Authors

Lynn Bitok

Adoh Baraza

Salome Wamaitha

Purity Mutunga

Meshack Omuda

Jusper Ageri

AI Development Workflow:

Part 1: Short Answer Questions

AI Development Workflow: Solar Panel Efficiency Prediction System

- **Problem Definition**

Problem: Predicting solar panel performance degradation and maintenance needs for residential solar installations in Kenya.

Context: Kenya's renewable energy expansion leads thousands of households to install rooftop solar. However, poor maintenance causes 20-30% energy output reduction within 3 years, discouraging adoption and reducing ROI.

Objectives:

- **Early Detection:** Identify panels with >5% efficiency loss within 7 days
- **Predictive Maintenance:** Reduce unplanned downtime by 40%
- **Cost Optimization:** Save homeowners 25% on maintenance costs

Stakeholders:

- **Homeowners:** Need reliable energy, minimized costs, SMS system health alerts
- **Solar Companies:** Require efficient maintenance scheduling, reduced emergency calls, proactive customer service

KPI: F1-Score ≥0.85 for degradation detection, balancing precision (avoiding false alarms) and recall (catching real degradation). Benchmark: manual inspection achieves ~0.60 F1-score.

- **Data Collection & Preprocessing**

Data Sources:

- **IoT Sensor Data:** Power output, voltage, current, temperature at 15-minute intervals from 10,000+ installations (3-5 years historical data). Includes panel specifications, installation date, orientation, tilt angle via cloud-connected smart inverters.
- **Weather Data:** Solar irradiance, temperature, humidity, cloud cover, dust/particulate levels from Kenya Meteorological Department and NASA POWER satellite data. GPS-matched to nearest weather stations (50km radius).

Potential Bias:

Geographic/Socioeconomic Imbalance: Urban installations (Nairobi, Mombasa, Kisumu) dominate training data—higher-income households have premium IoT systems, reliable connectivity, and frequent maintenance visits generating quality labels. Rural off-grid systems are underrepresented due to basic equipment, poor connectivity, and sparse maintenance records. Model learns well-maintained urban patterns but may not generalize to harsher rural conditions (more dust, extreme temperatures, wildlife damage), reinforcing energy access inequality.

Preprocessing Steps:

Step 1: Missing Data Handling Apply linear interpolation for gaps <2 hours; mark longer gaps with "data available" indicator feature. Fill weather gaps using spatial interpolation or satellite estimates. Missing data is informative (connectivity problems may correlate with degradation).

Step 2: Normalization & Feature Engineering Calculate Performance Ratio = (Actual/Expected Power) × 100%. Create temporal features (hour, day, season), rolling 7-day statistics, temperature correction (-0.5% efficiency per °C above 25°C), and age-based features (days since installation, ~0.5% annual degradation). Normalized metrics reveal actual performance issues independent of weather variations.

Step 3: Outlier Detection & Verification Remove physically impossible readings (>105% capacity, negative values). Validate maintenance labels against power output changes. Handle class imbalance (degradation ~5% of observations) with SMOTE oversampling or class weights.

- **Model Development Model:**

Random Forest Classifier

Justification:

1. Interpretable feature importance for technicians

2. Handles non-linear relationships and mixed-scale features naturally
3. Robust performance on tabular sensor data
4. Computationally efficient for resource-constrained deployment
5. Proven track record in predictive maintenance

Data Split (Temporal):

6. **Training (70%):** Jan 2019-Dec 2022 (~8M readings) - learn seasonal patterns
7. **Validation (15%):** Jan-Jun 2023 (~1M readings) - hyperparameter tuning
8. **Test (15%):** Jul 2023-Dec 2024 (~2M readings) - unbiased evaluation

Temporal split prevents data leakage, tests generalization to new time periods, and mirrors real deployment. Stratification ensures representation across installation types (urban/rural, brands, ages).

Hyperparameters:

- **Number of Trees (50-500)** Controls ensemble size. Too few (<50) causes unstable predictions; too many (>500) provides diminishing returns with slower inference. Optimal: 100- 300 trees, target 200. Tuning: Plot validation F1-score vs. tree count; select elbow point.

- **Max Depth (5-20)** Controls tree complexity. Too shallow (<5) cannot capture complex patterns like "IF (temp > 60°C AND dust > 7 AND age > 2yrs)". Too deep (>20) memorizes training specifics, overfitting to individual installations. Optimal: 10-15 for 3–4-way interactions while generalizing across installations. Tuning: 5-fold cross-validation; reduce depth if training-validation gap is large.

- **Evaluation & Deployment**

Metrics:

- **F1-Score (Primary)** Harmonic mean of precision and recall. Precision matters: false positives waste money (unnecessary visits, premature parts). Recall matters: false negatives cause harm (missed degradation, energy loss, failures). With a 5% degradation rate, accuracy misleads (95% by always predicting "no degradation"). Target ≥0.85 balances catching degradation while avoiding false alarms.

- **AUC-ROC** Threshold-independent evaluation across all decision points. Enables business flexibility: conservative threshold (fewer visits, may miss issues) vs. aggressive threshold (catch everything, more false alarms). Industry standards for comparing algorithms. Target ≥0.90 indicates excellent discrimination.

Concept Drift:

Changes in feature-target relationships over time degrading accuracy. Examples: new panel technologies (PERC vs. older silicon), climate change (extreme heat waves, changing dust patterns), aging fleet (beyond training data range), evolving maintenance practices.

Monitoring Drift:

- **Performance Dashboard:** Weekly F1-score/AUC tracking; alert if F1 <0.80; separate precision/recall monitoring
- **Feature Distribution:** Monthly Kolmogorov-Smirnov tests comparing current vs. training data; alert on significant shifts
- **Ground Truth Validation:** Quarterly confusion matrix from technician reports vs. predictions; trigger review if >20% mismatch
- **Retraining:** Automatic every 6 months; emergency if performance drops >10%

Deployment Challenge:

Intermittent Connectivity in Rural Areas

Problem: Rural installations lack reliable internet. Data arrives in delayed batches, incomplete feature windows (need 7 days, get 3), stale predictions, missed immediate alerts. Impact: delayed detection, energy loss, eroded trust, weakened value proposition for off-grid communities needing it most.

Solutions:

Edge Computing: Lightweight model on inverter hardware for local processing; SMS alerts via basic GSM; cloud model runs when connectivity allows detailed diagnostics.

Adaptive Windows: Model handles variable-length inputs (3-14 days); "data availability" features adjust confidence; uncertainty flagged when data sparse.

Hybrid Architecture: Rule-based critical alerts (>20% power drop) trigger immediate SMS; cloud AI handles nuanced degradation predictions. Fast simple alerts plus sophisticated analysis.

Part 2: Case Study Application

Hospital Readmission Predictor

Overview

The **Hospital Readmission Predictor** is an AI-powered web application designed to estimate a patient's risk of being readmitted within 30 days after discharge.
It uses a trained machine learning model built in **Python (scikit-learn)** and served via **FastAPI**, with a simple interactive dashboard for hospital staff to input patient data and receive real-time predictions.

Objectives

- Predict whether a discharged patient is **High risk** or **Low risk** for readmission within 30 days.
- Provide an easy-to-use interface for hospital administrators and clinicians.
- Support ethical and regulatory compliance (e.g., HIPAA, GDPR).

Stakeholders

Stakeholder	Role
Hospital administrators	Integrate the model into hospital workflows
Clinicians / Nurses	Use predictions to plan follow-up care
Patients	Benefit from proactive health management
Data scientists / developers	Maintain and optimize the AI system

Model Description

- **Model used:** Logistic Regression
- **Why:** It is interpretable, efficient, and suitable for binary outcomes (readmitted vs not readmitted).
- **Features:** Age, gender, blood pressure, BMI, cholesterol, diabetes, hypertension, medication count, length of stay, discharge destination, etc.
- **Output:**
    - `High risk` (probability ≥ 0.5)
    - `Low risk` (probability < 0.5)

Preprocessing Pipeline

9. Handle missing values and outliers
10. Encode categorical variables (e.g., gender, diabetes status)
11. Normalize numerical features

12. Feature engineering:

- Age grouped into ranges
- Derived features like BMI categories

Model Performance (Example)

Metric	Value
Accuracy	0.88
Precision	0.44
Recall	0.50
F1-score	0.47

Note: Metrics are based on the validation dataset from model training.

Tech Stack

Component	Technology
Backend	FastAPI
Frontend	TailwindCSS + Jinja2 templates
ML Library	scikit-learn
Data Processing	pandas, NumPy
Deployment	Uvicorn server
Language	Python 3.10+

Installation and Setup

1. Clone the project

```
git clone https://github.com/aladeen5/hospital-readmission-predictor.git
```

2. Create and activate virtual environment

```
python3 -m venv .venv
source .venv/bin/activate    # (Linux/Mac)
.venv\Scripts\activate       # (Windows)
```

3. Install dependencies

```
pip install -r requirements.txt
```

4. Run the app

```
uvicorn app:app --reload
```

5. Access the dashboard

- Open: **http://127.0.0.1:8000/dashboard**
- API Docs: **http://127.0.0.1:8000/docs**

Directory Structure

```
AI_developmentWorkflow/
│
├── app.py                    # Main FastAPI application
├── readmission_model.pkl             # Trained ML model
├── requirements.txt          # Dependencies
├── templates/
│   └── index.html            # Dashboard UI
├── static/                   # You can add Static assets (e.g., images, css)
├── data/
│   └── hospital_readmissions_30k.csv
└── README.md
```

Example API Request

Endpoint: `/predict_readmission`
Method: POST
Body (JSON):

```
{
  "age": 55,
  "gender": "Female",
  "blood_pressure": "120/80",
  "cholesterol": 210,
  "bmi": 27.5,
  "diabetes": "Yes",
  "hypertension": "No",
  "medication_count": 3,
  "length_of_stay": 5,
  "discharge_destination": "Home"
}
```

Response:

```
{
  "prediction": "High risk",
  "readmission_probability": 0.78
}
```

Ethical & Regulatory Compliance

- **Data Privacy:** Patient data anonymized and stored securely.
- **HIPAA Compliance:** Only de-identified or aggregated data used for model training.
- **Transparency:** Clinicians can understand and audit model predictions.
- **Fairness:** Model tested for bias across demographic groups.

Optimization Plan

To address **overfitting**:

- Use **cross-validation**

- Apply **regularization (L2 penalty)** in Logistic Regression
- Tune hyperparameters using **GridSearchCV**
- Increase dataset diversity and balance classes via **SMOTE**

Future Improvements

- Integrate into hospital EHR systems via API endpoints
- Build an admin dashboard for monitoring predictions
- Extend to multi-hospital networks
- Include SHAP explainability for interpretability

Part 3: Critical Thinking

1. How might biased training data affect patient outcomes in the case study?

Biased training data can lead to **systematic errors** that disproportionately harm certain patient groups. For example:

- If historical hospital data underrepresents rural patients, the model may **underestimate readmission risk** for them, leading to insufficient follow-up care.
- If certain demographic groups (e.g., older adults, women, or specific ethnic communities) had lower access to care historically, the model might **misinterpret this as lower risk**, further reinforcing inequality.
- Biased data can influence triage decisions, resulting in **unfair allocation of resources**, where some patients receive extra monitoring while others who truly need it are ignored.
- If socioeconomic factors are not properly represented, patients from low-income backgrounds may be flagged incorrectly, leading to **unnecessary interventions**, increased cost, or distrust.

In healthcare, this is especially dangerous because model decisions directly affect **patient safety, fairness, and quality of care**.

2. Suggest 1 strategy to mitigate this bias.

Strategy: Perform bias and fairness audits on subgroups (e.g., age, gender, location) and retrain using rebalancing techniques.

A fairness audit evaluates model performance across different demographic groups. If performance gaps are found, techniques such as:

- **Reweighting** (giving more weight to underrepresented groups),
- **Oversampling** minority groups, or
- **Using fairness-aware loss functions**

Part 4 Reflection & Workflow Diagram

1. What was the most challenging part of the workflow? Why?

The most challenging part of the workflow was the **data preprocessing and feature engineering** stage. Hospital EHR data is usually messy, contains many missing values, and includes complex categorical variables such as diagnosis codes, medication lists, and comorbidity indicators. Properly cleaning and transforming this information into meaningful machine-readable features required multiple iterations and domain understanding. Additionally, balancing the dataset was difficult because readmissions are often a minority class, which can hurt model performance.

2. How would you improve your approach with more time/resources?

With more time and resources, I would improve the workflow in several ways:

- **Collect richer clinical data** such as lab time-series trends and discharge notes, which could enhance prediction accuracy.
- **Use advanced models**, including transformer-based models for clinical notes or sequence models (LSTMs) for time-dependent data.
- **Collaborate with clinicians** to refine feature engineering so that key clinical risk factors (e.g., frailty scores) are captured.
- **Implement continuous monitoring** to track model drift and recalibrate the model regularly with new hospital data.
- **Strengthen fairness testing**, ensuring the model performs consistently across demographic subgroups to avoid bias.

AI Development Workflow Diagram