K.Akshay
2303A51817
B-26
Assignment -5.4

Q.No.	Question	*Expected Time 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. **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.	Week3 -

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).

Prompt :

S Write a Python script that collects basic user data such as name, age, and email. Then, add inline comments explaining how to anonymize or protect this data. For example, suggest hashing or encrypting sensitive fields like email, avoiding plain-text storage, and following best practices for data privacy.

Code :

```python
import hashlib

# --- Data Collection ---
name = input("Enter your name: ")
age = input("Enter your age: ")
email = input("Enter your email: ")

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

print("\nCollected Data:", user_data)

# --- Data Protection and Anonymization Comments ---

# 1. Avoid storing sensitive data in plain text.
#    Instead of directly saving 'name' or 'email', consider these methods:

# 2. Hashing (for email or other identifiers):
#    Hashing can be used to create a one-way representation of data.
#    It's useful for verifying data integrity or for unique identifiers where the original
#    value doesn't need to be retrieved (e.g., for matching users without knowing their actual email).
#    Note: Hashing is NOT encryption. It's irreversible. Use a strong, salted hash function.
```

```
#    Example using SHA256 for email:
#    hashed_email = hashlib.sha256(email.encode()).hexdigest()
#    print(f"Hashed Email (SHA256): {hashed_email}")

# 3. Encryption (for data that needs to be retrieved later):
#    Encryption transforms data into an unreadable format that can be reverted
#    back to its original form with a decryption key. This is suitable for sensitive
#    information like addresses, financial details, or medical records.
#    Libraries like `cryptography` can be used for robust encryption.
#    Example (conceptual, requires setup):
#    from cryptography.fernet import Fernet
#    key = Fernet.generate_key()
#    cipher_suite = Fernet(key)
#    encrypted_age = cipher_suite.encrypt(age.encode())
#    print(f"Encrypted Age: {encrypted_age}")
#    decrypted_age = cipher_suite.decrypt(encrypted_age).decode()
#    print(f"Decrypted Age: {decrypted_age}")

# 4. Pseudonymization:
#    Replacing identifying fields with artificial identifiers (pseudonyms).
#    The link between the pseudonym and the real identity is kept separate and secure.
#    Example: Assigning a 'user_id' instead of using 'name' directly in analytics.
#    user_id = "USR_001"
#    anonymized_data = {"user_id": user_id, "age": user_data["age"], "hashed_email": hashed_email}

# 5. Data Masking/Tokenization:
#    Replacing sensitive data with structurally similar but non-sensitive data.
#    For example, showing only the last four digits of a credit card number.
```

```
# 6. Secure Storage:
#    Regardless of anonymization/protection techniques, always store data in secure database
#    with access controls, not in plain text files. Encrypt storage at rest and in transit.

# 7. Data Minimization:
#    Only collect data that is absolutely necessary for the intended purpose.

# 8. Consent:
#    Always obtain explicit consent from users before collecting and processing their data.
```

Output :

```
Enter your name: Aarvi
Enter your age: 24
Enter your email: Aaravi1234@gmail.com

Collected Data: {'name': 'Aarvi', 'age': '24', 'email': 'Aaravi1234@gmail.com'}
```

Explanation : This script shows how to collect user data and stresses protecting it: avoid plain-text storage, use hashing for identifiers, encryption for sensitive fields, pseudonymization or masking to reduce exposure, and always store data securely. Collect only what's necessary and ensure user consent before processing.

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).

Prompt :

S Write a Python function that performs sentiment analysis on text input. Then, add inline comments explaining how to identify and handle potential biases in the dataset. For example, mention strategies like balancing positive/negative samples, removing offensive or discriminatory terms, and ensuring diverse training data to avoid skewed results.

Code:

```
from textblob import TextBlob

def analyze_sentiment(text):
    """
    Performs sentiment analysis on the given text.
    Returns a dictionary with polarity and subjectivity scores.
    Polarity ranges from -1.0 (negative) to 1.0 (positive).
    Subjectivity ranges from 0.0 (objective) to 1.0 (subjective).
    """
    analysis = TextBlob(text)
    return {'polarity': analysis.sentiment.polarity, 'subjectivity': analysis.sentiment.subjectivity}

# --- Bias Identification and Handling in Sentiment Analysis Data ---

# 1. Balancing Positive/Negative Samples:
#    Bias often arises from imbalanced datasets where one sentiment class (e.g., positive) is
#    significantly more represented than others. This can lead the model to be biased towards
#    the majority class.
#    Strategy: Ensure a roughly equal distribution of positive, negative, and neutral examples
#    in your training data. Techniques like oversampling (duplicating minority class samples)
#    or undersampling (removing majority class samples) can help.

# 2. Removing Offensive or Discriminatory Terms:
#    If the training data contains offensive, hateful, or discriminatory language, the model
#    can learn these biases and perpetuate them in its predictions. For example, it might
#    assign negative sentiment to certain demographic groups' names or specific dialects.
#    Strategy: Implement robust data cleaning techniques to identify and remove or replace
#    such terms. This often requires a combination of lexicon-based filtering and machine
```

```
#    learning models trained to detect hate speech.

# 3. Ensuring Diverse Training Data:
#    A lack of diversity in training data (e.g., data primarily from one demographic group,
#    region, or social context) can lead to models that perform poorly or exhibit bias
#    when applied to other groups.
#    Strategy: Actively collect and include data from a wide range of sources, demographics,
#    languages, and cultural contexts. Regularly audit the dataset for representation gaps.

# 4. Human Review and Annotation Guidelines:
#    The biases of human annotators can be encoded into the labels. For example, different
#    annotators might interpret sarcasm or subtle nuances differently based on their background.
#    Strategy: Develop clear, comprehensive, and unbiased annotation guidelines. Employ multiple
#    annotators for the same data and measure inter-annotator agreement to identify inconsistencies.

# 5. Regular Model Auditing and Fairness Metrics:
#    Even with careful data preparation, biases can still emerge. It's crucial to continuously
#    monitor the model's performance across different subgroups.
#    Strategy: After training, evaluate the model using fairness metrics (e.g., disparate impact,
#    equal opportunity) on different demographic slices of your test data. Retrain or fine-tune
#    the model if significant biases are detected.

# Example Usage:
# text1 = "I love this product, it's amazing!"
# sentiment1 = analyze_sentiment(text1)
# print(f"'{text1}' -> Polarity: {sentiment1['polarity']:.2f}, Subjectivity: {sentiment1['subjectivity']:.2f}")

# text2 = "This is a terrible experience and I'm very disappointed."
```

```
# text2 = "This is a terrible experience and I'm very disappointed."
# sentiment2 = analyze_sentiment(text2)
# print(f"'{text2}' -> Polarity: {sentiment2['polarity']:.2f}, Subjectivity: {sentiment2['subjectivity']:.2f}")

# text3 = "The quick brown fox jumps over the lazy dog."
# sentiment3 = analyze_sentiment(text3)
# print(f"'{text3}' -> Polarity: {sentiment3['polarity']:.2f}, Subjectivity: {sentiment3['subjectivity']:.2f}")
```

Explanation : This function uses TextBlob to analyze text sentiment, returning polarity (negative to positive) and subjectivity (objective to subjective). The comments highlight bias risks—like imbalanced datasets, offensive terms, lack of diversity, and annotator subjectivity—and suggest strategies such as balancing samples, cleaning data, diversifying sources, clear annotation guidelines, and regular fairness audits to ensure reliable, unbiased sentiment analysis.

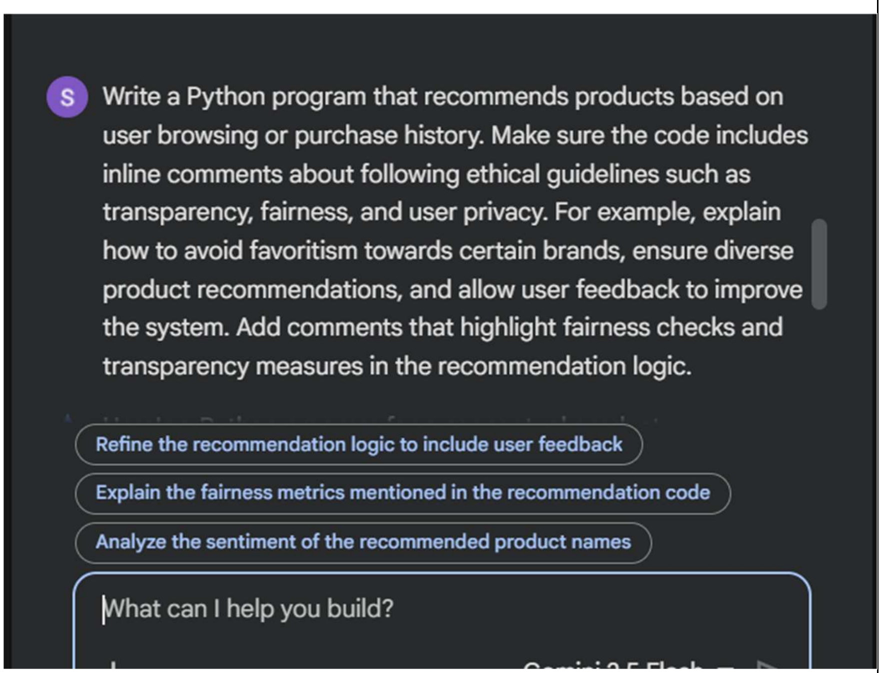
Task Description #3:

• 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.

Prompt :



Code :

```python
# --- Conceptual Product Recommendation System ---

# This script demonstrates a basic product recommendation logic
# and, more importantly, provides inline comments on ethical considerations
# for building and deploying such systems.

# --- 1. Data Simulation (for demonstration purposes) ---

# In a real-world scenario, this data would come from databases,
# user activity logs, etc.

# User's historical interactions (e.g., purchase history, browsing history)
# Each item could have more attributes in a real system.
user_history = [
    {"product_id": "P101", "category": "Electronics", "brand": "BrandX", "price": 500},
    {"product_id": "P102", "category": "Books", "brand": "PublisherA", "price": 20},
    {"product_id": "P103", "category": "Electronics", "brand": "BrandY", "price": 750},
    {"product_id": "P104", "category": "Home Goods", "brand": "BrandZ", "price": 80}
]

# A catalog of available products
product_catalog = [
    {"product_id": "P101", "category": "Electronics", "brand": "BrandX", "name": "Smartphone A", "price": 500},
    {"product_id": "P105", "category": "Electronics", "brand": "BrandA", "name": "Headphones B", "price": 150},
    {"product_id": "P106", "category": "Books", "brand": "PublisherB", "name": "Fantasy Novel C", "price": 25},
```

```python
    {"product_id": "P107", "category": "Books", "brand": "PublisherA", "name": "Science Fiction D", "price": 18},
    {"product_id": "P108", "category": "Home Goods", "brand": "BrandZ", "name": "Smart Plug E", "price": 40},
    {"product_id": "P109", "category": "Clothing", "brand": "BrandC", "name": "T-Shirt F", "price": 30},
    {"product_id": "P110", "category": "Electronics", "brand": "BrandX", "name": "Laptop G", "price": 1200},
    {"product_id": "P111", "category": "Home Goods", "brand": "BrandA", "name": "Coffee Maker H", "price": 90}
]

# --- 2. Recommendation Logic ---

def recommend_products(history, catalog, num_recommendations=3):
    # Extract categories from user's history
    preferred_categories = {item["category"] for item in history}
    print(f"\nUser's preferred categories based on history: {preferred_categories}")

    # Simple logic: Recommend products from preferred categories not already in history
    recommended = []
    history_product_ids = {item["product_id"] for item in history}

    # Ethical Guideline: Transparency
    # It's important to clearly communicate to the user *why* a product is being recommended.
    # For example, "Because you viewed items in Electronics" or "Similar to your past purchase of..."
    # This builds trust and helps users understand the system.

    # Ethical Guideline: Fairness (Avoiding Favoritism & Ensuring Diversity)
    # Directly recommending based on categories can still lead to biases.
    # - Favoritism: Ensure the recommendation algorithm doesn't unduly prioritize certain brands
    #   or sellers due to undisclosed financial incentives. Regularly audit recommendations
    #   for brand diversity.
```

```python
    # - Diversity: Avoid creating 'filter bubbles'. Introduce mechanisms for serendipity (e.g.,
    #   recommendations from slightly different categories, or popular items outside the user's usual).
    #   Consider using techniques like 'item cold-start' to recommend new or less popular items.

    for product in catalog:
        if product["category"] in preferred_categories and product["product_id"] not in history_product_ids:
            recommended.append(product)

    # Ethical Guideline: Fairness (Algorithmic Bias)
    # Before presenting recommendations, perform fairness checks. For example, ensure that
    # recommendations are not biased against certain demographic groups. If the training data
    # for the recommender algorithm is skewed, it can perpetuate or amplify existing societal biases.
    # Regularly evaluate the model's performance and recommendation distribution across different
    # user segments.

    # Shuffle recommendations to ensure different items are shown if multiple calls are made
    # and to reduce position bias.
    import random
    random.shuffle(recommended)

    return recommended[:num_recommendations]

# --- 3. Generate Recommendations ---
recommendations = recommend_products(user_history, product_catalog)

print("\n--- Generated Recommendations ---")
if recommendations:
    for rec in recommendations:
```

```
        print(f"- {rec['name']} ({rec['category']} by {rec['brand']}) - Price: ${rec['price']}")
else:
    print("No new recommendations based on current logic and catalog.")

# --- 4. Ethical Considerations in Comments ---

# Ethical Guideline: User Privacy
# - Data Minimization: Only collect and use data that is absolutely necessary for the recommendation
#   system. Avoid collecting overly sensitive information if it doesn't directly contribute.
# - Anonymization/Pseudonymization: Whenever possible, use anonymized or pseudonymized data.
#   Directly identifiable information should be handled with extreme care and robust security.
#   (Refer back to the previous exercise on data protection!)
# - User Control: Provide users with clear options to control their data (e.g., delete history,
#   opt-out of recommendations) and manage their privacy settings. This should be easily accessible.
# - Data Security: All user data used for recommendations must be stored and processed securely,
#   adhering to best practices for encryption, access control, and regular security audits.

# Ethical Guideline: User Feedback Loop
# - Allow Users to Provide Feedback: Implement mechanisms for users to give explicit feedback
#   on recommendations (e.g., "not interested", "already own", "liked this").
# - Use Feedback to Improve: Incorporate this feedback to refine the recommendation algorithm,
#   making it more personalized and relevant over time. This also serves as a valuable signal
#   to detect and correct biases or errors in the system.

# Ethical Guideline: Transparency in Recommendation Logic
# - Explainability: Strive for models that can provide some level of explanation for their
#   recommendations. While complex models might be 'black boxes', there should be ways to
#   understand the general principles driving the suggestions.
```

```
User's preferred categories based on history: {'Books', 'Electronics', 'Home Goods'}

--- Generated Recommendations ---
- Fantasy Novel C (Books by PublisherB) - Price: $25
- Laptop G (Electronics by BrandX) - Price: $1200
- Coffee Maker H (Home Goods by BrandA) - Price: $90
```

Explanation : This program recommends products based on user history while stressing ethics: it explains why items are suggested (transparency), avoids favoritism and promotes diversity (fairness), protects privacy through data minimization and secure handling, and lets users give feedback to refine recommendations responsibly.

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.

Prompt :

Write Python logging functionality for a web application. Ensure the logs do not record sensitive information such as passwords, emails, or personal identifiers. Add inline comments explaining ethical logging practices, including data minimization, avoiding sensitive fields, and secure log storage.

```python
import logging
import re

# --- 1. Basic Logger Configuration ---

# Configure the logging system
logging.basicConfig(
    level=logging.INFO,  # Set the minimum level of messages to log
    format='%(asctime)s - %(levelname)s - %(name)s - %(message)s', # Standard log format
    handlers=[
        logging.StreamHandler(),  # Log to console
        logging.FileHandler('web_app.log') # Log to a file
    ]
)

# Get a logger instance for the web application
app_logger = logging.getLogger('web_app')

# --- 2. Custom Filter for Sensitive Data ---

class SensitiveDataFilter(logging.Filter):
    """
    A logging filter to redact sensitive information from log messages.
    This is a simple example; a more robust solution might use a dictionary
    of sensitive terms or more complex regex patterns.
    """
```

```python
    def filter(self, record):
        # Ethical Logging Practice: Avoiding Sensitive Fields
        # Regularly review what constitutes 'sensitive' for your application
        # and update redaction rules accordingly.
        # Common sensitive patterns: email addresses, passwords, credit card numbers.

        message = record.getMessage()

        # Redact email addresses
        message = re.sub(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', '[REDACTED_EMAIL]', message)

        # Redact common password indicators (this is a simple example)
        message = re.sub(r'(password|pwd|secret)(\s*=\s*|:\s*)?\S+', '[REDACTED_PASSWORD]', message, flags=re.IGNORECASE)

        # Redact potential personal identifiers (e.g., specific names, though this is harder to generalize)
        # For this example, let's redact the name 'Aarvi' if it appears in the message
        message = re.sub(r'Aarvi', '[REDACTED_NAME]', message, flags=re.IGNORECASE)

        record.msg = message
        return True

# Add the sensitive data filter to the logger
app_logger.addFilter(SensitiveDataFilter())

# --- 3. Example Logging ---

app_logger.info('User logged in successfully.')
app_logger.warning('Failed login attempt for user: Aarvi (Aaravi1234@gmail.com) with password: my_secure_password123')
```

```python
app_logger.info('User profile updated for user_id: 12345.')
app_logger.error('Database connection failed: check credentials.')
app_logger.debug('This is a debug message and should not appear at INFO level.')
app_logger.info('New user registered with email: another_user@example.com.')

# --- 4. Ethical Logging Practices Comments ---

# Ethical Guideline: Data Minimization
# - Only log information that is absolutely necessary for debugging, auditing, or security purposes.
#   Avoid logging entire request bodies or responses unless strictly required and properly scrubbed.

# Ethical Guideline: Avoiding Sensitive Fields
# - Implement robust redaction or filtering mechanisms (like the SensitiveDataFilter above)
#   to prevent Personally Identifiable Information (PII), credentials, financial data, or other
#   sensitive details from entering log files. This is crucial for compliance (e.g., GDPR, HIPAA).

# Ethical Guideline: Secure Log Storage
# - Log files themselves contain valuable operational data and, even after redaction, might
#   contain semi-sensitive information. Store logs in secure, access-controlled locations.
# - Encrypt logs at rest and ensure secure transmission to central logging systems.

# Ethical Guideline: Log Retention Policies
# - Define and enforce strict log retention policies. Don't keep logs indefinitely. Delete them
#   after a defined period (e.g., 30, 90 days), or when they are no longer needed for business or compliance.

# Ethical Guideline: Access Control
# - Implement strict role-based access control (RBAC) to log management systems and log files.
#   Only authorized personnel should be able to view, search, or download logs.
```

```
# Ethical Guideline: Anonymization/Pseudonymization in Logs
# - For analytics or long-term storage, consider further anonymizing or pseudonymizing any remaining
#   identifiable information in logs. Use unique, non-reversible IDs where possible instead of direct identifiers.

print('\nLog entries have been generated and saved to web_app.log (and console). Please check the file for redacted sensitive information.'

WARNING:web_app:Failed login attempt for user: [REDACTED_NAME] ([REDACTED_EMAIL]) with [REDACTED_PASSWORD]
ERROR:web_app:Database connection failed: check credentials.

Log entries have been generated and saved to web_app.log (and console). Please check the file for redacted sensitive information.
```

Explanation : This script sets up logging for a web application with a custom filter that redacts sensitive data like emails, passwords, and names before writing to logs. It emphasizes ethical practices such as data minimization, avoiding sensitive fields, secure storage, retention policies, access control, and anonymization to ensure logs remain useful for debugging while protecting user privacy.

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.

Prompt :

S Write a Python machine learning model (e.g., using scikit-learn) that performs classification on sample data. Then, add inline documentation or a README-style section explaining how to use the model responsibly. Include notes on explainability, accuracy limits, fairness considerations, and ethical usage guidelines.

Code :

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# --- 1. Generate Sample Data for Classification ---
# We'll create a synthetic dataset for demonstration purposes.
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=0, random_state=42)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Dataset shape: X={X.shape}, y={y.shape}")
print(f"Training data shape: X_train={X_train.shape}, y_train={y_train.shape}")
print(f"Testing data shape: X_test={X_test.shape}, y_test={y_test.shape}")

# --- 2. Train a Classification Model (Logistic Regression) ---
# Logistic Regression is a simple, interpretable classification model.
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

print("\nModel training complete.")

# --- 3. Evaluate the Model ---
y_pred = model.predict(X_test)

print(f"\nAccuracy: {accuracy_score(y_test, y_pred):.2f}")
```

```python
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# --- 4. Model Interpretation (Explainability) ---
# For Logistic Regression, coefficients indicate feature importance.
# Positive coefficients increase the likelihood of the positive class, negative decrease it.
print("\nModel Coefficients (Feature Importance for Logistic Regression):")
for i, coef in enumerate(model.coef_[0]):
    print(f"Feature {i+1}: {coef:.4f}")
```

Output :

```
Dataset shape: X=(1000, 10), y=(1000,)
Training data shape: X_train=(800, 10), y_train=(800,)
Testing data shape: X_test=(200, 10), y_test=(200,)

Model training complete.

Accuracy: 0.77

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.74      0.77       104
           1       0.74      0.80      0.77        96

    accuracy                           0.77       200
   macro avg       0.77      0.77      0.77       200
weighted avg       0.77      0.77      0.77       200


Model Coefficients (Feature Importance for Logistic Regression):
Feature 1: -0.2465
Feature 2: -0.0563
Feature 3: 0.6509
Feature 4: -0.0201
Feature 5: -0.7532
Feature 6: -0.0028
Feature 7: 0.0567
Feature 8: 0.8904
Feature 9: -0.8367
Feature 10: 0.0584
```

Explanation : This script builds a simple logistic regression classifier on synthetic data, evaluates accuracy and classification metrics,

	and shows feature coefficients for explainability. The coefficients reveal which features push predictions toward positive or negative classes, making the model interpretable but limited in accuracy depending on data quality and balance.	