

Part 1: Short Answer Questions (30 points)

1. Problem Definition (6 points)

Hypothetical AI Problem:

"Predicting Grain Shortages in Kenya's National Cereals Board (NCPB) Warehouses."

3 Objectives:

1. Forecast grain demand 3–6 months in advance to prevent shortages.
2. Optimize stock levels to reduce wastage (e.g., spoilage, overstocking).
3. Improve pricing strategies based on predicted supply/demand fluctuations.

2 Stakeholders:

1. NCPB Managers – Need accurate forecasts for procurement and distribution.
2. Farmers – Benefit from fair pricing and timely purchases by NCPB.

1 Key Performance Indicator (KPI):

- "Mean Absolute Percentage Error (MAPE)" – Measures forecast accuracy (e.g., <10% error = success).

2. Data Collection & Preprocessing (8 points)

2 Data Sources

1. USDA Global Grain Production Data
 - *Link:* [USDA Production, Supply, and Distribution \(PSD\) Database](#)
 - *Relevance:* Provides maize/wheat production, imports, and stock levels for 200+ countries. Use data from African nations (e.g., Ethiopia, Tanzania) as proxies for Kenya.
2. CHIRPS Rainfall Data (Africa-Focused)
 - *Link:* [Climate Hazards Group Infrared Precipitation \(CHIRPS\)](#)
 - *Relevance:* High-resolution rainfall data for East Africa (1981–present). Adjust coordinates to mimic Kenyan growing regions.

1 Potential Bias

- "Commercial Farm Bias"
 - *Explanation:* USDA data primarily reflects large-scale farms, underrepresenting smallholder yields (which dominate Kenya's agriculture). This could skew production forecasts.

3 Preprocessing Steps

1. Handling Missing Data

- *Action:* Use interpolation for gaps in rainfall data or forward-fill last known production values.
- *Code:* `df['Production_MT'].fillna(method='ffill', inplace=True)`

2. Normalization

- *Action:* Scale numerical features (e.g., `Rainfall_mm`, `Price_USD`) to [0, 1] range to balance model weights.

Code:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['Rainfall_mm', 'Price_USD']] = scaler.fit_transform(df[['Rainfall_mm',
'Price_USD']])
```

3. Outlier Detection

- *Action:* Remove extreme values (e.g., a 3000% price spike due to data entry errors) using IQR.

3. Model Development (8 points)

Chosen Model: Long Short-Term Memory (LSTM) Neural Network

Justification:

- **Temporal Dependence:** Grain supply/demand relies on time-series patterns (e.g., seasonal rainfall, harvest cycles). LSTMs excel at learning sequential dependencies.
- **Non-Linear Relationships:** Can capture complex interactions (e.g., drought → low yields → price spikes) better than linear models.
- **Proven Use Case:** LSTMs are widely used in agricultural forecasting (e.g., [NASA's crop yield predictions](#)).

Alternatives Considered:

- **Random Forest:** Less effective for time-series data (ignores temporal order).
- **ARIMA:** Only handles linear trends, struggles with external variables (e.g., weather).

Data Splitting Strategy

1. Training Set (70%)
 - *Why?* Maximize learning from historical patterns (e.g., 2010–2018 data).
2. Validation Set (15%)
 - *Why?* Tune hyperparameters (e.g., LSTM units, dropout) without overfitting.
3. Test Set (15%)
 - *Why?* Final evaluation on unseen data (e.g., 2019–2020) to simulate real-world performance.

```
from sklearn.model_selection import train_test_split
```

code:

```
# Split temporally (no shuffling!)
```

```
train, test = train_test_split(df, test_size=0.15, shuffle=False)
```

```
train, val = train_test_split(train, test_size=0.176) # 15% of original
```

2 Hyperparameters to Tune

1. Number of LSTM Units
 - *What?* Controls model complexity (e.g., 32 vs. 64 units).
 - *Why?* Too few → underfitting; too many → overfitting.
 - *Tuning Method:* Grid search (e.g., [32, 64, 128]).
2. Dropout Rate
 - *What?* Fraction of neurons randomly disabled during training (e.g., 0.2 = 20%).
 - *Why?* Prevents overfitting to noise in small datasets.
 - *Tuning Method:* Try [0.1, 0.2, 0.3].

4. Evaluation & Deployment (8 points)

2 Evaluation Metrics

1. Mean Absolute Error (MAE)
 - *Relevance:* Measures the average absolute difference between predicted and actual grain supply/demand values (e.g., in metric tons).
 - *Why?* Easy to interpret (e.g., "Model is off by ±500 tons on average").
2. Root Mean Squared Error (RMSE)
 - *Relevance:* Penalizes larger errors more heavily than MAE (e.g., a 2000-ton error is 4x worse than a 500-ton error).
 - *Why?* Critical for avoiding severe shortages (big mistakes are costly).

Concept Drift & Monitoring

What is Concept Drift?

- When the statistical properties of the target variable (e.g., grain prices) change over time, making the model less accurate.
- *Example:* A new government subsidy suddenly reduces maize prices, breaking past trends.

How to Monitor Post-Deployment:

1. Statistical Tests:
 - Use the Kolmogorov-Smirnov (KS) test to compare distributions of predicted vs. actual values monthly.
2. Threshold Alerts:
 - Flag drift if RMSE increases by >15% over a 3-month window.
3. Retraining Protocol:
 - Automatically retrain the model if drift is detected (e.g., using new data from the last 6 months).

1 Technical Deployment Challenge

Challenge: Scalability for Real-Time Predictions

- *Issue:* The LSTM model may be too slow to process real-time data from 100+ NCPB warehouses simultaneously.
- *Solution:*
 1. Model Optimization:
 - Convert the model to TensorFlow Lite for faster inference.
 2. Edge Deployment:
 - Run predictions on regional servers (not centralized) to reduce latency.
 3. Caching:
 - Pre-compute forecasts for stable variables (e.g., weather) to reduce load.

Part 2: Case Study Application (40 points)

1. Problem Scope (5 points)

Problem Definition:

Predict which patients are at high risk of readmission within 30 days post-discharge using EHR data.

Objectives:

1. Reduce preventable readmissions by flagging high-risk patients for follow-up care.
2. Optimize hospital resource allocation.
3. Achieve $\geq 80\%$ recall (minimize false negatives).

Stakeholders:

- Hospital Administrators (Cost reduction)
- Clinicians (Care planning)
- Patients (Improved outcomes)

2. Data Strategy (10 points)

Proposed Data Sources:

1. Electronic Health Records (EHRs):
 - Lab results, medications, discharge summaries.
 - *Example Feature:* `Number_of_Previous_Readmissions`
2. Demographics:
 - Age, insurance type, ZIP code (proxy for socioeconomic status).
 - *Example Feature:* `Social_Vulnerability_Index`

Ethical Concerns:

1. Patient Privacy:
 - *Mitigation:* De-identify data (remove PHI per HIPAA). Use federated learning if possible.
2. Bias in Training Data:
 - *Mitigation:* Audit for disparities in race/insurance status. Apply reweighting techniques.

Preprocessing Pipeline:

1. Handling Missing Data:
 - Impute missing lab values with population medians.
2. Feature Engineering:
 - Create `Medication_Adherence_Score` from prescription refill data.

- Calculate `Comorbidity_Index` using ICD-10 codes.
3. Normalization:
- Scale numerical features (e.g., `Length_of_Stay`) using `RobustScaler`.

3. Model Development (10 points)

Selected Model: Gradient Boosted Trees (XGBoost)

- *Justification:*
 - Handles mixed data types (numeric/categorical).
 - Provides feature importance for clinical interpretability.
 - Outperforms logistic regression on imbalanced data (typical in healthcare).

Confusion Matrix (Hypothetical):

	Predicted Readmit	Predicted Not Readmit
Actual Readmit	85 (TP)	15 (FN)
Actual Not Readmit	20 (FP)	180 (TN)

- - Precision = $TP / (TP + FP) = 85/105 = 81\%$
- Recall = $TP / (TP + FN) = 85/100 = 85\%$

4. Deployment (10 points)

Integration Steps:

1. API Development:
 - Deploy model as a REST API using `FastAPI/Flask`.
2. EHR Integration:
 - Trigger predictions when discharge orders are signed (HL7/FHIR interface).
3. Alert System:
 - Flag high-risk patients in clinician dashboards (Epic/Cerner integration).

Regulatory Compliance (HIPAA):

1. Data Encryption:

- Use AES-256 for data in transit/at rest.
- 2. Access Controls:
 - Role-based access (e.g., only care teams can see predictions).
- 3. Audit Logs:
 - Track all model accesses per HIPAA Security Rule §164.312(b).

5. Optimization (5 points)

Overfitting Solution: Stratified Cross-Validation + Early Stopping

- *Implementation:*
- python

```
from xgboost import XGBClassifier
model = XGBClassifier(
    early_stopping_rounds=10, # Stop if no improvement in 10 epochs
    eval_metric='logloss',
    subsample=0.8 # Stochastic gradient boosting
)
```

- `model.fit(X_train, y_train, eval_set=[(X_val, y_val)])`
- *Why?* Ensures generalization by validating on multiple data splits and preventing overtraining.

Part 3: Critical Thinking (20 points)

Ethics & Bias (10 points)

Impact of Biased Training Data:

1. Demographic Disparities:
 - If historical data overrepresents insured patients, the model may underestimate readmission risks for uninsured patients (who often face barriers to follow-up care).
 - *Example:* A model trained on urban hospital data may fail rural patients with limited access to post-discharge services.
2. Clinical Consequences:
 - False negatives (missing high-risk patients) could lead to preventable complications.
 - False positives (over-flagging low-risk patients) may strain resources, diverting attention from truly at-risk cases.

Mitigation Strategy: Adversarial Debiasing




- *Implementation:*
- python

```
from aif360.algorithms.inprocessing import AdversarialDebiasing
```

- `debiased_model = AdversarialDebiasing(scope_name='debiased_model').fit(X_train, y_train)`
- *Why?* Actively reduces bias during training by penalizing the model for correlations between protected attributes (e.g., insurance type) and predictions.

Trade-offs (10 points)

Interpretability vs. Accuracy:

Model Type	Interpretability	Accuracy	Clinical Use Case
Logistic Regression	High 	Moderate	Ethics reviews, regulatory approval
XGBoost	Moderate (SHAP)	High 	Operational decision support
Neural Networks	Low	Highest 	Research-only settings

- *Key Consideration:* In healthcare, false negatives (missed risks) can be life-threatening. A slightly less interpretable but more accurate model (e.g., XGBoost with SHAP) is often justified if clinicians can understand key decision drivers.

Limited Computational Resources:

1. Model Choice:
 - Avoid deep learning—opt for lightweight models (e.g., L1-penalized logistic regression or Random Forest with limited depth).
2. Optimizations:
 - Use feature selection (e.g., SelectKBest) to reduce dimensionality.
 - Deploy ONNX runtime for faster inference on low-power devices.
 - *Example:*

- python

```
from sklearn.feature_selection import SelectKBest, f_classif  
selector = SelectKBest(f_classif, k=10).fit(X_train, y_train)
```

- X_train_reduced = selector.transform(X_train)

Part 4: Reflection & Workflow Diagram (10 points)

Reflection (5 points)

1. Most Challenging Part:

- Data Integration & Feature Engineering
 - *Why?* Merging disparate data sources (EHRs, demographics) required handling missing values, temporal alignment, and creating clinically meaningful features (e.g., Comorbidity_Index). Ensuring ethical use of sensitive variables (e.g., ZIP code as a proxy for socioeconomic status) added complexity.

2. Improvements with More Time/Resources:

- Real-Time Data Pipelines:
 - Replace synthetic data with live EHR integrations (HL7/FHIR APIs) for dynamic model updating.
- Bias Auditing Tools:
 - Integrate IBM's AI Fairness 360 for automated bias detection across patient subgroups.
- Explainability Enhancements:
 - Add LIME for local interpretability alongside SHAP for global insights.

Diagram (5 points)

AI Development Workflow Flowchart

Stage	Key Actions	Tools/Techniques	Healthcare-Specific Considerations
1. Problem Definition	Define clinical objective (e.g., reduce readmissions), success metrics	Stakeholder interviews, literature review	Align with hospital priorities (cost/outcomes)
2. Data Collection	Extract EHRs (ICD codes, labs), demographics, socioeconomic proxies	SQL, FHIR/HL7 APIs, de-identification tools	HIPAA compliance, audit logs
3. Preprocessing	Handle missing values, normalize numerical features, encode categories	RobustScaler, OneHotEncoder, KNN imputation	Protect PHI, address bias in imputation
4. Feature Engineering	Create clinical features (e.g., Comorbidity_Score), temporal aggregations	Pandas, featuretools	Clinician validation of feature relevance

5. Model Training	Train XGBoost with stratified CV, early stopping	XGBoost, scikit-learn	Class weighting for imbalance
6. Evaluation	Calculate precision/recall, SHAP analysis, bias audits	SHAP, AIF360, confusion matrix	Prioritize recall to minimize false negatives
7. Deployment Decision	Ethics review, regulatory approval (if required)	Model cards, explainability reports	HIPAA/GDPR compliance checks
8. API Integration	Deploy as REST API with role-based access control	FastAPI, Docker, OAuth2	Audit trails for data access
9. Monitoring	Track concept drift, model performance decay	Evidently, Prometheus	Real-time alerts for high-risk predictions
10. Feedback Loop	Clinician feedback → retrain model quarterly	CI/CD pipelines, A/B testing	Update model cards with new versions

