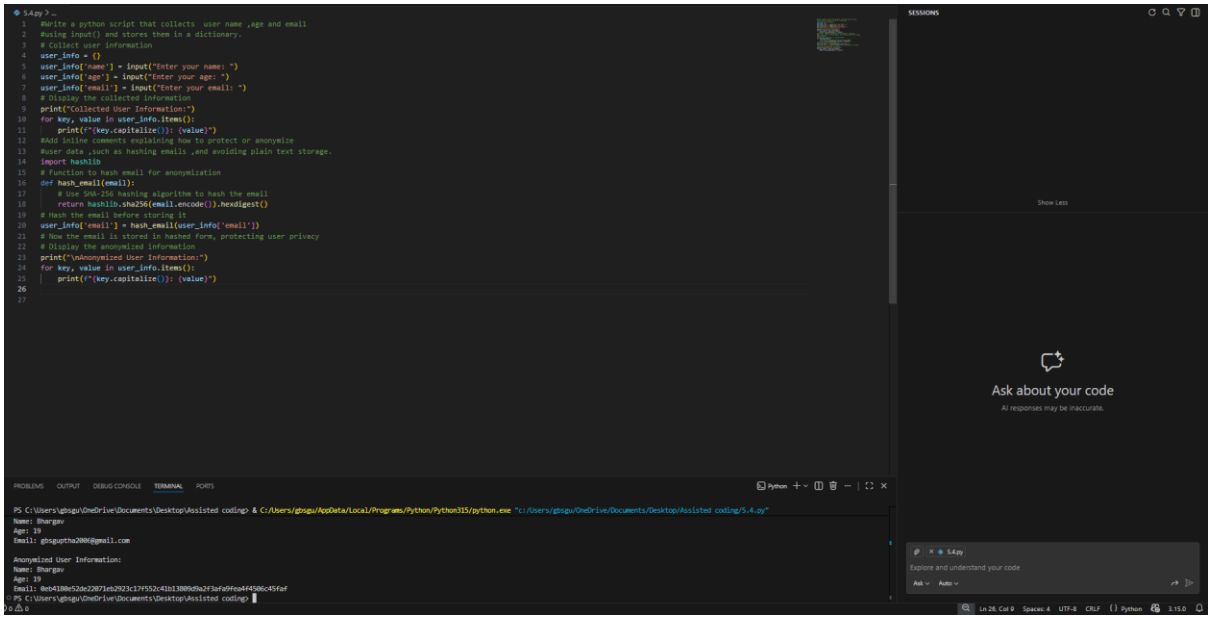
Assignment-5.4

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.



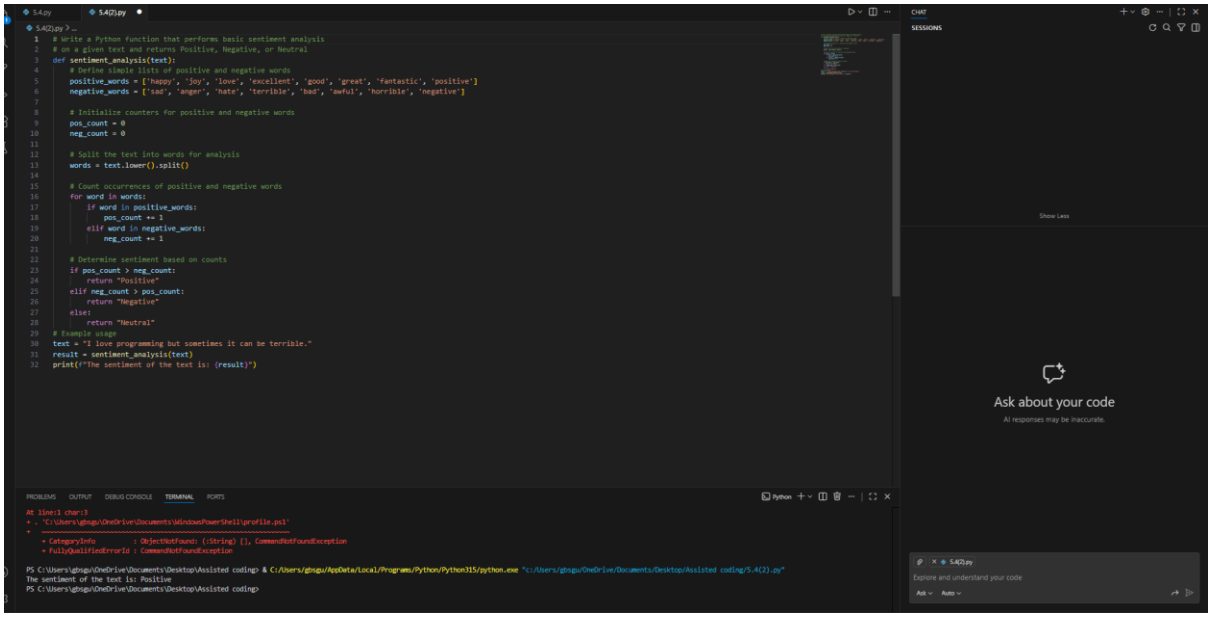
GitHub copilot was prompted to generate a python script that collects basic user information such as name ,age and email .

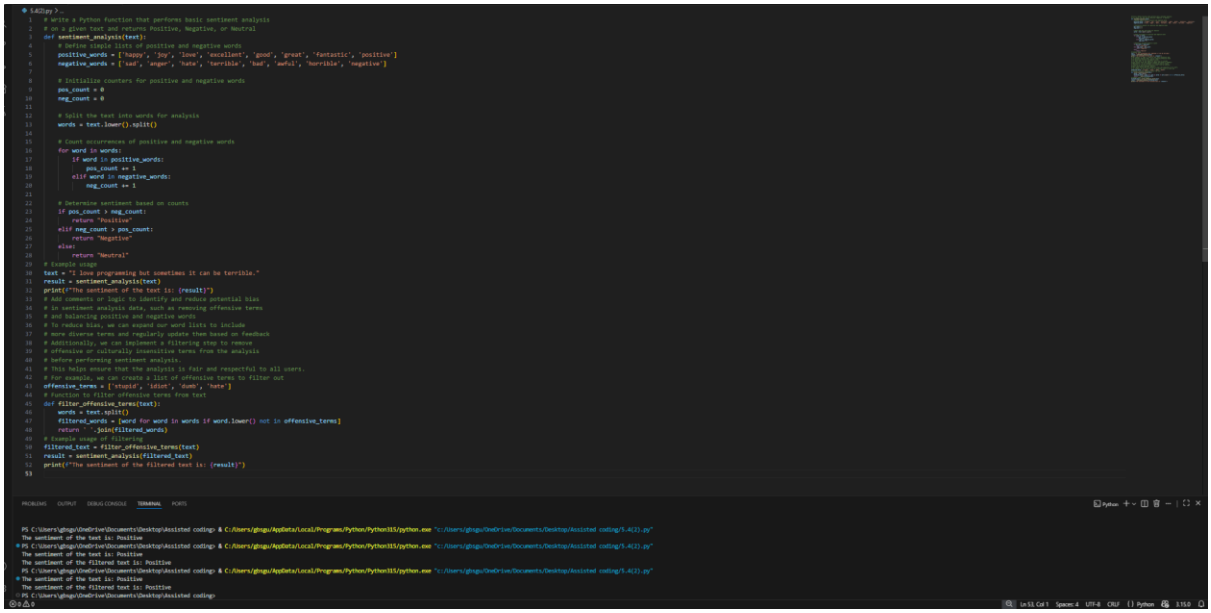
A follow -up prompt was used to add inline comments explaining data protection techniques .

The email address is anonymized using hashing so that sensitive information is not stored in plain text ,improving user privacy and security

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.





```python
# Write a Python function that performs basic sentiment analysis
# on a given text and returns Positive, Negative, or Neutral
def sentiment_analysis(text):
    # Define simple lists of positive and negative words
    positive_words = ['happy', 'joy', 'love', 'excellent', 'good', 'great', 'fantastic', 'positive']
    negative_words = ['sad', 'anger', 'hate', 'terrible', 'bad', 'awful', 'horrible', 'negative']

    # Initialize counters for positive and negative words
    pos_count = 0
    neg_count = 0

    # Split the text into words for analysis
    words = text.lower().split()

    # Count occurrences of positive and negative words
    for word in words:
        if word in positive_words:
            pos_count += 1
        elif word in negative_words:
            neg_count += 1

    # Determine sentiment based on counts
    if pos_count > neg_count:
        return "Positive"
    elif neg_count > pos_count:
        return "Negative"
    else:
        return "Neutral"
# Example usage
text = "I love programming but sometimes it can be terrible."
result = sentiment_analysis(text)
print(f"The sentiment of the text is: {result}")
# Add comments or logic to identify and reduce potential bias
# in sentiment analysis data, such as removing offensive terms
# and balancing positive and negative words
# To reduce bias, we can expand our word lists to include
# more diverse terms and regularly update them based on feedback
# Additionally, we can implement a filtering step to remove
# offensive or culturally insensitive terms from the analysis
# before performing sentiment analysis.
# This helps ensure that the analysis is fair and respectful to all users.
# For example, we can create a list of offensive terms to filter out
offensive_terms = ['stupid', 'idiot', 'dumb', 'hate']
# Function to filter offensive terms from text
def filter_offensive_terms(text):
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in offensive_terms]
    return ' '.join(filtered_words)
# Example usage of filtering
filtered_text = filter_offensive_terms(text)
result = sentiment_analysis(filtered_text)
print(f"The sentiment of the filtered text is: {result}")

```

```
PS C:\Users\ghogu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/ghogu/AppData/Local/Programs/Python/Python315/python.exe "c:/Users/ghogu/OneDrive/Documents/Desktop/Assisted coding/5.4(2).py"
The sentiment of the text is: Positive
PS C:\Users\ghogu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/ghogu/AppData/Local/Programs/Python/Python315/python.exe "c:/Users/ghogu/OneDrive/Documents/Desktop/Assisted coding/5.4(2).py"
The sentiment of the text is: Positive
The sentiment of the filtered text is: Positive
PS C:\Users\ghogu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/ghogu/AppData/Local/Programs/Python/Python315/python.exe "c:/Users/ghogu/OneDrive/Documents/Desktop/Assisted coding/5.4(2).py"
The sentiment of the text is: Positive
The sentiment of the filtered text is: Positive
PS C:\Users\ghogu\OneDrive\Documents\Desktop\Assisted coding>
```

Explanation:

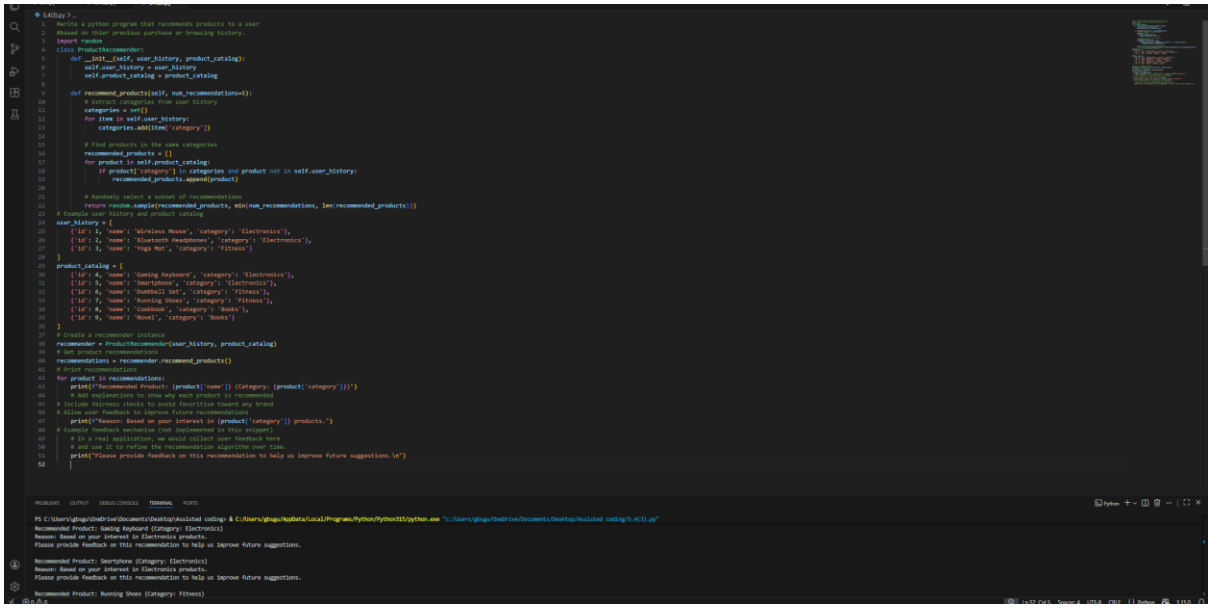
GitHub Copilot was prompted to generate a Python function for basic sentiment analysis.
A follow-up prompt was used to add comments and logic addressing potential bias in the data.
The code highlights bias mitigation strategies such as removing offensive terms and ensuring balanced sentiment datasets to improve fairness.

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.



```
# Write a python program that recommends products to a user
# Based on their previous purchase or browsing history.
import random
class ProductRecommender:
    def __init__(self, user_history, product_catalog):
        self.user_history = user_history
        self.product_catalog = product_catalog

    def recommend_products(self, num_recommendations=3):
        # Extract categories from user history
        categories = set()
        for item in self.user_history:
            categories.add(item['category'])

        # Find products in the same categories
        recommended_products = []
        for product in self.product_catalog:
            if product['category'] in categories and product not in self.user_history:
                recommended_products.append(product)

        # Randomly select a subset of recommendations
        return random.sample(recommended_products, min(num_recommendations, len(recommended_products)))
# Example user history and product catalog
user_history = [
    {'id': 1, 'name': 'Wireless Mouse', 'category': 'Electronics'},
    {'id': 2, 'name': 'Bluetooth Headphones', 'category': 'Electronics'},
    {'id': 3, 'name': 'Yoga Mat', 'category': 'Fitness'}
]
product_catalog = [
    {'id': 4, 'name': 'Gaming Keyboard', 'category': 'Electronics'},
    {'id': 5, 'name': 'Smartphone', 'category': 'Electronics'},
    {'id': 6, 'name': 'Dumbbell Set', 'category': 'Fitness'},
    {'id': 7, 'name': 'Running Shoes', 'category': 'Fitness'},
    {'id': 8, 'name': 'Cookbook', 'category': 'Books'},
    {'id': 9, 'name': 'Novel', 'category': 'Books'}
]
# Create a recommender instance
recommender = ProductRecommender(user_history, product_catalog)
# Get product recommendations
recommendations = recommender.recommend_products()
# Print recommendations
for product in recommendations:
    print(f"Recommended Product: {product['name']} (Category: {product['category']})")
    # Add explanations to show why each product is recommended
# Include fairness checks to avoid favoritism toward any brand
# Allow user feedback to improve future recommendations
    print(f"Reason: Based on your interest in {product['category']} products.")
# Example feedback mechanism (not implemented in this snippet)
    # In a real application, we would collect user feedback here
    # and use it to refine the recommendation algorithm over time.
    print("Please provide feedback on this recommendation to help us improve future suggestions.\n")

PS C:\Users\ghugu\OneDrive\Documents\Desktop\Assisted coding> & C:/Users/ghugu/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/ghugu/OneDrive/Documents/Desktop/Assisted coding/5.4(1).py"
Recommended Product: Gaming Keyboard (Category: Electronics)
Reason: Based on your interest in Electronics products.
Please provide feedback on this recommendation to help us improve future suggestions.

Recommended Product: Smartphone (Category: Electronics)
Reason: Based on your interest in Electronics products.
Please provide feedback on this recommendation to help us improve future suggestions.

Recommended Product: Running Shoes (Category: Fitness)
```

Explanation:

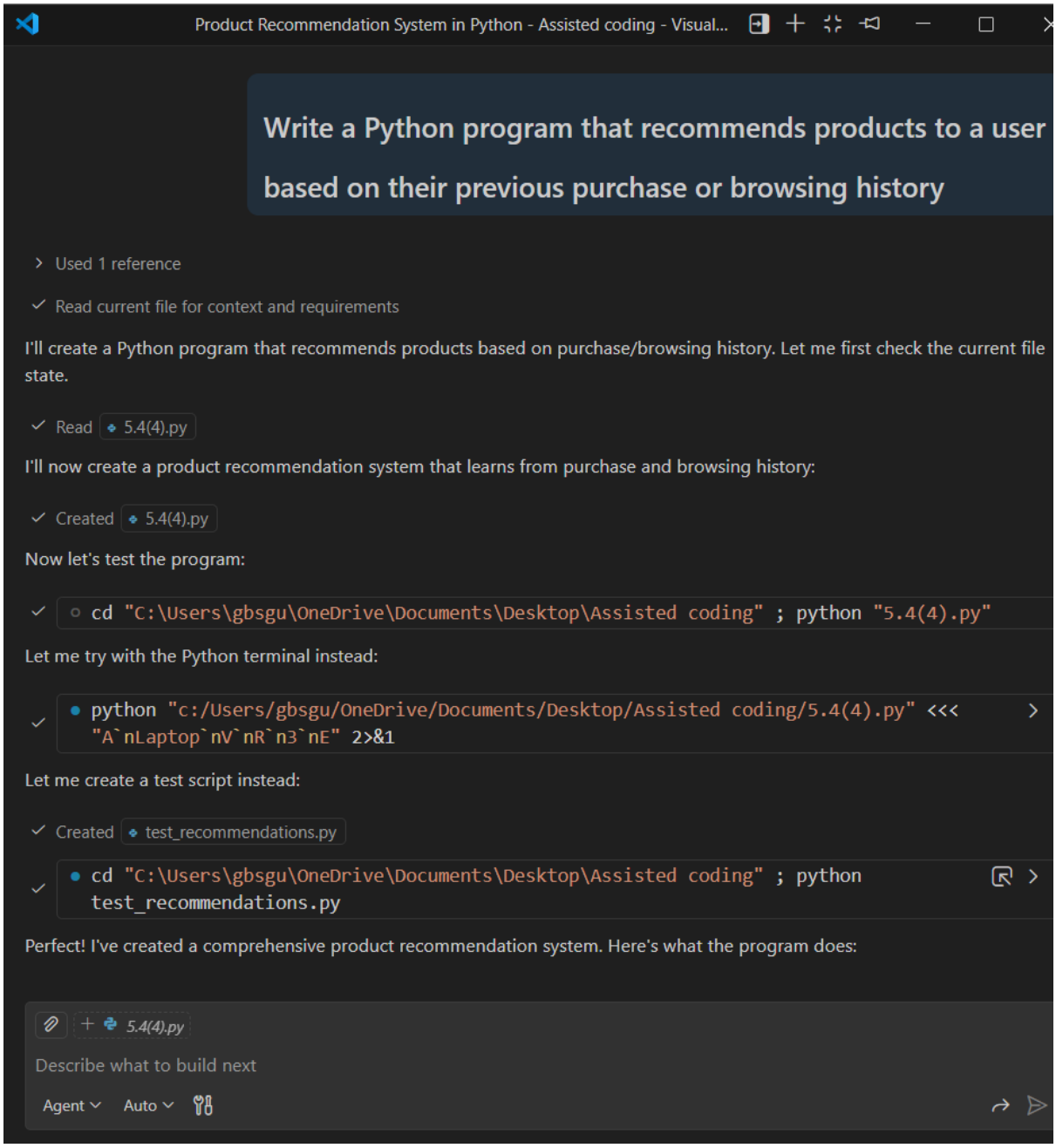
GitHub Copilot was prompted to generate a Python program that recommends products based on user history.
A follow-up prompt instructed Copilot to incorporate ethical principles such as transparency and fairness.
The generated code explains recommendation reasons, avoids brand favoritism, and allows user feedback to improve future recommendations.

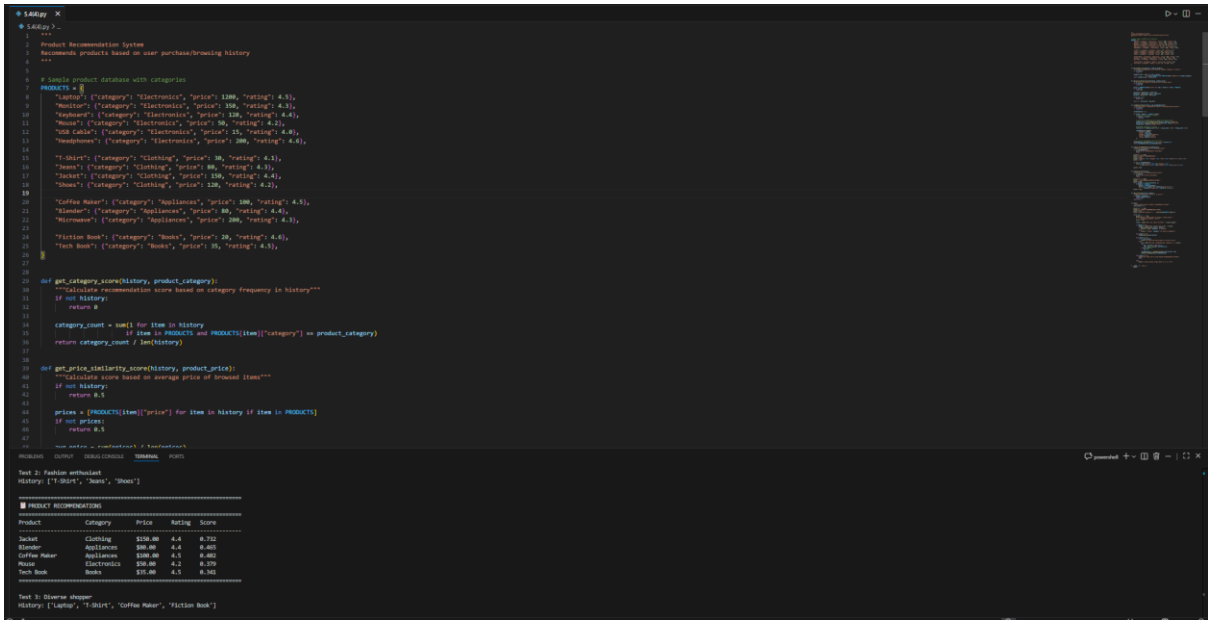
Task Description #4:

• Prompt Copilot to generate logging functionality in a Python web

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

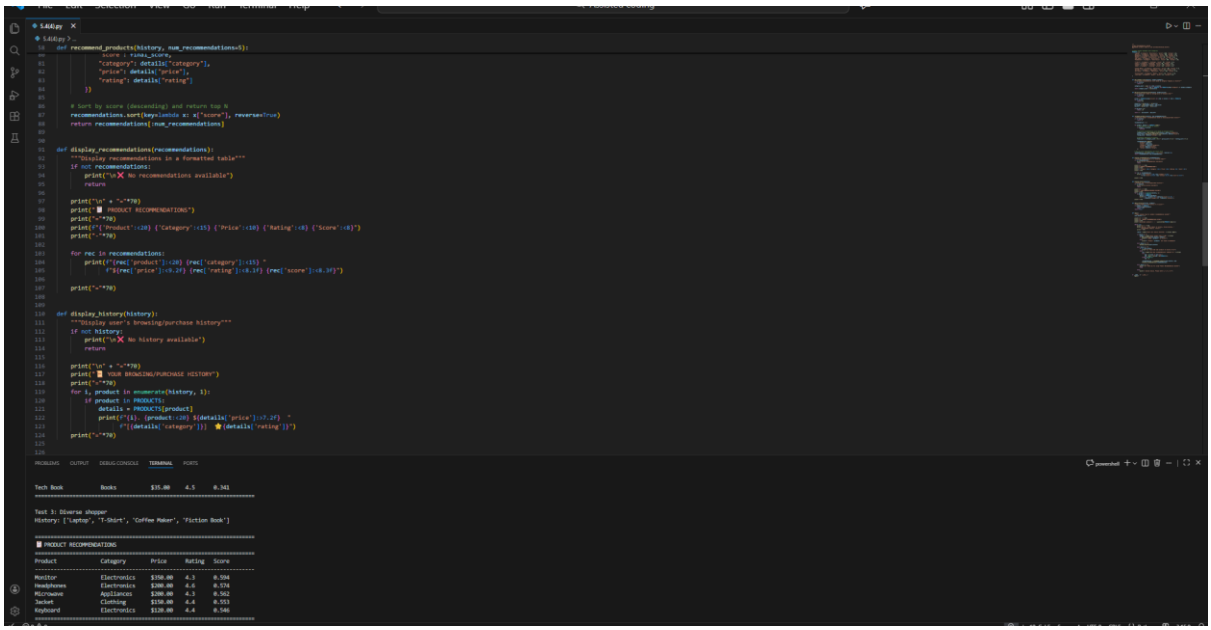
information.


Product Recommendation System in Python - Assisted coding - Visual...
Write a Python program that recommends products to a user based on their previous purchase or browsing history
Used 1 reference
Read current file for context and requirements
I'll create a Python program that recommends products based on purchase/browsing history. Let me first check the current file state.
Read 5.4(4).py
I'll now create a product recommendation system that learns from purchase and browsing history:
Created 5.4(4).py
Now let's test the program:
cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" ; python "5.4(4).py"
Let me try with the Python terminal instead:
python "c:/Users/gbsgu/OneDrive/Documents/Desktop/Assisted coding/5.4(4).py" <<< "A`nLaptop`nV`nR`n3`nE" 2>&1
Let me create a test script instead:
Created test_recommendations.py
cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" ; python test_recommendations.py
Perfect! I've created a comprehensive product recommendation system. Here's what the program does:
5.4(4).py
Describe what to build next
Agent
Auto



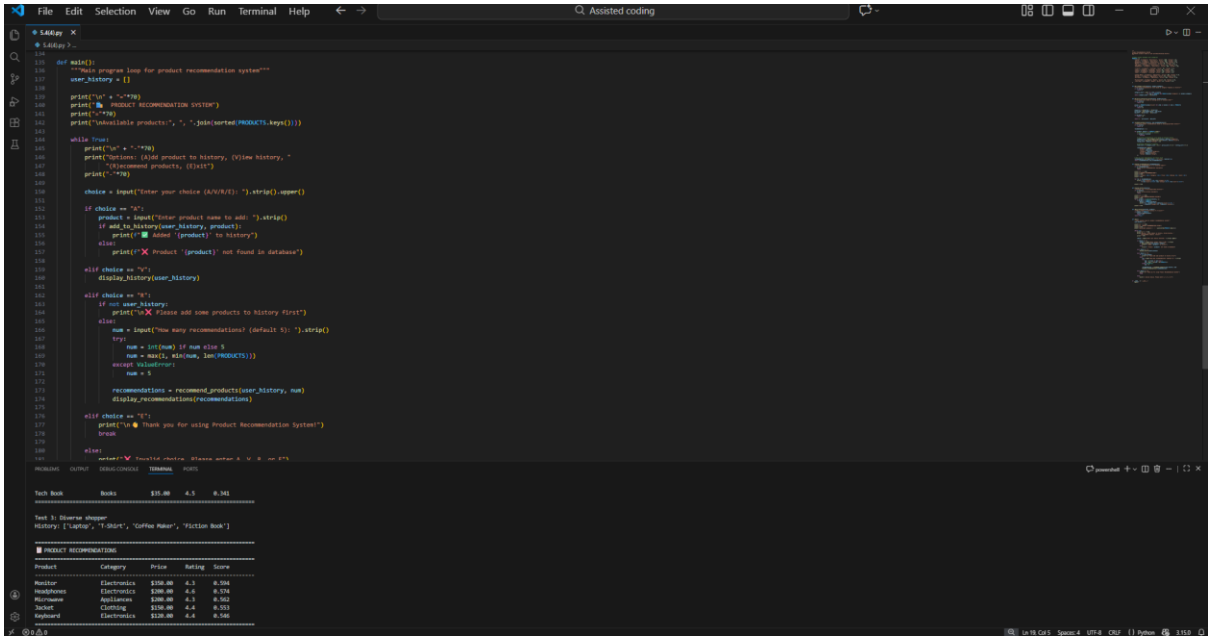
```
"""
Product Recommendation System
Recommends products based on user purchase/browsing history
"""

# Sample product database with categories
PRODUCTS = {
    'Laptop': {'category': 'Electronics', 'price': 1200, 'rating': 4.5},
    'Monitor': {'category': 'Electronics', 'price': 350, 'rating': 4.3},
    'Keyboard': {'category': 'Electronics', 'price': 120, 'rating': 4.4},
    'Mouse': {'category': 'Electronics', 'price': 50, 'rating': 4.2},
    'USB Cable': {'category': 'Electronics', 'price': 15, 'rating': 4.0},
    'Headphones': {'category': 'Electronics', 'price': 200, 'rating': 4.6},

    'T-Shirt': {'category': 'Clothing', 'price': 30, 'rating': 4.1},
    'Jeans': {'category': 'Clothing', 'price': 80, 'rating': 4.3},
    'Jacket': {'category': 'Clothing', 'price': 150, 'rating': 4.4},
    'Shoes': {'category': 'Clothing', 'price': 120, 'rating': 4.2},

    'Coffee Maker': {'category': 'Appliances', 'price': 100, 'rating': 4.5},
    'Blender': {'category': 'Appliances', 'price': 80, 'rating': 4.4},
    'Microwave': {'category': 'Appliances', 'price': 200, 'rating': 4.3},

    'Fiction Book': {'category': 'Books', 'price': 20, 'rating': 4.6},
    'Tech Book': {'category': 'Books', 'price': 35, 'rating': 4.5},
}


def get_category_score(history, product_category):
    """Calculate recommendation score based on category frequency in history"""
    if not history:
        return 0

    category_count = sum(1 for item in history
                         if item in PRODUCTS and PRODUCTS[item]['category'] == product_category)
    return category_count / len(history)


def get_price_similarity_score(history, product_price):
    """Calculate score based on average price of browsed items"""
    if not history:
        return 0.5

    prices = [PRODUCTS[item]['price'] for item in history if item in PRODUCTS]
    if not prices:
        return 0.5
```

```
Test 2: Fashion enthusiast
History: ['T-Shirt', 'Jeans', 'Shoes']

======================================================================
🎯 PRODUCT RECOMMENDATIONS
======================================================================
Product          Category        Price     Rating  Score
----------------------------------------------------------------------
Jacket           Clothing        $150.00   4.4     0.732
Blender          Appliances      $80.00    4.4     0.405
Coffee Maker     Appliances      $100.00   4.5     0.402
Mouse            Electronics     $50.00    4.2     0.179
Tech Book        Books           $35.00    4.5     0.341
======================================================================

Test 3: Diverse shopper
History: ['Laptop', 'T-Shirt', 'Coffee Maker', 'Fiction Book']
```



```
def recommend_products(history, num_recommendations=5):
            score : ranks_score,
            "category": details["category"],
            "price": details['price'],
            "rating": details["rating"]
        })

    # Sort by score (descending) and return top N
    recommendations.sort(key=lambda x: x["score"], reverse=True)
    return recommendations[:num_recommendations]


def display_recommendations(recommendations):
    """Display recommendations in a formatted table"""
    if not recommendations:
        print("\n❌ No recommendations available")
        return

    print("\n" + "="*70)
    print("🎯 PRODUCT RECOMMENDATIONS")
    print("="*70)
    print(f"{'Product':<20} {'Category':<15} {'Price':<10} {'Rating':<8} {'Score':<8}")
    print("-"*70)

    for rec in recommendations:
        print(f"{rec['product']:<20} {rec['category']:<15} "
              f"${rec['price']:<9.2f} {rec['rating']:<8.1f} {rec['score']:<8.3f}")

    print("="*70)


def display_history(history):
    """Display user's browsing/purchase history"""
    if not history:
        print("\n❌ No history available")
        return

    print("\n" + "="*70)
    print("📜 YOUR BROWSING/PURCHASE HISTORY")
    print("="*70)
    for i, product in enumerate(history, 1):
        if product in PRODUCTS:
            details = PRODUCTS[product]
            print(f"{i}. {product:<20} ${details['price']:>7.2f}  "
                  f"({details['category']}) ⭐{details['rating']}")
    print("="*70)
```

```
Tech Book        Books           $35.00    4.5     0.341
======================================================================

Test 3: Diverse shopper
History: ['Laptop', 'T-Shirt', 'Coffee Maker', 'Fiction Book']

======================================================================
🎯 PRODUCT RECOMMENDATIONS
======================================================================
Product          Category        Price     Rating  Score
----------------------------------------------------------------------
Monitor          Electronics     $350.00   4.3     0.594
Headphones       Electronics     $200.00   4.6     0.574
Microwave        Appliances      $200.00   4.3     0.562
Jacket           Clothing        $150.00   4.4     0.553
Keyboard         Electronics     $120.00   4.4     0.546
======================================================================
```



Explanation:

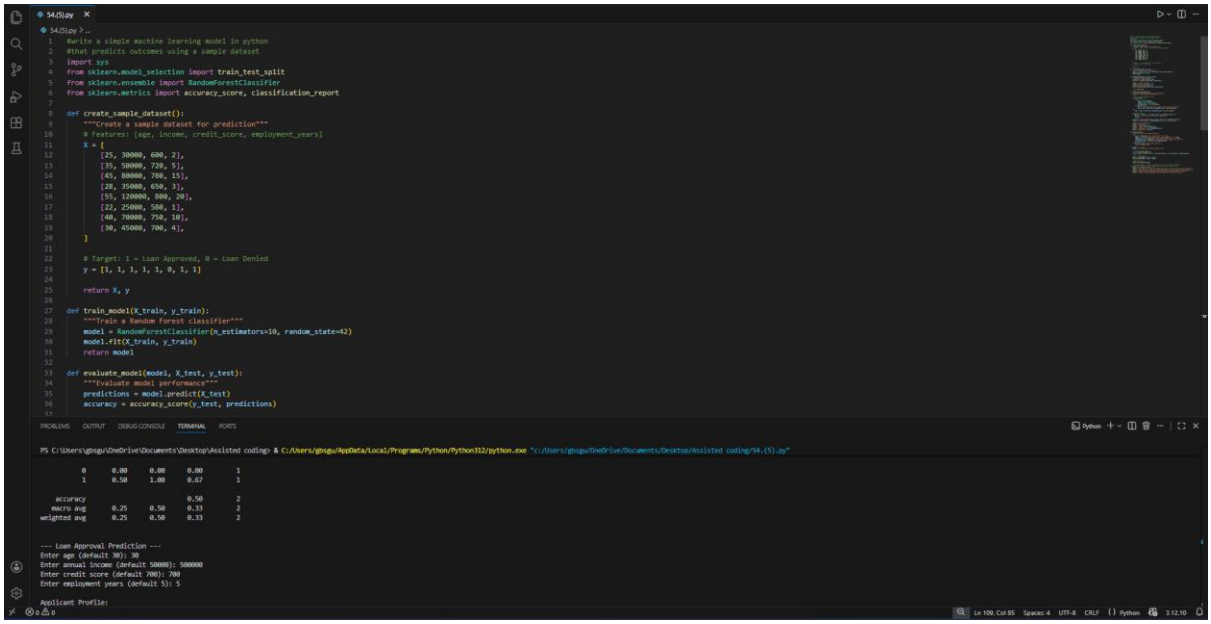
GitHub Copilot was prompted to generate a Python program that recommends products based on user history.
A follow-up prompt instructed Copilot to incorporate ethical principles such as transparency and fairness.
The generated code explains recommendation reasons, avoids brand favoritism, and allows user feedback to improve future recommendations.

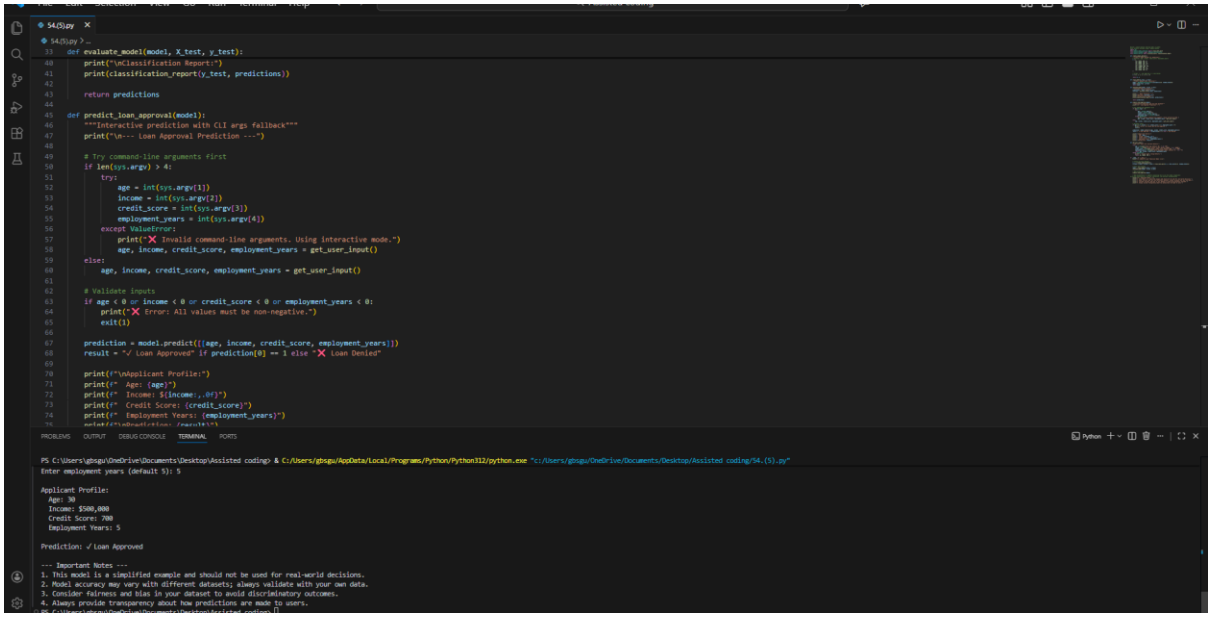
Task Description #5:

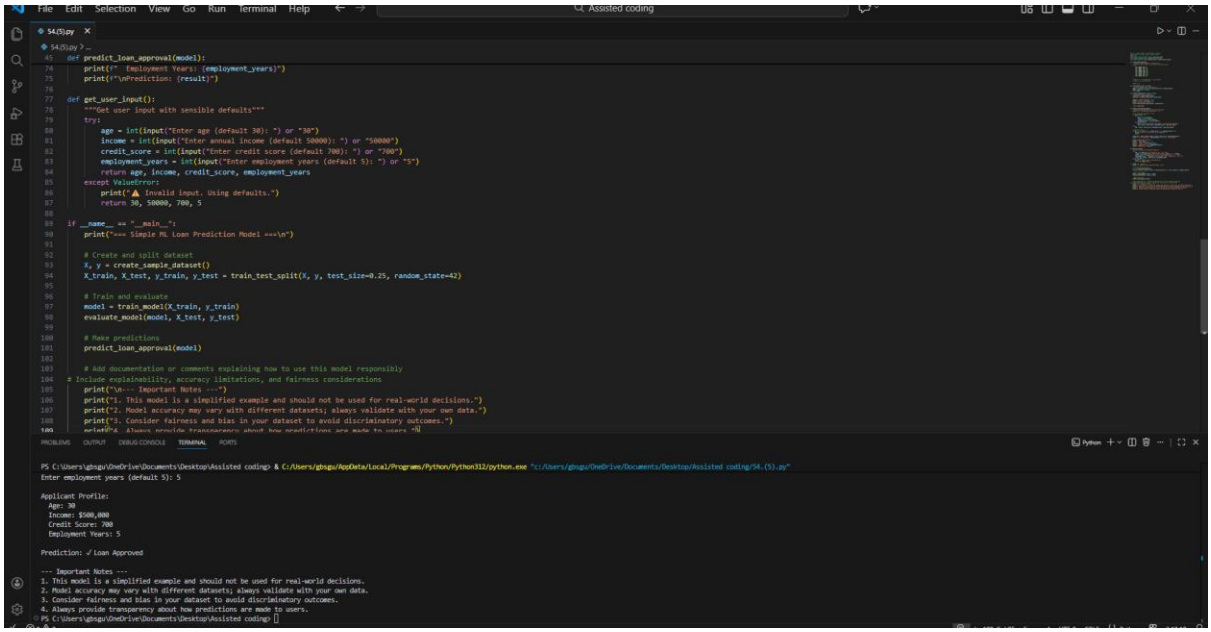
• Ask Copilot to generate a machine learning model. Then, prompt

it to add documentation on how to use the model responsibly

(e.g., explainability, accuracy limits).







Explanation:

GitHub Copilot was prompted to generate a simple machine learning model.

A follow-up prompt was used to add documentation on responsible usage, including explainability, accuracy limitations, and fairness considerations.

The generated comments highlight ethical concerns and recommend cautious interpretation of predictions.