```
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
from sqlalchemy import create_engine
import requests
from datetime import datetime, timedelta

# OpenWeatherMap API setup
API_KEY = 'YOUR_OPENWEATHERMAP_API_KEY'  # Replace with your actual API key from
openweathermap.org
WEATHER_URL = f'http://api.openweathermap.org/data/2.5/forecast/daily?
lat=39.0646&lon=-94.5877&cnt=7&units=metric&appid={API_KEY}'

# Fetch 7-day weather forecast
def fetch_weather():
    try:
        response = requests.get(WEATHER_URL)
        response.raise_for_status()
        data = response.json()
        forecast = []
        for day in data['list']:
            date = datetime.fromtimestamp(day['dt']).strftime('%Y-%m-%d')
            temp_high = day['temp']['max']
            temp_low = day['temp']['min']
            condition = day['weather'][0]['main']
            rain_prob = day.get('pop', 0) * 100  # Probability of precipitation
            forecast.append({
                'date': date,
                'temp_high': temp_high,
                'temp_low': temp_low,
                'condition': condition,
                'rain_prob': rain_prob
            })
        return forecast
    except Exception as e:
        print(f"Error fetching weather: {e}")
        return []

# Connect to MySQL database using SQLAlchemy
engine = create_engine('mysql+pymysql://root:Kiku@999@localhost/iot_farming')
query = "SELECT * FROM sensor_data WHERE timestamp BETWEEN '2025-04-19' AND '2025-
05-03'"
df = pd.read_sql(query, engine)

# Preprocess IoT data
df['timestamp'] = pd.to_datetime(df['timestamp'])
df = df.sort_values('timestamp')

# Define crop health label (stressed if soil_moisture < 30 or temperature > 30)
df['health_status'] = np.where(
    (df['soil_moisture'] < 30) | (df['temperature'] > 30),
    'stressed', 'healthy'
)

# Simulated crop recommendation dataset
crops = ['Corn', 'Tomatoes', 'Wheat']
crop_conditions = pd.DataFrame([
    {'crop': 'Corn', 'temp_min': 15, 'temp_max': 30, 'humidity_min': 60,
'humidity_max': 90, 'soil_moisture_min': 30, 'soil_moisture_max': 80},
    {'crop': 'Tomatoes', 'temp_min': 18, 'temp_max': 32, 'humidity_min': 50,
'humidity_max': 85, 'soil_moisture_min': 40, 'soil_moisture_max': 90},
    {'crop': 'Wheat', 'temp_min': 10, 'temp_max': 25, 'humidity_min': 55,
'humidity_max': 80, 'soil_moisture_min': 25, 'soil_moisture_max': 70}
])

# Assign crop labels to IoT data
df['recommended_crop'] = 'Corn'  # Default
for _, crop in crop_conditions.iterrows():
    mask = (
        (df['temperature'] >= crop['temp_min']) & (df['temperature'] <=
crop['temp_max']) &
        (df['humidity'] >= crop['humidity_min']) & (df['humidity'] <=
crop['humidity_max']) &
        (df['soil_moisture'] >= crop['soil_moisture_min']) & (df['soil_moisture']
<= crop['soil_moisture_max'])
    )
    df.loc[mask, 'recommended_crop'] = crop['crop']

# Train health status model
features = ['temperature', 'humidity', 'soil_moisture', 'water_level',
'ambient_light']
X_health = df[features]
y_health = df['health_status']
X_train_h, X_test_h, y_train_h, y_test_h = train_test_split(X_health, y_health,
test_size=0.2, random_state=42)
clf_health = RandomForestClassifier(n_estimators=100, random_state=42)
clf_health.fit(X_train_h, y_train_h)
health_accuracy = accuracy_score(y_test_h, clf_health.predict(X_test_h))

# Train crop recommendation model
X_crop = df[features]
y_crop = df['recommended_crop']
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X_crop, y_crop,
test_size=0.2, random_state=42)
clf_crop = RandomForestClassifier(n_estimators=100, random_state=42)
clf_crop.fit(X_train_c, y_train_c)
crop_accuracy = accuracy_score(y_test_c, clf_crop.predict(X_test_c))

# Get latest IoT data and predict
latest_data = df.tail(100)[features]
health_pred = clf_health.predict(latest_data)
health_status = 'healthy' if (health_pred == 'healthy').mean() > 0.5 else
'stressed'
crop_pred = clf_crop.predict(latest_data)
recommended_crop = pd.Series(crop_pred).mode()[0]

# Fetch weather forecast
weather_forecast = fetch_weather()
weather_alert = 'No Alerts'
rain_expected = any(day['rain_prob'] > 50 for day in weather_forecast[:2])  # Check
next 48 hours
if rain_expected:
    weather_alert = 'Rain Expected'

# Generate irrigation schedule and suggestions
latest_stats = df.tail(100)[features].mean(numeric_only=True)
```

```
irrigation_schedule = '8:00 AM'
irrigation_desc = 'Optimal watering time based on soil moisture.'
recommendations = []

if latest_stats['soil_moisture'] < 30:
    if not rain_expected:
        recommendations.append("Irrigate for 20 minutes, soil moisture critically
low.")
    else:
        irrigation_schedule = 'Skip'
        irrigation_desc = 'Rain expected, irrigation not needed.'
if latest_stats['temperature'] > 30:
    recommendations.append("Provide shade or misting, high temperature stress
detected.")
if latest_stats['water_level'] < 1000:
    recommendations.append("Refill water reservoir, low water level detected.")
if weather_alert == 'Rain Expected':
    recommendations.append("Prepare for rainfall in next 48 hours.")

# Save results for dashboard
results = {
    'health_status': health_status,
    'recommended_crop': recommended_crop,
    'irrigation_schedule': irrigation_schedule,
    'irrigation_desc': irrigation_desc,
    'weather_alert': weather_alert,
    'weather_forecast': weather_forecast,
    'recommendations': recommendations,
    'model_accuracy': {
        'health': round(health_accuracy * 100, 1),
        'crop': round(crop_accuracy * 100, 1)
    }
}
pd.DataFrame([results]).to_json('ai_results.json')

print("AI analysis complete. Results saved to ai_results.json")
```