🎓 Professional ML Project Documentation: Student Dropout Risk Prediction

This document outlines the end-to-end plan for developing, deploying, and maintaining a machine learning model designed to predict student dropout risk. It follows a robust MLOps framework, integrating data versioning (DVC), experiment tracking (MLflow/W&B), and containerized deployment (Docker/FastAPI/AWS).

1. Project Repository Structure

The following structure ensures all artifacts—code, data, models, configurations, and documentation—are organized and version-controlled.

```
Student_dropout_risk/
├── .github/                    # CI/CD workflows (GitHub Actions)
├── configs/                    # YAML/JSON configs for pipeline
│   └── training_config.yaml      # Hyperparameters, feature lists, thresholds
├── data/
│   ├── raw/                    # Original uncleaned data (DVC-tracked)
│   └── processed/              # Cleaned and engineered data (DVC-tracked)
├── models/                     # Trained model artifacts (DVC/MLflow)
│   └── dropout_model.pkl
├── src/                        # Source code for the entire project
│   ├── data/                   # Scripts for data collection and initial loading
│   ├── preprocessing/          # Cleaning and feature engineering scripts (e.g., encoders,
scalers)
│   ├── models/                 # Training, tuning, and evaluation code
│   ├── api/                    # FastAPI application for serving predictions
│   │   └── app.py
│   ├── ui/                     # Streamlit/Gradio UI code for a simple interface
│   └── monitoring/             # Scripts for model drift and latency checks
├── tests/                      # Unit & integration tests
│   ├── test_preprocessing.py
│   ├── test_model.py
│   └── test_api.py
├── notebooks/                  # Jupyter notebooks for EDA and experimentation
├── reports/                    # Visual artifacts, evaluation reports, and model comparisons
├── experiments/                # MLflow/W&B logs and run metadata
├── .dvc/                       # DVC configuration files
├── Dockerfile                  # Docker container definition
├── requirements.txt            # Python dependencies
```

```
├── README.md              # Project overview & instructions
└── deployment.md          # Step-by-step deployment guide
```

2. End-to-End ML Project Roadmap & Tracking

This section details the nine phases of the project, including the specific actions taken for the Student Dropout Risk Prediction task and the professional tools used for tracking and reproducibility.

Phase 1: Problem Definition & Planning

Goal	Dropout Project Specifics	Success Tracking & Tools
Business Goal	Reduce student dropout rate by 20% through proactive intervention.	Define business goal, document in **Confluence/Notion**.
ML Type	**Binary Classification** (Dropout = 1, Continue = 0).	**GitHub Wiki** documentation.
Success Metric	**F1-score** or **ROC-AUC** (Target: ROC-AUC > 0.85).	Log criteria in `configs/training_config.yaml`.
Constraints	Predictions must be highly interpretable (explainable) for intervention teams.	Document assumptions & constraints in `docs/constraints.md`.

Phase 2: Data Collection & Versioning

Goal	Dropout Project Specifics	Success Tracking & Tools
Source	Open datasets (UCI Student Performance) or Synthetic generation.	Log source, date, and schema in `data/schema.json`.

Features	Demographics, Academics (Grades, Attendance), Behavior (Disciplinary records).	Ensure scripts are in `src/data/`.
Storage	Initial storage as CSV files.	Data stored in `data/raw/`.
Versioning	Use Data Version Control (DVC) to track the raw dataset.	Use **DVC** (`dvc add data/raw/`) for versioning.

Phase 3: Data Preprocessing & Cleaning

Goal	Dropout Project Specifics	Success Tracking & Tools
Cleaning	Handle missing values (median/mode imputation), duplicates, and outliers.	Keep scripts in `src/preprocessing/`.
Engineering	OneHotEncoding for categories; MinMaxScaler/StandardScaler for continuous features.	Use a reproducible pipeline (e.g., Scikit-Learn Pipeline).
Imbalance	Handle class imbalance using **SMOTE** on the training set or via model **class weights**.	Log transformations and class distribution using **MLflow/W&B**.
Splitting	Train/Test Split (80-20), ensuring stratification on the target variable.	Store processed data in `data/processed/` (DVC-tracked).
Testing	Validate preprocessing logic (e.g., no data leakage).	Write unit tests in `tests/test_preprocessing.py`.

Phase 4: Exploratory Data Analysis (EDA)

Goal	Dropout Project Specifics	Success Tracking & Tools
Visualization	Generate **Correlation heatmap** of features vs. dropout. Distribution plots for attendance and grades.	Save plots and summary in `reports/eda_report.html`.
Biases/Trends	Detect potential biases (e.g., income, gender). Analyze feature distributions.	Document findings in the EDA report.
Scripts	Keep analysis scripts versioned.	Analysis performed in `notebooks/` and versioned via Git.

Phase 5: Model Selection & Training

Goal	Dropout Project Specifics	Success Tracking & Tools
Model	**LightGBM Classifier** (for speed and robust performance on structured data).	Training script in `src/models/train_model.py`.
Tuning	Hyperparameter search using **GridSearchCV / Optuna** for optimization.	**MLflow / W&B** to log experiments (hyperparameters, metrics, training time).
Code		

```
import lightgbm as lgb
# ... (rest of the code)
model = lgb.LGBMClassifier(n_estimators=500, learning_rate=0.05, max_depth=7)
model.fit(X_train, y_train)
```

| Log training environment and dependencies. |

Phase 6: Model Evaluation

Goal	Dropout Project Specifics	Success Tracking & Tools
Metrics	Compare models based on **ROC-AUC**, F1-score, Precision, and Recall.	Log confusion matrix and ROC curves in **MLflow / W&B**.
Generalization	Use k-fold cross-validation during tuning to confirm model stability.	Save evaluation reports in `reports/model_comparison.md`.
Artifacts	Save the final, best-performing model.	Save model to `models/dropout_model.pkl`. Use **DVC** to version the model file.
Testing	Ensure the model output is correct given various inputs.	Write prediction tests in `tests/test_model.py`.

Phase 7: Deployment Preparation (API Development)

Goal	Dropout Project Specifics	Success Tracking & Tools
API Framework	**FastAPI** to create a high-performance, asynchronous prediction service.	API code in `src/api/app.py`.
Endpoint	Define a `POST /predict` endpoint that accepts a JSON payload of features and returns the dropout risk (0 or 1).	Document API specification in `docs/api_spec.yaml`.

API Code

```
from fastapi import FastAPI
import joblib
# ... (imports)
```

```
app = FastAPI()
model = joblib.load("models/dropout_model.pkl")

@app.post("/predict")
def predict(features: dict):
    # ... preprocessing and prediction logic
    prediction = model.predict(df)[0]
    return {"dropout_risk": int(prediction)}
```

| Ensure `requirements.txt` is up-to-date for reproducibility. |

Phase 8: Deployment & CI/CD

Goal	Dropout Project Specifics	Success Tracking & Tools
Containerization	**Docker** to package the FastAPI app, model, and all dependencies.	**Dockerfile** in root directory.
Deployment Target	**AWS ECS (Fargate)** for scalable, managed container orchestration.	Deployment steps detailed in `deployment.md`.
CI/CD	Use **GitHub Actions** to automate testing, Docker image building, and pushing to **Amazon ECR**.	CI/CD pipeline definition in `.github/workflows/ci.yml`.
Build & Run	`docker build -t dropout-api .` and `docker run -p 8080:8080 dropout-api`.	Version Docker images (`dropout-api:1.0`).

Phase 9: Monitoring & Maintenance

Goal	Dropout Project Specifics	Success Tracking & Tools

Monitoring	Track model drift, data drift, API latency, and error rates in production.	Use **AWS CloudWatch** for logs and **Prometheus/Grafana** for dashboarding. Monitoring logic in `src/monitoring/`.
Data Logging	Log all prediction inputs and outputs (with timestamps) for future labeling and retraining.	Prediction logs saved to a dedicated S3 bucket.
Retraining Plan	Schedule quarterly retraining or trigger retraining when ROC-AUC drops by 5%.	Maintain schedule in `docs/retraining_plan.md`.
UI	Use **Streamlit** (code in `src/ui/`) for a simple, non-technical dashboard to visualize risk and monitoring metrics.	Log model updates and versions in the monitoring dashboard.

This comprehensive documentation covers the entire lifecycle of the ML project, from the initial business definition to production monitoring. Let me know if you would like to start drafting the code for a specific file, like `src/api/app.py` or `src/models/train_model.py`!