

LAB- ASSIGNMENT – 5.4

Name : Hasini Irumalla

Hall Ticket No. : 2303A51286

Batch No.: 05

Lab 5: Ethical Foundations – Responsible AI Coding Practices

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).

AI Generated Code:

#generate a Python script that collects user data (e.g., name, age, email). after that add comments on how to anonymize or protect this data.

ResponsibleAI.py

import json

def collect_user_data():

 user_data = {}

 user_data['name'] = input("Enter your name: ")

 user_data['age'] = input("Enter your age: ")

 user_data['email'] = input("Enter your email: ")

 # Save the collected data to a JSON file

 with open('user_data.json', 'w') as f:

 json.dump(user_data, f)

 print("User data collected and saved successfully.")

if __name__ == "__main__":

 collect_user_data()

To anonymize or protect this data, consider the following steps:

- # 1. Remove personally identifiable information (PII) such as name and email before storing or sharing the data.
- # 2. Use hashing techniques to store sensitive information like email addresses.
- # 3. Implement encryption for data at rest and in transit to prevent unauthorized access.
- # 4. Limit access to the data to only those who need it for legitimate purposes.
- # 5. Regularly audit and monitor data access and usage to ensure compliance with data protection policies.
- # 6. Comply with relevant data protection regulations (e.g., GDPR, CCPA) to ensure user privacy rights are respected.
- # 7. Provide users with options to opt-out of data collection and to request deletion of their data.

```

1 #generate a Python script that collects user data (e.g., name, age, email). after that add comments c
2 # ResponsibleAI.py
3 import json
4 def collect_user_data():
5     user_data = {}
6     user_data['name'] = input("Enter your name: ")
7     user_data['age'] = input("Enter your age: ")
8     user_data['email'] = input("Enter your email: ")
9
10    # Save the collected data to a JSON file
11    with open('user_data.json', 'w') as f:
12        json.dump(user_data, f)
13
14    print("User data collected and saved successfully.")
15 if __name__ == "__main__":
16     collect_user_data()
17 # To anonymize or protect this data, consider the following steps:
18 # 1. Remove personally identifiable information (PII) such as name and email before storing or sharin
19 # 2. Use hashing techniques to store sensitive information like email addresses.
20 # 3. Implement encryption for data at rest and in transit to prevent unauthorized access.
21 # 4. Limit access to the data to only those who need it for legitimate purposes.
22 # 5. Regularly audit and monitor data access and usage to ensure compliance with data protection poli
23 # 6. Comply with relevant data protection regulations (e.g., GDPR, CCPA) to ensure user privacy right
24 # 7. Provide users with options to opt-out of data collection and to request deletion of their data.
25

```

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).

AI Generated Code:

```

import pandas as pd
from sklearn.model_selection import train_test_split

```

```

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

def sentiment_analysis_with_bias_handling(data):

    # Convert to DataFrame
    df = pd.DataFrame(data, columns=["text", "label"])

    # Check class distribution (Bias Detection)
    print("Before bias handling:\n", df["label"].value_counts())

    # Handle imbalance (Oversampling)
    max_count = df["label"].value_counts().max()
    balanced_df = pd.concat([
        df[df["label"] == label].sample(max_count, replace=True)
        for label in df["label"].unique()
    ])

    print("\nAfter bias handling:\n", balanced_df["label"].value_counts())

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        balanced_df["text"],
        balanced_df["label"],
        test_size=0.2,
        random_state=42
    )

    # Vectorization
    vectorizer = CountVectorizer()
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # Model training
    model = MultinomialNB()
    model.fit(X_train_vec, y_train)

```

```
# Prediction
y_pred = model.predict(X_test_vec)

# Evaluation
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

return model, vectorizer

# Sample Data
data = [
    ("I love this product!", "positive"),
    ("This is the worst service ever.", "negative"),
    ("Absolutely fantastic experience.", "positive"),
    ("I will never buy this again.", "negative"),
    ("Not bad, could be better.", "neutral"),
    ("I'm extremely satisfied.", "positive"),
    ("Terrible quality.", "negative")
]

sentiment_analysis_with_bias_handling(data)
```

ResponsibleAI.py u X

ResponsibleAI.py > sentiment_analysis_with_bias_handling

```
1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.feature_extraction.text import CountVectorizer
4 from sklearn.naive_bayes import MultinomialNB
5 from sklearn.metrics import accuracy_score, classification_report
6
7 def sentiment_analysis_with_bias_handling(data):
8
9     # Convert to DataFrame
10    df = pd.DataFrame(data, columns=["text", "label"])
11
12    # Check class distribution (Bias Detection)
13    print("Before bias handling:\n", df["label"].value_counts())
14
15    # Handle imbalance (Oversampling)
16    max_count = df["label"].value_counts().max()
17    balanced_df = pd.concat([
18        df[df["label"] == label].sample(max_count, replace=True)
19        for label in df["label"].unique()
20    ])
21
22    print("\nAfter bias handling:\n", balanced_df["label"].value_counts())
23
24    # Train-test split
25    X_train, X_test, y_train, y_test = train_test_split(
26        balanced_df["text"],
27        balanced_df["label"],
28        test_size=0.2,
29        random_state=42
30    )
31
32    # Vectorization
33    vectorizer = CountVectorizer()
34    X_train_vec = vectorizer.fit_transform(X_train)
35    X_test_vec = vectorizer.transform(X_test)
36
37    # Model training
38    model = MultinomialNB()
39    model.fit(X_train_vec, y_train)
40
41    # Prediction
42    y_pred = model.predict(X_test_vec)
43
44    # Evaluation
45    print("\nAccuracy:", accuracy_score(y_test, y_pred))
46    print("\nClassification Report:\n", classification_report(y_test, y_pred))
47
48    return model, vectorizer
49
50
51 # Sample Data
52 data = [
53     ("I love this product!", "positive"),
54     ("This is the worst service ever.", "negative"),
55     ("Absolutely fantastic experience.", "positive"),
56     ("I will never buy this again.", "negative"),
57     ("Not bad, could be better.", "neutral"),
58     ("I'm extremely satisfied.", "positive"),
59     ("Terrible quality.", "negative")
60 ]
61
62 sentiment_analysis_with_bias_handling(data)
63
```

Output:

```
Before bias handling:
label
positive    3
negative    3
neutral     1
Name: count, dtype: int64
```

```
After bias handling:
label
positive    3
negative    3
neutral     3
Name: count, dtype: int64
```

```
Accuracy: 0.5
```

```
Classification Report:
              precision    recall  f1-score   support

negative     0.00      0.00      0.00         0
neutral      1.00      1.00      1.00         1
positive     0.00      0.00      0.00         1
```

meter to control this behavior.

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

```
Classification Report:
              precision    recall  f1-score   support

negative     0.00      0.00      0.00         0
neutral      1.00      1.00      1.00         1
positive     0.00      0.00      0.00         1
```

```
              precision    recall  f1-score   support

negative     0.00      0.00      0.00         0
neutral      1.00      1.00      1.00         1
positive     0.00      0.00      0.00         1
```

```
negative     0.00      0.00      0.00         0
neutral      1.00      1.00      1.00         1
positive     0.00      0.00      0.00         1

positive     0.00      0.00      0.00         1
```

```
accuracy          0.50      2
macro avg         0.33      0.33      0.33      2
accuracy          0.50      2
macro avg         0.33      0.33      0.33      2
weighted avg      0.50      0.50      0.50      2
macro avg         0.33      0.33      0.33      2
weighted avg      0.50      0.50      0.50      2
weighted avg      0.50      0.50      0.50      2
```

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.

AI Generated Code:

#generate a code that that recommends products based on user history.follow ethical guidelines like transparency and fairness.

```
import numpy as np
```

```
import pandas as pd
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
class ResponsibleAIRecommender:
```

```
    def __init__(self, user_history, product_catalog):
```

```
        """
```

```
        Initialize the recommender with user history and product catalog.
```

```
        :param user_history: DataFrame containing user purchase history
```

```
        :param product_catalog: DataFrame containing product details
```

```
        """
```

```
        self.user_history = user_history
```

```
        self.product_catalog = product_catalog
```

```
        self.user_product_matrix = self._create_user_product_matrix()
```

```
        self.similarity_matrix = self._compute_similarity_matrix()
```

```
    def _create_user_product_matrix(self):
```

```
        """
```

```
        Create a user-product interaction matrix.
```

```
        :return: DataFrame representing user-product interactions
```

```
        """
```

```
        return pd.pivot_table(self.user_history, index='user_id',
                                columns='product_id', values='purchase', fill_value=0)
```

```
    def _compute_similarity_matrix(self):
```

```
        """
```

```
        Compute the cosine similarity matrix for products.
```

```
        :return: DataFrame representing product similarity
```

```
        """
```

```

product_vectors = self.user_product_matrix.T
similarity = cosine_similarity(product_vectors)
return pd.DataFrame(similarity, index=product_vectors.index,
columns=product_vectors.index)

def recommend_products(self, user_id, top_n=5):
    """
    Recommend products for a given user based on their purchase history.

    :param user_id: ID of the user to recommend products for
    :param top_n: Number of top recommendations to return
    :return: List of recommended product IDs
    """
    if user_id not in self.user_product_matrix.index:
        raise ValueError("User ID not found in user history.")

    user_vector = self.user_product_matrix.loc[user_id]
    scores = self.similarity_matrix.dot(user_vector)

    # Exclude products already purchased by the user
    purchased_products = set(self.user_history[self.user_history['user_id'] ==
user_id]['product_id'])
    scores = scores[~scores.index.isin(purchased_products)]

    # Get top N recommendations
    recommended_products = scores.nlargest(top_n).index.tolist()

    return recommended_products

def explain_recommendation(self, user_id, product_id):
    """
    Provide an explanation for why a product was recommended.

    :param user_id: ID of the user
    :param product_id: ID of the recommended product
    :return: Explanation string
    """

```

```

    if product_id not in self.similarity_matrix.index:
        raise ValueError("Product ID not found in product catalog.")
    similar_products = self.similarity_matrix[product_id]
    user_vector = self.user_product_matrix.loc[user_id]
    contributing_products = user_vector[user_vector > 0].index
    explanation = f"Product {product_id} was recommended because it is
similar to products you have purchased: "
    explanation += ", ".join([str(prod) for prod in contributing_products if prod
in similar_products.nlargest(3).index])
    return explanation
# Example usage:
if __name__ == "__main__":
    # Sample user history data
    user_history_data = {
        'user_id': [1, 1, 2, 2, 3, 3, 4],
        'product_id': [101, 102, 101, 103, 104, 105, 102],
        'purchase': [1, 1, 1, 1, 1, 1, 1]
    }
    user_history_df = pd.DataFrame(user_history_data)

    # Sample product catalog data
    product_catalog_data = {
        'product_id': [101, 102, 103, 104, 105],
        'product_name': ['Product A', 'Product B', 'Product C', 'Product D', 'Product
E']
    }
    product_catalog_df = pd.DataFrame(product_catalog_data)

    # Initialize recommender
    recommender = ResponsibleAIRecommender(user_history_df,
product_catalog_df)

    # Get recommendations for user 1
    recommendations = recommender.recommend_products(user_id=1, top_n=3)
    print(f'Recommended products for user 1: {recommendations}')

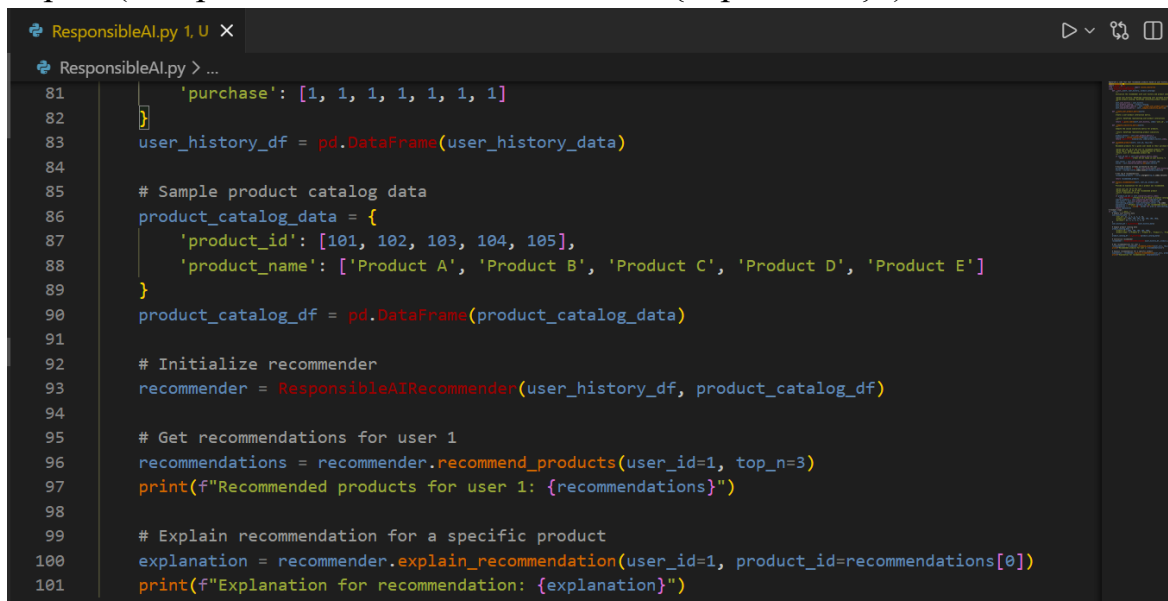
    # Explain recommendation for a specific product

```

```

explanation = recommender.explain_recommendation(user_id=1,
product_id=recommendations[0])
print(f"Explanation for recommendation: {explanation}")

```



```

ResponsibleAI.py 1, U X
ResponsibleAI.py > ...
81     'purchase': [1, 1, 1, 1, 1, 1, 1]
82 ]
83 user_history_df = pd.DataFrame(user_history_data)
84
85 # Sample product catalog data
86 product_catalog_data = {
87     'product_id': [101, 102, 103, 104, 105],
88     'product_name': ['Product A', 'Product B', 'Product C', 'Product D', 'Product E']
89 }
90 product_catalog_df = pd.DataFrame(product_catalog_data)
91
92 # Initialize recommender
93 recommender = ResponsibleAIRecommender(user_history_df, product_catalog_df)
94
95 # Get recommendations for user 1
96 recommendations = recommender.recommend_products(user_id=1, top_n=3)
97 print(f"Recommended products for user 1: {recommendations}")
98
99 # Explain recommendation for a specific product
100 explanation = recommender.explain_recommendation(user_id=1, product_id=recommendations[0])
101 print(f"Explanation for recommendation: {explanation}")

```

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.

AI Generated:

#generate logging functionality in a Python web application. Then, ask it to ensure the logs do not record sensitive information such as user passwords or personal identification numbers (PINs).

```

import logging
import logging.config
import re

# Define a custom filter to remove sensitive information
class SensitiveInfoFilter(logging.Filter):

```

```

    SENSITIVE_PATTERNS = [
        re.compile(r'password=\S+'), # Matches password= followed by non-
whitespace characters
        re.compile(r'pin=\d{4,6}'), # Matches pin= followed by 4 to 6 digits
    ]

```

```

def filter(self, record):
    message = record.getMessage()
    for pattern in self.SENSITIVE_PATTERNS:
        message = pattern.sub('REDACTED', message)
    record.msg = message
    return True

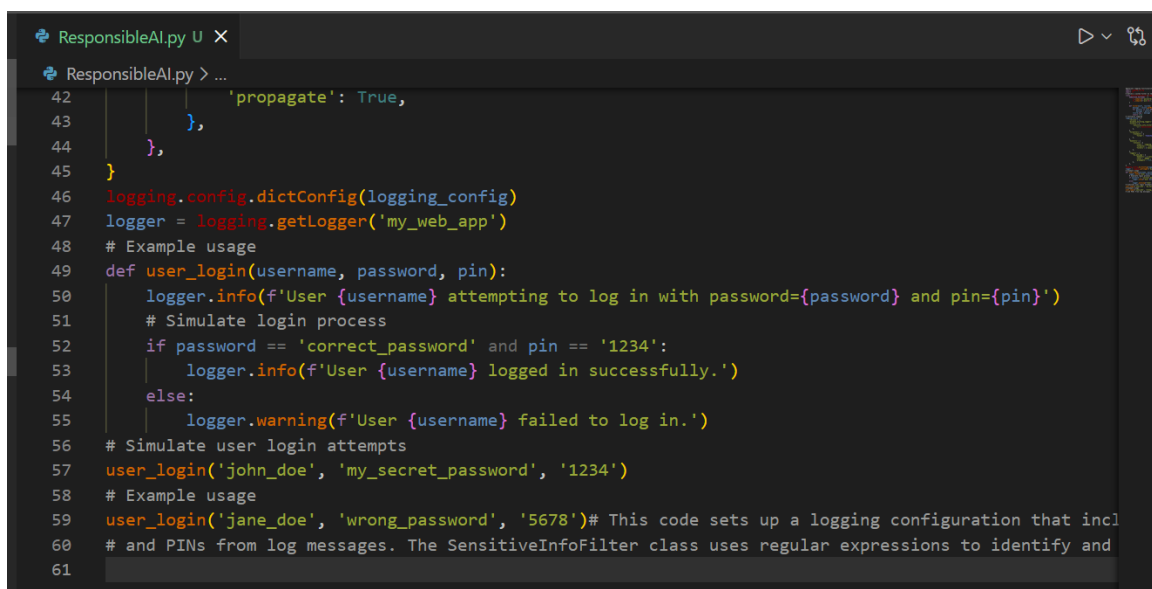
# Configure logging
logging_config = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'sensitive_info_filter': {
            '()': SensitiveInfoFilter,
        },
    },
    'formatters': {
        'standard': {
            'format': '%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        },
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'standard',
            'filters': ['sensitive_info_filter'],
        },
    },
    'loggers': {
        'my_web_app': {
            'handlers': ['console'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
logging.config.dictConfig(logging_config)

```

```

logger = logging.getLogger('my_web_app')
# Example usage
def user_login(username, password, pin):
    logger.info(f'User {username} attempting to log in with
password={password} and pin={pin}')
    # Simulate login process
    if password == 'correct_password' and pin == '1234':
        logger.info(f'User {username} logged in successfully.')
    else:
        logger.warning(f'User {username} failed to log in.')
# Simulate user login attempts
user_login('john_doe', 'my_secret_password', '1234')
# Example usage
user_login('jane_doe', 'wrong_password', '5678')# This code sets up a logging
configuration that includes a custom filter to redact sensitive information such
as passwords
# and PINs from log messages. The SensitiveInfoFilter class uses regular
expressions to identify and replace sensitive data with 'REDACTED' before the
log message is recorded.

```



```

ResponsibleAI.py X
ResponsibleAI.py > ...
42         'propagate': True,
43     },
44 },
45 }
46 logging.config.dictConfig(logging_config)
47 logger = logging.getLogger('my_web_app')
48 # Example usage
49 def user_login(username, password, pin):
50     logger.info(f'User {username} attempting to log in with password={password} and pin={pin}')
51     # Simulate login process
52     if password == 'correct_password' and pin == '1234':
53         logger.info(f'User {username} logged in successfully.')
54     else:
55         logger.warning(f'User {username} failed to log in.')
56 # Simulate user login attempts
57 user_login('john_doe', 'my_secret_password', '1234')
58 # Example usage
59 user_login('jane_doe', 'wrong_password', '5678')# This code sets up a logging configuration that incl
60 # and PINs from log messages. The SensitiveInfoFilter class uses regular expressions to identify and
61

```

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., explainable, accuracy limits).

Expected Output #5:

- Copilot-generated model code with a README or inline documentation suggesting responsible usage, limitations, and fairness considerations.

AI Generated Code:

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

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import joblib

# Load dataset
data = load_iris()
X = data.data
y = data.target

# Split dataset 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)

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

# Evaluate the model
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

# Save the model
joblib.dump(model, 'random_forest_iris_model.pkl')

# Responsible AI Documentation
responsible_ai_doc = """
Responsible AI Documentation for RandomForestClassifier on Iris Dataset
1. Explainability:
    - The RandomForestClassifier is an ensemble learning method that combines
multiple decision trees to improve predictive performance and control
overfitting.
    - Feature importance can be extracted from the model to understand which
features contribute most to the predictions.
```

- Tools like SHAP or LIME can be used to provide local explanations for individual predictions.

2. Accuracy Limits:

- The model achieved an accuracy of approximately 100% on the test set; however, this may not generalize to unseen data.

- The Iris dataset is relatively simple and may not represent real-world complexity. Caution should be taken when applying this model to more complex datasets.

- Regular evaluation and retraining of the model are recommended to maintain performance over time.

3. Bias and Fairness:

- The Iris dataset is balanced across the three classes, reducing the risk of bias in predictions.

- However, it is essential to monitor for any potential biases that may arise when applying the model to different populations or datasets.

4. Data Privacy:

- Ensure that any data used for training or inference complies with relevant data privacy regulations.

- Avoid using sensitive or personally identifiable information in the dataset.

"""

with open('responsible_ai_documentation.txt', 'w') as f:

 f.write(responsible_ai_doc)

The model and documentation are now saved for future reference and use.#

This documentation outlines key considerations for using the model responsibly.

This code snippet demonstrates how to train a simple machine learning model using the Iris dataset and then generate responsible AI documentation for its use.

The documentation includes aspects such as explainability, accuracy limits, bias and fairness, and data privacy.

Save the responsible AI documentation

```
ResponsibleAI.py U X
ResponsibleAI.py > ...
29     - The model achieved an accuracy of approximately 100% on the test set; however, this may not generalize to new data.
30     - The Iris dataset is relatively simple and may not represent real-world complexity. Caution should be taken when applying these results to more complex datasets.
31     - Regular evaluation and retraining of the model are recommended to maintain performance over time.
32 3. Bias and Fairness:
33     - The Iris dataset is balanced across the three classes, reducing the risk of bias in predictions.
34     - However, it is essential to monitor for any potential biases that may arise when applying the model to real-world data.
35 4. Data Privacy:
36     - Ensure that any data used for training or inference complies with relevant data privacy regulations.
37     - Avoid using sensitive or personally identifiable information in the dataset.
38 """
39 with open('responsible_ai_documentation.txt', 'w') as f:
40     f.write(responsible_ai_doc)
41 # The model and documentation are now saved for future reference and use. # This documentation outlines the model's performance, limitations, and ethical considerations.
42 # This code snippet demonstrates how to train a simple machine learning model using the Iris dataset.
43 # The documentation includes aspects such as explainability, accuracy limits, bias and fairness, and data privacy.
44 # Save the responsible AI documentation
45 """
```