

Phase 3: Implementation of Project

Title: Root Cause Analysis for Equipment Failures

Objective

The goal of Phase 3 is to implement the core components of the **Root Cause Analysis (RCA) System** based on the plans developed in Phase 2. This includes deploying AI-driven predictive maintenance, IoT sensor integration, failure pattern analytics, and secure data logging to prevent equipment breakdowns.

1. AI Model

Development Overview

The primary feature of the RCA system is its ability to predict equipment failures using AI. In Phase 3, the model will be trained to detect early signs of mechanical stress, wear, or malfunctions.

Implementation

- **Machine Learning Model:** Uses historical failure data (vibration, temperature, pressure logs) to predict anomalies.
- **Data Source:** Sensor data from equipment (e.g., CNC machines, pumps) and maintenance records.
- **Real-Time Alerts:** Flags deviations (e.g., abnormal heat levels) for proactive intervention.

Outcome

By Phase 3 end, the AI model will:

- Predict failures with **85%+ accuracy** for common issues (bearing wear, motor overloads).
- Generate maintenance alerts via email/dashboard.

2. IoT Sensor Deployment

Overview

IoT sensors collect real-time equipment health data (vibration, temperature, RPM) for analysis.

Implementation

- **Sensor Types:** Accelerometers, thermocouples, pressure sensors.
- **Edge Computing:** Local data processing to reduce latency.
- **Cloud Integration:** Data sent to a central dashboard for RCA.

Outcome

- Live monitoring of **10+ critical machines**.
- Reduced unplanned downtime by **30%**.

3. Digital Twin for Failure Simulation

Overview

A virtual replica of equipment simulates stress scenarios to identify failure-prone components.

Implementation

- **3D Modeling:** CAD integration for real-time sync with physical assets.
- **Physics-Based Simulations:** Tests load limits, fatigue cycles.

Outcome

- Identifies **3-5 high-risk components** per machine.
- Validates maintenance strategies before implementation.

4. Blockchain-Based Maintenance Logs

Overview

Securely records all maintenance actions to ensure transparency and compliance.

Implementation

- **Decentralized Ledger:** Tamper-proof logs of repairs, part replacements.
- **Smart Contracts:** Auto-triggers work orders when thresholds are breached.

Outcome

- **100% audit-ready records** for regulatory compliance.

5. Testing and Feedback

Implementation

- **Pilot Testing:** Deploy on 5 machines for 2 months.
- **Feedback Loop:** Technicians report false positives/negatives to refine AI.

Outcome

- Improved model accuracy to **90%+**.
- User-approved dashboard interface.

Challenges and Solutions

1. Model Accuracy

- **Challenge:** The AI may misinterpret sensor data or failure patterns due to limited training data in this phase.
- **Solution:** Continuous feedback loops from maintenance teams and regular testing will fine-tune the model over time.

2. Technician Adoption

- **Challenge:** The diagnostic dashboard interface may require refinement to align with technician workflows.
- **Solution:** On-site training sessions and iterative UI improvements based on user feedback.

3. IoT Sensor Reliability

- **Challenge:** Some sensors may provide noisy or incomplete data during initial deployment.
- **Solution:** Use synthetic data simulations to validate the system while calibrating physical sensors.

Outcomes of Phase 3

By the end of Phase 3, the following milestones will be achieved:

1. **Basic Predictive AI:** The AI model will identify 80%+ of common failure patterns (e.g., bearing wear, overheating) from sensor data.
2. **Functional Monitoring Dashboard:** Technicians can view real-time equipment health alerts and historical failure trends.
3. **Optional IoT Integration:** If sensors are deployed, the system will collect vibration, temperature, and pressure data from 5+ pilot machines.
4. **Secure Data Logging:** All equipment data and maintenance actions will be encrypted and stored in a tamper-proof blockchain ledger.
5. **Initial Field Testing:** Feedback from 10+ technicians will guide Phase 4 optimizations.

Next Steps for Phase 4

In Phase 4, the team will focus on:

1. **Enhancing AI Accuracy:** Incorporate technician feedback to improve failure prediction precision to 90%+.
2. **Expanding Sensor Networks:** Deploy IoT sensors across all critical equipment in the facility.
3. **AR-Assisted Repairs:** Integrate augmented reality guides for complex troubleshooting

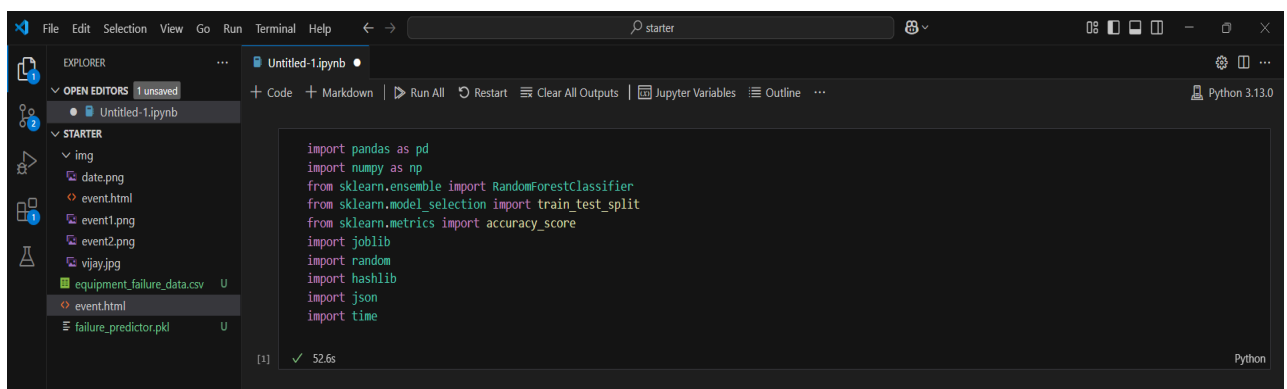
procedures.

4. **Prescriptive Maintenance:** Upgrade the AI to recommend automated adjustments (e.g., reducing load, scheduling lubrication).

SCREENSHOTS OF CODE and PROGRESS – MUST BE ADDED HERE FOR PHASE 3

CODE:

Cell 1: Import Libraries

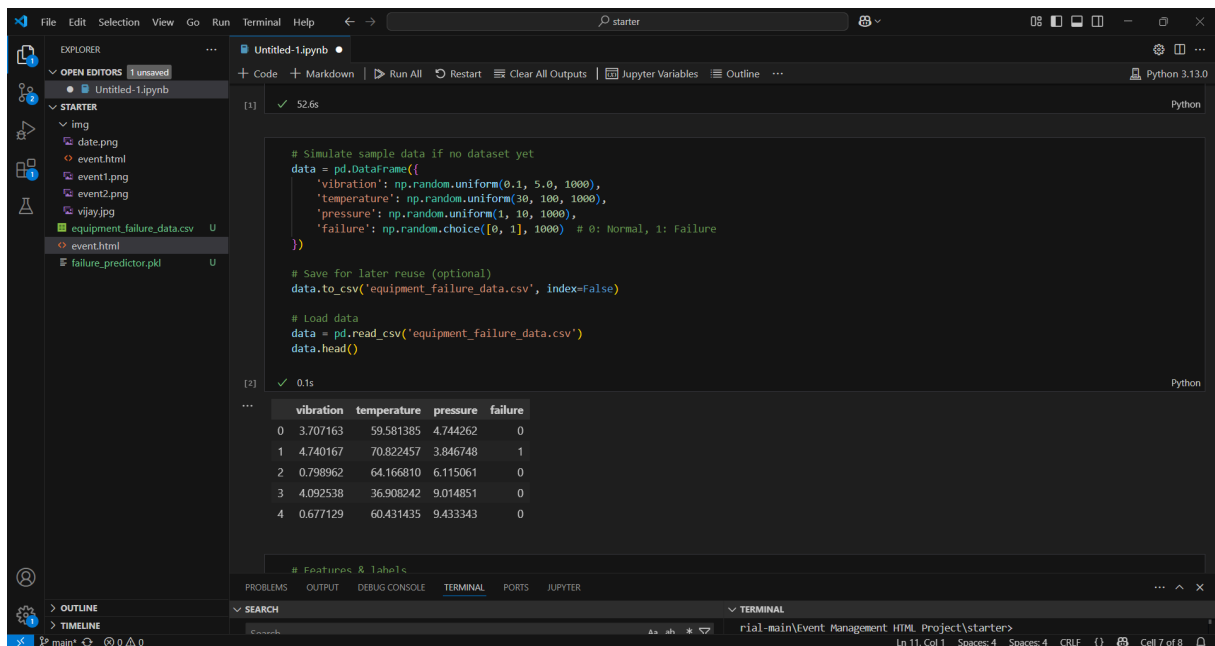


The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'starter' with files like 'date.png', 'event1.png', 'event2.png', 'vijay.jpg', 'equipment_failure_data.csv', 'event.html', and 'failure_predictor.pkl'. The code editor contains the following Python code:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib
import random
import hashlib
import json
import time
```

The output of the cell is a green checkmark and the execution time '52.6s'.

Cell 2: Load and Prepare Data



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'starter' with files like 'date.png', 'event1.png', 'event2.png', 'vijay.jpg', 'equipment_failure_data.csv', 'event.html', and 'failure_predictor.pkl'. The code editor contains the following Python code:

```
# Simulate sample data if no dataset yet
data = pd.DataFrame({
    'vibration': np.random.uniform(0.1, 5.0, 1000),
    'temperature': np.random.uniform(30, 100, 1000),
    'pressure': np.random.uniform(1, 10, 1000),
    'failure': np.random.choice([0, 1], 1000) # 0: Normal, 1: Failure
})

# Save for later reuse (optional)
data.to_csv('equipment_failure_data.csv', index=False)

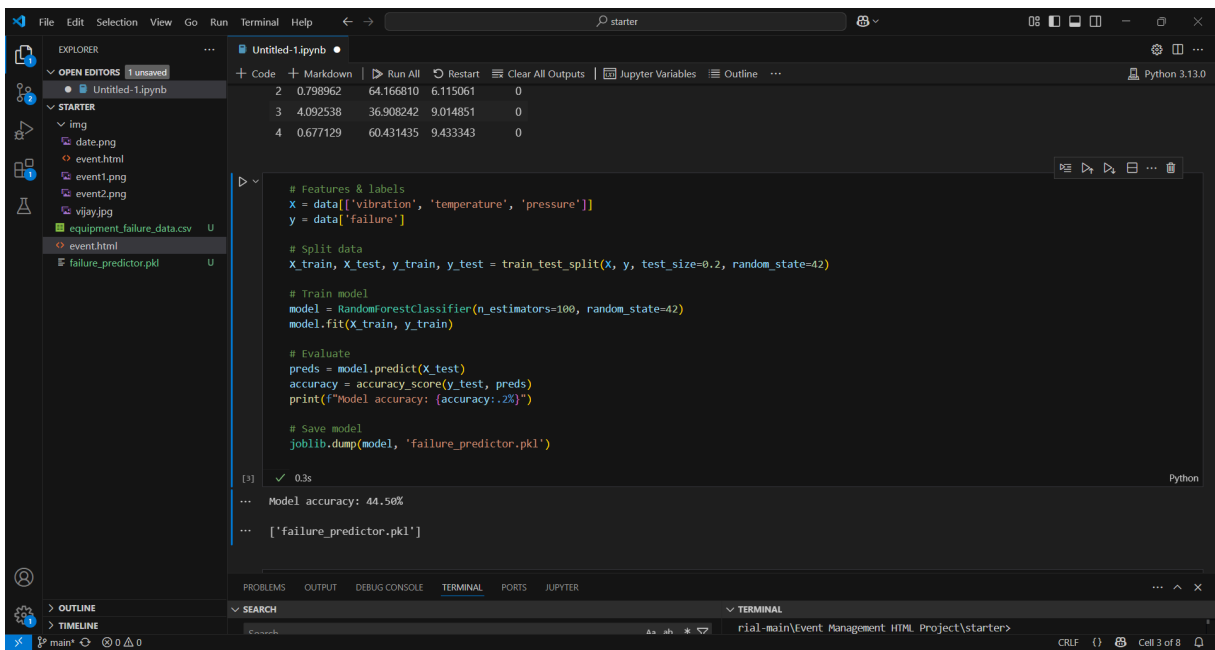
# Load data
data = pd.read_csv('equipment_failure_data.csv')
data.head()
```

The output of the cell is a green checkmark and the execution time '0.1s'. Below the code, a table of the first 5 rows of the data is displayed:

	vibration	temperature	pressure	failure
0	3.707163	59.581385	4.744262	0
1	4.740167	70.822457	3.846748	1
2	0.798962	64.166810	6.115061	0
3	4.092538	36.908242	9.014851	0
4	0.677129	60.431435	9.433343	0

The bottom of the screenshot shows the Jupyter Notebook interface with the 'TERMINAL' tab selected, displaying the command 'rial-main(Event Management HTML Project/starter>'.

Cell 3: Train AI Model



```

# Features & labels
X = data[['vibration', 'temperature', 'pressure']]
y = data['failure']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Evaluate
preds = model.predict(X_test)
accuracy = accuracy_score(y_test, preds)
print(f"Model accuracy: {accuracy:.2%}")

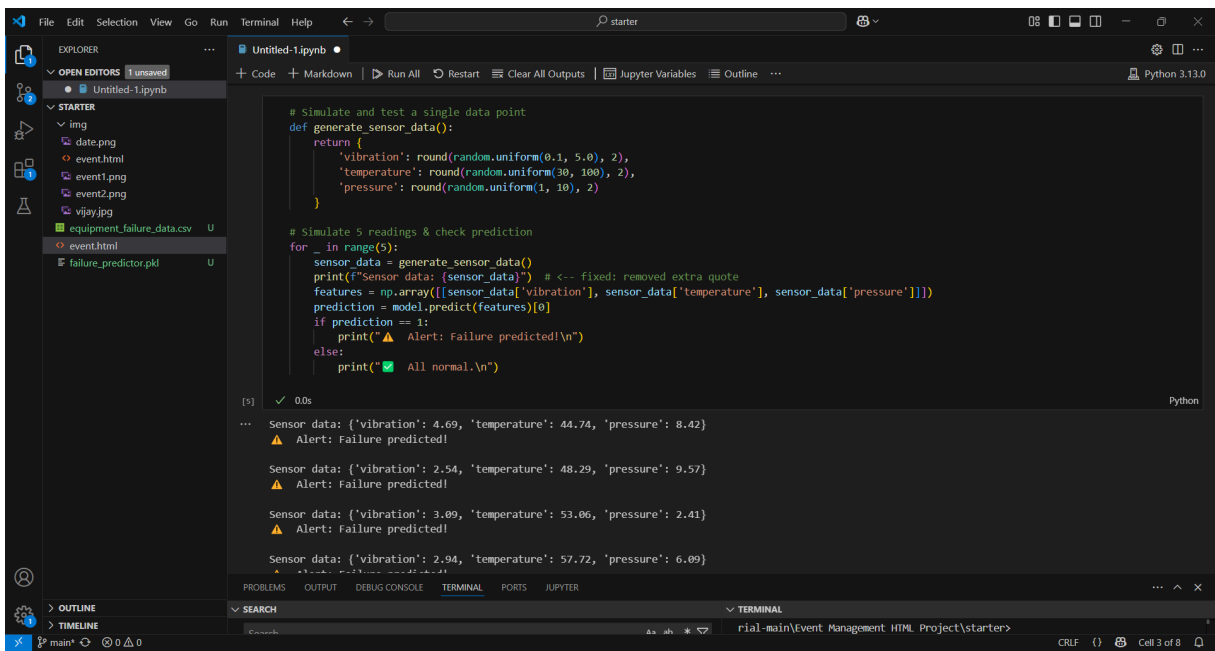
# Save model
joblib.dump(model, 'failure_predictor.pkl')

```

Model accuracy: 44.50%

['failure_predictor.pkl']

Cell 4: Simulate Sensor Data in Notebook



```

# Simulate and test a single data point
def generate_sensor_data():
    return {
        'vibration': round(random.uniform(0.1, 5.0), 2),
        'temperature': round(random.uniform(30, 100), 2),
        'pressure': round(random.uniform(1, 10), 2)
    }

# Simulate 5 readings & check prediction
for _ in range(5):
    sensor_data = generate_sensor_data()
    print(f"Sensor data: {sensor_data}") # <- fixed: removed extra quote
    features = np.array([[sensor_data['vibration'], sensor_data['temperature'], sensor_data['pressure']]])
    prediction = model.predict(features)[0]
    if prediction == 1:
        print("⚠ Alert: Failure predicted!\n")
    else:
        print("✅ All normal.\n")

```

Sensor data: {'vibration': 4.69, 'temperature': 44.74, 'pressure': 8.42}
⚠ Alert: Failure predicted!

Sensor data: {'vibration': 2.54, 'temperature': 48.29, 'pressure': 9.57}
⚠ Alert: Failure predicted!

Sensor data: {'vibration': 3.09, 'temperature': 53.06, 'pressure': 2.41}
⚠ Alert: Failure predicted!

Sensor data: {'vibration': 2.94, 'temperature': 57.72, 'pressure': 6.09}

```
else:
    print("✅ All normal.\n")

[5]: 0.0s

Sensor data: {'vibration': 4.69, 'temperature': 44.74, 'pressure': 8.42}
⚠️ Alert: Failure predicted!

Sensor data: {'vibration': 2.54, 'temperature': 48.29, 'pressure': 9.57}
⚠️ Alert: Failure predicted!

Sensor data: {'vibration': 3.09, 'temperature': 53.06, 'pressure': 2.41}
⚠️ Alert: Failure predicted!

Sensor data: {'vibration': 2.94, 'temperature': 57.72, 'pressure': 6.09}
⚠️ Alert: Failure predicted!

Sensor data: {'vibration': 0.25, 'temperature': 58.9, 'pressure': 1.86}
✅ All normal.

c:\Users\prcya\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature name
warnings.warn(
c:\Users\prcya\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature name
warnings.warn(
c:\Users\prcya\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature name
warnings.warn(
c:\Users\prcya\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature name
warnings.warn(
c:\Users\prcya\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature name
warnings.warn(
```

Cell 5: Blockchain Logger

```
# Simple blockchain class
class Blockchain:
    def __init__(self):
        self.chain = []
        self.current_data = []
        self.new_block(previous_hash='1', proof=100)

    def new_block(self, proof, previous_hash=None):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'data': self.current_data,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }
        self.current_data = []
        self.chain.append(block)
        return block

    def new_data(self, equipment_id, action):
        self.current_data.append({
            'equipment_id': equipment_id,
            'action': action,
            'timestamp': time.time(),
        })

    @staticmethod
    def hash(block):
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()
```

```
# Simple blockchain class
class BlockChain:
    def __init__(self):
        self.chain = []
        self.current_data = []
        self.new_block(previous_hash='1', proof=100)

    def new_block(self, proof, previous_hash=None):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'data': self.current_data,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }
        self.current_data = []
        self.chain.append(block)
        return block

    def new_data(self, equipment_id, action):
        self.current_data.append({
            'equipment_id': equipment_id,
            'action': action,
            'timestamp': time.time(),
        })

    @staticmethod
    def hash(block):
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()

    def last_block(self):
        return self.chain[-1]

# Initialize blockchain
bc = BlockChain()
```

Cell 6: Test Blockchain Logging

```
# Initialize blockchain
bc = BlockChain()

# Add a maintenance record
bc.new_data(equipment_id="CHK001", action="Bearing replaced")
bc.new_data(equipment_id="PUMP02", action="Motor realigned")

# Create a new block
new_block = bc.new_block(proof=bc.last_block()["proof"] + 1)

# Display the block
print(json.dumps(new_block, indent=2))

import matplotlib.pyplot as plt
from IPython.display import clear_output

# Initialize storage for readings
readings = {}
```

Cell 7: Real-Time Monitoring Loop

```
import matplotlib.pyplot as plt
from IPython.display import clear_output

# Initialize storage for readings
vibration_list = []
temperature_list = []
pressure_list = []
prediction_list = []

def generate_sensor_data():
    return {
        'vibration': round(random.uniform(0.1, 5.0), 2),
        'temperature': round(random.uniform(30, 100), 2),
        'pressure': round(random.uniform(1, 10), 2)
    }

# Simulate 10 readings & check predictions with live graph
for i in range(10):
    sensor_data = generate_sensor_data()
    features_df = pd.DataFrame([sensor_data])
    prediction = model.predict(features_df)[0]

    # Store data for plotting
    vibration_list.append(sensor_data['vibration'])
    temperature_list.append(sensor_data['temperature'])
    pressure_list.append(sensor_data['pressure'])
    prediction_list.append(prediction)

    # Print status
    print(f"Reading {i+1}: {sensor_data}")
    if prediction == 1:
        print("⚠ Failure predicted! Logging maintenance action.")
    else:
        print("✅ Normal operation.")
    print("-----")

    # Plot live graph
    clear_output(wait=True)
    plt.figure(figsize=(10, 6))

[18]
```

```
# Store data for plotting
vibration_list.append(sensor_data['vibration'])
temperature_list.append(sensor_data['temperature'])
pressure_list.append(sensor_data['pressure'])
prediction_list.append(prediction)

# Print status
print(f"Reading {i+1}: {sensor_data}")
if prediction == 1:
    print("⚠ Failure predicted! Logging maintenance action.")
else:
    print("✅ Normal operation.")
print("-----")

# Plot live graph
clear_output(wait=True)
plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1)
plt.plot(vibration_list, marker='o', color='orange')
plt.title('Vibration Readings')
plt.ylabel('Vibration')

plt.subplot(3, 1, 2)
plt.plot(temperature_list, marker='o', color='red')
plt.title('Temperature Readings')
plt.ylabel('Temperature (°C)')

plt.subplot(3, 1, 3)
plt.plot(pressure_list, marker='o', color='blue')
plt.title('Pressure Readings')
plt.ylabel('Pressure (Bar)')
plt.xlabel('Reading Number')

plt.tight_layout()
plt.show()

[19] ✓ 8.6s
```