PART 1: SHORT ANSWER QUESTIONS

Problem Definition

Hypothetical AI problem:

- Predicting equipment failure in rural water pumping stations.

Explanation:

- In many rural areas, water supply depends on mechanical pumping stations. Unexpected failures can disrupt access to clean water, affecting health and productivity.

AI role:

- Use historical sensor data and maintenance logs to predict failures before they occur.

Objectives:

- Minimize downtime by forecasting failures early, allowing preventive maintenance.
- Optimize maintenance schedules using predictive analytics to reduce unnecessary inspections.
- Reduce operational costs by avoiding emergency repairs and improving resource allocation.

Stakeholders:

- County water engineers: Responsible for infrastructure reliability and maintenance planning.
- Local communities: End-users who rely on consistent water access for daily needs.

Key Performance Indicator (KPI):

- Downtime reduction rate: Measure the percentage decrease in unplanned outages over a 6-month period post-deployment.

Data collection & Preprocessing

Data sources:

- IoT sensor logs: Real-time data from pumps (pressure, flow rate, vibration).
- Maintenance records: Technician reports, repair history, and service logs.

Potential bias:

- Connectivity bias: Remote stations may have intermittent data due to poor network coverage, leading to underrepresentation and skewed model performance favoring urban stations.

Preprocessing steps:

- Missing data imputation: Use interpolation or time-series methods to fill gaps in sensor logs.
- Normalization: Scale sensor readings to a uniform range to ensure model stability.
- Categorical encoding: Convert maintenance actions (“replaced valve”) into numerical format using one-hot encoding for model compatibility.

Model development

Model choice: Random Forest

Justification:

- Handles non-linear relationships and mixed data types.
- Robust to missing values and noise.
- Offers interpretability through feature importance—valuable for engineering teams.

Data splitting strategy:

- Training Set (70%): Used to fit the model.
- Validation Set (15%): Used for hyperparameter tuning.
- Test Set (15%): Used to evaluate final model performance.

Hyperparameters to tune:

- n_estimators: Number of trees—affects accuracy and generalization.
- max_depth: Controls tree depth—prevents overfitting and improves interpretability.

Evaluation & deployment

Evaluation metrics:

- F1 Score: Balances precision and recall—important when false negatives (missed failures) are costly.
- ROC-AUC: Measures model's ability to distinguish between failure and non-failure across thresholds.

Concept drift:

- When the statistical properties of input data change over time, reducing model accuracy.

Monitoring strategy:

- Use dashboards to track prediction accuracy and retrain the model periodically using fresh data.

Deployment challenge:

- Scalability: Ensuring real-time inference across distributed stations with limited bandwidth and compute resources. Solution may involve edge computing or lightweight model deployment.

PART 2: CASE STUDY APPLICATION

Problem scope

Problem:

- Predict patient readmission risk within 30 days of hospital discharge.

Objectives:

- Improve patient outcomes by identifying high-risk individuals for follow-up care.
- Reduce hospital costs and avoid penalties associated with high readmission rates.

Stakeholders:

- Hospital administrators: Oversee operational efficiency and compliance.
- Medical staff: Use predictions to guide discharge planning and post-care interventions.

Data strategy

Data sources:

- Electronic Health Records (EHRs): Diagnoses, procedures, medications, lab results.
- Demographics: Age, gender, income level, insurance type.

Ethical concerns:

- Privacy: EHRs contain sensitive data—must comply with HIPAA and local regulations.
- Bias: Historical data may reflect systemic inequalities, leading to unfair predictions for marginalized groups.

Preprocessing Pipeline:

- Clean raw encounters, normalize missing values, drop non-applicable discharges, and label readmissions.
- Engineer utilization, medication, diagnosis, admission/discharge, and demographic features; convert age bins, group specialties, add risk flags and comorbidity counts.
- Encode categoricals via SimpleImputer + OneHotEncoder, scale numerics, and carry class-ratio metadata for downstream modeling.

Model development

Model: Gradient Boosting (XGBoost)

Justification:

- High performance on tabular data.
- Handles missing values and imbalanced classes.
- Offers feature importance for interpretability.

Hypothetical Confusion Matrix Example

Imagine 100 cases:

	Predicted positive	*Predicted negative*
Actual positive	18 (TP)	7 (FN)
Actual negative	12 (FP)	63 (TN)

Precision = TP / (TP + FP) = 18 / (18 + 12) = 0.60

Recall = TP / (TP + FN) = 18 / (18 + 7) = 0.72

Interpretation:

Precision answers "when the model predicts positive, how often is it right?" High precision means few false alarms (low false positives).

Recall answers "how many of the actual positives did the model catch?" High recall means few misses (low false negatives).

Deployment

Integration steps:

- Package the trained pipeline plus preprocessing artifacts into a versioned service (joblib bundle + REST/Batch endpoint).
- Build an interface that pulls encounter data from the hospital's EHR, runs preprocessing/model scoring, and writes risk scores back to clinical dashboards or alerts.
- Add monitoring hooks: capture inputs, outputs, latency, and drift metrics; schedule periodic retraining and threshold reviews with stakeholders.
- Train clinicians and IT staff, document model behavior, fallback plans, and support channels; pilot on a limited unit before wider rollout

Regulatory compliance:

- Enforce HIPAA controls end to end: encrypted storage and transport, role-based access, audit logging, and PHI minimization (use only necessary fields, mask identifiers in analytic logs).
- Maintain a Data Use Agreement and documented risk assessments; run regular security tests and privacy reviews before every release.

Optimization

Overfitting solution:

Introduce stronger regularization/early stopping on XGBoost (tune max_depth, min_child_weigh t, gamma, enable early stopping rounds) to curb variance without sacrificing recall.

PART 3: CRITICAL THINKING

Ethics & Bias

Impact of biased data:

- Biased training data would skew the model's risk scores toward the groups that are overrepresented in the historical records. Underrepresented populations could be misclassified as low risk, so they'd miss early follow-up, while groups tagged too aggressively as high risk might undergo unnecessary interventions; patient outcomes and resource allocation would both suffer.

Mitigation strategy:

- Enforce stratified auditing and rebalancing during preprocessing: examine performance metrics across demographic slices, then resample or reweight the training data (class-conditioned sampling per sub-group) until precision and recall stay within acceptable bounds for every cohort.

Trade-offs

Trade-off between model interpretability and accuracy in healthcare.

- In healthcare, clinicians need to understand why a model flags a patient, so highly interpretable approaches (logistic regression, rules, scorecards) make adoption and trust easier. The trade-off is that the most accurate models on messy, nonlinear data are often complex ensembles or deep nets whose reasoning is opaque; they may squeeze out a few accuracy points but can be harder to validate, explain to clinicians, or defend to regulators.

Limited resources impact:

- With limited compute, the hospital would likely favor lighter models that fit and score quickly; logistic regression, gradient-boosted trees with modest depth, or even calibrated rules, over heavyweight neural nets or massive ensembles. That constraint nudges selection toward algorithms that balance acceptable accuracy with small memory footprint, faster inference, and easier deployment on available hardware.

PART 4: REFLECTION & WORKFLOW DIAGRAM

Reflection

Most challenging part:

- The toughest part was reconciling the evaluation constraints with the model's behavior. The original fallback picked the highest-precision threshold, which—all the way up at 0.9—collapsed recall to ~0.0. Finding why the system "chose" an obviously bad operating point required digging through the training code, re-running the pipeline, and checking the generated reports; that feedback loop is slow and adds friction.

Improvement plan:

- With more time, we would harden the evaluation loop: surface the full precision/recall trade-off explicitly (plots, summary stats), add sanity checks that block saving models with near-zero recall, and maybe run a sweep or AutoML job focused on recall-oriented objectives.
- We would also invest in better tooling around threshold selection—e.g., optimizing a weighted metric or letting stakeholders choose from a dashboard—so we're not relying on a brittle rule baked into the script.

Diagram

Flowchart of the AI Development Workflow