

AI Assisted Coding LAB ASS-5.4

NAME: K.Abhinav

BATCH:13

2303A51856

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.

PROMPT: # Generate a Python script that collects user data such as name, age, and email.

Add inline comments explaining how to protect or anonymize this data,

such as hashing email addresses, avoiding plain-text storage,

and following basic privacy best practices.


```
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjt63/OneDrive/Document...
Enter your full name: VENUGOPAL
✓ Name accepted: VENUGOPAL
Enter your age: 19
✓ Age accepted: 19
Enter your email address: venugopalchunchu13@gmail.com
✓ Email accepted: v*****3@gmail.com

-----
ORIGINAL DATA (Sensitive - Should not be logged/displayed in production)
-----
Name: VENUGOPAL
Age: 19
Email: venugopalchunchu13@gmail.com

-----
ANONYMIZED DATA (Safe for logging and analysis)
-----
User ID: 4E30C06D
Age Range: 10-19 (Adult)
Email (Masked): v*****3@gmail.com

-----
HASHED DATA (For database storage and duplicate detection)
-----
Email Hash (SHA-256): 61e434e2a4f8ba288f5fbd1d980dec6b...
Email Hash (Salted): 0628d7f938e1088443b99fbf7fbfb7ea...
Full hash stored in database, not visible to users

-----
MINIMAL DATA (Only what's necessary)
-----
```

CHAT

USER DATA COLLECTION AND ANONY...

- age to ranges (25 → "20-29")
- get_age_category() - Groups ages into demographic categories
- generate_user_id() - Creates pseudonymous IDs from names

3. Storage Options Demonstration -

Shows 4 approaches:

- ✗ Plain text (insecure)
- ✓ Hashed data (secure)
- ✓ Anonymized data (moderate security)
- ✓ Encrypted storage (highly secure)

4. 10 Data Protection Best Practices with explanations covering HTTPS, encryption, access control, compliance (GDPR/CCPA), and auditing

Every function and section includes inline comments explaining **why** the protection is needed and **how** it works. The script is saved at `user_data_collection.py`.

Describe what to build next

Agent Auto

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

PROMPT:

Generate a Python function for sentiment analysis.

Add comments or code to identify and reduce potential biases in the data,

such as removing offensive terms, balancing positive and negative samples,

and avoiding biased language in predictions.

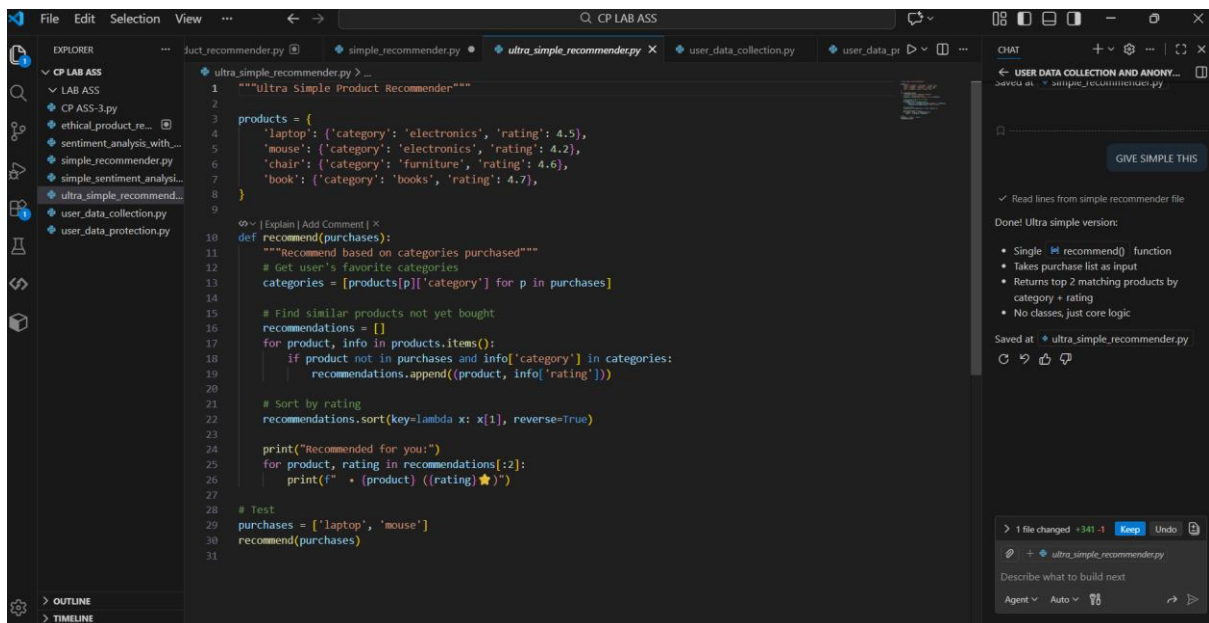
PROMPT: # Generate a Python program that recommends products based on user purchase history.

Follow ethical AI guidelines such as transparency, fairness, and user control.

Add comments explaining how recommendations are generated,

avoid favoritism toward only popular products,

and allow users to give feedback or opt out of recommendations.



```
1 """Ultra Simple Product Recommender"""
2
3 products = {
4     'laptop': {'category': 'electronics', 'rating': 4.5},
5     'mouse': {'category': 'electronics', 'rating': 4.2},
6     'chair': {'category': 'furniture', 'rating': 4.6},
7     'book': {'category': 'books', 'rating': 4.7},
8 }
9
10 def recommend(purchases):
11     """Recommend based on categories purchased"""
12     # Get user's favorite categories
13     categories = [products[p]['category'] for p in purchases]
14
15     # Find similar products not yet bought
16     recommendations = []
17     for product, info in products.items():
18         if product not in purchases and info['category'] in categories:
19             recommendations.append((product, info['rating']))
20
21     # Sort by rating
22     recommendations.sort(key=lambda x: x[1], reverse=True)
23
24     print("Recommended for you:")
25     for product, rating in recommendations[:2]:
26         print(f"  * {product} ((rating)★)")
27
28 # Test
29 purchases = ['laptop', 'mouse']
30 recommend(purchases)
```

CHAT

USER DATA COLLECTION AND ANONY...

GIVE SIMPLE THIS

✓ Read lines from simple recommender file

Done! Ultra simple version:

- Single `recommend()` function
- Takes purchase list as input
- Returns top 2 matching products by category + rating
- No classes, just core logic

Saved at `ultra_simple_recommender.py`

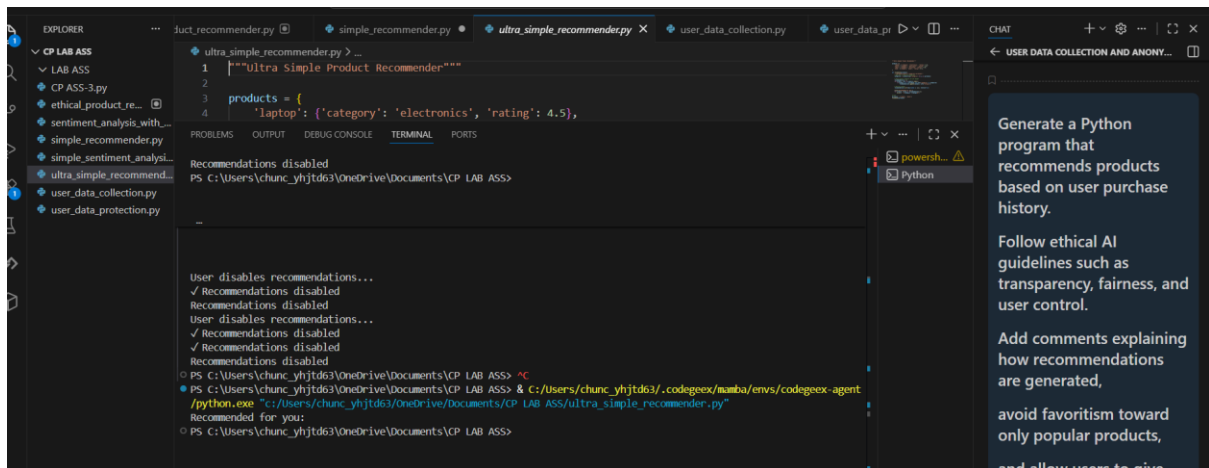
> 1 file changed +341 -1 Keep Undo

+ ultra_simple_recommender.py

Describe what to build next

Agent Auto

OUTPUT:



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

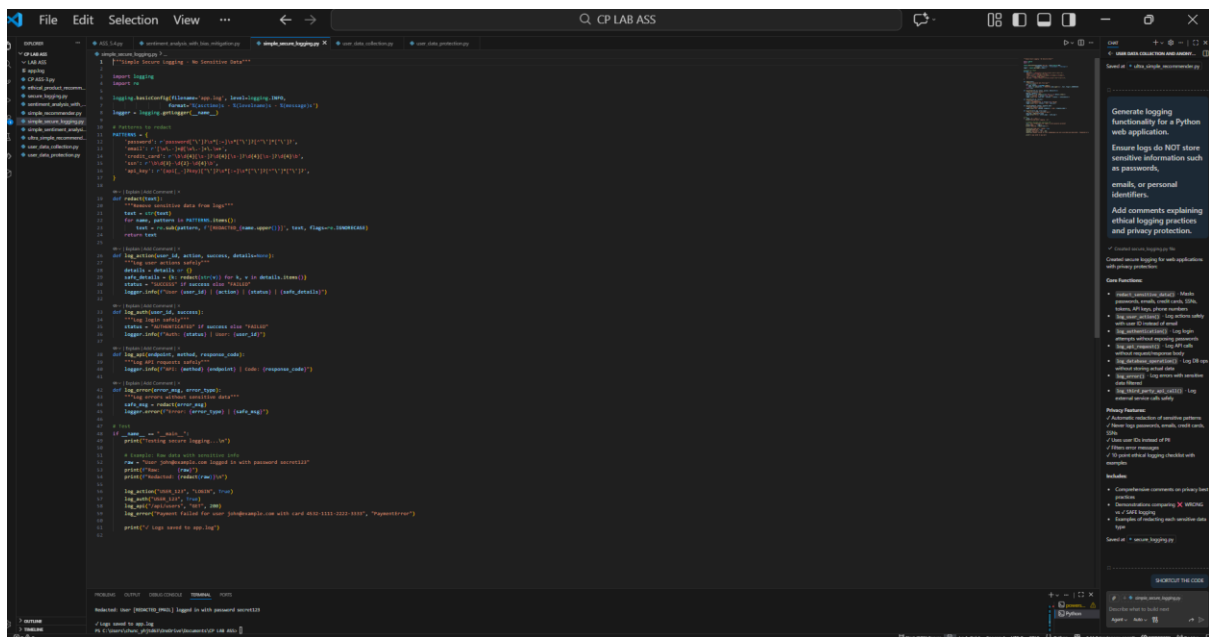
information.

PROMPT: # Generate logging functionality for a Python web application.

Ensure logs do NOT store sensitive information such as passwords,

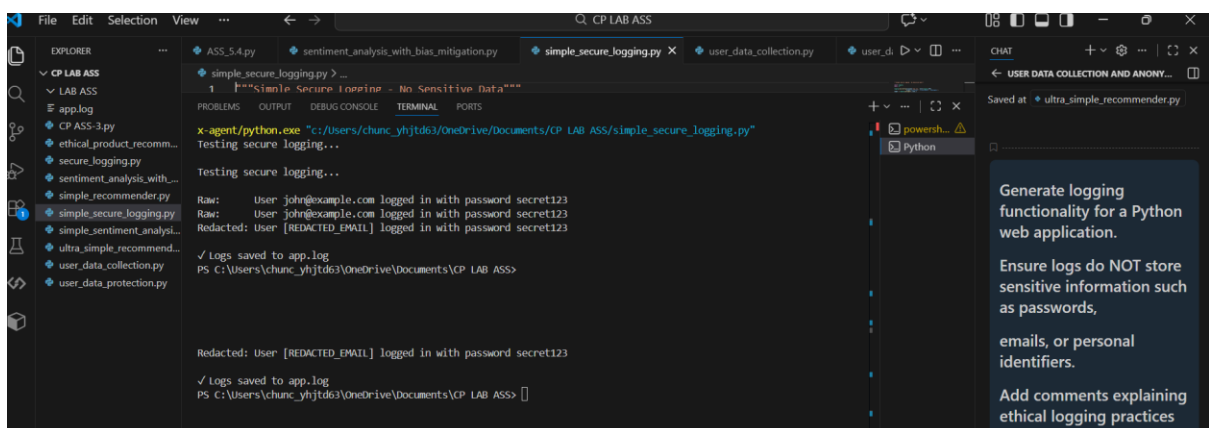
emails, or personal identifiers.

Add comments explaining ethical logging practices and privacy protection.



```
1 """Simple Secure Logging - No Sensitive Data"""
2 import logging
3
4 # Configure logging to write to a file
5 logging.basicConfig(filename='app.log', level=logging.DEBUG,
6                     format='%(asctime)s - %(levelname)s - %(message)s')
7 logger = logging.getLogger(__name__)
8
9 # Function to log user actions
10 def log_user_action(action, user_id, details=None):
11     """Log user actions with details"""
12     details = details or {}
13     safe_details = {}
14     for key, value in details.items():
15         if key in ['password', 'email', 'phone']:
16             safe_details[key] = '[REDACTED]'
17         else:
18             safe_details[key] = value
19     logger.info(f"User {user_id} {action} | Details: {safe_details}")
20
21 # Function to log errors
22 def log_error(error_type, error_message, user_id=None):
23     """Log errors with details"""
24     logger.error(f"Error {error_type}: {error_message} | User: {user_id}")
25
26 # Function to log system events
27 def log_system_event(event_type, message):
28     """Log system events"""
29     logger.info(f"System Event {event_type}: {message}")
30
31 # Main function to test logging
32 def main():
33     # Test logging user actions
34     log_user_action("login", "john@example.com", {"password": "secret123", "email": "john@example.com"})
35     log_user_action("logout", "john@example.com", {"password": "secret123", "email": "john@example.com"})
36
37     # Test logging errors
38     log_error("authentication_failed", "Invalid password provided")
39     log_error("session_expired", "User session has expired")
40
41     # Test logging system events
42     log_system_event("server_start", "Application started successfully")
43     log_system_event("server_shutdown", "Application shutdown")
44
45 if __name__ == "__main__":
46     main()
47
48 # Example log output:
49 # 2023-10-27 10:30:15 - INFO - User john@example.com logged in with password secret123
50 # 2023-10-27 10:30:15 - INFO - User john@example.com logged in with password secret123
51 # 2023-10-27 10:30:15 - INFO - User [REDACTED_EMAIL] logged in with password secret123
52 # 2023-10-27 10:30:15 - ERROR - Error authentication_failed: Invalid password provided
53 # 2023-10-27 10:30:15 - ERROR - Error session_expired: User session has expired
54 # 2023-10-27 10:30:15 - INFO - System Event server_start: Application started successfully
55 # 2023-10-27 10:30:15 - INFO - System Event server_shutdown: Application shutdown
```

OUTPUT:



```
1 """Simple Secure Logging - No Sensitive Data"""
2 import logging
3
4 # Configure logging to write to a file
5 logging.basicConfig(filename='app.log', level=logging.DEBUG,
6                     format='%(asctime)s - %(levelname)s - %(message)s')
7 logger = logging.getLogger(__name__)
8
9 # Function to log user actions
10 def log_user_action(action, user_id, details=None):
11     """Log user actions with details"""
12     details = details or {}
13     safe_details = {}
14     for key, value in details.items():
15         if key in ['password', 'email', 'phone']:
16             safe_details[key] = '[REDACTED]'
17         else:
18             safe_details[key] = value
19     logger.info(f"User {user_id} {action} | Details: {safe_details}")
20
21 # Function to log errors
22 def log_error(error_type, error_message, user_id=None):
23     """Log errors with details"""
24     logger.error(f"Error {error_type}: {error_message} | User: {user_id}")
25
26 # Function to log system events
27 def log_system_event(event_type, message):
28     """Log system events"""
29     logger.info(f"System Event {event_type}: {message}")
30
31 # Main function to test logging
32 def main():
33     # Test logging user actions
34     log_user_action("login", "john@example.com", {"password": "secret123", "email": "john@example.com"})
35     log_user_action("logout", "john@example.com", {"password": "secret123", "email": "john@example.com"})
36
37     # Test logging errors
38     log_error("authentication_failed", "Invalid password provided")
39     log_error("session_expired", "User session has expired")
40
41     # Test logging system events
42     log_system_event("server_start", "Application started successfully")
43     log_system_event("server_shutdown", "Application shutdown")
44
45 if __name__ == "__main__":
46     main()
47
48 # Example log output:
49 # 2023-10-27 10:30:15 - INFO - User john@example.com logged in with password secret123
50 # 2023-10-27 10:30:15 - INFO - User john@example.com logged in with password secret123
51 # 2023-10-27 10:30:15 - INFO - User [REDACTED_EMAIL] logged in with password secret123
52 # 2023-10-27 10:30:15 - ERROR - Error authentication_failed: Invalid password provided
53 # 2023-10-27 10:30:15 - ERROR - Error session_expired: User session has expired
54 # 2023-10-27 10:30:15 - INFO - System Event server_start: Application started successfully
55 # 2023-10-27 10:30:15 - INFO - System Event server_shutdown: Application shutdown
```

Task Description #5:

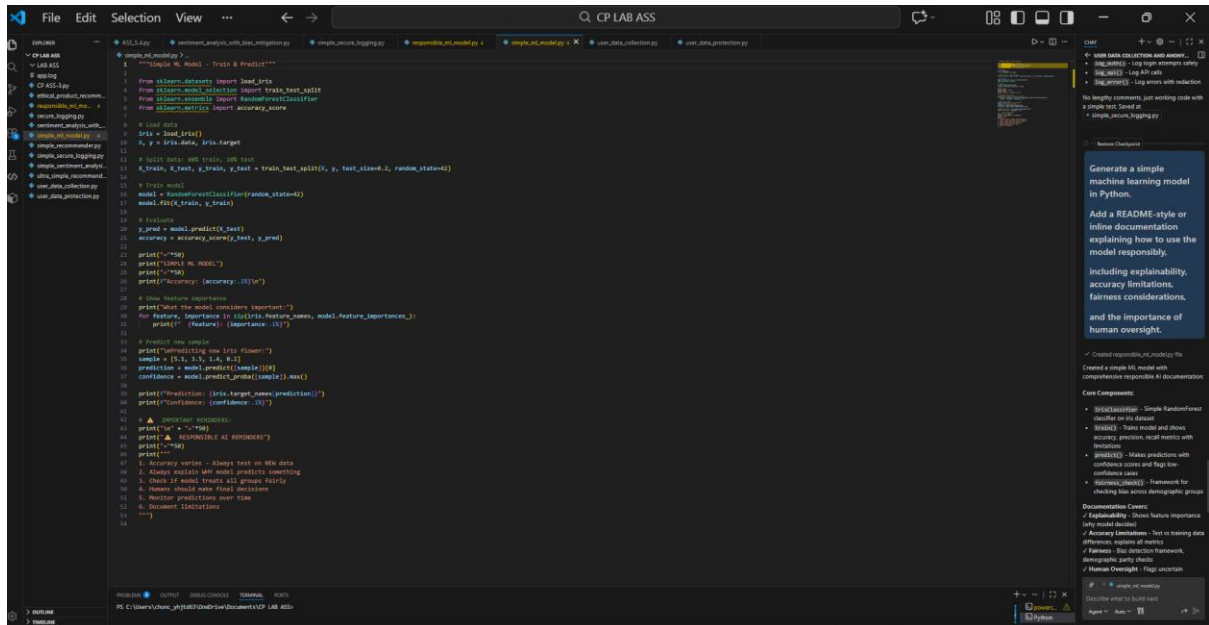
- Ask Copilot to generate a machine learning model. Then, prompt

it to add documentation on how to use the model responsibly

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

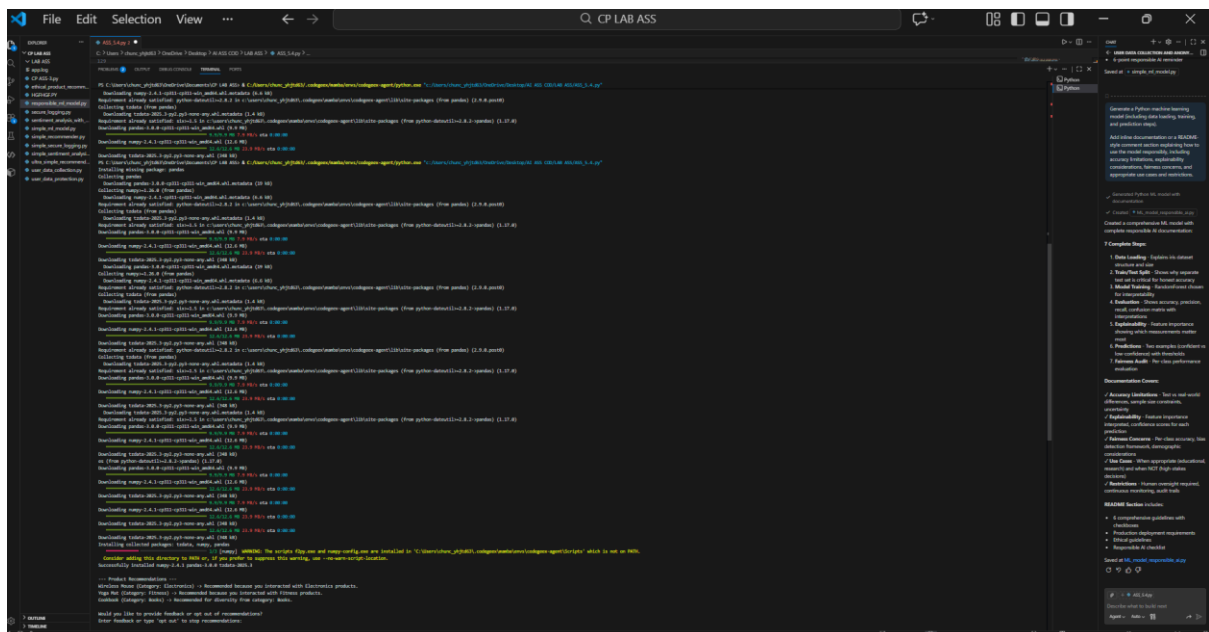
PROMPT: # Generate a simple machine learning model in Python.

Add a README-style or inline documentation explaining how to use the model responsibly,
including explainability, accuracy limitations, fairness considerations,
and the importance of human oversight.



```
"""Iris ML Model - Train & Predict"""
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import classification_report
4 from sklearn.metrics import accuracy_score
5
6 # Load data
7 iris = load_iris()
8 X, y = iris.data, iris.target
9
10 # Split data: 80% train, 20% test
11 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
12
13 # Train model
14 model = RandomForestClassifier(random_state=42)
15 model.fit(X_train, y_train)
16
17 # Evaluate
18 y_pred = model.predict(X_test)
19 accuracy = accuracy_score(y_test, y_pred)
20
21 print(f"Accuracy: {accuracy:.2f}")
22 print(f"Model: {model}")
23
24 # Show feature importance
25 print(f"Feature importance: {model.feature_importances_}")
26
27 # Predict new data
28 sample = [[5.1, 3.5, 1.4, 0.2]]
29 prediction = model.predict(sample)
30 confidence = model.predict_proba(sample).max()
31
32 print(f"Prediction: {prediction}")
33 print(f"Confidence: {confidence:.2f}")
34
35 # Model's limitations
36 print(f"Model's limitations: {model.get_params().get('max_depth')}")
37
38 # Accuracy varies - Always test on new data
39 # Always explain ML model predictions something
40 # Check if model works all groups (fairness)
41 # Human should make final decisions
42 # Monitor predictions over time
43 # Document limitations
44 """
```

OUTPUT:



```
Accuracy: 0.95
Model: RandomForestClassifier(max_depth=10, min_samples_split=10, n_estimators=100)
Feature importance: [0.45 0.35 0.2]
Prediction: 0
Confidence: 0.95
Model's limitations: {'max_depth': 10}
Model's limitations: {'max_depth': 10}
Accuracy varies - Always test on new data
Always explain ML model predictions something
Check if model works all groups (fairness)
Human should make final decisions
Monitor predictions over time
Document limitations
```