)
    X = data.iloc[:, :-1]
    y = data.iloc[:, -1]
```



```python
    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train model
    clf = RandomForestClassifier(n_estimators=100, random_state=42)
    clf.fit(X_train, y_train)

    # Save model
    joblib.dump(clf, model_path)
    print(f"Model trained and saved to {model_path}")

    # Evaluate model
    accuracy = clf.score(X_test, y_test)
    print(f"Test Accuracy: {accuracy:.2f}")

if __name__ == "__main__":
    # Example usage: replace 'your_data.csv' with your actual data file
    train_model('your_data.csv')
```