Part 1 — Short Answer Questions

1. Problem definition

Problem: Predicting which university students are at risk of dropping out within the next academic year.

Objectives:

1. Identify at-risk students early (within a semester) so interventions can be offered.
2. Reduce overall dropout rate by X% (operational objective — e.g., 10% in year 1).
3. Provide interpretable risk factors per student to guide targeted support (actionable insights).

Stakeholders:

- University Student Support Services / Advisors.
- Students (and optionally parents / guardians).

Key Performance Indicator (KPI):
F1-score on the "dropout" class (harmonizes precision and recall so both false negatives and false positives are considered).

2. Data Collection & Preprocessing

Two data sources:

1. LMS logs (learning management system): assignment submissions, time-on-task, forum participation.
2. Administrative records: demographics, prior academic performance (GPA), financial aid status, attendance.

One potential bias in the data:
Students from lower-income backgrounds may have sparser LMS activity (e.g., no reliable internet), causing the model to equate low log activity with low engagement — this confounds socioeconomic status with “engagement” and could unfairly label disadvantaged students as high-risk.

Three preprocessing steps (outline):

1. **Handle missing data:**
    - Distinguish *Missing Completely At Random* vs. *Missing Not At Random*.
    - Impute numeric features with median or model-based imputation; for categorical, add a “missing” category where appropriate. For features where missingness is itself predictive (e.g., no submission at all), create a binary “missing” indicator.
2. **Feature scaling / normalization:**

- Scale continuous predictors (e.g., time-on-task, GPA) using robust scaling (median & IQR) to reduce influence of outliers.

3. **Encoding and feature engineering:**
    - One-hot or target encoding for categorical variables (program, country).
    - Create engineered features: submission rate (% of assignments submitted), trend features (GPA change over last 2 terms), and interaction terms (financial-aid × time-on-task).

3. Model Development (8 pts)

Model choice & justification:
Gradient Boosted Trees (e.g., XGBoost). Reasons: handles heterogeneous feature types, robust to missing values, strong predictive performance on tabular data, and provides feature importance (and SHAP values) for interpretability.

Data split strategy:

- Training set: 70%
- Validation set: 15%
- Test set: 15%
  If time-series/semester ordering matters, split temporally: train on earlier semesters, validate on a later semester, test on the most recent semester to avoid leakage.

Two hyperparameters to tune & why:

1. **Learning rate (eta):** controls step size — affects convergence speed and overfitting.
2. **Max_depth (or num_leaves):** controls tree complexity — affects bias/variance tradeoff and model capacity.

4. Evaluation & Deployment (8 pts)

Two evaluation metrics & relevance:

1. **Recall (sensitivity) for the dropout class:** important because missing an at-risk student (false negative) means a missed opportunity for intervention.
2. **Precision for the dropout class:** important to avoid wasting scarce support resources on many false positives (keeps interventions targeted).

What is concept drift? How to monitor it post-deployment:

- **Definition:** Concept drift occurs when the relationship between input features and target changes over time (e.g., new curriculum, pandemic changes behavior), degrading model performance.
- **Monitoring:** Continuously track key performance metrics (e.g., recall, precision, AUC) on recent labeled data; monitor statistical distributions of important features (population stability index, PSI) and model score distributions. Set alerts for significant metric drops or distribution shifts and schedule retraining when thresholds are crossed.

One technical challenge during deployment (scalability):
Serving predictions in near-real-time for tens of thousands of students may require autoscaling inference infrastructure and caching (e.g., batch scoring nightly vs. real-time scoring for newly-submitted behavior). Ensuring low-latency, horizontally scalable model servers and efficient feature stores is non-trivial.

Part 2 — Case Study: Hospital readmission within 30 days (40 pts)

Problem scope (5 pts)

Problem: Predict whether a discharged patient will be readmitted within 30 days.

Objectives:

1. Accurately identify high-risk patients before discharge so care plans / follow-up can be scheduled.
2. Reduce avoidable 30-day readmissions by enabling targeted post-discharge interventions.
3. Provide interpretable drivers of risk so clinicians can act (medication reconciliation, social support, follow-up appointments).

Stakeholders:

- Clinicians and discharge planners.
- Hospital administration (quality & compliance).
- Patients and family caregivers.

Data strategy (10 pts)

Proposed data sources:

1. Electronic Health Records (EHR): diagnoses (ICD codes), labs, vitals, medications, procedures, length of stay, prior admissions.
2. Demographics & social determinants: age, gender, address-level socioeconomic indicators, insurance type.

(Also: claims data, post-discharge outpatient visit records, and patient-reported outcomes if available.)

Two ethical concerns:

1. **Patient privacy & data protection:** handling PHI (protected health information) requires strict safeguards.
2. **Bias and fairness:** models trained on historical care may reflect access disparities (e.g., certain groups historically under-admitted or underfollowed), causing unequal predictions and potentially worsening outcomes for disadvantaged groups.

Preprocessing pipeline (including feature engineering):

1. **Data ingestion & linking:** Merge inpatient EHR data with prior admissions, outpatient follow-ups, and claims where available (use patient identifiers under secure controls).
2. **De-identification / pseudonymization** for development datasets (keep linkage keys in secure vault for production).
3. **Missing values handling:**
    - Labs/vitals: impute with clinically sensible values (e.g., last observation carried forward for short stays) and add missing flags.
4. **Temporal features & windowing:**
    - Create features from last n days/weeks (e.g., mean vitals in last 48 hours, trend in creatinine).
    - Count features: number of ED visits in past 6 months, number of comorbidities.
5. **Feature engineering:**
    - **Comorbidity scores:** compute Charlson or Elixhauser comorbidity index.
    - **Medication complexity:** count of unique medications at discharge.
    - **Social risk proxies:** distance to hospital, missed appointments history, insurance type.
6. **Categorical encoding:** target or one-hot encode diagnosis groups, admission type (elective vs emergency).
7. **Normalization / scaling:** scale continuous features as appropriate.
8. **Label creation:** readmission within 30 days (binary), ensure correct handling of transfers, death, and planned readmissions.
9. **Train/val/test split:** temporal split by discharge date to simulate future performance.

Model development (10 pts)

Model selection & justification:
Gradient Boosted Trees (XGBoost). Justification: excellent performance on tabular healthcare data, handles mixed feature types, robust to missing values, relatively fast training, and interpretable via SHAP for clinician-facing explanations.

Hypothetical confusion matrix and precision/recall calculation:
Suppose on test set (N = 1,000 discharges) we have:

	Predicted Readmit	**Predicted Not Readmit**
Actual Readmit	TP = 80	FN = 30
Actual Not Readmit	FP = 20	TN = 870

Compute precision and recall for the "readmit" class:

- Precision = TP / (TP + FP) = 80 / (80 + 20) = 80 / 100 = 0.80 = **80%**.
  (digit-by-digit: 80 ÷ 100 = 0.8)
- Recall (sensitivity) = TP / (TP + FN) = 80 / (80 + 30) = 80 / 110.
  Compute 80 ÷ 110: reduce by 10 → 8 ÷ 11 = 0.727272... → **~72.7%**.

So precision = 80%, recall ≈ 72.7%.

(You could also compute F1 = 2 * precision * recall / (precision + recall) ≈ 2 * 0.8 * 0.72727 / (1.52727) ≈ 0.7619 ≈ **76.2%**.)

Deployment (10 pts)

Steps to integrate model into hospital system:

1. **Feature store & ETL:** Build an automated, auditable pipeline that extracts required features from EHR at discharge time. Use FHIR/HL7 interfaces for standardized data exchange.
2. **Model serving:** Containerize the model and expose a secure API (e.g., internal REST/gRPC) for the EHR system to request risk scores at discharge. Use a model server (KFServing, Seldon, or simple Flask/gunicorn behind an API gateway).
3. **Clinical UI integration:** Embed risk scores and concise explanations (top contributing features/SHAP summary) into the clinician workflow (discharge summary screen) so it's visible at the point of decision-making.
4. **Alerts & action triggers:** Configure decision support actions (e.g., schedule follow-up within 7 days if risk > threshold) and allow clinicians to override or provide feedback.
5. **Logging & audit:** Log predictions, features used, clinician actions, and outcomes for monitoring and compliance.
6. **Monitoring & retraining:** Implement monitoring for data drift, performance degradation, and a process for regular retraining and validation.

7. **Testing & validation:** Run parallel (silent) deployment for a pilot period to compare model predictions to clinician assessments and collect outcome data before full rollout.

Ensuring compliance with healthcare regulations (e.g., HIPAA):

- **Data access controls & least privilege:** restrict who and what systems can access PHI; use role-based access controls.
- **Encryption:** encrypt PHI both at rest and in transit (TLS for APIs, AES-256 for data storage).
- **Business Associate Agreements (BAA):** ensure any cloud or third-party vendor signs BAAs.
- **De-identification for development:** use de-identified or limited datasets for model development whenever possible; keep re-identification keys in a secure vault.
- **Audit logs & breach response plan:** comprehensive logging of access and operations; documented policies for breach detection and reporting.
- **Clinical validation & governance:** multidisciplinary review (clinicians, compliance, legal) and institutional approval processes before use in care decisions.

Optimization (5 pts)

One method to address overfitting:
Apply **L2 regularization (weight decay)** on the model and use **early stopping** based on validation AUC/recall. In practice with gradient-boosted trees, tune the regularization parameters (lambda / min_child_weight), limit tree depth, and use early stopping on a hold-out validation set to prevent overfitting.

Part 3 — Critical Thinking (20 pts)

Ethics & Bias (10 pts)

How biased training data might affect patient outcomes:
If the training data under-represents certain groups (e.g., patients from low-income neighborhoods, non-English speakers, or certain ethnicities), the model may systematically understate or overstate their readmission risk. Consequences: disadvantaged patients might be denied extra follow-up (false negatives) or overstressed with unnecessary readmissions or interventions (false positives). Historical biases in care patterns (e.g., differential admission practices) can embed inequities into predictions, reinforcing disparities.

One strategy to mitigate bias:
Reweighing / fairness-aware training: adjust sample weights or apply constraint-based learning so the model optimizes both accuracy and a fairness metric (e.g., equalized odds). Complement this with targeted data augmentation (collect more labeled examples for under-represented groups) and stratified performance monitoring by subgroup with threshold adjustments or subgroup-specific calibration.

Trade-offs (10 pts)

Interpretability vs accuracy in healthcare:

- Highly complex models (deep ensembles, deep neural nets) may achieve higher raw accuracy but are harder to interpret. In healthcare, interpretability is often critical for clinical trust, legal accountability, and actionable recommendations. Therefore, a slightly less accurate but interpretable model (e.g., logistic regression, decision tree, or gradient-boosted tree with SHAP explanations) may be preferable because clinicians can understand and contest decisions — improving adoption and patient safety. The optimal balance depends on context: for diagnostic decisions where stakes are very high, interpretability usually wins; for low-risk triage, predictive accuracy may be prioritized.

If the hospital has limited computational resources, impact on model choice:

- Choose lighter-weight models: logistic regression, small decision trees, or small gradient-boosted models with limited depth and fewer trees. Consider model compression / distillation (train a lightweight student model to mimic a heavier teacher model). Also use batch scoring (nightly) instead of realtime inference and optimize feature computation (precompute heavy features offline).

Part 4 — Reflection & Workflow Diagram (10 pts)

Reflection (5 pts)

Most challenging part of the workflow:
Dealing with **data quality and label correctness** is the hardest — real-world EHRs are messy (inconsistent coding, missingness, mislabelled planned readmissions). Without high-quality labels and consistent feature extraction, even the best modeling choices can fail.

How to improve with more time/resources:

- Run prospective data collection and a pilot RCT to validate interventions.
- Invest in better data engineering: a robust feature store, standardized FHIR-based data ingestion, and linkage to post-discharge outcomes (claims).
- Engage clinicians early for feature validation and to design actionable outputs.
- Conduct subgroup fairness audits and collect more representative data.