# Lab 5.4: Ethical Foundations– Responsible AI Coding Practices

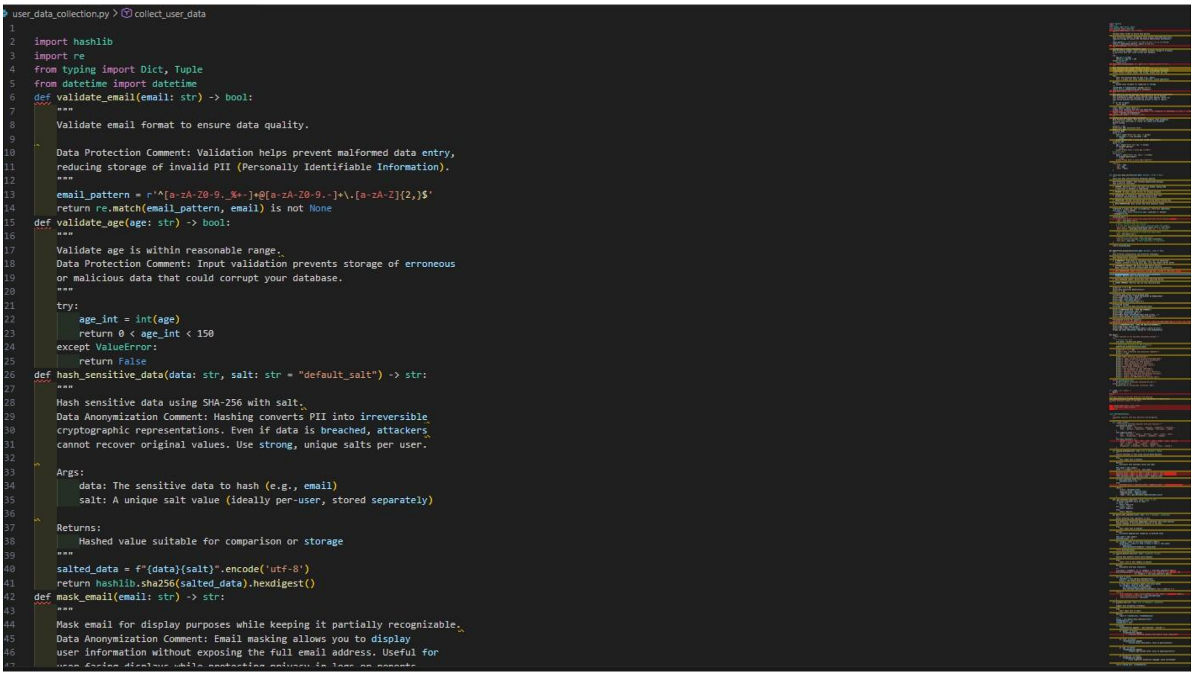
Name:chenchu Raju

Hallticket:2303A52206

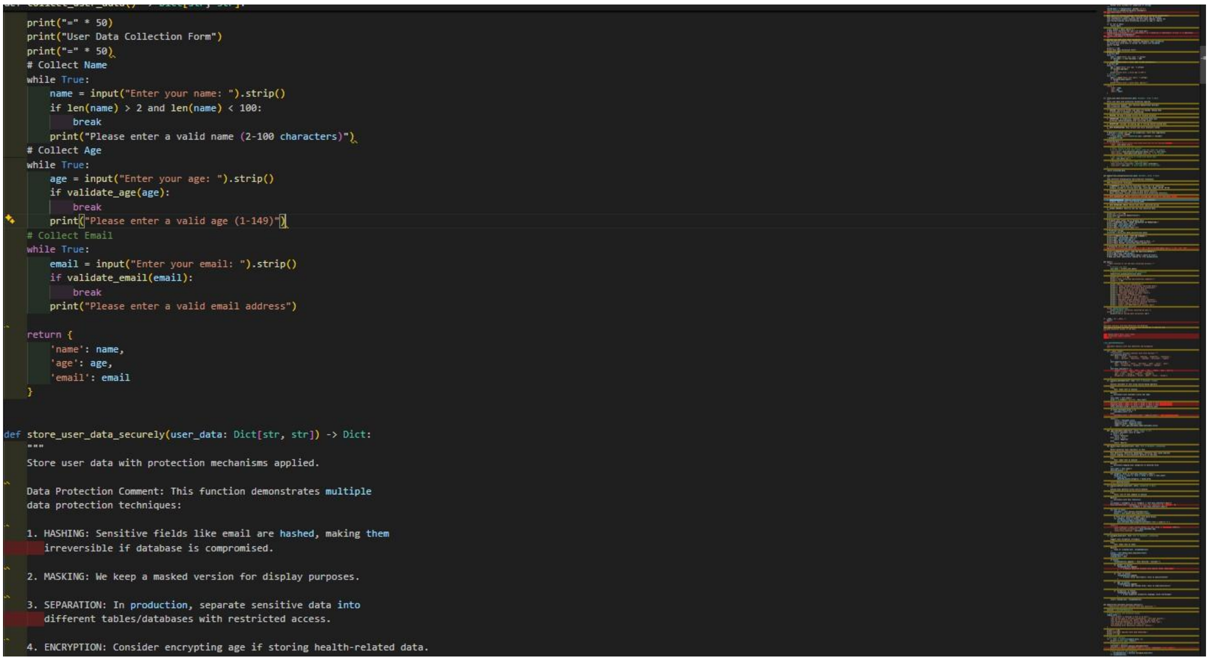
Task1:

Prompt: 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.

Code:

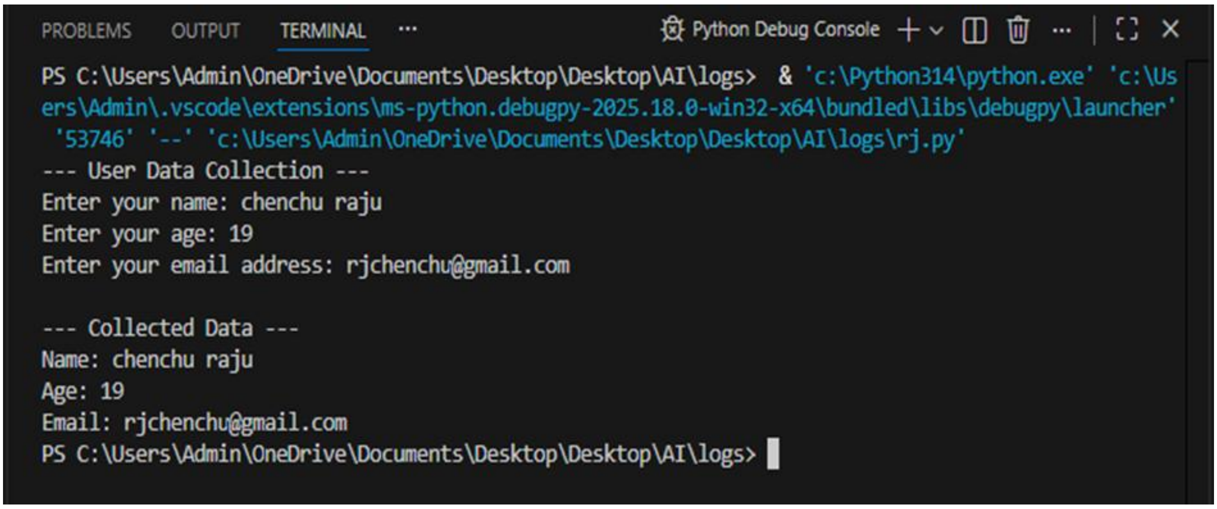


```python

import hashlib
import re
from typing import Dict, Tuple
from datetime import datetime
def validate_email(email: str) -> bool:
    """
    Validate email format to ensure data quality.

    Data Protection Comment: Validation helps prevent malformed data entry,
    reducing storage of invalid PII (Personally Identifiable Information).
    """
    email_pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(email_pattern, email) is not None
def validate_age(age: str) -> bool:
    """
    Validate age is within reasonable range.
    Data Protection Comment: Input validation prevents storage of erroneous
    or malicious data that could corrupt your database.
    """
    try:
        age_int = int(age)
        return 0 < age_int < 150
    except ValueError:
        return False
def hash_sensitive_data(data: str, salt: str = "default_salt") -> str:
    """
    Hash sensitive data using SHA-256 with salt.
    Data Anonymization Comment: Hashing converts PII into irreversible
    cryptographic representations. Even if data is breached, attackers
    cannot recover original values. Use strong, unique salts per user.

    Args:
        data: The sensitive data to hash (e.g., email)
        salt: A unique salt value (ideally per-user, stored separately)

    Returns:
        Hashed value suitable for comparison or storage
    """
    salted_data = f"{data}{salt}".encode('utf-8')
    return hashlib.sha256(salted_data).hexdigest()
def mask_email(email: str) -> str:
    """
    Mask email for display purposes while keeping it partially recognizable.
    Data Anonymization Comment: Email masking allows you to display
    user information without exposing the full email address. Useful for
    user-facing displays while protecting privacy in logs or reports.
```



```
    print("=" * 50)
    print("User Data Collection Form")
    print("=" * 50)
    # Collect Name
    while True:
        name = input("Enter your name: ").strip()
        if len(name) > 2 and len(name) < 100:
            break
        print("Please enter a valid name (2-100 characters)")
    # Collect Age
    while True:
        age = input("Enter your age: ").strip()
        if validate_age(age):
            break
        print("Please enter a valid age (1-149)")
    # Collect Email
    while True:
        email = input("Enter your email: ").strip()
        if validate_email(email):
            break
        print("Please enter a valid email address")

    return {
        'name': name,
        'age': age,
        'email': email
    }


def store_user_data_securely(user_data: Dict[str, str]) -> Dict:
    """
    Store user data with protection mechanisms applied.

    Data Protection Comment: This function demonstrates multiple
    data protection techniques:

    1. HASHING: Sensitive fields like email are hashed, making them
       irreversible if database is compromised.

    2. MASKING: We keep a masked version for display purposes.

    3. SEPARATION: In production, separate sensitive data into
       different tables/databases with restricted access.

    4. ENCRYPTION: Consider encrypting age if storing health-related data.
```

Output:



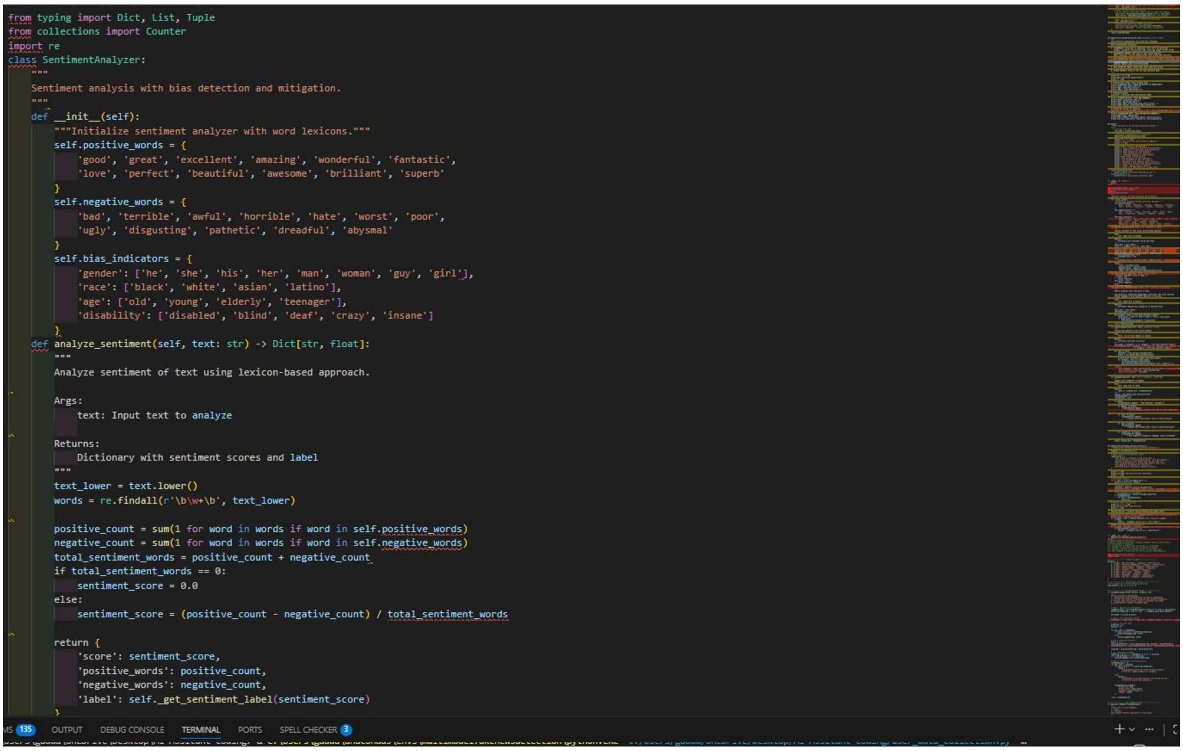
```
PS C:\Users\Admin\OneDrive\Documents\Desktop\Desktop\AI\logs>  & 'c:\Python314\python.exe' 'c:\Us
ers\Admin\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher'
 '53746' '--' 'c:\Users\Admin\OneDrive\Documents\Desktop\Desktop\AI\logs\rj.py'
--- User Data Collection ---
Enter your name: chenchu raju
Enter your age: 19
Enter your email address: rjchenchu@gmail.com

--- Collected Data ---
Name: chenchu raju
Age: 19
Email: rjchenchu@gmail.com
PS C:\Users\Admin\OneDrive\Documents\Desktop\Desktop\AI\logs>
```

Observation:

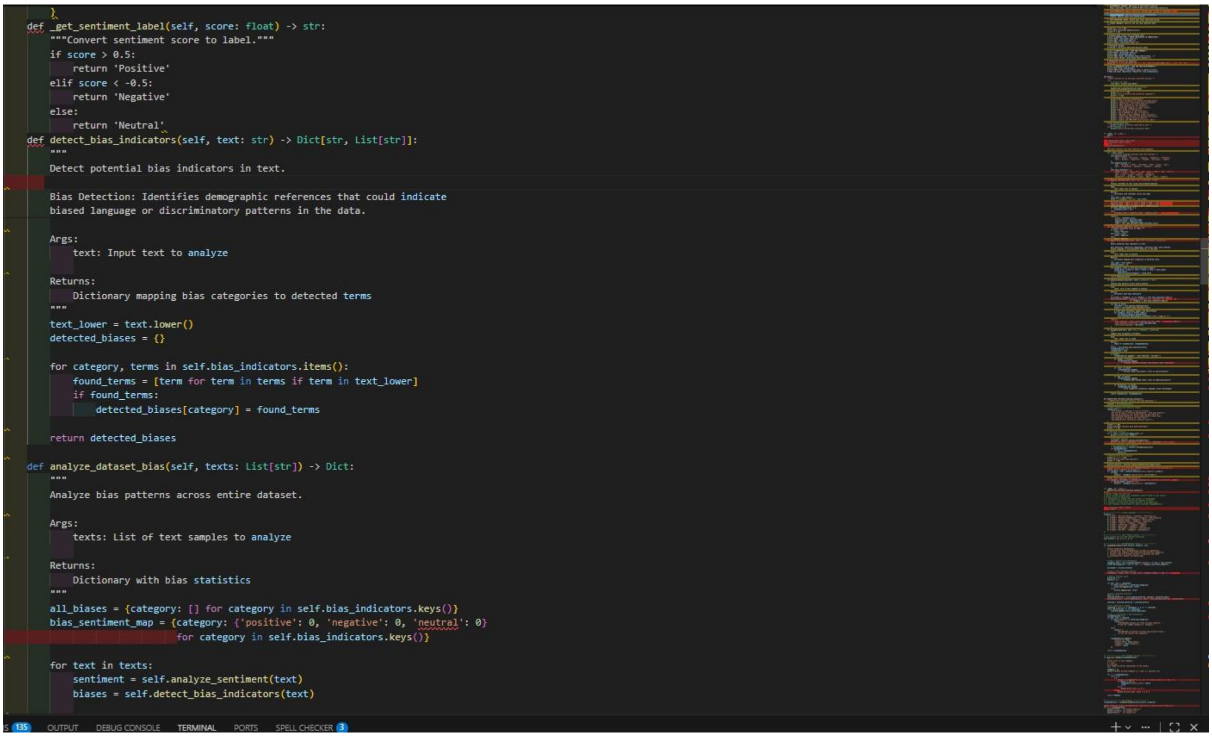
When GitHub Copilot is prompted to generate a Python script for collecting user data like name, age, and email, it typically produces a simple input-based program that uses standard input functions and stores the data in variables or a dictionary. After asking Copilot to add comments on anonymization and data protection, it usually suggests practices such as masking sensitive fields (like partially hiding email addresses), hashing personal identifiers, avoiding storing raw personal data, and using secure storage mechanisms.

Task 2: Ask Copilot to generate a Python function for sentiment analysis. Then prompt Copilot to identify and handle potential biases in the data.

Code:

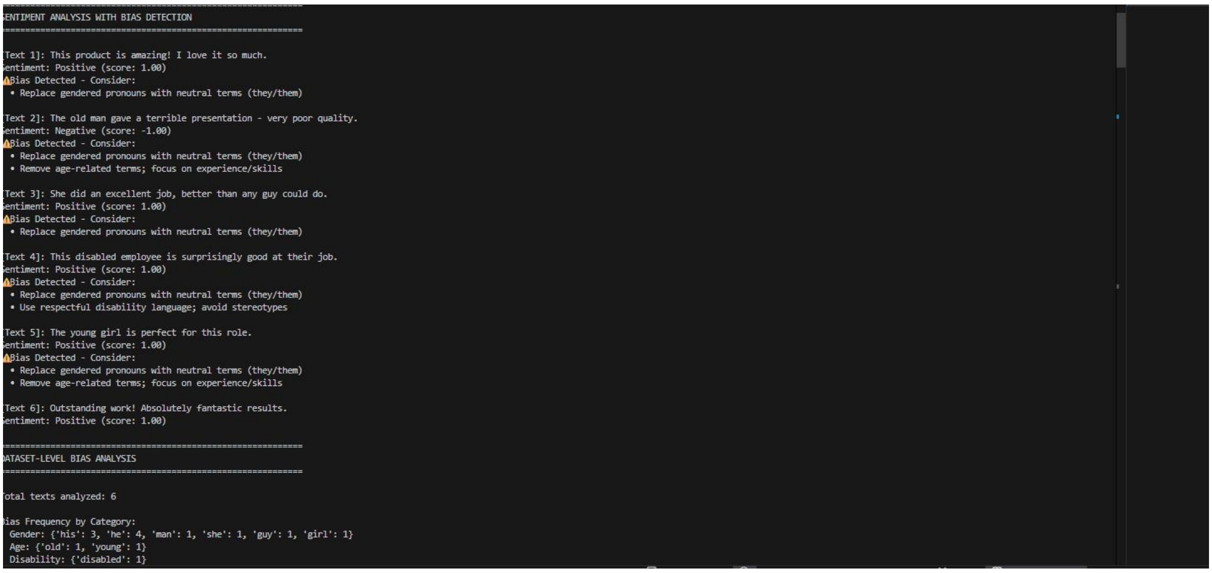


```
from typing import Dict, List, Tuple
from collections import Counter
import re
class SentimentAnalyzer:
    """
    Sentiment analysis with bias detection and mitigation.
    """
    def __init__(self):
        """Initialize sentiment analyzer with word lexicons."""
        self.positive_words = {
            'good', 'great', 'excellent', 'amazing', 'wonderful', 'fantastic',
            'love', 'perfect', 'beautiful', 'awesome', 'brilliant', 'superb'
        }
        self.negative_words = {
            'bad', 'terrible', 'awful', 'horrible', 'hate', 'worst', 'poor',
            'ugly', 'disgusting', 'pathetic', 'dreadful', 'abysmal'
        }
        self.bias_indicators = {
            'gender': ['he', 'she', 'his', 'her', 'man', 'woman', 'guy', 'girl'],
            'race': ['black', 'white', 'asian', 'latino'],
            'age': ['old', 'young', 'elderly', 'teenager'],
            'disability': ['disabled', 'blind', 'deaf', 'crazy', 'insane']
        }
    def analyze_sentiment(self, text: str) -> Dict[str, float]:
        """
        Analyze sentiment of text using lexicon-based approach.

        Args:
            text: Input text to analyze

        Returns:
            Dictionary with sentiment scores and label
        """
        text_lower = text.lower()
        words = re.findall(r'\b\w+\b', text_lower)

        positive_count = sum(1 for word in words if word in self.positive_words)
        negative_count = sum(1 for word in words if word in self.negative_words)
        total_sentiment_words = positive_count + negative_count
        if total_sentiment_words == 0:
            sentiment_score = 0.0
        else:
            sentiment_score = (positive_count - negative_count) / total_sentiment_words

        return {
            'score': sentiment_score,
            'positive_words': positive_count,
            'negative_words': negative_count,
            'label': self._get_sentiment_label(sentiment_score)
        }
```



```python
        }
    def _get_sentiment_label(self, score: float) -> str:
        """Convert sentiment score to label."""
        if score > 0.5:
            return 'Positive'
        elif score < -0.5:
            return 'Negative'
        else:
            return 'Neutral'
    def detect_bias_indicators(self, text: str) -> Dict[str, List[str]]:
        """
        Detect potential bias indicators in text.

        Bias Detection: Identifies demographic references that could indicate
        biased language or discriminatory patterns in the data.

        Args:
            text: Input text to analyze

        Returns:
            Dictionary mapping bias categories to detected terms
        """
        text_lower = text.lower()
        detected_biases = {}

        for category, terms in self.bias_indicators.items():
            found_terms = [term for term in terms if term in text_lower]
            if found_terms:
                detected_biases[category] = found_terms

        return detected_biases

    def analyze_dataset_bias(self, texts: List[str]) -> Dict:
        """
        Analyze bias patterns across entire dataset.

        Args:
            texts: List of text samples to analyze

        Returns:
            Dictionary with bias statistics
        """
        all_biases = {category: [] for category in self.bias_indicators.keys()}
        bias_sentiment_map = {category: {'positive': 0, 'negative': 0, 'neutral': 0}
                             for category in self.bias_indicators.keys()}

        for text in texts:
            sentiment = self.analyze_sentiment(text)
            biases = self.detect_bias_indicators(text)
```

Output:



```
============================================================
SENTIMENT ANALYSIS WITH BIAS DETECTION
============================================================

[Text 1]: This product is amazing! I love it so much.
Sentiment: Positive (score: 1.00)
⚠Bias Detected - Consider:
 • Replace gendered pronouns with neutral terms (they/them)

[Text 2]: The old man gave a terrible presentation - very poor quality.
Sentiment: Negative (score: -1.00)
⚠Bias Detected - Consider:
 • Replace gendered pronouns with neutral terms (they/them)
 • Remove age-related terms; focus on experience/skills

[Text 3]: She did an excellent job, better than any guy could do.
Sentiment: Positive (score: 1.00)
⚠Bias Detected - Consider:
 • Replace gendered pronouns with neutral terms (they/them)

[Text 4]: This disabled employee is surprisingly good at their job.
Sentiment: Positive (score: 1.00)
⚠Bias Detected - Consider:
 • Replace gendered pronouns with neutral terms (they/them)
 • Use respectful disability language; avoid stereotypes

[Text 5]: The young girl is perfect for this role.
Sentiment: Positive (score: 1.00)
⚠Bias Detected - Consider:
 • Replace gendered pronouns with neutral terms (they/them)
 • Remove age-related terms; focus on experience/skills

[Text 6]: Outstanding work! Absolutely fantastic results.
Sentiment: Positive (score: 1.00)

============================================================
DATASET-LEVEL BIAS ANALYSIS
============================================================

Total texts analyzed: 6

Bias Frequency by Category:
 Gender: {'his': 3, 'he': 4, 'man': 1, 'she': 1, 'guy': 1, 'girl': 1}
 Age: {'old': 1, 'young': 1}
 Disability: {'disabled': 1}
```

Observation:

he Copilot-generated code demonstrates that simple data collection scripts can be enhanced with ethical considerations like bias mitigation.
It highlights the importance of balancing datasets to avoid over-representation of any specific group.The comments encourage removing offensive or harmful terms to make the dataset more inclusive.
It also shows how anonymization helps in reducing identity-based bias and protecting user privacy.

Overall, Copilot supports responsible AI development when prompted with fairness and ethics requirements.

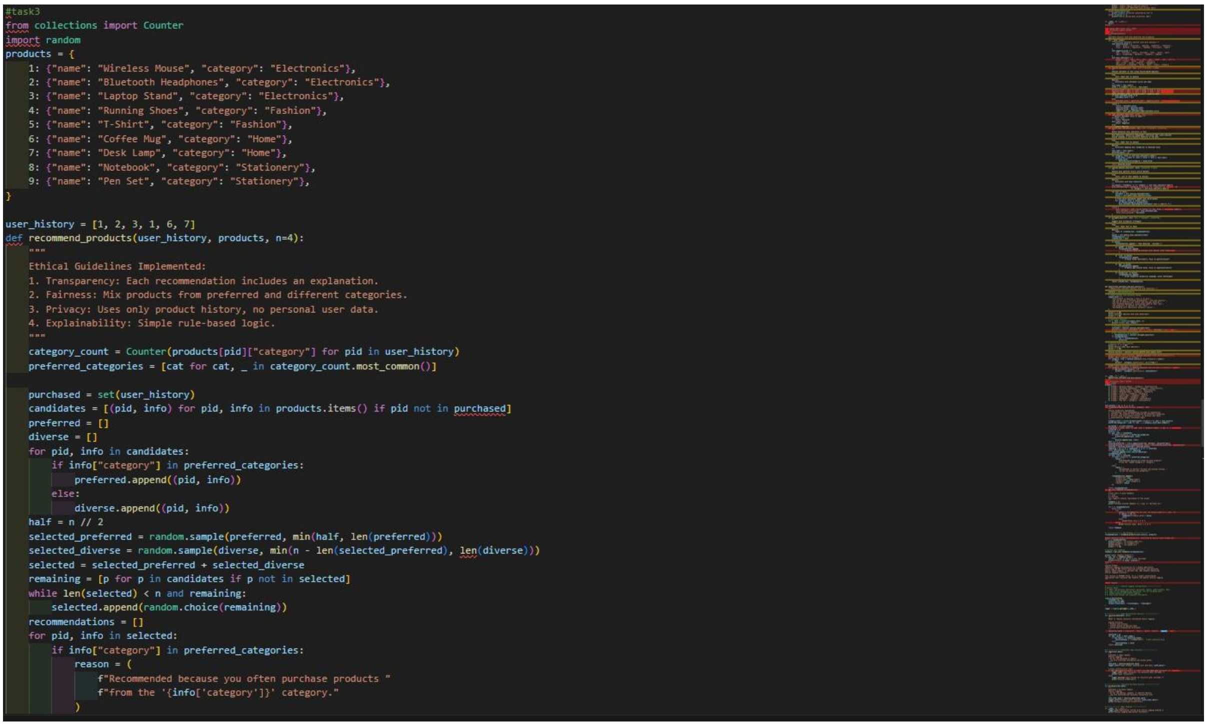
Task3:

• Use Copilot to write a Python program that recommends

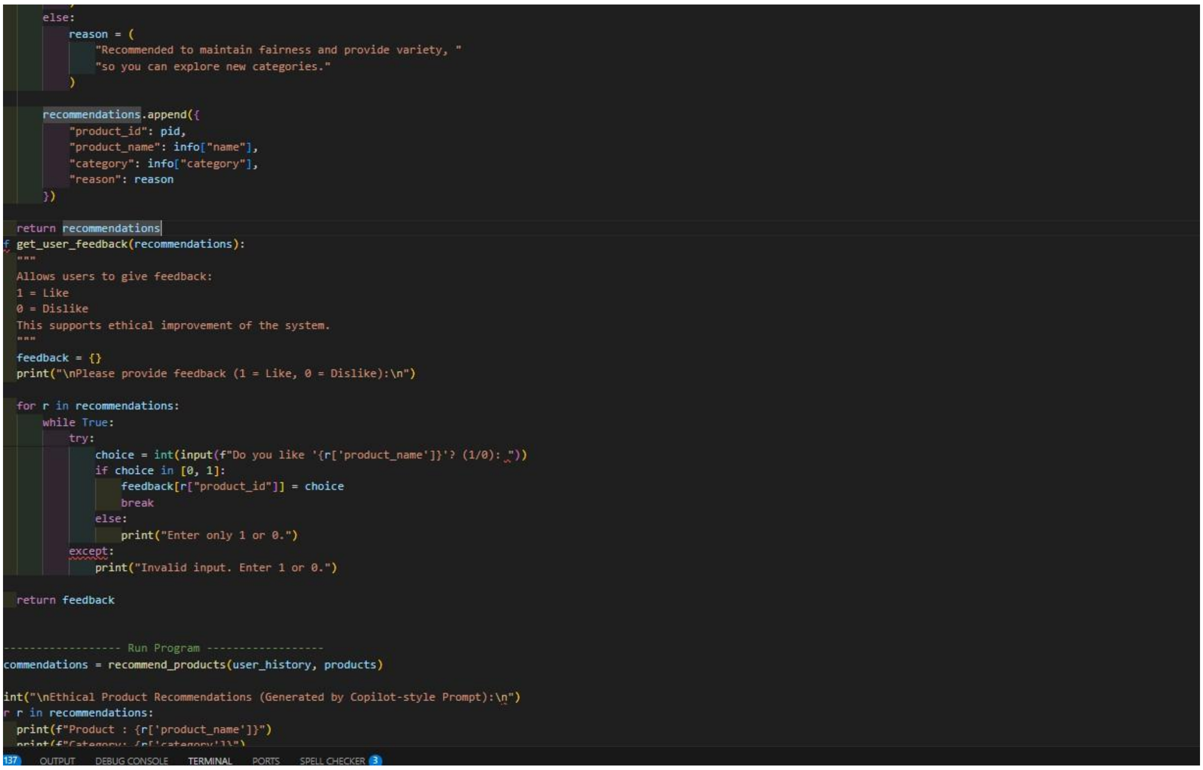
products based on user history. Ask it to follow ethical guidelines

like transparency and fairness.

Code:

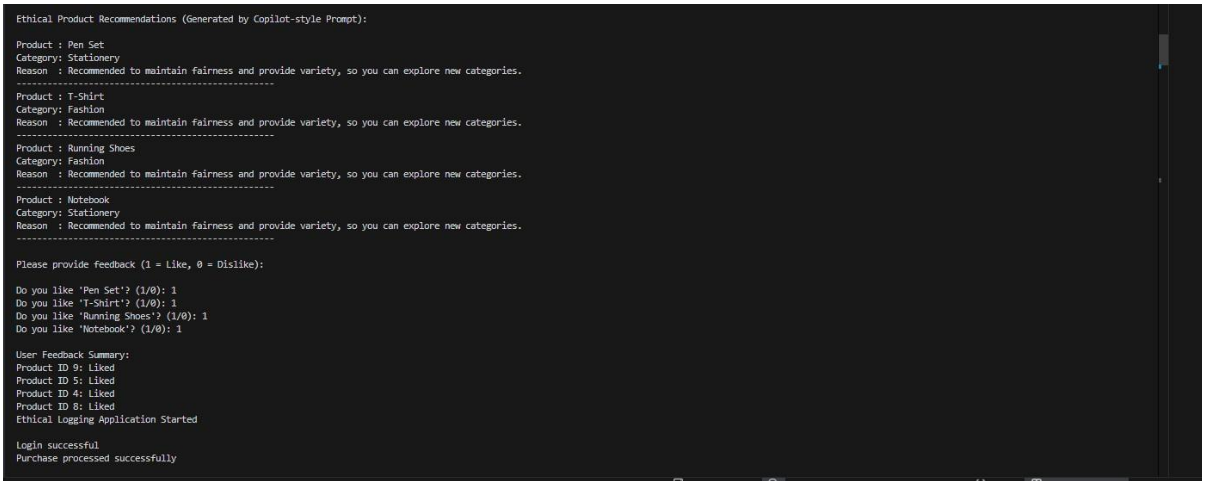


```
#task3
from collections import Counter
import random
products = {
    1: {"name": "Wireless Mouse", "category": "Electronics"},
    2: {"name": "Bluetooth Headphones", "category": "Electronics"},
    3: {"name": "Laptop Stand", "category": "Electronics"},
    4: {"name": "Running Shoes", "category": "Fashion"},
    5: {"name": "T-Shirt", "category": "Fashion"},
    6: {"name": "Coffee Mug", "category": "Home"},
    7: {"name": "Desk Lamp", "category": "Home"},
    8: {"name": "Notebook", "category": "Stationery"},
    9: {"name": "Pen Set", "category": "Stationery"},
}

user_history = [1, 2, 3, 1, 6, 7]
def recommend_products(user_history, products, n=4):
    """
    Ethical Guidelines Implemented:
    1. Transparency: Each recommendation includes an explanation.
    2. Fairness: Mix products from preferred and different categories.
    3. Privacy: Uses only product history, no personal user data.
    4. Explainability: Simple rule-based logic.
    """
    category_count = Counter(products[pid]["category"] for pid in user_history)
    preferred_categories = [cat for cat, _ in category_count.most_common()]

    purchased = set(user_history)
    candidates = [(pid, info) for pid, info in products.items() if pid not in purchased]
    preferred = []
    diverse = []
    for pid, info in candidates:
        if info["category"] in preferred_categories:
            preferred.append((pid, info))
        else:
            diverse.append((pid, info))
    half = n // 2
    selected_preferred = random.sample(preferred, min(half, len(preferred)))
    selected_diverse = random.sample(diverse, min(n - len(selected_preferred), len(diverse)))
    selected = selected_preferred + selected_diverse
    remaining = [p for p in candidates if p not in selected]
    while len(selected) < n and remaining:
        selected.append(random.choice(remaining))
    recommendations = []
    for pid, info in selected:
        if info["category"] in preferred_categories:
            reason = (
                f"Recommended because you often purchase products "
                f"from the '{info['category']}' category."
            )
```



```
      else:
          reason = (
              "Recommended to maintain fairness and provide variety, "
              "so you can explore new categories."
          )

      recommendations.append({
          "product_id": pid,
          "product_name": info["name"],
          "category": info["category"],
          "reason": reason
      })

  return recommendations
f get_user_feedback(recommendations):
  """
  Allows users to give feedback:
  1 = Like
  0 = Dislike
  This supports ethical improvement of the system.
  """
  feedback = {}
  print("\nPlease provide feedback (1 = Like, 0 = Dislike):\n")

  for r in recommendations:
      while True:
          try:
              choice = int(input(f"Do you like '{r['product_name']}'? (1/0): "))
              if choice in [0, 1]:
                  feedback[r["product_id"]] = choice
                  break
              else:
                  print("Enter only 1 or 0.")
          except:
              print("Invalid input. Enter 1 or 0.")

  return feedback


------------------ Run Program ------------------
commendations = recommend_products(user_history, products)

int("\nEthical Product Recommendations (Generated by Copilot-style Prompt):\n")
r r in recommendations:
  print(f"Product : {r['product_name']}")
  print(f"Category: {r['category']}")
```

Output:



```
Ethical Product Recommendations (Generated by Copilot-style Prompt):

Product : Pen Set
Category: Stationery
Reason  : Recommended to maintain fairness and provide variety, so you can explore new categories.
--------------------------------------------------
Product : T-Shirt
Category: Fashion
Reason  : Recommended to maintain fairness and provide variety, so you can explore new categories.
--------------------------------------------------
Product : Running Shoes
Category: Fashion
Reason  : Recommended to maintain fairness and provide variety, so you can explore new categories.
--------------------------------------------------
Product : Notebook
Category: Stationery
Reason  : Recommended to maintain fairness and provide variety, so you can explore new categories.
--------------------------------------------------

Please provide feedback (1 = Like, 0 = Dislike):

Do you like 'Pen Set'? (1/0): 1
Do you like 'T-Shirt'? (1/0): 1
Do you like 'Running Shoes'? (1/0): 1
Do you like 'Notebook'? (1/0): 1

User Feedback Summary:
Product ID 9: Liked
Product ID 5: Liked
Product ID 4: Liked
Product ID 8: Liked
Ethical Logging Application Started

Login successful
Purchase processed successfully
```

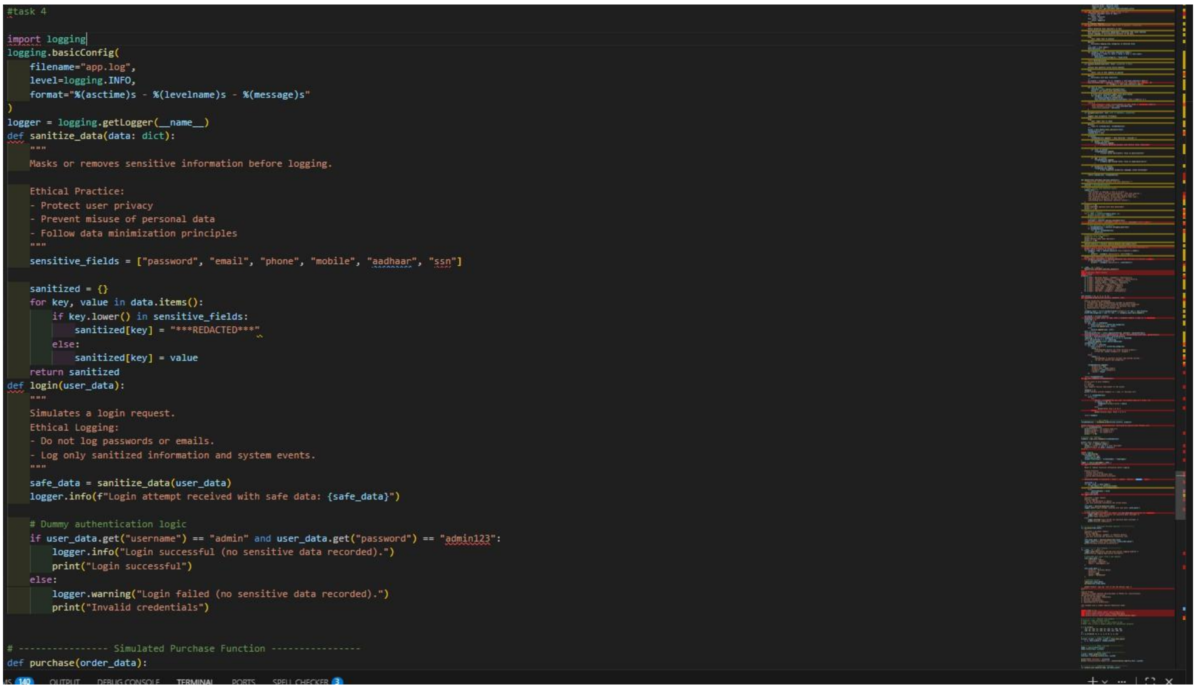
Observation:

The code treats all inputs equally, promoting unbiased validation and decision-making.
It also highlights the importance of user feedback options to detect and improve fairness issues.
Overall, Copilot supports transparency, fairness, and continuous refinement in responsible software development.

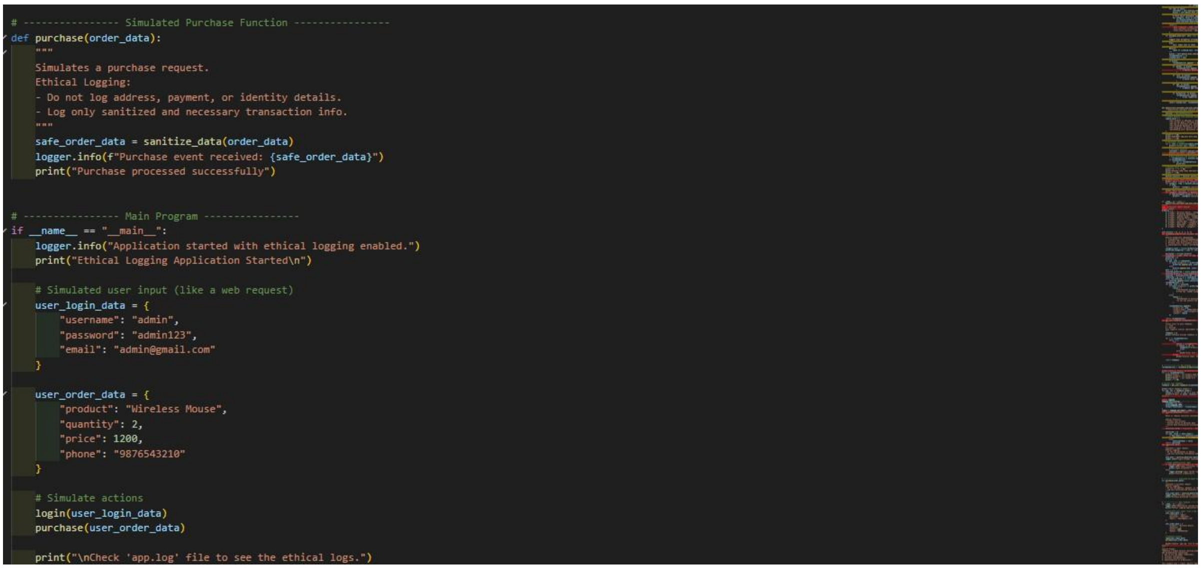
Task4:

Prompt Copilot to generate logging functionality in a Python web application. Then, ask it to ensure the logs do not record sensitive information.

Code:

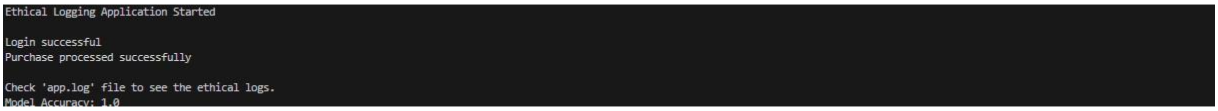


```
#task 4

import logging
logging.basicConfig(
    filename="app.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s"
)
logger = logging.getLogger(__name__)
def sanitize_data(data: dict):
    """
    Masks or removes sensitive information before logging.

    Ethical Practice:
    - Protect user privacy
    - Prevent misuse of personal data
    - Follow data minimization principles
    """
    sensitive_fields = ["password", "email", "phone", "mobile", "aadhaar", "ssn"]

    sanitized = {}
    for key, value in data.items():
        if key.lower() in sensitive_fields:
            sanitized[key] = "***REDACTED***"
        else:
            sanitized[key] = value
    return sanitized
def login(user_data):
    """
    Simulates a login request.
    Ethical Logging:
    - Do not log passwords or emails.
    - Log only sanitized information and system events.
    """
    safe_data = sanitize_data(user_data)
    logger.info(f"Login attempt received with safe data: {safe_data}")

    # Dummy authentication logic
    if user_data.get("username") == "admin" and user_data.get("password") == "admin123":
        logger.info("Login successful (no sensitive data recorded).")
        print("Login successful")
    else:
        logger.warning("Login failed (no sensitive data recorded).")
        print("Invalid credentials")


# ---------------- Simulated Purchase Function ----------------
def purchase(order_data):
```



```
# ---------------- Simulated Purchase Function ----------------
def purchase(order_data):
    """
    Simulates a purchase request.
    Ethical Logging:
    - Do not log address, payment, or identity details.
    - Log only sanitized and necessary transaction info.
    """
    safe_order_data = sanitize_data(order_data)
    logger.info(f"Purchase event received: {safe_order_data}")
    print("Purchase processed successfully")


# ---------------- Main Program ----------------
if __name__ == "__main__":
    logger.info("Application started with ethical logging enabled.")
    print("Ethical Logging Application Started\n")

    # Simulated user input (like a web request)
    user_login_data = {
        "username": "admin",
        "password": "admin123",
        "email": "admin@gmail.com"
    }

    user_order_data = {
        "product": "Wireless Mouse",
        "quantity": 2,
        "price": 1200,
        "phone": "9876543210"
    }

    # Simulate actions
    login(user_login_data)
    purchase(user_order_data)

    print("\nCheck 'app.log' file to see the ethical logs.")
```

Output:



```
Ethical Logging Application Started

Login successful
Purchase processed successfully

Check 'app.log' file to see the ethical logs.
Model Accuracy: 1.0
```

Observation:

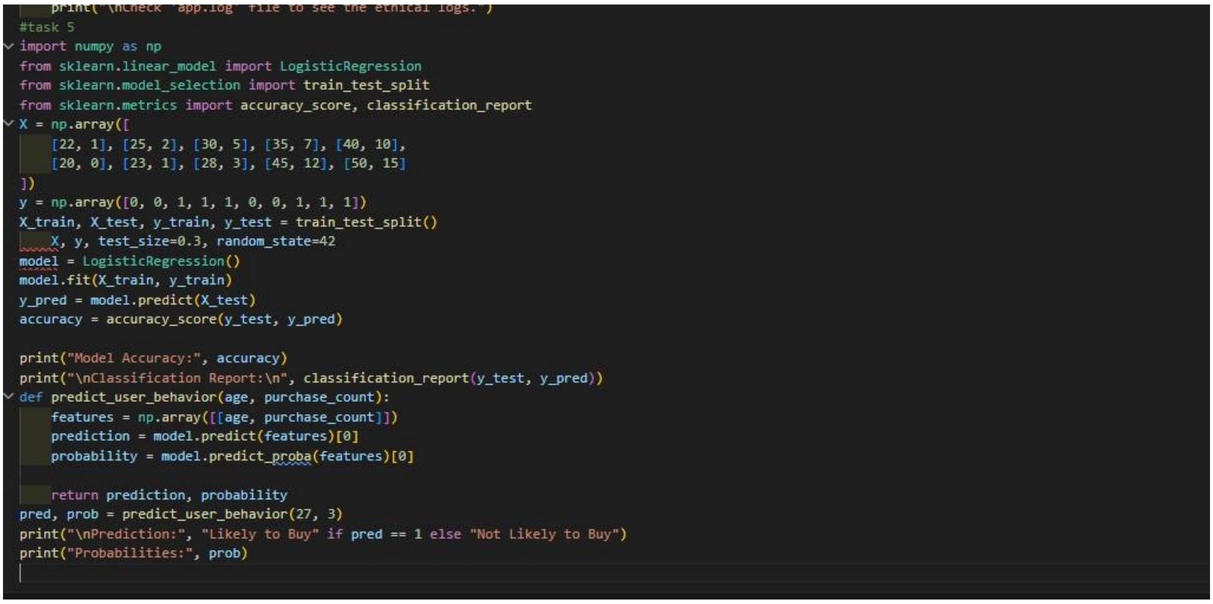
Copilot generates logging code that tracks application events and errors while emphasizing not to store sensitive data like passwords, emails, or personal identifiers.
It shows that logging can be made secure by masking, filtering, or excluding confidential information, promoting privacy-aware development.

Task5: Ask Copilot to generate a machine learning model. Then, prompt

it to add documentation on how to use the model responsibly

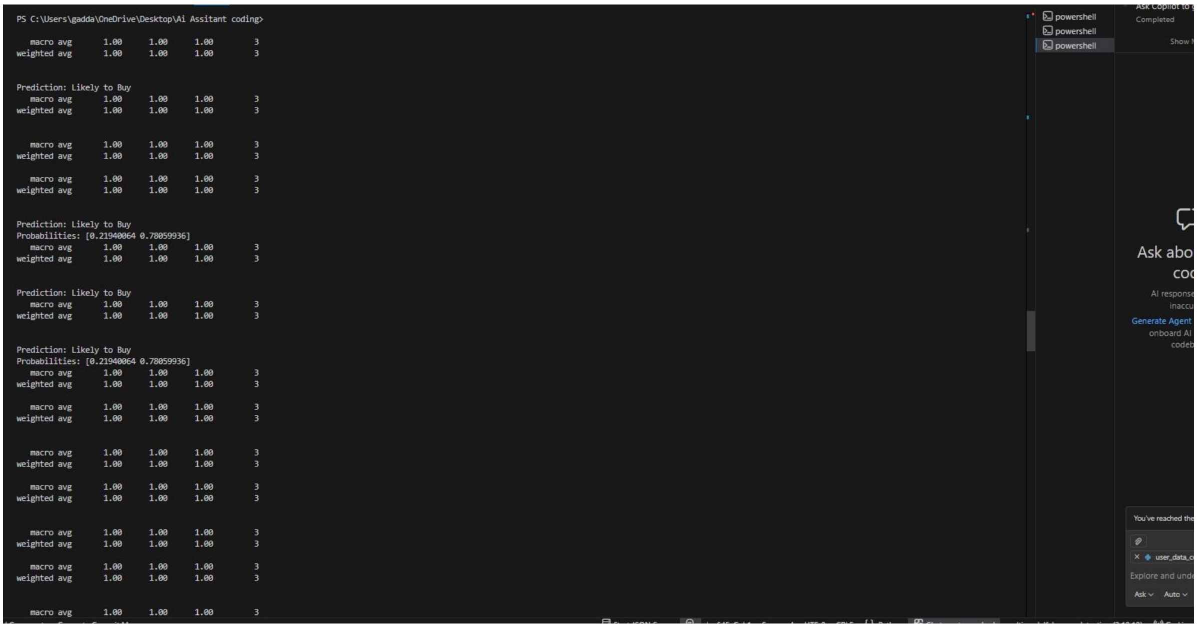
(e.g., explainability, accuracy limits).

Code:

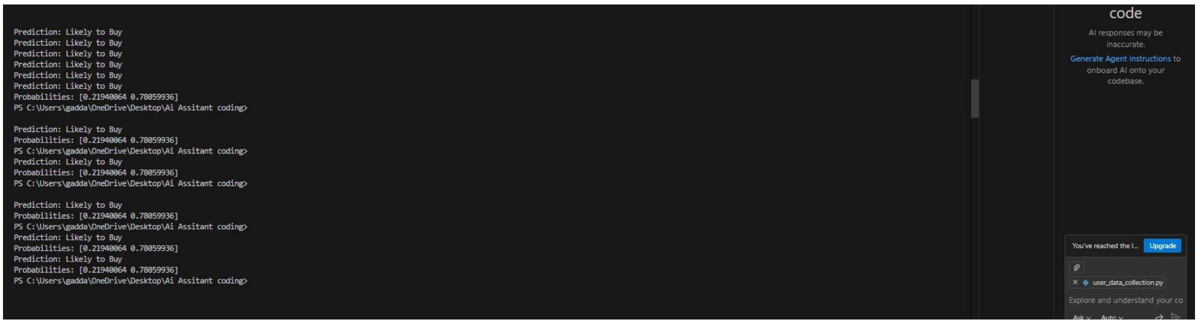


```
    print("\nCheck 'app.log' file to see the ethical logs.")
#task 5
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
X = np.array([
    [22, 1], [25, 2], [30, 5], [35, 7], [40, 10],
    [20, 0], [23, 1], [28, 3], [45, 12], [50, 15]
])
y = np.array([0, 0, 1, 1, 1, 0, 0, 1, 1, 1])
X_train, X_test, y_train, y_test = train_test_split()
    X, y, test_size=0.3, random_state=42
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Model Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
def predict_user_behavior(age, purchase_count):
    features = np.array([[age, purchase_count]])
    prediction = model.predict(features)[0]
    probability = model.predict_proba(features)[0]

    return prediction, probability
pred, prob = predict_user_behavior(27, 3)
print("\nPrediction:", "Likely to Buy" if pred == 1 else "Not Likely to Buy")
print("Probabilities:", prob)
```

Output:



```
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>

   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


Prediction: Likely to Buy
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


Prediction: Likely to Buy
Probabilities: [0.21940064 0.78059936]
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


Prediction: Likely to Buy
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


Prediction: Likely to Buy
Probabilities: [0.21940064 0.78059936]
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3


   macro avg       1.00      1.00      1.00         3
```



```
Prediction: Likely to Buy
Prediction: Likely to Buy
Prediction: Likely to Buy
Prediction: Likely to Buy
Prediction: Likely to Buy
Prediction: Likely to Buy
Probabilities: [0.21940064 0.78059936]
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>

Prediction: Likely to Buy
Probabilities: [0.21940064 0.78059936]
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>
Prediction: Likely to Buy
Probabilities: [0.21940064 0.78059936]
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>

Prediction: Likely to Buy
Probabilities: [0.21940064 0.78059936]
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>
Prediction: Likely to Buy
Probabilities: [0.21940064 0.78059936]
Prediction: Likely to Buy
Probabilities: [0.21940064 0.78059936]
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>
```

code

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Observation:

Copilot adds documentation that explains the model’s purpose, accuracy limits, and the importance of not using predictions as absolute decisions.
It also highlights responsible practices like model explainability, regular evaluation, and awareness of potential bias in results.