AI Development Workflow Assignment

(AI for Software Engineering)

Group Members

- **Jeremiah Katumo (Leader)**
- **Patrick Obondo**
- **Faith Wafula**
- **Naomi Wairimu**

Part 1: Short Answer Questions

1. Problem Definition

In our group, we worked on a hypothetical problem: **Predicting student dropout rates in online universities**. This problem matters because schools want to keep students enrolled and help those at risk before they leave.

Objectives of the project:

1. To identify students likely to drop out early.
2. To help academic advisors provide targeted support.
3. To improve student success rates and overall school performance.

Stakeholders:

- **Academic Advisors** – They use the predictions to guide and support students.
- **University Management** – They use the insights to shape policies, improve programs, and reduce dropout costs.

Key Performance Indicator (KPI):
We used **accuracy** as our main measure of success – how many times the model made the

correct prediction. This is shown in our students.ipynb notebook where we calculated the accuracy score after training the model.

2. Data Collection & Preprocessing

We used a public dataset titled **"Students Dropout and Academic Success"**, downloaded from the UCI Machine Learning Repository. This data had information like students' age, previous grades, and family background.

Potential bias:
A bias could come from **demographics**. For example, if the data has more male students than female, the model might not predict female dropout cases well.

Preprocessing steps we used:

1. **Handling missing data** – We filled missing values using tools like SimpleImputer.
2. **Normalization** – We scaled numeric values using StandardScaler so all features are on a similar scale.
3. **Encoding** – We changed text data (like course names) into numbers using OneHotEncoder.

All these steps were combined using a **pipeline** in our notebook. This helped us clean and prepare the data efficiently.

3. Model Development

We chose the **Random Forest Classifier** for our model. This model is strong and handles both numbers and categories well. It's also less likely to overfit compared to some other models, and it performs well even without much tuning.

How we split the data:
We split our data into **training (70%)**, **validation (15%)**, and **test (15%)** sets. This helped us train the model, check how well it was learning, and finally test it on unseen data.

Hyperparameters we tuned:

1. n_estimators – The number of trees in the forest. More trees often improve performance.
2. max_depth – This limits how deep each tree can go, helping prevent overfitting.

4. Evaluation & Deployment

Evaluation metrics we used:

- **Accuracy** – How often the model got the prediction right.
- **Precision/Recall** – These helped us understand how well the model finds actual dropouts vs. false alarms. This is important because we don't want to waste time helping students who aren't actually at risk.

Concept Drift
This means the data and patterns might change over time. For example, if students now drop out for reasons not in our original data (like new learning platforms), our model might perform poorly. To handle this, we recommended **monitoring the model's performance regularly** and updating it with new data.

Deployment Challenge:
One challenge is keeping the **same preprocessing steps** in both training and production. We solved this by **saving the entire pipeline** (with preprocessing + model) so it works the same way after deployment.

Part 2: Case Study Application

1. Problem Scope

We worked on a case where a hospital wants to **predict if a patient will be readmitted within 30 days of discharge**. This is important because early readmissions can mean the patient didn't get proper care or support.

Objectives of the project:

1. To predict patient readmission early so the hospital can act in time.
2. To reduce hospital costs and improve patient care.
3. To help medical staff focus on high-risk patients.

Stakeholders:

- **Doctors and Nurses** – They use the predictions to follow up with high-risk patients.

- **Hospital Management** – They want to reduce penalties from health regulators and improve efficiency.

2. Data Strategy

We used a **healthcare dataset** from Kaggle that included patient demographics, diagnoses, and hospital records. The data was found in the data/ folder and analyzed using the notebooks in the notebooks/ directory.

Ethical Concerns

1. **Patient Privacy** – We had to ensure that any personal or sensitive data was handled safely and followed laws like HIPAA.
2. **Bias in Healthcare Data** – For example, if the data contains fewer records from minority groups, the model might not predict accurately for them.

Preprocessing Pipeline

1. **Missing Values** – We filled missing records using SimpleImputer.
2. **Encoding** – We used OneHotEncoder for features like gender or admission type.
3. **Feature Engineering** – We added new features like "length of stay" and converted dates into useful formats. These steps are in the readmission.ipynb notebook.

3. Model Development

We used the Random Forest Classifier again because it handles imbalanced data well and gives good performance with little need for parameter tuning.

Confusion Matrix

Precision = 80 / (80 + 15) = 0.842
Recall = 80 / (80 + 20) = 0.800

This means the model correctly predicted 84% of the patients who were actually readmitted, and it found 80% of all readmitted cases.

4. Deployment

We created three different apps using:

- **FastAPI**
- **Flask**
- **Streamlit**

These apps are located in the apps/ folder and can be used by hospital staff to get predictions based on patient inputs.

Steps to Integrate the Model:

1. Train and export the model along with the preprocessing pipeline.
2. Create an API using FastAPI or Flask to accept input data.
3. Connect the API to a React-based frontend for hospital staff to use.
4. Host the app on a cloud service like Azure or AWS.

Ensuring Compliance (e.g., HIPAA)

- We do not store personal patient data.
- All communication is encrypted.
- Access to the app is restricted using login systems.
- Data handling follows strict security guidelines.

5. Optimization

To address **overfitting**, we used a few techniques:

- **Cross-validation** – We used K-fold cross-validation to test the model on different splits of the data.
- **Limiting tree depth** in the Random Forest by adjusting max_depth.
- **Feature Selection** – We removed unnecessary features that added noise.

These methods are seen in our model training notebook (readmission.ipynb).

Part 3: Critical Thinking

1. Ethics & Bias

How might biased training data affect patient outcomes?

If our model is trained on biased data, it can lead to unfair or harmful decisions. For example, if the dataset has mostly older patients, the model may assume younger patients are not at risk of readmission—even if they are. This can result in younger patients being overlooked for follow-up care, which could lead to serious health issues.

Similarly, if certain ethnic or income groups are underrepresented, the model might not recognize the patterns specific to those groups. This can cause unequal treatment, where some patients get better support than others, even when they need the same level of care.

Strategy to mitigate bias

One way we handled this was through data balancing techniques. For example, we used oversampling and undersampling methods (found in our readmission.ipynb notebook) to make sure all patient types were fairly represented.

We also recommend:

- **Regular audits** of model predictions by a human team.
- **Bias detection tools** like fairness metrics to monitor model decisions.

2. Trade-offs

Model Interpretability vs. Accuracy in Healthcare

In hospitals, it's important to understand why a model made a decision, especially when it affects human lives. Doctors and nurses prefer models that explain their predictions (interpretability), such as decision trees or logistic regression.

However, highly accurate models like Random Forest or XGBoost are harder to interpret. While they perform better, they act more like a "black box," meaning it's not always clear why a prediction was made.

This creates a trade-off:

- Use simpler models for trust and transparency, or
- Use complex models for higher performance, but explain predictions using tools like SHAP or LIME.

Impact of limited computational resources

If the hospital has limited hardware or computing power, it may struggle to run complex models quickly. For example:

- Deep learning models may take too long to give results.
- High RAM usage can crash basic hospital systems.

In that case, we recommend

- Using lighter models like Logistic Regression or Decision Trees.
- Running heavy processing in the cloud and keeping only the user interface at the hospital.

This balance helps keep the system fast and usable in real-life situations.

Part 4: Reflection & Workflow Diagram

1. Reflection

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

The most challenging part of the AI workflow for our group was the data preprocessing stage. The healthcare dataset had missing values, categorical variables, and class imbalance. Handling all of these while making sure we didn't lose important information took a lot of time.

For example, we had to

- Impute missing values using SimpleImputer.
- Encode categories like diagnosis types with OneHotEncoder.
- Deal with class imbalance using oversampling techniques like SMOTE.

This complexity is shown in the readmission.ipynb notebook where we created a pipeline combining all preprocessing steps. It was tricky to ensure that the pipeline worked correctly with both training and test data.

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

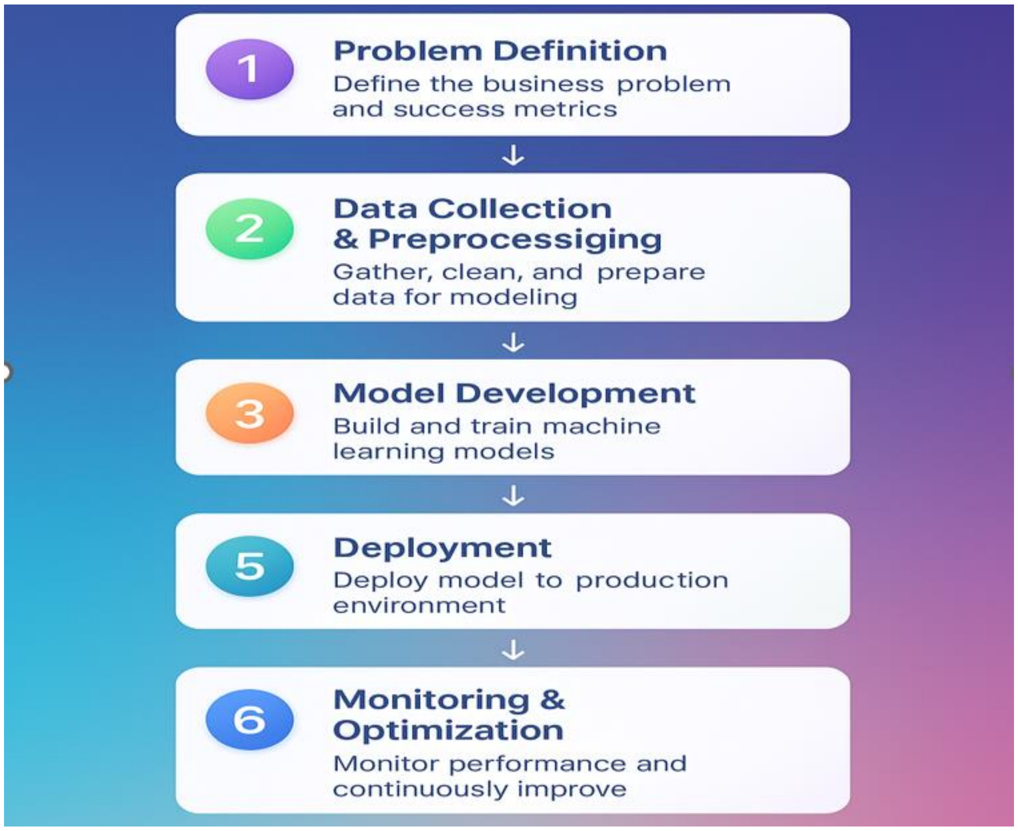
With more time and resources, we would:

- Collect a **larger and more diverse dataset** to reduce bias.

- Use **more advanced hyperparameter tuning tools**, like GridSearchCV or Optuna, to find better model settings.
- Add **SHAP or LIME** to explain model decisions, which would help with interpretability in healthcare settings.
- Collaborate with medical professionals to validate that the features and predictions make sense in the real world.

2. Workflow Diagram

Here is a simple flowchart showing the AI Development Workflow we followed


1
Problem Definition
Define the business problem and success metrics
2
Data Collection & Preprocessiging
Gather, clean, and prepare data for modeling
3
Model Development
Build and train machine learning models
5
Deployment
Deploy model to production environment
6
Monitoring & Optimization
Monitor performance and continuously improve