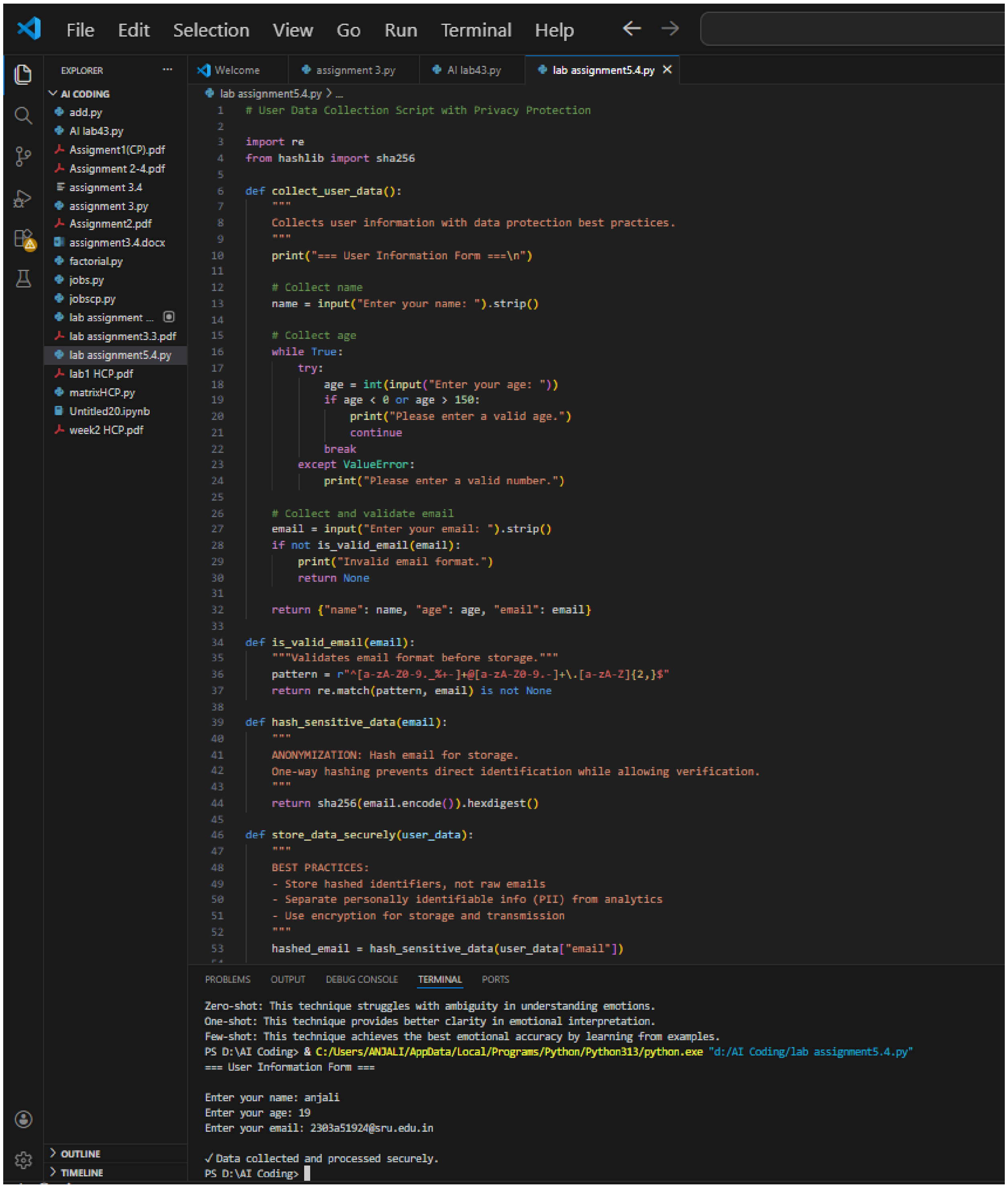
2303a51797

Lab assignment 5.4

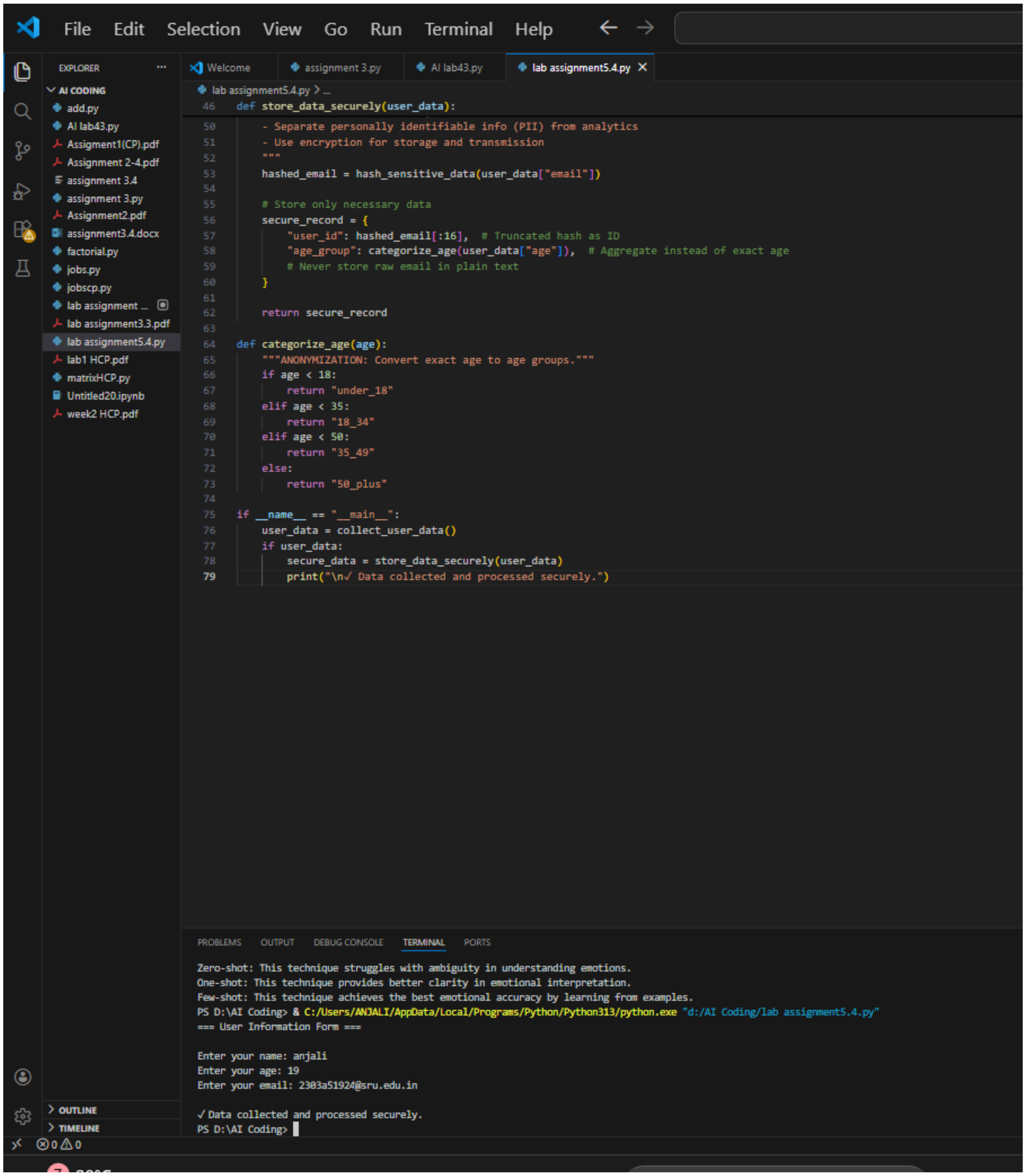
Lab 5: Ethical Foundations – Responsible AI Coding Practices

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.



```python
# User Data Collection Script with Privacy Protection

import re
from hashlib import sha256

def collect_user_data():
    """
    Collects user information with data protection best practices.
    """
    print("=== User Information Form ===\n")

    # Collect name
    name = input("Enter your name: ").strip()

    # Collect age
    while True:
        try:
            age = int(input("Enter your age: "))
            if age < 0 or age > 150:
                print("Please enter a valid age.")
                continue
            break
        except ValueError:
            print("Please enter a valid number.")

    # Collect and validate email
    email = input("Enter your email: ").strip()
    if not is_valid_email(email):
        print("Invalid email format.")
        return None

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

def is_valid_email(email):
    """Validates email format before storage."""
    pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
    return re.match(pattern, email) is not None

def hash_sensitive_data(email):
    """
    ANONYMIZATION: Hash email for storage.
    One-way hashing prevents direct identification while allowing verification.
    """
    return sha256(email.encode()).hexdigest()

def store_data_securely(user_data):
    """
    BEST PRACTICES:
    - Store hashed identifiers, not raw emails
    - Separate personally identifiable info (PII) from analytics
    - Use encryption for storage and transmission
    """
    hashed_email = hash_sensitive_data(user_data["email"])
```

```
Zero-shot: This technique struggles with ambiguity in understanding emotions.
One-shot: This technique provides better clarity in emotional interpretation.
Few-shot: This technique achieves the best emotional accuracy by learning from examples.
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment5.4.py"
=== User Information Form ===

Enter your name: anjali
Enter your age: 19
Enter your email: 2303a51924@sru.edu.in

✓ Data collected and processed securely.
PS D:\AI Coding>
```

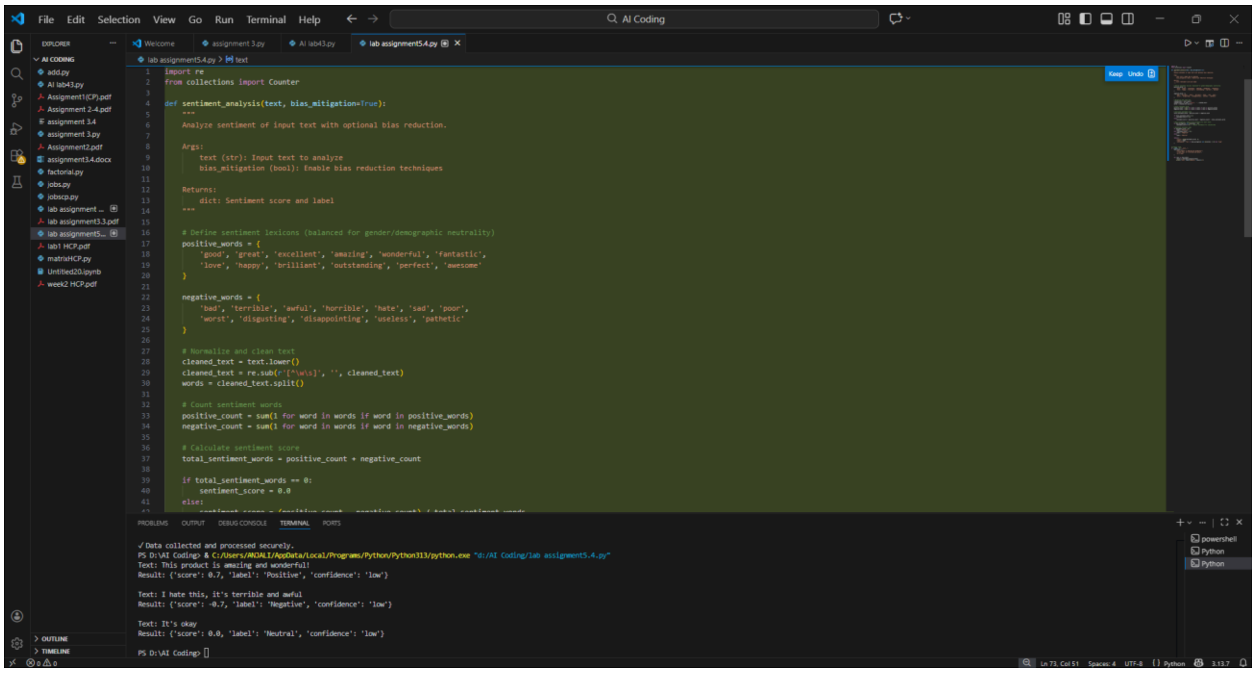


```python
def store_data_securely(user_data):
    - Separate personally identifiable info (PII) from analytics
    - Use encryption for storage and transmission
    """
    hashed_email = hash_sensitive_data(user_data["email"])

    # Store only necessary data
    secure_record = {
        "user_id": hashed_email[:16],  # Truncated hash as ID
        "age_group": categorize_age(user_data["age"]),  # Aggregate instead of exact age
        # Never store raw email in plain text
    }

    return secure_record

def categorize_age(age):
    """ANONYMIZATION: Convert exact age to age groups."""
    if age < 18:
        return "under_18"
    elif age < 35:
        return "18_34"
    elif age < 50:
        return "35_49"
    else:
        return "50_plus"

if __name__ == "__main__":
    user_data = collect_user_data()
    if user_data:
        secure_data = store_data_securely(user_data)
        print("\n✓ Data collected and processed securely.")
```

```
Zero-shot: This technique struggles with ambiguity in understanding emotions.
One-shot: This technique provides better clarity in emotional interpretation.
Few-shot: This technique achieves the best emotional accuracy by learning from examples.
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment5.4.py"
=== User Information Form ===

Enter your name: anjali
Enter your age: 19
Enter your email: 2303a51924@sru.edu.in

✓ Data collected and processed securely.
PS D:\AI Coding> 
```

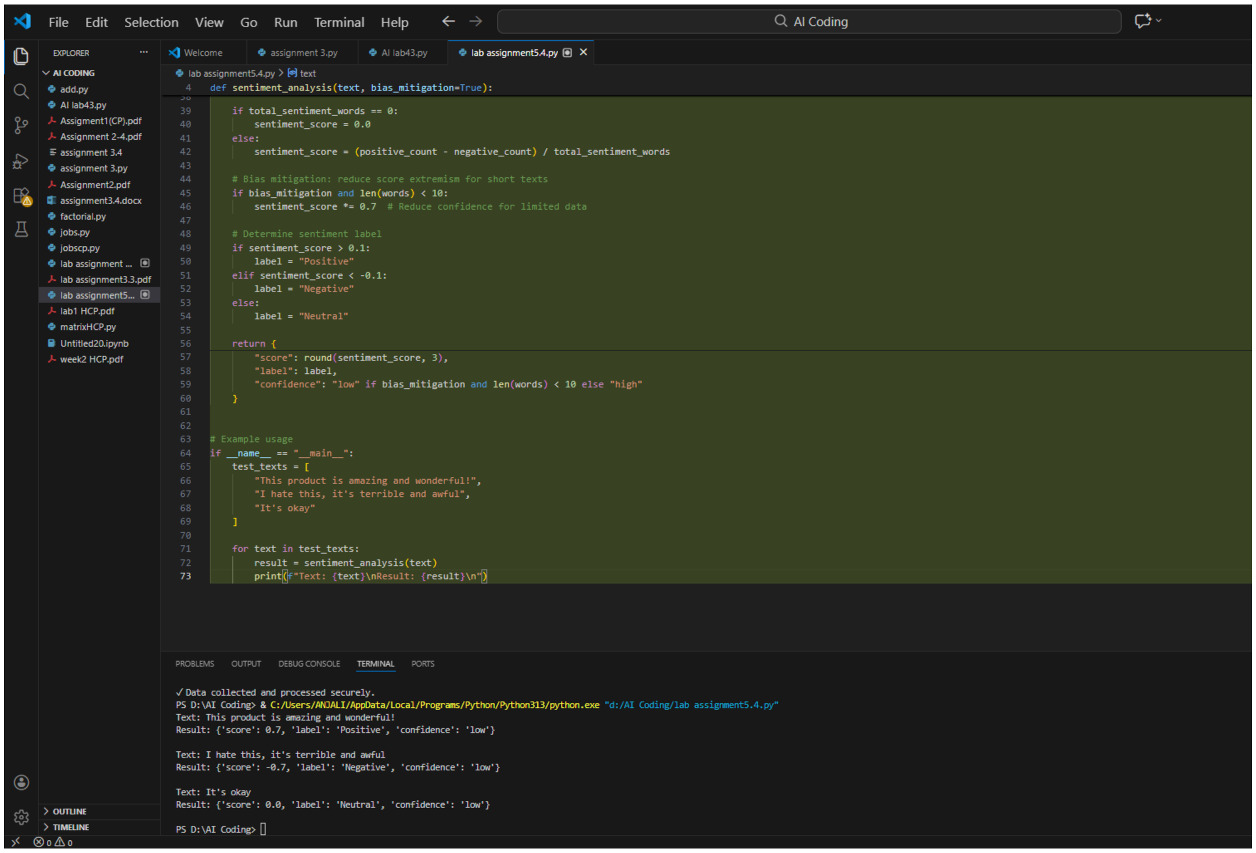
2) Task Description #2:

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



```python
import re
from collections import Counter

def sentiment_analysis(text, bias_mitigation=True):
    """
    Analyze sentiment of input text with optional bias reduction.

    Args:
        text (str): Input text to analyze
        bias_mitigation (bool): Enable bias reduction techniques

    Returns:
        dict: Sentiment score and label
    """

    # Define sentiment lexicons (balanced for gender/demographic neutrality)
    positive_words = {
        'good', 'great', 'excellent', 'amazing', 'wonderful', 'fantastic',
        'love', 'happy', 'brilliant', 'outstanding', 'perfect', 'awesome'
    }

    negative_words = {
        'bad', 'terrible', 'awful', 'horrible', 'hate', 'sad', 'poor',
        'worst', 'disgusting', 'disappointing', 'useless', 'pathetic'
    }

    # Normalize and clean text
    cleaned_text = text.lower()
    cleaned_text = re.sub(r'[^\w\s]', '', cleaned_text)
    words = cleaned_text.split()

    # Count sentiment words
    positive_count = sum(1 for word in words if word in positive_words)
    negative_count = sum(1 for word in words if word in negative_words)

    # Calculate sentiment score
    total_sentiment_words = positive_count + negative_count

    if total_sentiment_words == 0:
        sentiment_score = 0.0
    else:
```

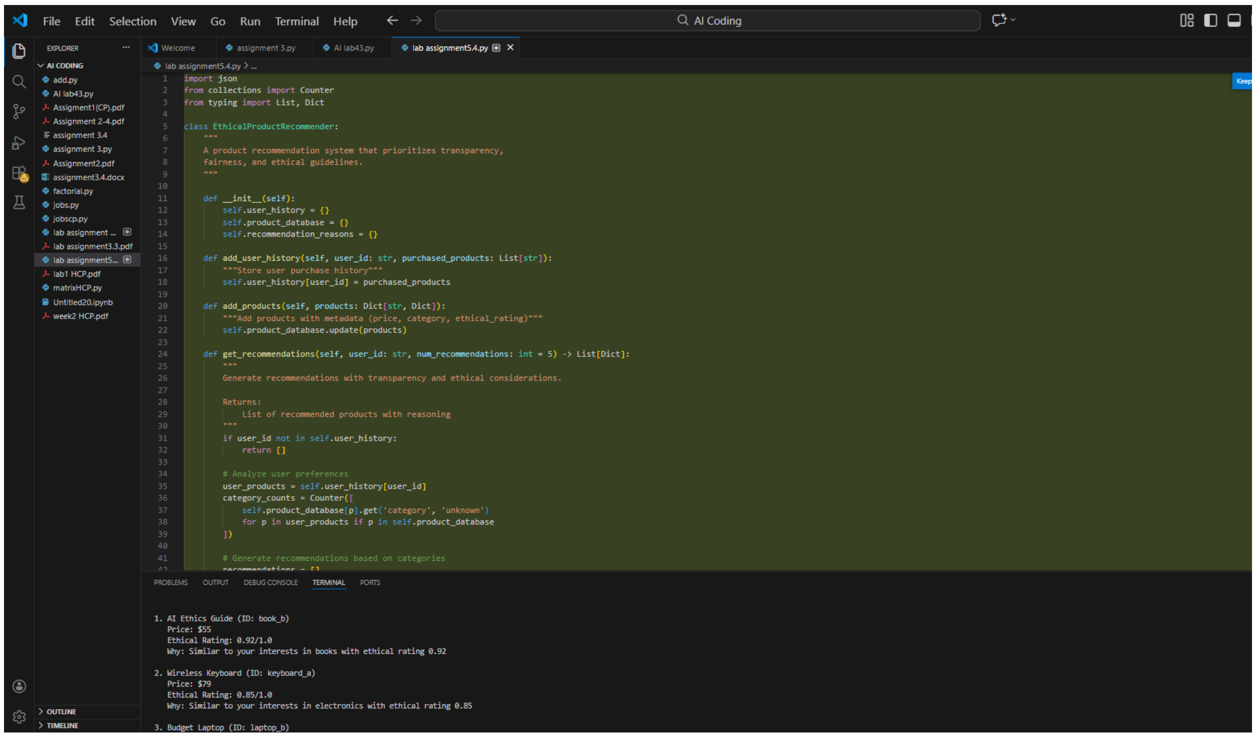
```
✓ Data collected and processed securely.
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment5.4.py"
Text: This product is amazing and wonderful!
Result: {'score': 0.7, 'label': 'Positive', 'confidence': 'low'}

Text: I hate this, it's terrible and awful
Result: {'score': -0.7, 'label': 'Negative', 'confidence': 'low'}

Text: It's okay
Result: {'score': 0.0, 'label': 'Neutral', 'confidence': 'low'}

PS D:\AI Coding> 
```



```python
def sentiment_analysis(text, bias_mitigation=True):
    if total_sentiment_words == 0:
        sentiment_score = 0.0
    else:
        sentiment_score = (positive_count - negative_count) / total_sentiment_words

    # Bias mitigation: reduce score extremism for short texts
    if bias_mitigation and len(words) < 10:
        sentiment_score *= 0.7  # Reduce confidence for limited data

    # Determine sentiment label
    if sentiment_score > 0.1:
        label = "Positive"
    elif sentiment_score < -0.1:
        label = "Negative"
    else:
        label = "Neutral"

    return {
        "score": round(sentiment_score, 3),
        "label": label,
        "confidence": "low" if bias_mitigation and len(words) < 10 else "high"
    }


# Example usage
if __name__ == "__main__":
    test_texts = [
        "This product is amazing and wonderful!",
        "I hate this, it's terrible and awful",
        "It's okay"
    ]

    for text in test_texts:
        result = sentiment_analysis(text)
        print(f"Text: {text}\nResult: {result}\n")
```

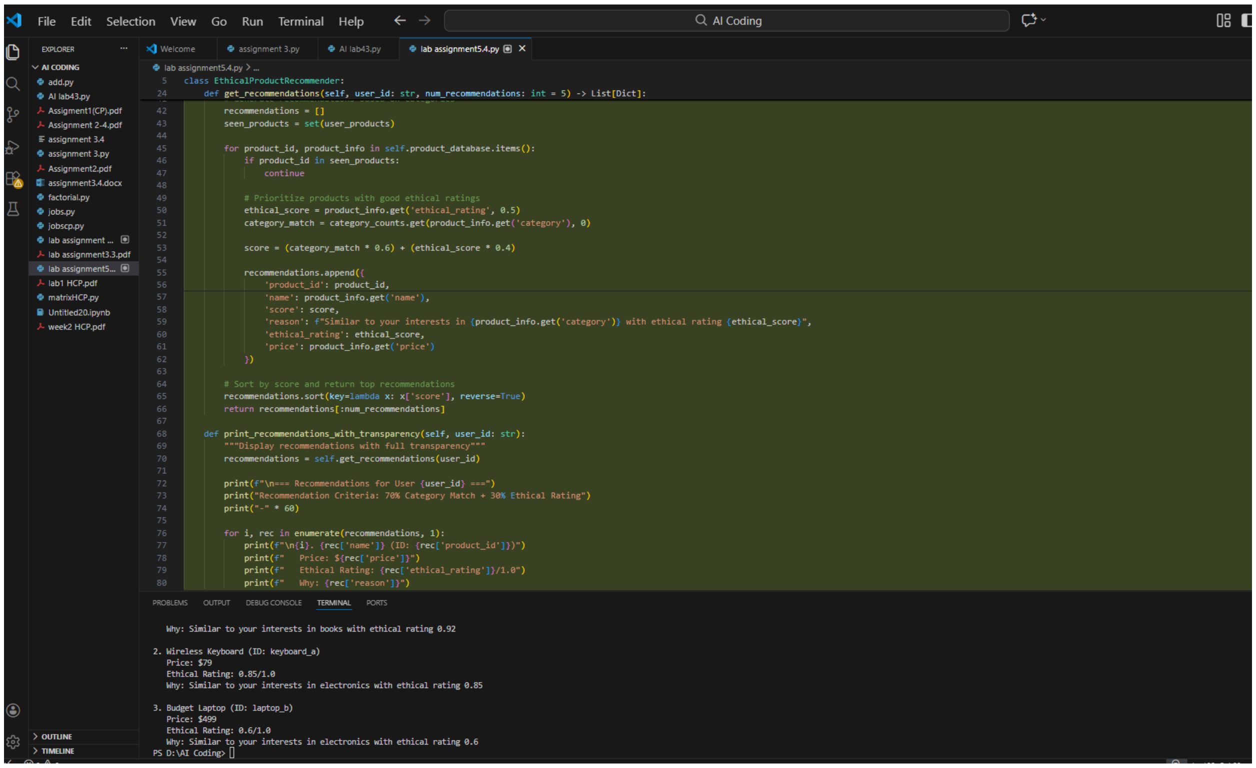
```
✓ Data collected and processed securely.
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment5.4.py"
Text: This product is amazing and wonderful!
Result: {'score': 0.7, 'label': 'Positive', 'confidence': 'low'}

Text: I hate this, it's terrible and awful
Result: {'score': -0.7, 'label': 'Negative', 'confidence': 'low'}

Text: It's okay
Result: {'score': 0.0, 'label': 'Neutral', 'confidence': 'low'}

PS D:\AI Coding> 
```

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.



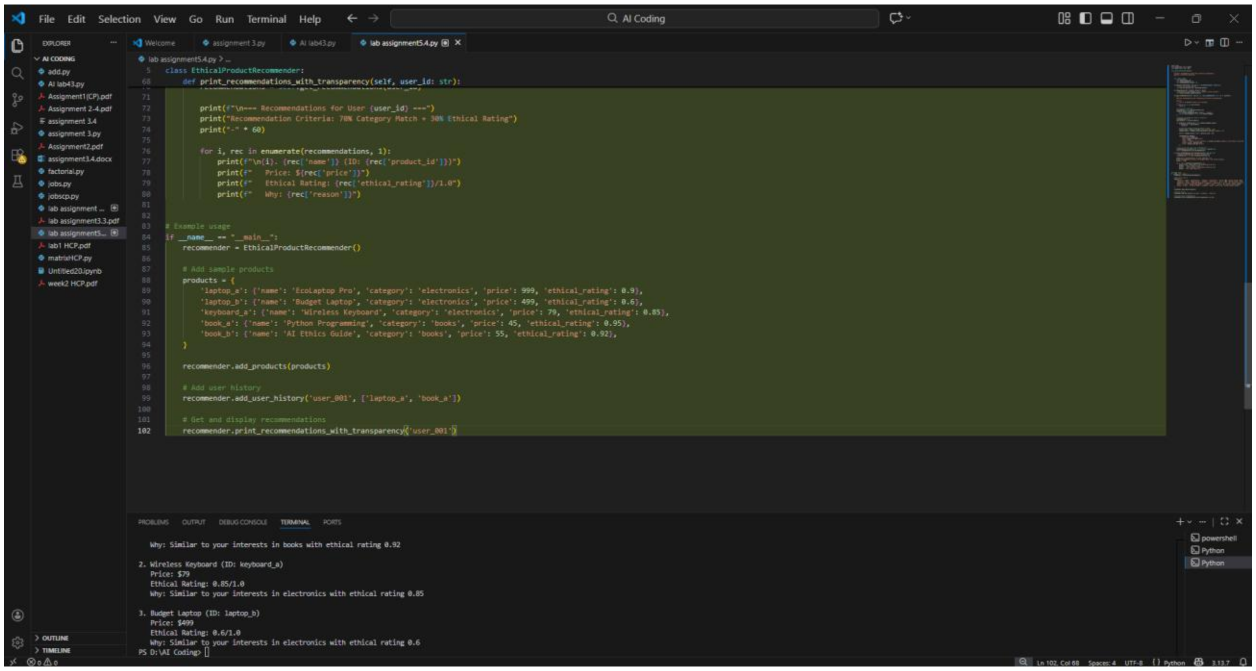
```python
import json
from collections import Counter
from typing import List, Dict

class EthicalProductRecommender:
    """
    A product recommendation system that prioritizes transparency,
    fairness, and ethical guidelines.
    """

    def __init__(self):
        self.user_history = {}
        self.product_database = {}
        self.recommendation_reasons = {}

    def add_user_history(self, user_id: str, purchased_products: List[str]):
        """Store user purchase history"""
        self.user_history[user_id] = purchased_products

    def add_products(self, products: Dict[str, Dict]):
        """Add products with metadata (price, category, ethical_rating)"""
        self.product_database.update(products)

    def get_recommendations(self, user_id: str, num_recommendations: int = 5) -> List[Dict]:
        """
        Generate recommendations with transparency and ethical considerations.

        Returns:
            List of recommended products with reasoning
        """
        if user_id not in self.user_history:
            return []

        # Analyze user preferences
        user_products = self.user_history[user_id]
        category_counts = Counter([
            self.product_database[p].get('category', 'unknown')
            for p in user_products if p in self.product_database
        ])

        # Generate recommendations based on categories
```

```
1. AI Ethics Guide (ID: book_b)
   Price: $55
   Ethical Rating: 0.92/1.0
   Why: Similar to your interests in books with ethical rating 0.92

2. Wireless Keyboard (ID: keyboard_a)
   Price: $79
   Ethical Rating: 0.85/1.0
   Why: Similar to your interests in electronics with ethical rating 0.85

3. Budget Laptop (ID: laptop_b)
```



```python
class EthicalProductRecommender:
    def get_recommendations(self, user_id: str, num_recommendations: int = 5) -> List[Dict]:
        recommendations = []
        seen_products = set(user_products)

        for product_id, product_info in self.product_database.items():
            if product_id in seen_products:
                continue

            # Prioritize products with good ethical ratings
            ethical_score = product_info.get('ethical_rating', 0.5)
            category_match = category_counts.get(product_info.get('category'), 0)

            score = (category_match * 0.6) + (ethical_score * 0.4)

            recommendations.append({
                'product_id': product_id,
                'name': product_info.get('name'),
                'score': score,
                'reason': f"Similar to your interests in {product_info.get('category')} with ethical rating {ethical_score}",
                'ethical_rating': ethical_score,
                'price': product_info.get('price')
            })

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

    def print_recommendations_with_transparency(self, user_id: str):
        """Display recommendations with full transparency"""
        recommendations = self.get_recommendations(user_id)

        print(f"\n=== Recommendations for User {user_id} ===")
        print("Recommendation Criteria: 70% Category Match + 30% Ethical Rating")
        print("-" * 60)

        for i, rec in enumerate(recommendations, 1):
            print(f"\n{i}. {rec['name']} (ID: {rec['product_id']})")
            print(f"   Price: ${rec['price']}")
            print(f"   Ethical Rating: {rec['ethical_rating']}/1.0")
            print(f"   Why: {rec['reason']}")
```

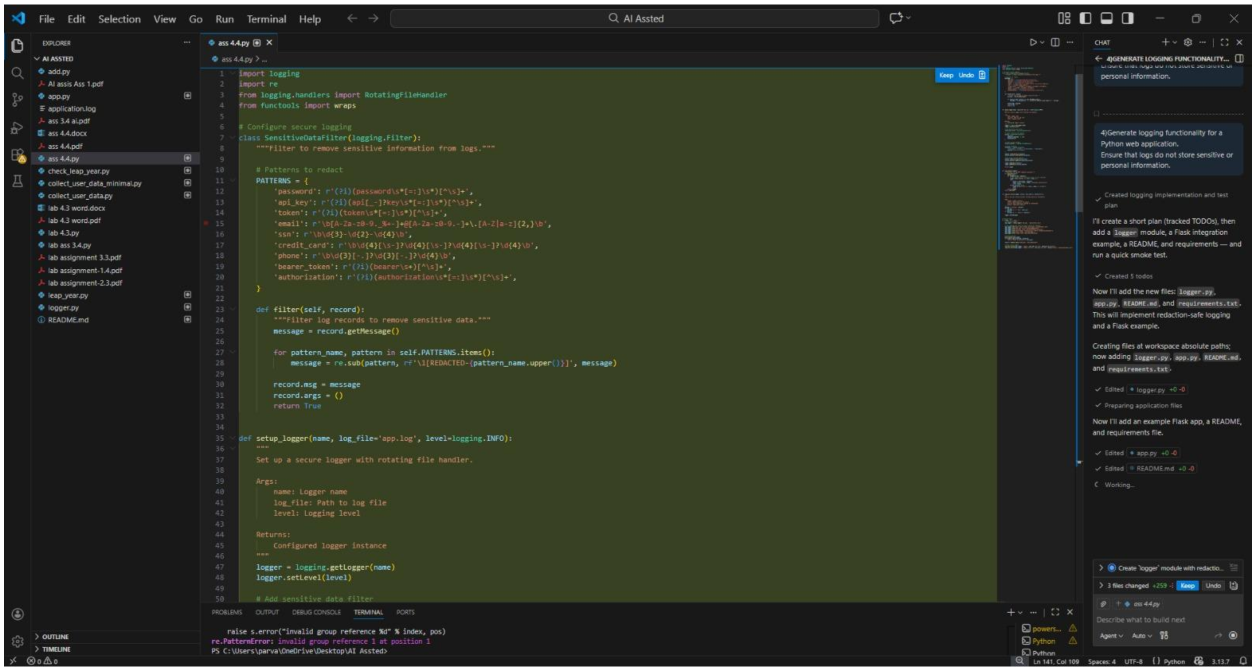
```
   Why: Similar to your interests in books with ethical rating 0.92

2. Wireless Keyboard (ID: keyboard_a)
   Price: $79
   Ethical Rating: 0.85/1.0
   Why: Similar to your interests in electronics with ethical rating 0.85

3. Budget Laptop (ID: laptop_b)
   Price: $499
   Ethical Rating: 0.6/1.0
   Why: Similar to your interests in electronics with ethical rating 0.6
PS D:\AI Coding> 
```



```python
class EthicalProductRecommender:
    def print_recommendations_with_transparency(self, user_id: str):

        print(f"\n=== Recommendations for User {user_id} ===")
        print("Recommendation Criteria: 70% Category Match + 30% Ethical Rating")
        print("-" * 60)

        for i, rec in enumerate(recommendations, 1):
            print(f"\n{i}. {rec['name']} (ID: {rec['product_id']})")
            print(f"   Price: ${rec['price']}")
            print(f"   Ethical Rating: {rec['ethical_rating']}/1.0")
            print(f"   Why: {rec['reason']}")


# Example usage
if __name__ == "__main__":
    recommender = EthicalProductRecommender()

    # Add sample products
    products = {
        'laptop_a': {'name': 'EcoLaptop Pro', 'category': 'electronics', 'price': 999, 'ethical_rating': 0.9},
        'laptop_b': {'name': 'Budget Laptop', 'category': 'electronics', 'price': 499, 'ethical_rating': 0.6},
        'keyboard_a': {'name': 'Wireless Keyboard', 'category': 'electronics', 'price': 79, 'ethical_rating': 0.85},
        'book_a': {'name': 'Python Programming', 'category': 'books', 'price': 45, 'ethical_rating': 0.95},
        'book_b': {'name': 'AI Ethics Guide', 'category': 'books', 'price': 55, 'ethical_rating': 0.92},
    }

    recommender.add_products(products)

    # Add user history
    recommender.add_user_history('user_001', ['laptop_a', 'book_a'])

    # Get and display recommendations
    recommender.print_recommendations_with_transparency('user_001')
```

```
   Why: Similar to your interests in books with ethical rating 0.92

2. Wireless Keyboard (ID: keyboard_a)
   Price: $79
   Ethical Rating: 0.85/1.0
   Why: Similar to your interests in electronics with ethical rating 0.85

3. Budget Laptop (ID: laptop_b)
   Price: $499
   Ethical Rating: 0.6/1.0
   Why: Similar to your interests in electronics with ethical rating 0.6
PS D:\AI Coding> 
```

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



```python
import logging
import re
from logging.handlers import RotatingFileHandler
from functools import wraps

# Configure secure logging
class SensitiveDataFilter(logging.Filter):
    """Filter to remove sensitive information from logs."""

    # Patterns to redact
    PATTERNS = {
        'password': r'(?i)(password\s*[=:]\s*)[^\s]+',
        'api_key': r'(?i)(api[_-]?key\s*[=:]\s*)[^\s]+',
        'token': r'(?i)(token\s*[=:]\s*)[^\s]+',
        'email': r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b',
        'ssn': r'\b\d{3}-\d{2}-\d{4}\b',
        'credit_card': r'\b\d{4}[\s-]?\d{4}[\s-]?\d{4}[\s-]?\d{4}\b',
        'phone': r'\b\d{3}[-.]?\d{3}[-.]?\d{4}\b',
        'bearer_token': r'(?i)(bearer\s+)[^\s]+',
        'authorization': r'(?i)(authorization\s*[=:]\s*)[^\s]+',
    }

    def filter(self, record):
        """Filter log records to remove sensitive data."""
        message = record.getMessage()

        for pattern_name, pattern in self.PATTERNS.items():
            message = re.sub(pattern, rf'\1[REDACTED-{pattern_name.upper()}]', message)

        record.msg = message
        record.args = ()
        return True


def setup_logger(name, log_file='app.log', level=logging.INFO):
    """
    Set up a secure logger with rotating file handler.

    Args:
        name: Logger name
        log_file: Path to log file
        level: Logging level

    Returns:
        Configured logger instance
    """
    logger = logging.getLogger(name)
    logger.setLevel(level)

    # Add sensitive data filter
```

```
    raise s.error("invalid group reference %d" % index, pos)
re.PatternError: invalid group reference 1 at position 1
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

4)GENERATE LOGGING FUNCTIONALITY...

personal information.

4)Generate logging functionality for a Python web application.
Ensure that logs do not store sensitive or personal information.

✓ Created logging implementation and test plan

I'll create a short plan (tracked TODOs), then add a `logger` module, a Flask integration example, a README, and requirements — and run a quick smoke test.

✓ Created 5 todos

Now I'll add the new files: `logger.py`, `app.py`, `README.md`, and `requirements.txt`. This will implement redaction-safe logging and a Flask example.

Creating files at workspace absolute paths; now adding `logger.py`, `app.py`, `README.md`, and `requirements.txt`.

✓ Edited logger.py +0 -0

✓ Preparing application files

Now I'll add an example Flask app, a README, and requirements file.

✓ Edited app.py +0 -0

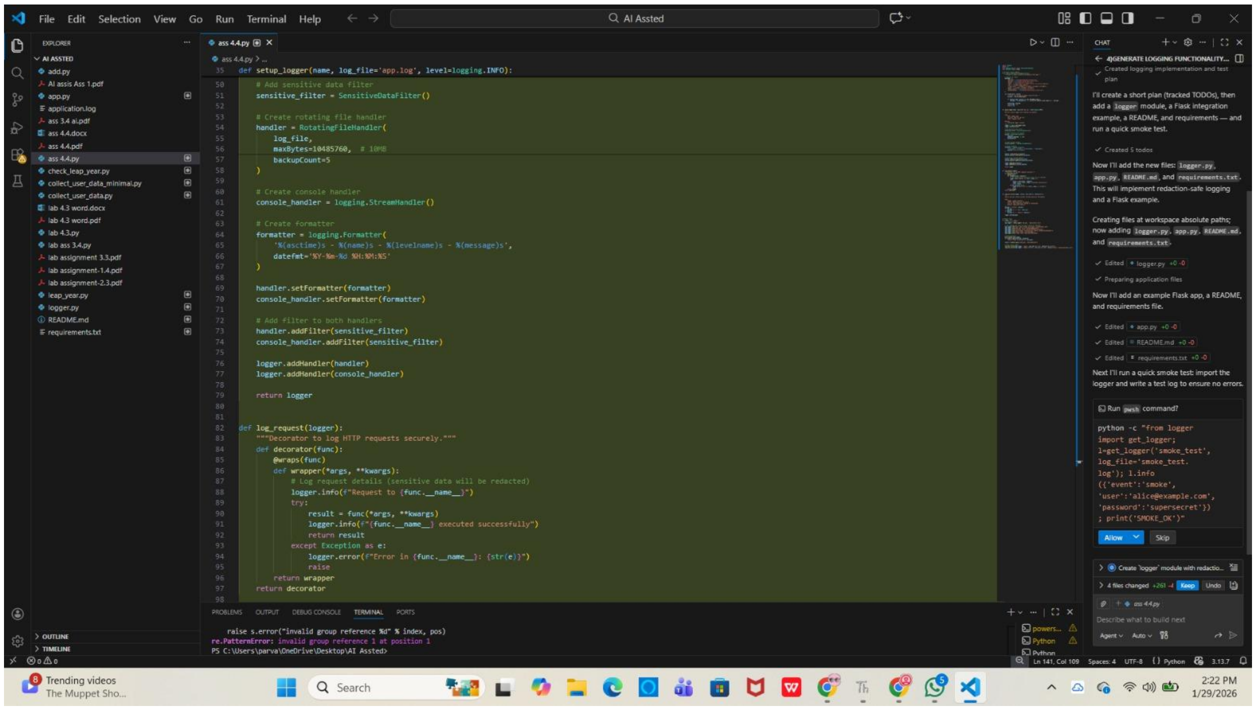
✓ Edited README.md +0 -0

Working...

Create `logger` module with redactio...

3 files changed +259 -1 Keep Undo

Describe what to build next



```python
def setup_logger(name, log_file='app.log', level=logging.INFO):
    # Add sensitive data filter
    sensitive_filter = SensitiveDataFilter()

    # Create rotating file handler
    handler = RotatingFileHandler(
        log_file,
        maxBytes=10485760,  # 10MB
        backupCount=5
    )

    # Create console handler
    console_handler = logging.StreamHandler()

    # Create formatter
    formatter = logging.Formatter(
        '%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        datefmt='%Y-%m-%d %H:%M:%S'
    )

    handler.setFormatter(formatter)
    console_handler.setFormatter(formatter)

    # Add filter to both handlers
    handler.addFilter(sensitive_filter)
    console_handler.addFilter(sensitive_filter)

    logger.addHandler(handler)
    logger.addHandler(console_handler)

    return logger


def log_request(logger):
    """Decorator to log HTTP requests securely."""
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            # Log request details (sensitive data will be redacted)
            logger.info(f"Request to {func.__name__}")
            try:
                result = func(*args, **kwargs)
                logger.info(f"{func.__name__} executed successfully")
                return result
            except Exception as e:
                logger.error(f"Error in {func.__name__}: {str(e)}")
                raise
        return wrapper
    return decorator
```

```
    raise s.error("invalid group reference %d" % index, pos)
re.PatternError: invalid group reference 1 at position 1
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

4)GENERATE LOGGING FUNCTIONALITY...

✓ Created logging implementation and test plan

I'll create a short plan (tracked TODOs), then add a `logger` module, a Flask integration example, a README, and requirements — and run a quick smoke test.

✓ Created 5 todos

Now I'll add the new files: `logger.py`, `app.py`, `README.md`, and `requirements.txt`. This will implement redaction-safe logging and a Flask example.

Creating files at workspace absolute paths; now adding `logger.py`, `app.py`, `README.md`, and `requirements.txt`.

✓ Edited logger.py +0 -0

✓ Preparing application files

Now I'll add an example Flask app, a README, and requirements file.

✓ Edited app.py +0 -0

✓ Edited README.md +0 -0

✓ Edited requirements.txt +0 -0

Next I'll run a quick smoke test: import the logger and write a test log to ensure no errors.

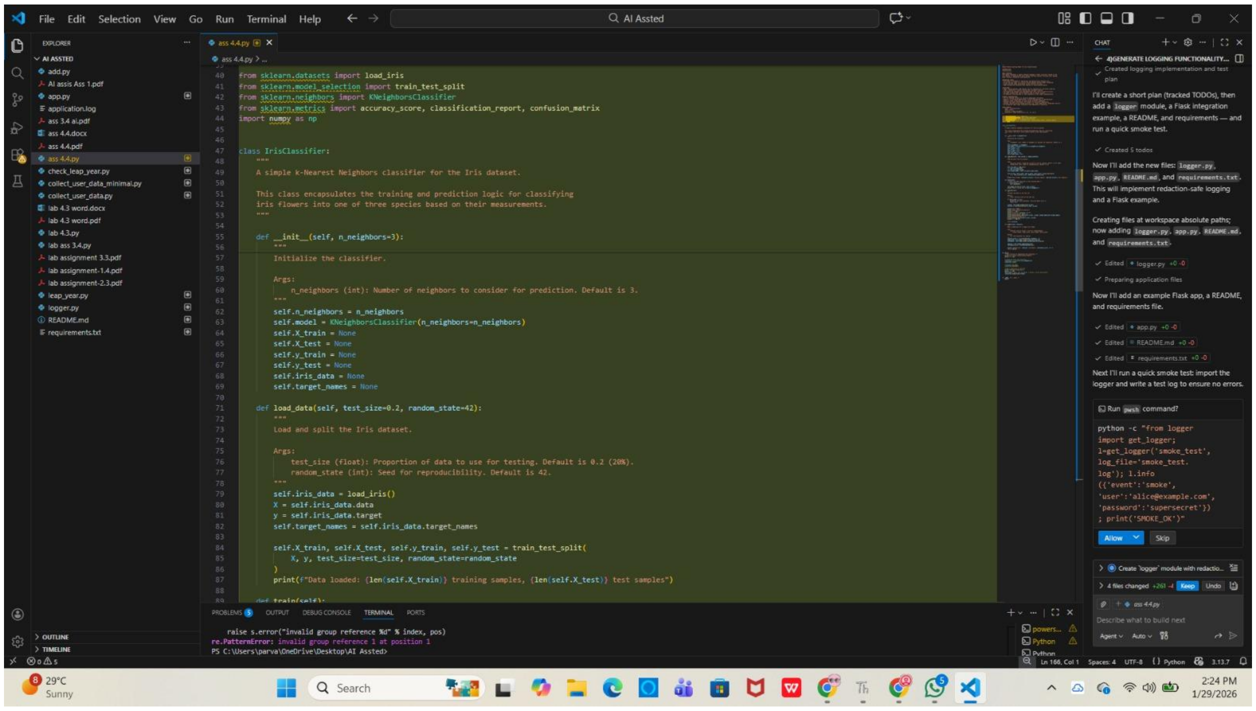
Run `pwsh` command?

```
python -c "from logger
import get_logger;
l=get_logger('smoke_test',
log_file='smoke_test.
log'); l.info
({'event':'smoke',
'user':'alice@example.com',
'password':'supersecret'})
; print('SMOKE_OK')"
```

Create `logger` module with redactio...

4 files changed +261 -4

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).



```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import numpy as np


class IrisClassifier:
    """
    A simple k-Nearest Neighbors classifier for the Iris dataset.

    This class encapsulates the training and prediction logic for classifying
    iris flowers into one of three species based on their measurements.
    """

    def __init__(self, n_neighbors=3):
        """
        Initialize the classifier.

        Args:
            n_neighbors (int): Number of neighbors to consider for prediction. Default is 3.
        """
        self.n_neighbors = n_neighbors
        self.model = KNeighborsClassifier(n_neighbors=n_neighbors)
        self.X_train = None
        self.X_test = None
        self.y_train = None
        self.y_test = None
        self.iris_data = None
        self.target_names = None

    def load_data(self, test_size=0.2, random_state=42):
        """
        Load and split the Iris dataset.

        Args:
            test_size (float): Proportion of data to use for testing. Default is 0.2 (20%).
            random_state (int): Seed for reproducibility. Default is 42.
        """
        self.iris_data = load_iris()
        X = self.iris_data.data
        y = self.iris_data.target
        self.target_names = self.iris_data.target_names

        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, y, test_size=test_size, random_state=random_state
        )
        print(f"Data loaded: {len(self.X_train)} training samples, {len(self.X_test)} test samples")

    def train(self):
```

```
    raise s.error("invalid group reference %d" % index, pos)
re.PatternError: invalid group reference 1 at position 1
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

4)GENERATE LOGGING FUNCTIONALITY...

✓ Created logging implementation and test plan

I'll create a short plan (tracked TODOs), then add a `logger` module, a Flask integration example, a README, and requirements — and run a quick smoke test.

✓ Created 5 todos

Now I'll add the new files: `logger.py`, `app.py`, `README.md`, and `requirements.txt`. This will implement redaction-safe logging and a Flask example.

Creating files at workspace absolute paths; now adding `logger.py`, `app.py`, `README.md`, and `requirements.txt`.

✓ Edited logger.py +0 -0

✓ Preparing application files

Now I'll add an example Flask app, a README, and requirements file.

✓ Edited app.py +0 -0

✓ Edited README.md +0 -0

✓ Edited requirements.txt +0 -0

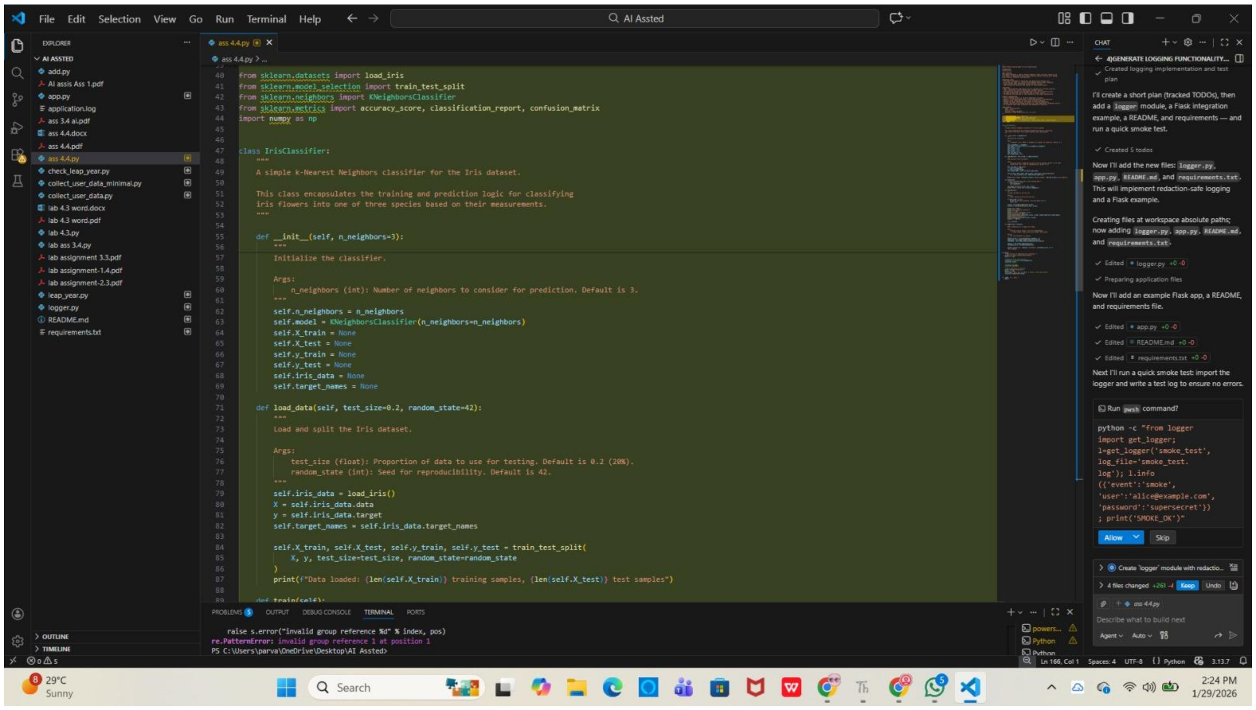
Next I'll run a quick smoke test: import the logger and write a test log to ensure no errors.

Run `pwsh` command?

```
python -c "from logger
import get_logger;
l=get_logger('smoke_test',
log_file='smoke_test.
log'); l.info
({'event':'smoke',
'user':'alice@example.com',
'password':'supersecret'})
; print('SMOKE_OK')"
```

Create `logger` module with redactio...

4 files changed +261 -4



```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import numpy as np


class IrisClassifier:
    """
    A simple k-Nearest Neighbors classifier for the Iris dataset.

    This class encapsulates the training and prediction logic for classifying
    iris flowers into one of three species based on their measurements.
    """

    def __init__(self, n_neighbors=3):
        """
        Initialize the classifier.

        Args:
            n_neighbors (int): Number of neighbors to consider for prediction. Default is 3.
        """
        self.n_neighbors = n_neighbors
        self.model = KNeighborsClassifier(n_neighbors=n_neighbors)
        self.X_train = None
        self.X_test = None
        self.y_train = None
        self.y_test = None
        self.iris_data = None
        self.target_names = None

    def load_data(self, test_size=0.2, random_state=42):
        """
        Load and split the Iris dataset.

        Args:
            test_size (float): Proportion of data to use for testing. Default is 0.2 (20%).
            random_state (int): Seed for reproducibility. Default is 42.
        """
        self.iris_data = load_iris()
        X = self.iris_data.data
        y = self.iris_data.target
        self.target_names = self.iris_data.target_names

        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, y, test_size=test_size, random_state=random_state
        )
        print(f"Data loaded: {len(self.X_train)} training samples, {len(self.X_test)} test samples")
```

```
    raise s.error("invalid group reference %d" % index, pos)
re.PatternError: invalid group reference 1 at position 1
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

4)GENERATE LOGGING FUNCTIONALITY...

✓ Created logging implementation and test plan

I'll create a short plan (tracked TODOs), then add a `logger` module, a Flask integration example, a README, and requirements — and run a quick smoke test.

✓ Created 5 todos

Now I'll add the new files: `logger.py`, `app.py`, `README.md`, and `requirements.txt`. This will implement redaction-safe logging and a Flask example.

Creating files at workspace absolute paths; now adding `logger.py`, `app.py`, `README.md`, and `requirements.txt`.

✓ Edited logger.py +0 -0

✓ Preparing application files

Now I'll add an example Flask app, a README, and requirements file.

✓ Edited app.py +0 -0

✓ Edited README.md +0 -0

✓ Edited requirements.txt +0 -0

Next I'll run a quick smoke test: import the logger and write a test log to ensure no errors.

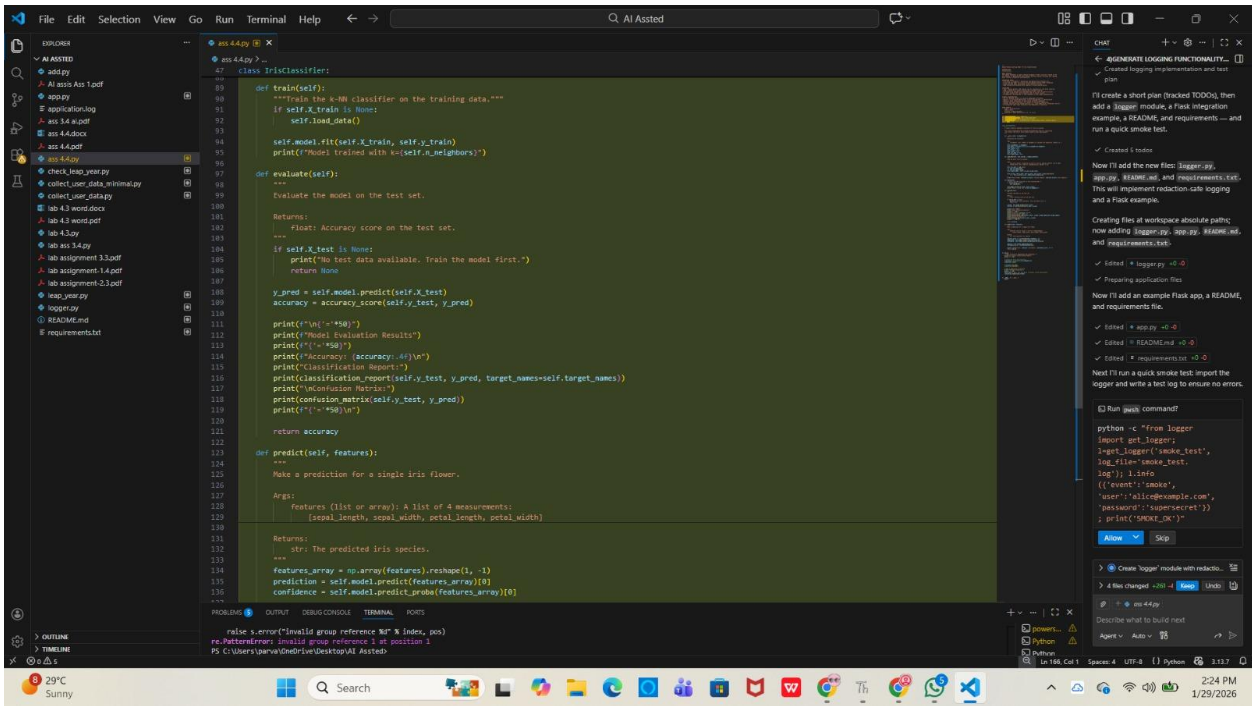
Run `pwsh` command?

```
python -c "from logger import get_logger; l=get_logger('smoke_test', log_file='smoke_test.log'); l.info({'event':'smoke', 'user':'alice@example.com', 'password':'supersecret'}); print('SMOKE_OK')"
```

Allow | Skip

Create `logger` module with redactio...

4 files changed +261 -4 Keep Undo



```python
    def train(self):
        """Train the k-NN classifier on the training data."""
        if self.X_train is None:
            self.load_data()

        self.model.fit(self.X_train, self.y_train)
        print(f"Model trained with k={self.n_neighbors}")

    def evaluate(self):
        """
        Evaluate the model on the test set.

        Returns:
            float: Accuracy score on the test set.
        """
        if self.X_test is None:
            print("No test data available. Train the model first.")
            return None

        y_pred = self.model.predict(self.X_test)
        accuracy = accuracy_score(self.y_test, y_pred)

        print(f"\n{'='*50}")
        print(f"Model Evaluation Results")
        print(f"{'='*50}")
        print(f"Accuracy: {accuracy:.4f}\n")
        print("Classification Report:")
        print(classification_report(self.y_test, y_pred, target_names=self.target_names))
        print("\nConfusion Matrix:")
        print(confusion_matrix(self.y_test, y_pred))
        print(f"{'='*50}\n")

        return accuracy

    def predict(self, features):
        """
        Make a prediction for a single iris flower.

        Args:
            features (list or array): A list of 4 measurements:
                [sepal_length, sepal_width, petal_length, petal_width]

        Returns:
            str: The predicted iris species.
        """
        features_array = np.array(features).reshape(1, -1)
        prediction = self.model.predict(features_array)[0]
        confidence = self.model.predict_proba(features_array)[0]
```

```
    raise s.error("invalid group reference %d" % index, pos)
re.PatternError: invalid group reference 1 at position 1
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```