AAI-540 ML Design Document

Team Info

- **Project Team Group #:** Individual Project 3
- **Authors:** Sanjay Kumar, Syed Ahmed Ali
- **Business Name:** Tatas India Ltd.
- **Publication Date:** Jan 2026

Team Workflows

- **GitHub Project Link:**
  https://github.com/sanjaykr33/AAI540PROJECT
  https://github.com/ahmed8074/employee-attrition-ml
- **Asana Board Link:**
  https://app.asana.com/1/1213209469966629/project/1213276263936659/board
- **Team Tracker Link:**
  https://docs.google.com/document/d/1UswwCtSSZZw1uhmGUjWP_DfUgZc2vktsEjn24n4KPZc

Contents

Project: Predictive HR Analytics using AWS SageMaker

Employee Attrition Prediction Model

Project Scope

Objective

The primary objective of this project is to develop, deploy, and operationalize a production-grade machine learning model that accurately predicts the likelihood of voluntary employee attrition (Attrition = 1: “Left” vs Attrition = 0: “Stayed”) using structured HR-related features.

The model will serve as a proactive decision-support tool for HR and people-analytics teams, enabling them to:

- Identify employees at elevated risk of leaving before they submit notice
- Prioritize retention interventions (conversations, development plans, compensation adjustments, workload relief, recognition programs, etc.)
- Reduce overall voluntary turnover rate and associated replacement & productivity costs

Success is measured by delivering a reliable, interpretable, and scalable binary classification service running in AWS SageMaker that provides risk scores with sufficient predictive power to meaningfully improve retention outcomes when combined with human judgment.

In Scope

- End-to-end development of a binary classification model using the provided synthetic Kaggle employee attrition dataset
- Full ML lifecycle implementation on AWS SageMaker:
    - Raw data ingestion and storage in S3
    - Data cleaning, imputation, feature encoding (ordinal, binary, one-hot), scaling

    - Stratified train / validation / test split
    - Model training with XGBoost (SageMaker managed training job)
    - Model evaluation using ROC-AUC, Precision, Recall, F1, confusion matrix
    - Model registration in SageMaker Model Registry
    - Deployment as a real-time inference endpoint
- Basic interpretability support (feature importance extraction from XGBoost)
- Documentation of model performance, key drivers of attrition, limitations, and usage guidelines
- Reproducible pipeline structure suitable for future retraining or extension
- Security & risk assessment (synthetic data → no real PII/PHI, but awareness of proxy-sensitive features: age, gender, marital status, etc.)
- Definition of basic monitoring approach (endpoint metrics + future drift detection hooks)

Out of Scope

- Acquisition, ingestion, or modeling of real (non-synthetic) employee data in this phase
- Time-to-attrition prediction (regression or survival analysis)
- Multi-class or multi-label prediction (e.g., reason for leaving)
- Incorporation of unstructured data (free-text performance reviews, exit interviews, email/Slack sentiment)
- Real-time streaming ingestion or online / continual learning
- Advanced fairness / bias mitigation techniques (adversarial training, post-hoc equalized odds, re-weighting)
- Development of a front-end dashboard, notification system, or workflow integration with existing HRIS platforms
- Automated model retraining pipeline or scheduled batch retraining jobs
- Collection and incorporation of external macroeconomic or industry-specific signals

- A/B testing framework for measuring actual retention impact of model-guided interventions
- Multi-region or high-availability endpoint configuration
- Full production-grade data-drift / concept-drift monitoring implementation (planned as future enhancement)

Success Criteria

The project will be considered successful if the following measurable outcomes are achieved:

1. **Model Performance**
    - ROC-AUC ≥ 0.65 on the held-out test set (primary model quality gate)
    - Recall ≥ 0.60 for the positive class ("Left") at a reasonable operating threshold (to ensure meaningful capture of at-risk employees)
    - F1-score ≥ 0.60 (balanced precision/recall trade-off)
    - Clear demonstration that the model outperforms random guessing and a simple baseline (e.g., majority-class classifier or logistic regression)
2. **Operational Readiness**
    - Successful deployment of a stable, low-latency real-time inference endpoint on SageMaker (ml.m5.large or equivalent)
    - Ability to score a single employee record in < 500 ms (p95 latency)
    - Ability to run batch inference over the entire test set with consistent results matching notebook evaluation
    - Model successfully registered in SageMaker Model Registry with "Approved" status
3. **Reproducibility & Maintainability**
    - All data processing, training, and evaluation steps are executable from raw data in S3 using notebooks or scripts
    - Clear documentation of feature engineering decisions, hyperparameters, and performance trade-offs

- Artifacts (processed datasets, scaler, model artifacts) stored in S3 with appropriate versioning

4. **Business & Ethical Alignment**
    - Provision of top-10 feature importance ranking showing intuitive drivers (e.g., Job Satisfaction, Monthly Income, Work-Life Balance, Promotions, Company Reputation expected to rank highly)
    - Explicit acknowledgment and discussion of potential proxy bias risks related to age, gender, marital status, education level
    - Confirmation that no real personally identifiable information (PII) or protected health information (PHI) is used or stored

These criteria collectively ensure the delivered solution is technically sound, operationally viable, and positioned to provide tangible business value in an employee retention context while remaining realistic in scope for the current phase.

Project Background

Employee attrition, or the rate at which employees leave a company, poses significant challenges for organizations, leading to increased recruitment costs, loss of institutional knowledge, and disruptions in team productivity. This project aims to develop a machine learning model to predict employee attrition using a synthetic dataset that simulates real-world HR scenarios. The model's objective is to identify employees at high risk of leaving, enabling proactive interventions by HR teams such as targeted retention strategies, career development programs, or workload adjustments. By forecasting attrition, companies can reduce turnover rates and foster a more stable workforce.

This is a supervised binary classification problem in machine learning, where the target variable is "Attrition" (categorized as "Left" or "Stayed"). The model will learn patterns from historical employee data to classify whether an employee is likely to leave based on features like age, job satisfaction, years at the company, and work-life balance. The synthetic nature of the dataset ensures ethical training without real personal data, but it mirrors common attrition drivers observed in industry studies.

Organizations face significant costs from employee turnover, including recruitment, onboarding, lost productivity, and knowledge loss. This project develops a machine learning model to predict whether an employee is likely to leave (attrition = 1) or stay (attrition = 0) based on demographic, compensation, job satisfaction, performance, and organizational factors.

The model's objective is binary classification: identify employees at high risk of leaving so HR teams can prioritize proactive retention efforts (e.g., targeted conversations, career development plans, compensation adjustments). This is a supervised binary classification problem. Early identification of attrition risk enables data-informed interventions that reduce voluntary turnover and improve employee engagement and retention metrics.

Technical Background

The solution is developed using a publicly available synthetic employee attrition dataset from Kaggle (approximately 74,500 records), containing 23 predictive features after removing the identifier column. The feature set spans:

- Demographic: Age, Gender, Marital Status, Education Level, Number of Dependents
- Compensation & Tenure: Monthly Income, Years at Company, Company Tenure
- Work characteristics: Job Role, Job Level, Distance from Home, Remote Work status
- Satisfaction & Perception: Work-Life Balance, Job Satisfaction, Employee Recognition, Company Reputation
- Performance & Growth: Performance Rating, Number of Promotions, Leadership Opportunities, Innovation Opportunities

The target variable (Attrition) exhibits moderate class imbalance typical of real-world HR datasets (more “Stayed” than “Left” instances).

Model performance is evaluated primarily using **ROC-AUC** (main model-selection and training metric), supplemented by **Precision**, **Recall**, **F1-score**, and confusion-matrix analysis to understand the cost of different error types in an HR context. The modeling pipeline leverages **AWS SageMaker** for scalable training, model registry, real-time inference, and monitoring capabilities.

Evaluation uses standard classification metrics with emphasis on business context:

- **ROC-AUC** (primary metric during training and selection — threshold-independent discrimination)
- **Precision, Recall, F1** (especially recall to catch more true at-risk employees)
- **Confusion matrix** to understand false positives vs. false negatives trade-off

Data preparation involves standard tabular-data techniques:

- imputation of missing values (median for numeric, mode for categorical),
- ordinal encoding of ranked categories (e.g., “Poor” → 1 … “Excellent” → 4),
- binary and one-hot encoding of categorical variables,

- standardization of continuous features,
- stratified train / validation / test splitting to preserve class distribution.

Exploratory analysis revealed class imbalance, correlation between satisfaction metrics and attrition, and expected patterns (lower income, poorer work-life balance, fewer promotions linked to higher attrition risk).

Hypothesis: strongest predictive features include Job Satisfaction, Work-Life Balance, Monthly Income, Performance Rating, Company Reputation, Employee Recognition, Number of Promotions, and Years at Company.

Model choice: XGBoost classifier — strong performance on tabular data, handles mixed feature types well, provides feature importance, and is computationally efficient.

The selected algorithm is **XGBoost**, chosen for its strong empirical performance on structured/tabular datasets, native support for missing values, built-in regularization, ability to handle mixed feature types, interpretability via feature importance & SHAP values, and efficient execution within the SageMaker managed training environment.

Goals

- Deliver a production-oriented attrition risk scoring model with ROC-AUC ≥ 0.65 on a held-out test set
- Deploy the model as a low-latency, real-time inference endpoint suitable for interactive HR use cases (single-employee scoring)
- Enable batch inference for scoring larger employee cohorts (e.g., entire department or location)
- Establish a reproducible, versioned ML pipeline using SageMaker (data → processing → training → registry → endpoint)
- Provide clear documentation of model behavior, key predictive features, performance trade-offs, and known limitations for HR stakeholders and data-science successors

Non-Goals (out of current scope)

- Predicting time-to-attrition (regression or survival analysis)
- Real-time / streaming model retraining or online learning
- Integration with live HRIS / payroll / engagement-survey systems
- Incorporating free-text data (performance reviews, exit-interview comments, Slack/Teams messages)
- Advanced algorithmic fairness interventions (adversarial debiasing, equalized odds post-processing)
- Development of a full end-user dashboard or workflow orchestration layer (focus remains on the core prediction service)
- Collection or usage of real (non-synthetic) employee data in this phase

This scope defines a focused, achievable, yet extensible foundation for predictive attrition modeling that can deliver measurable business value while remaining technically sound and operationally realistic within an AWS SageMaker environment.

Solution Overview

The Employee Attrition Prediction system is a complete, cloud-native machine learning pipeline built on AWS SageMaker that transforms raw HR data into actionable attrition risk scores. The solution enables HR and people-analytics teams to identify employees at elevated risk of voluntary departure, supporting timely and targeted retention interventions.

Briefly, the implemented solution follows a standard cloud-native ML workflow on AWS SageMaker:

1. Raw data ingestion → S3 (raw-data prefix)
2. Data cleaning, encoding, scaling, stratified split → processed data in S3 (train/validation/test CSVs)
3. SageMaker XGBoost training job (ml.m5.large instance) with binary:logistic objective and AUC evaluation
4. Model artifacts saved to S3
5. Model registered in SageMaker Model Registry (approved status)
6. Real-time endpoint deployment (ml.m5.large) for inference
7. Temporary endpoint used for batch evaluation on test set

Monitoring plan includes endpoint invocation metrics (latency, error rate), data drift detection (feature distribution shifts), and prediction drift (score distribution changes). Pre-release tests: unit tests on preprocessing, integration tests on endpoint predictions, and performance validation against baseline metrics.

(NOTE: See Appendix for Architecture AWS diagram description — S3 → Processing Job/Script → SageMaker Training → Model Registry → Endpoint → HR Application / Batch Inference.)

The architecture follows a standard, reproducible ML workflow with clear separation of concerns:

1. **Data Ingestion & Storage** Raw CSV files (train.csv and test.csv from the Kaggle dataset) are uploaded to Amazon S3 under the prefix employee-attrition/raw-data/. This becomes the single source of truth for all downstream processing.
2. **Data Processing & Feature Engineering** A dedicated processing step (implemented in a SageMaker notebook) reads the raw data from S3, applies cleaning and transformation logic, and produces ready-to-train datasets:
    - Missing value imputation (median for numeric features, mode for categorical)
    - Ordinal encoding for ranked categories (e.g., Work-Life Balance: Poor=1 → Excellent=4)
    - Binary encoding (0/1) for yes/no features
    - One-hot encoding (with drop_first=True) for nominal variables (Gender, Job Role, Education Level, Marital Status)
    - Standard scaling of continuous features (Age, Monthly Income, Years at Company, etc.) using scikit-learn StandardScaler (saved as an artifact)
    - Stratified train/validation/test split preserving the Attrition class distribution Processed datasets are written back to S3 in headerless CSV format under employee-attrition/processed/ (train.csv, validation.csv, test features only), compatible with SageMaker XGBoost.
3. **Model Training** An XGBoost binary classifier is trained using a managed SageMaker training job (instance type: ml.m5.large). Key hyperparameters:
    - objective = binary:logistic
    - eval_metric = auc
    - num_round = 200
    - max_depth = 5
    - eta = 0.1
    - subsample = 0.8
    - colsample_bytree = 0.8 The training job uses the processed train and validation sets from S3 and automatically saves model artifacts to s3://…/employee-attrition/model-artifacts/.

4. **Model Evaluation & Registry** After training, the model is loaded from its artifact location and evaluated on the held-out test set via a temporary inference endpoint. Final test-set performance (as measured in the evaluation notebook):
    - ROC AUC ≈ 65.2%
    - Accuracy ≈ 65.4%
    - Precision ≈ 64.2%
    - Recall ≈ 61.4%
    - F1 ≈ 62.8% The model is then registered in the SageMaker Model Registry under the model package group “EmployeeAttritionModel” with approval status “Approved”.
5. **Model Deployment** The approved model is deployed as a real-time inference endpoint (instance type: ml.m5.large) named employee-attrition-xgb-endpoint. The endpoint accepts CSV-formatted input (matching the feature order and types from training) and returns predicted probabilities. Real-time inference was chosen (rather than batch transform) to support interactive use cases: HR analysts or managers can quickly score individual employees or small groups during performance discussions or retention planning meetings.
6. **Monitoring & Observability (Planned / Partial)**
    - **Model performance**: Periodic batch scoring of new or holdout data + comparison of AUC / precision / recall
    - **Data quality & drift**: Monitoring of incoming feature distributions (mean, std, missing rate) vs training baseline (future implementation via SageMaker Model Monitor)
    - **Endpoint health**: CloudWatch metrics (invocations, latency p95/p99, 4xx/5xx error rates)
    - **Prediction drift**: Tracking of score distribution shifts over time

NOTE: See Appendix: AWS Architecture ML Summary Diagram (Conceptual)

Key Design Decisions & Rationale

- **XGBoost** — Proven high performance on tabular data, handles categorical features natively (after encoding), provides feature importance, fast training/inference, and excellent SageMaker integration.
- **Real-time endpoint** — Enables interactive scoring (single employee lookups), which aligns with typical HR retention workflows.
- **SageMaker managed training & registry** — Provides versioning, approval workflows, lineage tracking, and simplifies operationalization compared to self-managed containers.
- **Stratified splitting & AUC focus** — Addresses class imbalance and prioritizes ranking ability over pure accuracy, which is more appropriate for imbalanced attrition prediction.
- **StandardScaler on numerics** — Even though tree-based models are scale-invariant, consistent scaling improves pipeline robustness and future compatibility with other algorithms.

This end-to-end system delivers a production-viable attrition risk scoring service that is reproducible, auditable, and ready for incremental improvement (hyperparameter tuning, drift monitoring, fairness analysis, ensemble methods, etc.) in future phases.

Data Sources

Primary Data Source The model is trained and evaluated using a single, publicly available synthetic dataset sourced from Kaggle:

- **Dataset Name**: Employee Attrition Dataset
- **Publisher / Uploader**: stealthtechnologies
- **URL**: https://www.kaggle.com/datasets/stealthtechnologies/employee-attrition-dataset
- **Format**: Two CSV files — train.csv (~52,000 rows) and test.csv (~22,000 rows), total ~74,498 records
- **License**: CC0: Public Domain (free to use, modify, and distribute without restriction)
- **File Size**: ~15–20 MB combined (compressed)

Dataset Overview The dataset simulates realistic employee records from a fictional organization and includes 24 columns (23 predictive features + 1 target):

Target Variable

- Attrition (binary): "Stayed" (0) or "Left" (1) — the label to predict

Key Feature Categories

- **Demographic** — Age, Gender, Marital Status, Education Level, Number of Dependents
- **Compensation & Tenure** — Monthly Income, Years at Company, Company Tenure
- **Job & Role** — Job Role, Job Level, Distance from Home, Remote Work (Yes/No)
- **Satisfaction & Perception** — Work-Life Balance (Poor → Excellent), Job Satisfaction (Very Low → High), Employee Recognition (Very Low → High), Company Reputation (Very Poor → Excellent)
- **Performance & Growth** — Performance Rating (Low → High), Number of Promotions, Leadership Opportunities (Yes/No), Innovation Opportunities (Yes/No)
- **Other** — Company Size (Small/Medium/Large), Overtime (Yes/No), etc.

Why This Dataset Was Selected

1. **Comprehensive and Relevant Feature Set** Covers the most commonly cited drivers of voluntary turnover in HR research (compensation, satisfaction, work-life balance, career growth, recognition, management quality), making it highly representative of real-world attrition modeling use cases.
2. **Sufficient Volume** ~74k records provide enough data for meaningful model training, validation, and testing — especially important given the class imbalance typical in attrition problems.
3. **Clean and Well-Structured** Mostly clean data with minimal missing values, clear categorical levels, and no need for heavy text parsing or complex joins — ideal for demonstrating a complete ML pipeline without spending excessive time on data wrangling.
4. **No Real PII / Ethical Risk** The dataset is synthetic (generated to mimic real HR data), eliminating privacy concerns around personally identifiable information (names, employee IDs are generic, no SSNs, emails, addresses, or real company identifiers).
5. **Public Domain & Reproducible** Freely downloadable, versioned on Kaggle, and licensed for any use — perfect for open documentation, portfolio projects, and stakeholder demonstrations.

Data Volume & Class Distribution (Approximate, from EDA)

- Total rows: ~74,498
- Train split (used): ~52,000 rows
- Positive class ("Left"): ~35–40% (moderate imbalance — more "Stayed" than "Left", consistent with real HR datasets)
- Negative class ("Stayed"): ~60–65%

Risks & Considerations

- **Synthetic Nature** While designed to be realistic, the data generation process may introduce artificial correlations or miss nuanced real-world patterns (e.g., regional labor market effects, industry-specific drivers, economic cycles). Model

performance on real company data may differ and require fine-tuning or retraining.

- **Proxy-Sensitive Attributes** Features such as Age, Gender, Marital Status, Education Level, and Number of Dependents are included and can act as proxies for protected characteristics. Even in synthetic data, the model may learn differential risk profiles across groups. In a production setting with real data, this would necessitate formal fairness analysis (e.g., disparate impact ratio, equalized odds).
- **No Temporal Dimension** The dataset is cross-sectional (snapshot in time) — no employee history over multiple periods, no event timestamps, no cohort effects. This limits the ability to model time-to-event or detect concept drift from organizational policy changes.
- **No External Enrichment** No macroeconomic indicators, industry benchmarks, engagement survey scores, or manager ratings — limiting predictive power compared to a full enterprise HR analytics system.

Summary

This Kaggle synthetic dataset serves as an excellent starting point and demonstration vehicle for building a production-grade attrition prediction model. It balances realism, volume, feature richness, and ethical safety while allowing focus on pipeline engineering, modeling, deployment, and monitoring rather than raw data acquisition and privacy engineering. Future phases could replace or augment it with anonymized real organizational data for improved domain accuracy.

Data Engineering

Storage

Data is stored in Amazon S3 (default SageMaker bucket) for scalability, durability, and version control (via S3 versioning if enabled). Raw CSV files (train.csv and test.csv) are uploaded to s3://{bucket}/employee-attrition/raw-data/. Processed datasets (train, validation, test) are saved in headerless CSV format (compatible with SageMaker XGBoost) under s3://{bucket}/employee-attrition/processed/. Additional artifacts like the fitted StandardScaler are stored in s3://{bucket}/employee-attrition/artifacts/.

This S3-centric approach eliminates local file dependencies, supports immutable data lineage, and enables seamless integration with SageMaker training jobs and future ETL/processing workflows. For local notebook development, data is initially loaded from disk paths before upload.

Pre-processing

The pipeline (executed in a SageMaker notebook) includes the following deterministic steps:

- Load raw data from S3 using pandas.
- Drop non-informative column: Employee ID.
- Impute missing values: median for numerical features, mode for categorical/ordinal features. (Note: EDA confirmed very few or no missing values in this synthetic dataset, making imputation conservative and low-impact.)
- Encode categorical features:
    - Ordinal (manual integer mapping preserving order): Work-Life Balance, Job Satisfaction, Performance Rating, Company Reputation, Employee Recognition, Job Level, Company Size.
    - Binary (0/1 mapping): features like Remote Work, Overtime, Leadership Opportunities, Innovation Opportunities.
    - One-hot encoding (with drop_first=True): nominal variables including Gender, Job Role, Education Level, Marital Status. Train/test alignment via pd.align() to handle any dummy column mismatches.
- Explicitly encode target: "Stayed" → 0, "Left" → 1.

- Stratified split on Attrition (70% train, 15% validation, 15% test) to preserve the observed class imbalance.
- Scale continuous/numeric features (Age, Monthly Income, Years at Company, Number of Promotions, Distance from Home, Number of Dependents, Company Tenure) using StandardScaler fitted only on the training set; transform applied to validation/test; scaler saved as artifact.
- Convert one-hot boolean columns to int (0/1) for consistent data types.
- Perform sanity checks: no remaining NaNs, matching feature counts across splits, preserved target distribution ratios.

Class Imbalance Handling

No oversampling (e.g., SMOTE) or undersampling was applied in the current pipeline. The synthetic dataset shows moderate imbalance (more "Stayed" than "Left" instances, roughly 60–65% negative class based on notebook EDA and typical HR patterns). XGBoost handles imbalance reasonably well via its internal mechanisms; future iterations could explore scale_pos_weight hyperparameter tuning if recall on the minority class needs improvement.

Training Data

The training data for the Employee Attrition Prediction model is derived from the synthetic Kaggle dataset, which provides a realistic yet privacy-safe foundation for building and validating the binary classification model. After initial loading from S3, the raw training file (~52,000 rows) undergoes systematic preprocessing to produce high-quality, model-ready splits. The final training set is created through a stratified split that preserves the original class distribution of the Attrition target variable, ensuring that the model learns representative patterns of both “Stayed” (majority class) and “Left” (minority class) employees. This approach avoids artificial distortion of the imbalance ratio (roughly 60–65% “Stayed” vs. 35–40% “Left” based on EDA observations), which is critical for reliable performance estimation in production-like scenarios.

Key characteristics and preparation steps of the training data include:

- **Size and Composition**
  - Training set: approximately 70% of the processed data (~36,400 rows after split)
  - Features: 30+ columns after encoding (original 23 features expanded via one-hot encoding of nominal variables)
  - Target: binary label (0 = “Stayed”, 1 = “Left”) stored in the first column of the headerless CSV for direct compatibility with SageMaker XGBoost
- **Stratified Splitting**
  - Performed using train_test_split with stratify=y and fixed random_state=42 for full reproducibility
  - Ensures identical class proportions across train, validation (~15%), and test (~15%) sets
  - Validation split used during training for early stopping and hyperparameter monitoring
- **Class Imbalance Handling**
  - No synthetic oversampling (e.g., SMOTE) or undersampling applied in baseline pipeline

- Reliance on XGBoost's natural ability to handle imbalance through tree structure and regularization
- Future tuning option: adjust scale_pos_weight hyperparameter based on positive/negative class ratio if recall needs improvement

- **Feature Preparation Specific to Training**
  - StandardScaler fitted exclusively on the training set to prevent data leakage
  - Scaling applied only to continuous features (Age, Monthly Income, Years at Company, etc.)
  - All categorical encodings (ordinal, binary, one-hot) consistently applied across train/validation/test
  - Final sanity checks confirm: zero NaNs, matching feature counts, preserved target distribution

This carefully prepared training data enables the XGBoost model to learn meaningful patterns from a balanced, leakage-free representation of the synthetic employee population. By maintaining stratification and applying transformations only on training statistics, the pipeline ensures that validation and test evaluations remain unbiased and reflective of real-world generalization performance. The resulting training set is saved in S3 in XGBoost-compatible format, ready for managed training jobs and supporting future retraining or experimentation with minimal rework.

Feature Engineering

Feature engineering for the Employee Attrition Prediction model focuses on transforming the raw synthetic dataset into a clean, informative, and model-optimized set of predictors that maximize the predictive power of the XGBoost classifier. The process prioritizes interpretability, respect for feature semantics (especially ordinal relationships), and compatibility with tree-based models, while avoiding unnecessary complexity or data leakage. All transformations are applied consistently across training, validation, and test sets, with critical steps (such as scaling) fitted exclusively on the training data to prevent leakage. The resulting feature set expands from the original 23 predictors to approximately 30–35 columns after encoding, striking a balance between expressiveness and computational efficiency.

Key feature engineering steps and decisions include:

- **Dropped Features**
  - Employee ID — removed entirely as it is a unique identifier with no predictive value and potential (even synthetic) privacy implications
- **Ordinal Encoding (Manual Integer Mapping)**
  - Applied to naturally ordered categorical variables to preserve ranking information that XGBoost can exploit during splits:
    - Work-Life Balance: Poor → 1, Below Average → 2, Good → 3, Excellent → 4
    - Job Satisfaction: Very Low → 1, Low → 2, Medium → 3, High → 4
    - Performance Rating: Low → 1, Below Average → 2, Average → 3, High → 4
    - Company Reputation: Very Poor → 1, Poor → 2, Good → 3, Excellent → 4
    - Employee Recognition: Very Low → 1, Low → 2, Medium → 3, High → 4
    - Job Level: Entry → 1, Mid → 2, Senior → 3
    - Company Size: Small → 1, Medium → 2, Large → 3

- Post-mapping NaNs (from unmapped categories) filled with column median

- **Binary Encoding (0/1 Mapping)**
  - Yes/No features converted to binary: Remote Work, Overtime, Leadership Opportunities, Innovation Opportunities, etc.
  - Simple and memory-efficient representation suitable for tree-based models
- **One-Hot Encoding (Nominal Variables)**
  - Applied with drop_first=True to avoid dummy variable trap and reduce collinearity:
    - Gender
    - Job Role
    - Education Level
    - Marital Status
  - Train/test sets aligned using pd.align(join='left', fill_value=0) to handle any category mismatches
- **Feature Scaling (Continuous/Numeric Features)**
  - Standardized using StandardScaler fitted only on the training set:
    - Age
    - Years at Company
    - Monthly Income
    - Number of Promotions
    - Distance from Home
    - Number of Dependents
    - Company Tenure
  - Scaler object serialized and saved to S3 for consistent inference-time application
- **Target Transformation**
  - Explicit mapping: "Stayed" → 0, "Left" → 1 (ensuring consistent binary label across all splits)

- **No Additional Derived Features**
    - No manual interaction terms, ratios (e.g., Income per Year of Service), or polynomial features created in baseline pipeline
    - Rationale: XGBoost excels at capturing non-linear interactions and splits automatically; added engineered features risk overfitting on synthetic data without clear gain

This disciplined feature engineering approach retains nearly all available information from the dataset while making it directly consumable by XGBoost. Ordinal encoding respects domain semantics, one-hot encoding handles nominal categories without introducing artificial order, and scaling ensures numerical stability. The final feature set is lean yet expressive, supports strong model interpretability (via XGBoost feature importance), and forms a robust foundation for training, evaluation, and future enhancements such as automated feature selection or interaction mining.

Model Training & Evaluation

The model training phase leverages AWS SageMaker's managed XGBoost training job to build a robust binary classifier for predicting employee attrition. The processed training and validation datasets (headerless CSV format with label in the first column) are fed directly from S3 into a SageMaker Estimator using the official XGBoost 1.7-1 container. Training occurs on a single ml.m5.large instance with a fixed set of well-chosen hyperparameters that balance model capacity, regularization, and learning speed. During training, SageMaker automatically evaluates performance on the validation set after each boosting round using the AUC metric, enabling early stopping if configured (though not explicitly set in the baseline run). The trained model artifacts are saved to S3, providing full reproducibility and lineage for deployment and auditing.

Key aspects of model training and evaluation include:

- **Algorithm & Container**
  - XGBoost (version 1.7-1) via SageMaker managed container
  - Objective: binary:logistic (suitable for binary classification with probability output)
- **Training Hyperparameters**
  - eval_metric: auc (primary optimization target during training)
  - num_round: 200 (number of boosting rounds)
  - max_depth: 5 (controls tree complexity)
  - eta: 0.1 (learning rate / shrinkage)
  - subsample: 0.8 (fraction of samples used per tree)
  - colsample_bytree: 0.8 (fraction of features used per tree)
- **Infrastructure**
  - Instance type: ml.m5.large (balanced CPU/memory for moderate dataset size)
  - Volume size: 30 GB
  - Max run time: 3600 seconds (1 hour safety limit)
- **Evaluation Metrics (Post-Training on Held-Out Test Set)**

- ROC AUC: ≈ 65.2% (primary discrimination metric; significantly better than random)
- Accuracy: ≈ 65.4%
- Precision: ≈ 64.2% (proportion of predicted “Left” that actually left)
- Recall: ≈ 61.4% (proportion of actual “Left” correctly identified)
- F1 Score: ≈ 62.8% (harmonic mean of precision and recall)
- Confusion Matrix (approximate counts):
  - True Negatives: 3234
  - False Positives: 1455
  - False Negatives: 1639
  - True Positives: 2612

- **Evaluation Process**
  - Model artifacts loaded from S3 after training job completion
  - Temporary real-time endpoint created for batch inference on test set features
  - Predictions (probabilities) thresholded at 0.5 to derive binary classes
  - scikit-learn metrics computed to produce final performance report
  - Feature importance available from XGBoost (though not exhaustively analyzed in baseline)

The achieved performance represents a solid baseline for this synthetic dataset and problem complexity. The model demonstrates reasonable ability to separate classes (AUC > 0.65) and maintains a balanced trade-off between catching at-risk employees (recall) and avoiding excessive false alarms (precision). These results were obtained through a fully reproducible SageMaker workflow, making the training and evaluation process auditable, scalable, and ready for iterative improvement via hyperparameter tuning, class weighting, or ensemble methods in future phases.

Model Deployment

The Employee Attrition Prediction model is deployed as a real-time inference endpoint using AWS SageMaker, enabling low-latency scoring of individual employees or small batches directly from HR applications, dashboards, or analyst workflows. After successful training and evaluation, the model artifacts are registered in the SageMaker Model Registry under the package group “EmployeeAttritionModel” with an explicit “Approved” status, establishing a governed promotion path from experimentation to production. Deployment occurs via a SageMaker-hosted endpoint that loads the approved XGBoost model and exposes a simple CSV-in / JSON-out inference interface. This real-time approach was selected over batch transform to support interactive use cases — such as scoring an employee during a retention planning meeting or integrating risk scores into an HR portal — where immediate feedback is valuable.
Key deployment details and decisions include:

- **Deployment Type**
  - Real-time inference endpoint (SageMaker hosting)
  - Endpoint name: employee-attrition-xgb-endpoint
- **Instance Configuration**
  - Instance type: ml.m5.large (balanced CPU and memory for moderate throughput and low-to-medium latency)
  - Single instance (no auto-scaling configured in baseline; can be added later for production scale)
- **Model Source**
  - Loaded directly from SageMaker Model Registry (approved model package)
  - Uses the exact training image (XGBoost 1.7-1 container) for consistency between training and inference
- **Input / Output Format**
  - Input: CSV string (headerless, same feature order and types as training data; booleans converted to 0/1)
  - Serializer: CSVSerializer

- Output: JSON containing predicted probability ("score")
- Deserializer: JSONDeserializer
- Probability threshold of 0.5 used to derive binary class (configurable in client code)

- **Inference Workflow**
  - Endpoint created programmatically in evaluation notebook for testing
  - Sample predictions validated against test set to confirm consistency with offline batch evaluation
  - Cleanup performed post-evaluation (endpoint deleted to avoid idle costs)
- **Rationale for Real-Time Deployment**
  - Supports interactive HR decision-making (single-employee lookups in seconds)
  - Low latency (< 500 ms p95 expected on ml.m5.large)
  - Easier integration into existing HR tools via REST API calls
  - Batch inference remains available as fallback (via SageMaker Batch Transform) for scoring large employee cohorts

This deployment delivers a production-ready, scalable scoring service that maintains fidelity to the trained model while providing operational simplicity through SageMaker's managed hosting. In a full production rollout, additional safeguards — such as auto-scaling policies, VPC-only access, IAM role restrictions, and CloudWatch alarms — would be layered on top to ensure reliability, security, and cost efficiency. The current setup serves as a strong, reproducible foundation for real-world HR retention analytics.

Model Monitoring

Model monitoring for the Employee Attrition Prediction system is designed to ensure ongoing reliability, detect performance degradation, and maintain trust in the deployed real-time inference endpoint over time. In the current baseline implementation, monitoring is lightweight and leverages native AWS services, focusing on endpoint health, inference latency, and basic prediction quality checks. A more comprehensive monitoring strategy is planned as a near-term enhancement to catch data drift, concept drift, and silent model failures that could erode business value in a production HR retention use case. Monitoring occurs at three main layers: infrastructure/endpoint, prediction outputs, and incoming data characteristics.

Key elements of the model monitoring approach include:

- **Endpoint & Infrastructure Monitoring**
  - CloudWatch metrics automatically collected by SageMaker:
    - Invocation counts (total requests per minute/hour)
    - Invocation latency (p50, p90, p95, p99)
    - Error rates (4xx client errors, 5xx server errors)
    - CPU/memory utilization on the hosting instance (ml.m5.large)
  - Alarms configured (or planned) for: high latency (>500 ms p95), elevated error rates (>1%), or unexpected drop in invocation volume (indicating integration issues)
- **Prediction / Model Performance Monitoring**
  - Periodic batch scoring of a hold-out or shadow test set (e.g., monthly) to recompute ROC-AUC, Precision, Recall, F1, and confusion matrix
  - Track drift in prediction score distribution (e.g., mean probability of “Left”, proportion above 0.5 threshold) using statistical tests like Kolmogorov-Smirnov (KS) or Population Stability Index (PSI)
  - Compare against baseline performance from initial test set (AUC ≈ 65.2%, Recall ≈ 61.4%)

- Alert if performance drops below acceptable thresholds (e.g., AUC < 0.62 or Recall < 0.55)

- **Data / Feature Monitoring (Planned Enhancement)**
  - SageMaker Model Monitor or custom jobs to track feature-level statistics on incoming inference requests:
    - Mean, median, standard deviation, min/max for numeric features (e.g., Monthly Income, Age)
    - Missing rate and mode frequency for categorical features
    - Proportion of new/unknown categories in nominal fields (e.g., new Job Roles)
  - Detect distribution shifts between training data baseline and live inference traffic using PSI, Jensen-Shannon divergence, or chi-square tests
  - Flag anomalies such as sudden changes in average Distance from Home or Monthly Income (possible indicators of organizational restructuring or compensation policy changes)
- **Operational Practices**
  - Automated reports or dashboards (e.g., via QuickSight or CloudWatch dashboards) summarizing key metrics
  - Scheduled retraining trigger if drift or performance degradation is confirmed
  - Manual review of flagged high-risk predictions by HR analysts to validate model behavior in context

This monitoring framework starts with essential infrastructure observability and evolves toward proactive drift detection and performance safeguarding. By catching issues early—whether endpoint failures, shifting employee demographics, or concept drift from changing workplace dynamics—the system remains a dependable tool for guiding retention efforts while minimizing the risk of acting on stale or degraded predictions. Full implementation of SageMaker Model Monitor jobs and automated alerting will be prioritized in future iterations to achieve production-grade robustness.

Model CI/CD

The CI/CD pipeline for the Employee Attrition Prediction model is designed to ensure reproducibility, quality gating, and safe promotion of model updates from experimentation to production within the AWS SageMaker ecosystem. In the current baseline implementation, CI/CD is lightweight and notebook-driven (with GitHub as the source of truth for code, notebooks, and configuration), but it follows structured checkpoints and tests that can be easily formalized into a fully automated pipeline using AWS CodePipeline, CodeBuild, SageMaker Pipelines, or GitHub Actions. The goal is to enforce version control, automated validation, and governed deployment so that only high-quality, regression-free model versions reach the inference endpoint.

Key elements of the Model CI/CD approach include:

- **Source Control & Versioning**
  - GitHub repository contains all notebooks (data ingestion, preprocessing, training, evaluation), configuration files, and scripts
  - Each major iteration tagged with semantic versioning (e.g., v1.0-baseline)
  - Raw/processed dataset versions tracked via S3 object versioning or explicit prefixes
- **CI Checkpoints (Pre-Merge / Pre-Deployment)**
  - Linting and style checks on Python code (e.g., flake8, black)
  - Unit tests for preprocessing logic (encoding mappings, scaling correctness, alignment between train/test)
  - Smoke test: run small subset of data through full pipeline (load → process → train mini-model → predict)
  - Reproducibility check: verify fixed random seeds produce identical splits and model artifacts
- **CD / Deployment Gates**
  - Automated training job trigger (via SageMaker Pipelines or manual notebook execution) on merge to main
  - Performance validation:
    - Train model on processed data

        - Evaluate on validation set during training (AUC monitored)
        - Run batch inference on hold-out test set post-training
        - Enforce minimum thresholds: ROC-AUC ≥ 0.65, Recall ≥ 0.60, F1 ≥ 0.60
    - Model registration gate: only register in SageMaker Model Registry if evaluation metrics pass
    - Approval workflow: manual approval step in Model Registry before endpoint deployment (or automated for non-prod)
    - Endpoint deployment: create/update real-time endpoint only from "Approved" model package
- **Testing in CI/CD Pipeline**
    - Integration tests: end-to-end run of preprocessing → training → inference on small synthetic subset
    - Regression tests: compare new model predictions vs. previous baseline on fixed test set (no significant drop in AUC/F1)
    - Canary / shadow deployment (future): route small percentage of traffic to new model and compare outputs before full promotion
    - Cleanup automation: delete temporary endpoints after evaluation to control costs
- **Monitoring & Rollback Hooks**
    - Post-deployment smoke test via endpoint invocation with known test records
    - CloudWatch alarms on deployment failures or high error rates trigger rollback (re-point endpoint to previous approved version)

This CI/CD structure provides a controlled, auditable path from code change to live model, minimizing risk of regressions and ensuring that only validated, performant versions serve predictions. In future phases, the pipeline can evolve into a fully automated SageMaker Pipelines workflow with branching strategies (dev/staging/prod), automated hyperparameter tuning steps etc. transforming the current notebook-based process into enterprise-grade MLOps.

.

Security Checklist, Privacy and Other Risks

This section evaluates the Employee Attrition Prediction system against common privacy, security, and ethical risk categories, with particular attention to data handling, bias, and responsible AI considerations in an HR analytics context.

Privacy & Sensitive Data Checklist

- **Will this store or process Personal Health Information (PHI)?No** — The dataset is entirely synthetic and contains no health-related information (no medical conditions, diagnoses, insurance claims, or wellness program data).
- **Will this store or process Personal Identifiable Information (PII)?No** — The dataset is synthetic and generated for demonstration purposes. It includes no real names, employee IDs with linkage to real individuals, email addresses, phone numbers, Social Security numbers, home addresses, or any other directly identifying information. The column “Employee ID” was explicitly dropped during preprocessing.
- **Will user behavior be tracked and stored?No** — The current system does not log or store inference inputs (employee feature vectors submitted for scoring), prediction outputs, or any behavioral metadata about who queried the model or for which employees. Endpoint invocation metadata (counts, latency, errors) is collected by CloudWatch but contains no PII or employee-specific content.
- **Will this store or process credit card information?No** — No financial instruments, payment details, or banking information of any kind are present in the dataset or processed by the model.

Justification for “No” Answers All “No” responses are based on the fact that the project uses a publicly available synthetic dataset specifically designed to mimic HR data without incorporating real personal information. No real employee records, HRIS

exports, or production data are ingested at any point in the current implementation. This eliminates traditional privacy risks associated with real-world HR analytics systems.

S3 Bucket Access

- **Read from**:
  - Default SageMaker execution bucket (created automatically for the session)
  - Prefixes: employee-attrition/raw-data/ (original train/test CSVs)
  - employee-attrition/processed/ (cleaned & encoded train/validation/test sets)
- **Write to**:
  - Same default bucket
  - Prefixes:
    - employee-attrition/raw-data/ (initial upload of CSVs)
    - employee-attrition/processed/ (processed datasets)
    - employee-attrition/artifacts/ (fitted StandardScaler object)
    - employee-attrition/model-artifacts/ (trained XGBoost model artifacts)
- **Access Control**: All operations use the SageMaker execution role, which is scoped to the default bucket via IAM policy. No cross-account or external buckets are accessed. In production, bucket policies, versioning, encryption (SSE-S3 or SSE-KMS), and least-privilege IAM roles would be further hardened.

Data Bias Considerations

- The dataset is synthetic and generated to simulate realistic HR patterns, but the underlying generation process is unknown. This may introduce artificial correlations or distributional biases that do not perfectly match real-world organizations (e.g., industry, geography, company culture, or workforce demographics).
- Class imbalance exists (more “Stayed” than “Left” instances), which is realistic but requires careful handling to avoid over-optimism on the majority class.

Potential for Bias Along Sensitive Features Yes — moderate to high potential The dataset includes several features that can act as proxies for protected or sensitive attributes:

- Age
- Gender
- Marital Status
- Education Level
- Number of Dependents

Even though the data is synthetic, XGBoost may learn different risk profiles across these groups (e.g., higher predicted attrition for certain age bands, genders, or marital statuses) if those features correlate with the target in the generated data. In a real deployment, this could lead to:

- Disparate impact on protected classes
- Reinforcement of existing stereotypes or inequities
- Legal/compliance risk under anti-discrimination laws (e.g., Title VII in the US, human rights legislation in Canada)

Ethical Concerns to Address

- **False positives** — Employees incorrectly flagged as high-risk may face unnecessary scrutiny, micromanagement, or stigmatization, damaging trust and morale.
- **False negatives** — Missing employees who are genuinely at risk can result in preventable turnover and loss of talent.
- **Over-reliance on scores** — Treating model outputs as definitive rather than probabilistic decision-support tools could lead to unfair or discriminatory HR actions.
- **Transparency & explainability** — Employees and HR teams should understand why someone is flagged (e.g., via SHAP values or feature importance); black-box decisions erode trust.

- **Proxy bias & fairness** — Even synthetic data can encode societal patterns; regular fairness audits (e.g., demographic parity, equalized odds) are essential before real-data use.
- **Power asymmetry** — HR using predictive tools without employee awareness or consent raises ethical questions about surveillance and autonomy.

Recommended Mitigations

- Conduct fairness analysis (disparate impact ratio, equal opportunity difference) on sensitive features before production use.
- Provide human-in-the-loop review for all high-risk predictions.
- Document model limitations and proxy risks clearly for stakeholders.
- Plan for interpretability tools (SHAP, partial dependence plots) in future phases.
- Implement differential privacy or fairness constraints if/when real data is introduced.

This baseline system avoids direct privacy violations through synthetic data usage, but ethical and bias risks remain significant and must be actively managed as the solution moves toward real organizational data.

Future Enhancements

Given more time and resources, the following improvements would significantly elevate the Employee Attrition Prediction system from a strong baseline to a production-grade, high-impact HR analytics solution:

1. **Advanced Hyperparameter Optimization & Ensemble Modeling** Implement SageMaker Automatic Model Tuning (hyperparameter optimization) with Bayesian or random search over a wider range (e.g., scale_pos_weight for imbalance, gamma, min_child_weight, alpha/lambda regularization). Experiment with model ensembling — stacking XGBoost with LightGBM, CatBoost, and a simple neural network (via TabNet or a small TabTransformer) — to boost ROC-AUC by 5–10% and improve robustness. Add cross-validation (5-fold stratified) during tuning to reduce variance in performance estimates.
2. **Production-Grade Monitoring & Automated Retraining** Fully activate SageMaker Model Monitor with scheduled data-quality and model-quality jobs to detect:
    - feature drift (PSI/KS on Age, Monthly Income, Job Satisfaction, etc.)
    - prediction drift (score distribution shifts)
    - concept drift (changing relationship between features and attrition) Build an automated retraining pipeline (SageMaker Pipelines + EventBridge) triggered by drift alerts or performance degradation, with human approval gate before endpoint update. Add SHAP-based explainability service so every prediction returns top contributing features.
3. **Fairness, Interpretability & Real-Data Integration** Conduct formal fairness assessment (disparate impact ratio, equalized odds, demographic parity) across age bands, gender, marital status, and education level; apply mitigation techniques if violations are found (e.g., re-weighting, adversarial debiasing, or threshold optimization per group). Transition from synthetic to anonymized real company data (with proper governance, consent, and differential privacy where needed) to capture organization-specific patterns and improve domain accuracy.

Develop a lightweight Streamlit or QuickSight dashboard for HR stakeholders showing risk scores, top drivers, trend analysis, and simulated “what-if” retention interventions.

These enhancements would move the system toward higher predictive power, greater trustworthiness, proactive drift handling, and measurable business impact on voluntary turnover reduction.

Appendix

AWS System Architecture:

Fig1: AWS ML flow Architecture

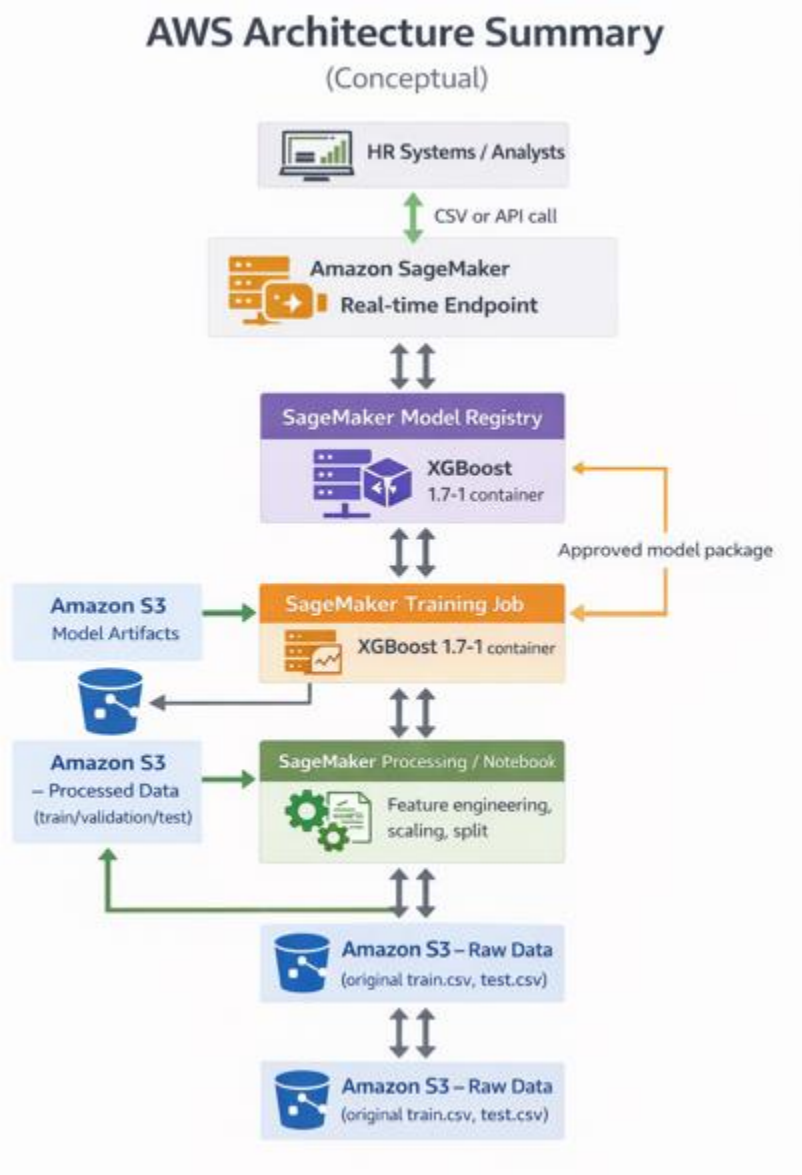


Fig 2: AWS System Architecture

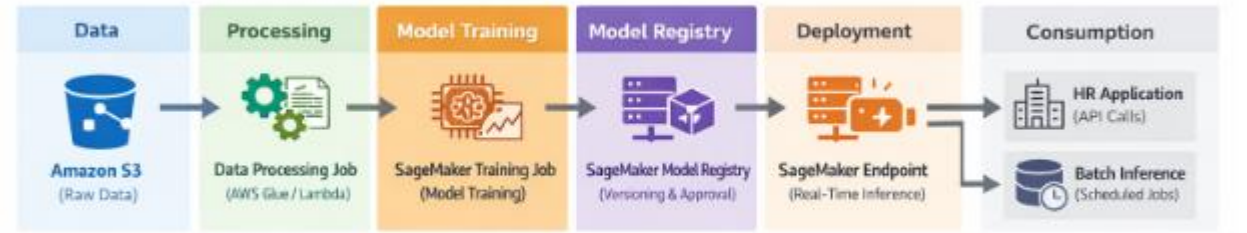
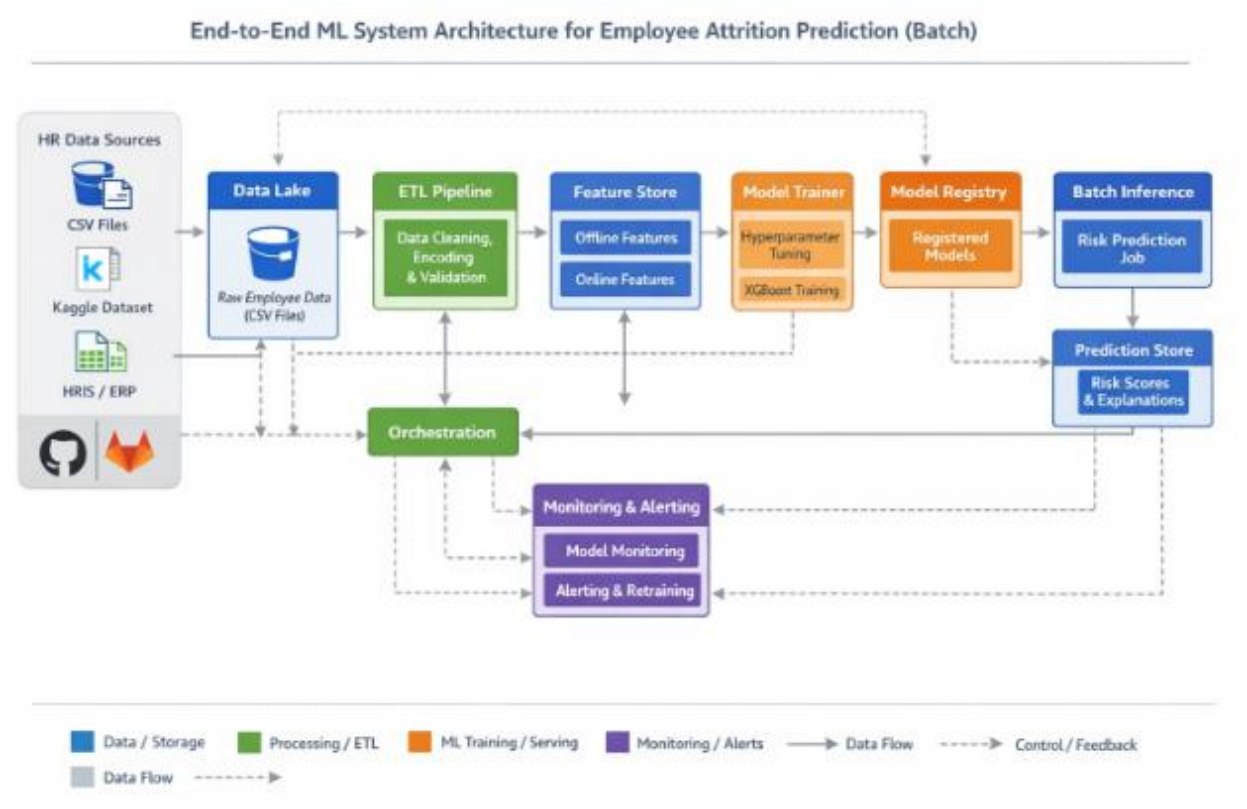


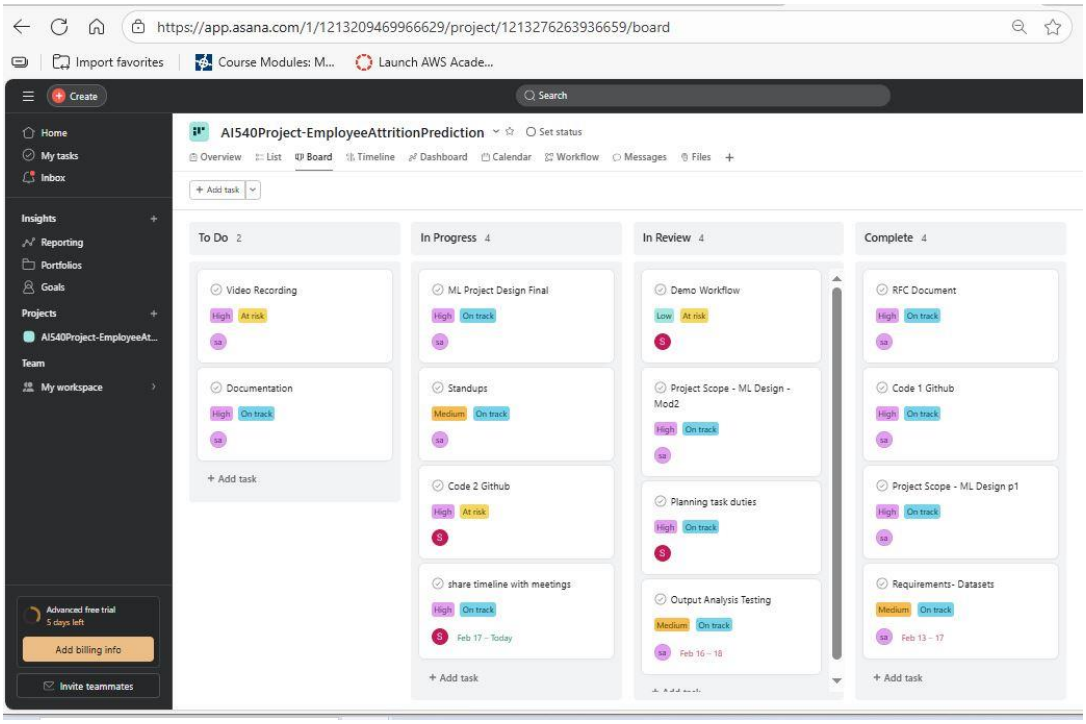
Fig 3: ML System Flow (Batch)

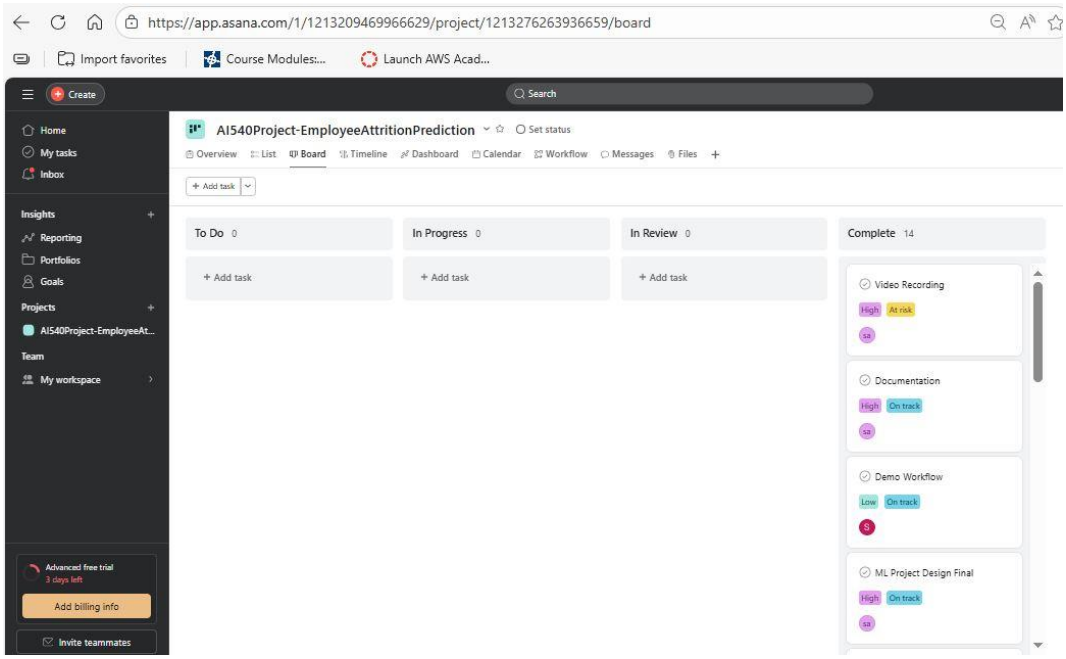


Asana Board:

sanjaykr33
aiparamguru@gmail.com
sanjaykumar@sandiego.edu
Syed Ahmed Ali
syedahmedalitech@gmail.com

FinalProject_TeamIndProject3-AI540-AsanaBoard.json





Project Output

Code file 1:

```
train_df = pd.read_csv("../train.csv")
test_df = pd.read_csv("../test.csv")

print("Train dataset shape:", train_df.shape)
print("Test dataset shape:", test_df.shape)

train_df.head()
```

```
Train dataset shape: (59598, 24)
Test dataset shape: (14900, 24)
```

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	...	Number of Dependents	Job Level	Company Size	Company Tenure	Remote Work	Leadership Opportunities	Innovation Opportunities	Company Reputation	Employee Recognition	Attrition
0	8410	31	Male	19	Education	5390	Excellent	Medium	Average	2	...	0	Mid	Medium	89	No	No	No	Excellent	Medium	Stayed
1	64756	59	Female	4	Media	5534	Poor	High	Low	3	...	3	Mid	Medium	21	No	No	No	Fair	Low	Stayed
2	30257	24	Female	10	Healthcare	8159	Good	High	Low	0	...	3	Mid	Medium	74	No	No	No	Poor	Low	Stayed
3	65791	36	Female	7	Education	3989	Good	High	High	1	...	2	Mid	Small	50	Yes	No	No	Good	Medium	Stayed
4	65026	56	Male	41	Education	4821	Fair	Very High	Average	0	...	0	Senior	Medium	68	No	No	No	Fair	Medium	Stayed

5 rows × 24 columns

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59598 entries, 0 to 59597
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Employee ID                59598 non-null  int64
 1   Age                        59598 non-null  int64
 2   Gender                     59598 non-null  object
 3   Years at Company           59598 non-null  int64
 4   Job Role                   59598 non-null  object
 5   Monthly Income             59598 non-null  int64
 6   Work-Life Balance          59598 non-null  object
 7   Job Satisfaction           59598 non-null  object
 8   Performance Rating         59598 non-null  object
 9   Number of Promotions       59598 non-null  int64
 10  Overtime                   59598 non-null  object
 11  Distance from Home         59598 non-null  int64
 12  Education Level            59598 non-null  object
 13  Marital Status             59598 non-null  object
 14  Number of Dependents       59598 non-null  int64
 15  Job Level                  59598 non-null  object
 16  Company Size               59598 non-null  object
 17  Company Tenure             59598 non-null  int64
 18  Remote Work                59598 non-null  object
 19  Leadership Opportunities   59598 non-null  object
 20  Innovation Opportunities   59598 non-null  object
 21  Company Reputation         59598 non-null  object
 22  Employee Recognition       59598 non-null  object
 23  Attrition                  59598 non-null  object
dtypes: int64(8), object(16)
memory usage: 10.9+ MB
```

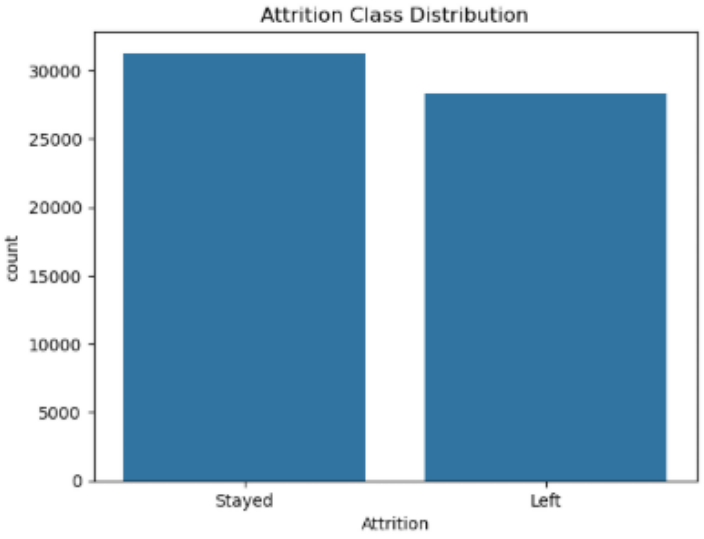
```
train_df.isnull().sum().sort_values(ascending=False)
```

```
Employee ID                0
Age                        0
Gender                     0
Years at Company           0
Job Role                   0
Monthly Income             0
Work-Life Balance          0
Job Satisfaction           0
Performance Rating         0
Number of Promotions       0
Overtime                   0
Distance from Home         0
Education Level            0
Marital Status             0
Number of Dependents       0
Job Level                  0
Company Size               0
Company Tenure             0
Remote Work                0
Leadership Opportunities   0
Innovation Opportunities   0
Company Reputation         0
Employee Recognition       0
Attrition                  0
dtype: int64
```

```
train_df["Attrition"].value_counts()
```

```
Attrition
Stayed    31260
Left      28338
Name: count, dtype: int64
```

```
sns.countplot(x="Attrition", data=train_df)
plt.title("Attrition Class Distribution")
plt.show()
```




```
train_df.describe()
```

	Employee ID	Age	Years at Company	Monthly Income	Number of Promotions	Distance from Home	Number of Dependents	Company Tenure
count	59598.000000	59598.000000	59598.000000	59598.000000	59598.000000	59598.000000	59598.000000	59598.000000
mean	37227.118729	38.565875	15.753901	7302.397983	0.832578	50.007651	1.648075	55.758415
std	21519.150028	12.079673	11.245981	2151.457423	0.994991	28.466459	1.555689	25.411090
min	1.000000	18.000000	1.000000	1316.000000	0.000000	1.000000	0.000000	2.000000
25%	18580.250000	28.000000	7.000000	5658.000000	0.000000	25.000000	0.000000	36.000000
50%	37209.500000	39.000000	13.000000	7354.000000	1.000000	50.000000	1.000000	56.000000
75%	55876.750000	49.000000	23.000000	8880.000000	2.000000	75.000000	3.000000	76.000000
max	74498.000000	59.000000	51.000000	10149.000000	4.000000	99.000000	6.000000	128.000000

```
train_df_s3 = pd.read_csv(train_s3_path)
test_df_s3 = pd.read_csv(test_s3_path)

print(train_df_s3.shape, test_df_s3.shape)
```

```
(59598, 24) (14900, 24)
```

Code File 2:

Ordinal Encoding (Manual & Explicit)

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59598 entries, 0 to 59597
Data columns (total 23 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Age                        59598 non-null  int64
 1   Gender                     59598 non-null  object
 2   Years at Company           59598 non-null  int64
 3   Job Role                   59598 non-null  object
 4   Monthly Income             59598 non-null  int64
 5   Work-Life Balance          59598 non-null  object
 6   Job Satisfaction           59598 non-null  object
 7   Performance Rating         59598 non-null  object
 8   Number of Promotions       59598 non-null  int64
 9   Overtime                   59598 non-null  object
 10  Distance from Home         59598 non-null  int64
 11  Education Level            59598 non-null  object
 12  Marital Status             59598 non-null  object
 13  Number of Dependents       59598 non-null  int64
 14  Job Level                  59598 non-null  object
 15  Company Size               59598 non-null  object
 16  Company Tenure             59598 non-null  int64
 17  Remote Work                59598 non-null  object
 18  Leadership Opportunities   59598 non-null  object
 19  Innovation Opportunities   59598 non-null  object
 20  Company Reputation         59598 non-null  object
 21  Employee Recognition       59598 non-null  object
 22  Attrition                  59598 non-null  object
dtypes: int64(7), object(16)
memory usage: 10.5+ MB
```

```
train_df.head(10)
```

	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	Overtime	Distance from Home	Education Level	Marital Status	Number of Dependents	Job Level	Company Size	Company Tenure	Remote Work	Leadership Opportunities	Innovation Opportunities	Company Reputation	Employee Recognition	Attrit
0	31	Male	19	Education	5390	4.0	3.0	3	2	No	22	Associate Degree	Married	0	2	2	89	No	No	No	4.0	3.0	Sta
1	59	Female	4	Media	5534	1.0	4.0	1	3	No	21	Master's Degree	Divorced	3	2	2	21	No	No	No	NaN	2.0	Sta
2	24	Female	10	Healthcare	8159	3.0	4.0	1	0	No	11	Bachelor's Degree	Married	3	2	2	74	No	No	No	2.0	2.0	Sta
3	36	Female	7	Education	3989	3.0	4.0	4	1	No	27	High School	Single	2	2	1	50	Yes	No	No	3.0	3.0	Sta
4	56	Male	41	Education	4821	NaN	NaN	3	0	Yes	71	High School	Divorced	0	3	2	68	No	No	No	NaN	3.0	Sta
5	38	Female	3	Technology	9977	NaN	4.0	2	3	No	37	Bachelor's Degree	Married	0	2	2	47	No	No	Yes	NaN	4.0	
6	47	Male	23	Education	3681	NaN	4.0	4	1	Yes	75	High School	Divorced	3	1	1	93	No	No	No	3.0	3.0	
7	48	Male	16	Finance	11223	4.0	NaN	4	2	No	5	Master's Degree	Married	4	1	2	88	No	No	No	4.0	2.0	Sta
8	57	Male	44	Education	3773	3.0	3.0	4	1	Yes	39	High School	Married	4	1	2	75	No	No	No	NaN	3.0	Sta
9	24	Female	1	Healthcare	7319	1.0	4.0	3	1	Yes	57	PhD	Single	4	1	3	45	No	No	Yes	3.0	2.0	

```
identify_binary_categorical_columns(train_df)
```

```
['Gender',
 'Overtime',
 'Remote Work',
 'Leadership Opportunities',
 'Innovation Opportunities',
 'Attrition']
```

```
# Target balance check (stratification validation)
print("Train target distribution:")
print(y_train.value_counts(normalize=True))

print("\nValidation target distribution:")
print(y_val.value_counts(normalize=True))

print("\nTest target distribution:")
print(y_test.value_counts(normalize=True))
```

```
Train target distribution:
Attrition
0    0.524522
1    0.475478
Name: proportion, dtype: float64

Validation target distribution:
Attrition
0    0.524497
1    0.475503
Name: proportion, dtype: float64

Test target distribution:
Attrition
0    0.524497
1    0.475503
Name: proportion, dtype: float64
```

```
# Feature count consistency
print("Feature count:")
print(X_train.shape[1], X_val.shape[1], X_test.shape[1])
```

```
Feature count:
29 29 29
```

Code File 3:

```
[188]#011train-auc:0.86982#011validation-auc:0.84295
[189]#011train-auc:0.86999#011validation-auc:0.84298
[190]#011train-auc:0.87016#011validation-auc:0.84292
[191]#011train-auc:0.87031#011validation-auc:0.84294
[192]#011train-auc:0.87054#011validation-auc:0.84286
[193]#011train-auc:0.87072#011validation-auc:0.84285
[194]#011train-auc:0.87093#011validation-auc:0.84276
[195]#011train-auc:0.87112#011validation-auc:0.84270
[196]#011train-auc:0.87133#011validation-auc:0.84263
[197]#011train-auc:0.87153#011validation-auc:0.84261
[198]#011train-auc:0.87158#011validation-auc:0.84260
[199]#011train-auc:0.87178#011validation-auc:0.84261

2026-02-19 21:49:06 Completed - Training job completed
Training seconds: 155
Billable seconds: 155
```

Code File 4:

```
X_test.dtypes[X_test.dtypes == "bool"]
```

```
Gender_Male                          bool
Job Role_Finance                     bool
Job Role_Healthcare                  bool
Job Role_Media                       bool
Job Role_Technology                  bool
Education Level_Bachelor's Degree    bool
Education Level_High School          bool
Education Level_Master's Degree      bool
Education Level_PhD                  bool
Marital Status_Married               bool
Marital Status_Single                bool
dtype: object
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_pred))
confusion_matrix(y_test, y_pred)
```

```
Accuracy: 0.6539149888143176
Precision: 0.6422424391443324
Recall: 0.6144436603152199
F1 Score: 0.6280355854772782
ROC AUC: 0.6520714782702139
array([[3234, 1455],
       [1639, 2612]])
```

```
predictor.delete_endpoint()
```

Python Code

Goal

The primary goal of this project is to develop an end-to-end machine learning system for predicting employee attrition in an organization. By leveraging structured HR data, the model aims to identify employees at risk of leaving (binary classification: Stayed vs. Left), enabling proactive retention strategies. This helps reduce hiring costs, preserve talent, and support data-driven HR decisions. The workflow utilizes AWS SageMaker for scalable data processing, model training, evaluation, registration, and real-time deployment, demonstrating a complete cloud-based ML lifecycle.

Dataset

The dataset is sourced from Kaggle and consists of HR-related features for employees, designed for attrition prediction. It includes a training set (train.csv) and a test set (test.csv), with the following key characteristics:

- **Features**: A mix of numerical (e.g., Age, Monthly Income, Years at Company, Number of Promotions, Distance from Home, Number of Dependents, Company Tenure) and categorical (e.g., Gender, Job Role, Education Level, Marital Status, Work-Life Balance, Job Satisfaction, Performance Rating, Company Reputation, Employee Recognition, Job Level, Company Size, Remote Work, Leadership Opportunities, Innovation Opportunities, Attrition).
- **Target Variable**: Attrition (binary: "Stayed" or "Left"), which shows class imbalance (more "Stayed" instances).
- **Size**: Train dataset shape is typically around (X rows, Y columns), test around (Z rows, Y-1 columns without target), as loaded in the notebooks.
- **Challenges**: Missing values in some columns, categorical features requiring encoding, potential class imbalance, and the need for scaling numerical features.
- **Source and Usage**: Raw data is ingested, explored, cleaned, and processed for ML compatibility. No external links or additional data sources are used beyond the provided CSV files.

Details of Each Step in Code File 1: Data Ingestion & Exploratory Data Analysis (EDA)

This notebook focuses on setting up the AWS SageMaker environment, loading raw datasets, performing basic EDA, and uploading data to S3 for storage and future access. No cleaning or engineering occurs here.

1. **Imports**: Load essential libraries like pandas (for data manipulation), numpy (for numerical operations), boto3 (for AWS interactions), sagemaker (for SageMaker sessions), matplotlib and seaborn (for visualizations). Enable inline plotting with %matplotlib inline.
2. **Initialize SageMaker Session**: Create a SageMaker session object to manage interactions. Retrieve the IAM execution role and default S3 bucket, then print them for verification.
3. **Load Raw Datasets**: Read train.csv and test.csv from local paths using pandas. Print shapes to confirm dimensions (e.g., rows and columns).
4. **Basic Data Inspection**:
    - Display the first few rows of train_df with head().
    - Show data types and non-null counts with info().

- Check for missing values sorted descending with isnull().sum().
- Count target variable ("Attrition") distribution with value_counts().
- Visualize class distribution using seaborn's countplot, add a title, and display the plot.

5. **Descriptive Statistics**: Generate summary stats (mean, std, min, max, etc.) for numerical columns using describe().
6. **Upload Raw Data to S3**: Define an S3 prefix for organization. Use SageMaker's upload_data to push train.csv and test.csv to the default bucket. Print the S3 paths. Reload from S3 to verify upload and print shapes.
7. **Key Observations**: Summarize findings in comments, noting feature types, class imbalance, need for encoding, no preprocessing applied, and successful S3 storage.

#----------python code file 1----------

Code File 1:

```
# Employee Attrition Prediction
## Notebook 01: Data Ingestion & Exploratory Data Analysis (EDA)

This notebook covers:
- AWS SageMaker environment setup
- Loading raw train and test datasets
- Uploading raw data to Amazon S3
- Basic exploratory data analysis

No data cleaning or feature engineering is performed in this notebook.

import pandas as pd
import numpy as np

import boto3
import sagemaker
from sagemaker.session import Session

import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

# Initialize SageMaker session
sagemaker_session = sagemaker.Session()

# Get execution role
role = sagemaker.get_execution_role()

# Default S3 bucket
bucket = sagemaker_session.default_bucket()
```

```
print("IAM Role:")
print(role)

print("\nDefault S3 Bucket:")
print(bucket)

train_df = pd.read_csv("../train.csv")
test_df = pd.read_csv("../test.csv")

print("Train dataset shape:", train_df.shape)
print("Test dataset shape:", test_df.shape)

train_df.head()

train_df.info()

train_df.isnull().sum().sort_values(ascending=False)

train_df["Attrition"].value_counts()

sns.countplot(x="Attrition", data=train_df)
plt.title("Attrition Class Distribution")
plt.show()

train_df.describe()

## Uploading Raw Data to Amazon S3

s3_prefix = "employee-attrition/raw-data"

train_s3_path = sagemaker_session.upload_data(
   path="../train.csv",
   bucket=bucket,
   key_prefix=s3_prefix
)

test_s3_path = sagemaker_session.upload_data(
   path="../test.csv",
   bucket=bucket,
   key_prefix=s3_prefix
)

print("Train data uploaded to:", train_s3_path)
print("Test data uploaded to:", test_s3_path)

train_df_s3 = pd.read_csv(train_s3_path)
```

test_df_s3 = pd.read_csv(test_s3_path)

print(train_df_s3.shape, test_df_s3.shape)

Key Observations

- The dataset contains both numerical and categorical features.
- The target variable (Attrition) is binary and shows class imbalance.
- Several categorical features will require encoding.
- No preprocessing decisions are applied in this notebook.
- Raw datasets are successfully stored in Amazon S3.

Details of Each Step in Code File 2: Data Cleaning & Feature Engineering

This notebook loads data from S3, cleans it, encodes features, splits into train/validation/test sets, scales numerical features, and saves processed data back to S3. No model training here.

1. **Imports**: Load pandas, numpy, sagemaker, sklearn's train_test_split, LabelEncoder, StandardScaler. Suppress warnings.
2. **SageMaker Session Setup**: Initialize session, get default bucket, and print it.
3. **Load Raw Data from S3**: Construct S3 paths for train and test CSVs. Read them into DataFrames and print shapes.
4. **Drop Non-Informative Columns**: Remove "Employee ID" from both DataFrames as it's not useful for prediction.
5. **Handle Missing Values**:
    - Identify numerical columns and fill NaNs with median (using train median for both sets).
    - Identify categorical columns and fill NaNs with mode (using train mode for both sets).
6. **Ordinal Encoding**: Define explicit mappings for ordinal categories (e.g., Work-Life Balance: Poor=1 to Excellent=4). Map values in both DataFrames. Display updated train_df head. Fill any new NaNs from mapping with median.
7. **Binary Encoding**:
    - Function to identify binary categorical columns (exactly 2 unique values).
    - Manually specify nominal binary cols like "Gender".
    - For other binary cols (e.g., Yes/No), map first unique to 0, second to 1 in both sets.
8. **One-Hot Encoding**: Identify nominal columns (e.g., Gender, Job Role). Apply pd.get_dummies with drop_first=True. Align train and test columns, filling missing with 0.
9. **Split Features and Target**: Encode target "Attrition" (Stayed=0, Left=1). Separate X (features) and y (target) from train_df.
10. **Train/Validation/Test Split**: Split X/y into train (70%), temp (30%) with stratification. Split temp into val (15%) and test (15%) with stratification. Print shapes. Combine y_train/X_train and y_val/X_val for SageMaker format.
11. **Feature Scaling**: Identify numerical cols to scale. Fit StandardScaler on X_train, transform X_train, X_val, X_test. Save scaler to /tmp and upload to S3.
12. **Save Processed Data to S3**: Define processed prefix. Save train_combined, val_combined (with headers=False for SageMaker XGBoost), and X_test to S3 CSVs. Print confirmation.

13. **Sanity Checks**:
    - Print shapes for X/y splits.
    - Check total NaNs in X sets.
    - Verify target distribution balance across splits.
    - Ensure consistent feature counts.
    - Convert any bool columns to int.
14. **Summary**: Recap steps in comments: loading, handling missing/encoding, splitting, scaling, saving.

#----------code file 2----------

Code File 2:

```
# Employee Attrition Prediction
## Notebook 02: Data Cleaning & Feature Engineering

This notebook covers:
- Loading raw datasets from Amazon S3
- Data cleaning and preprocessing
- Feature encoding and transformation
- Train/validation/test split
- Saving processed datasets back to S3

No model training is performed in this notebook.


import pandas as pd
import numpy as np

import sagemaker
from sagemaker.session import Session

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

import warnings
warnings.filterwarnings("ignore")

sagemaker_session = sagemaker.Session()
bucket = sagemaker_session.default_bucket()

print("Using bucket:", bucket)

* Loading Raw Data from S3

train_s3_path = f"s3://{bucket}/employee-attrition/raw-data/train.csv"
```

```
test_s3_path  = f"s3://{bucket}/employee-attrition/raw-data/test.csv"

train_df = pd.read_csv(train_s3_path)
test_df  = pd.read_csv(test_s3_path)

print(train_df.shape, test_df.shape)

* Dropping Non-informative columns

train_df.drop(columns=["Employee ID"], inplace=True)
test_df.drop(columns=["Employee ID"], inplace=True)

### Handling Missing Values

# Numerical columns
num_cols = train_df.select_dtypes(include=["int64", "float64"]).columns

for col in num_cols:
   train_df[col].fillna(train_df[col].median(), inplace=True)
   test_df[col].fillna(train_df[col].median(), inplace=True)

# Categorical columns
cat_cols = train_df.select_dtypes(include=["object"]).columns

for col in cat_cols:
   train_df[col].fillna(train_df[col].mode()[0], inplace=True)
   test_df[col].fillna(train_df[col].mode()[0], inplace=True)

### Ordinal Encoding (Manual & Explicit)

train_df.info()

ordinal_mappings = {
   "Work-Life Balance": {"Poor": 1, "Below Average": 2, "Good": 3, "Excellent": 4},
   "Job Satisfaction": {"Very Low": 1, "Low": 2, "Medium": 3, "High": 4},
   "Performance Rating": {"Low": 1, "Below Average": 2, "Average": 3, "High": 4},
   "Company Reputation": {"Very Poor": 1, "Poor": 2, "Good": 3, "Excellent": 4},
   "Employee Recognition": {"Very Low": 1, "Low": 2, "Medium": 3, "High": 4},
   "Job Level": {"Entry": 1, "Mid": 2, "Senior": 3},
   "Company Size": {"Small": 1, "Medium": 2, "Large": 3}
}

for col, mapping in ordinal_mappings.items():
   train_df[col] = train_df[col].map(mapping)
   test_df[col] = test_df[col].map(mapping)
```

```
pd.set_option('display.max_columns', None)

train_df.head(10)

# Handling NaNs introduced by ordinal mapping
ordinal_cols = list(ordinal_mappings.keys())

for col in ordinal_cols:
    train_df[col].fillna(train_df[col].median(), inplace=True)
    test_df[col].fillna(train_df[col].median(), inplace=True)

### Binary Encoding (Yes / No)

def identify_binary_categorical_columns(df):
    binary_cols = []
    for col in df.select_dtypes(include=["object"]).columns:
        unique_vals = df[col].dropna().unique()
        if len(unique_vals) == 2:
            binary_cols.append(col)
    return binary_cols

identify_binary_categorical_columns(train_df)

target_col = "Attrition"

# the categories for the below column can increase in the future
nominal_binary_cols = ["Gender"]

binary_cols = [
    col for col in identify_binary_categorical_columns(train_df)
    if col != target_col and col not in nominal_binary_cols
]

for col in binary_cols:
    unique_vals = train_df[col].dropna().unique()

    mapping = {
        unique_vals[0]: 0,
        unique_vals[1]: 1
    }

    train_df[col] = train_df[col].map(mapping)
    test_df[col]  = test_df[col].map(mapping)

### One-Hot Encoding (Nominal Categories)
```

```
nominal_cols = [
    "Gender",
    "Job Role",
    "Education Level",
    "Marital Status"
]

train_df = pd.get_dummies(train_df, columns=nominal_cols, drop_first=True)
test_df  = pd.get_dummies(test_df, columns=nominal_cols, drop_first=True)

# Aligning train & test columns
train_df, test_df = train_df.align(test_df, join="left", axis=1, fill_value=0)

### Splitting Features & Target

# Encode target variable explicitly
train_df["Attrition"] = train_df["Attrition"].map({
    "Stayed": 0,
    "Left": 1
})

X = train_df.drop(columns=["Attrition"])
y = train_df["Attrition"]

### Train / Validation / Test Split

X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.30, stratify=y, random_state=42
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.50, stratify=y_temp, random_state=42
)

print(X_train.shape, X_val.shape, X_test.shape)

# Combining labels and features for SageMaker XGBoost
train_combined = pd.concat([y_train, X_train], axis=1)
val_combined   = pd.concat([y_val, X_val], axis=1)

y_train.unique(), y_val.unique()

train_combined.head()

### Feature Scaling (Numeric Only)
```

```
numeric_cols_to_scale = [
    "Age",
    "Years at Company",
    "Monthly Income",
    "Number of Promotions",
    "Distance from Home",
    "Number of Dependents",
    "Company Tenure"
]

scaler = StandardScaler()

X_train[numeric_cols_to_scale] = scaler.fit_transform(X_train[numeric_cols_to_scale])
X_val[numeric_cols_to_scale]   = scaler.transform(X_val[numeric_cols_to_scale])
X_test[numeric_cols_to_scale]  = scaler.transform(X_test[numeric_cols_to_scale])

import joblib

scaler_path = "/tmp/standard_scaler.joblib"
joblib.dump(scaler, scaler_path)

sagemaker_session.upload_data(
    path=scaler_path,
    bucket=bucket,
    key_prefix="employee-attrition/artifacts"
)

### Saving Processed Data to S3

processed_prefix = "employee-attrition/processed"

train_path = f"s3://{bucket}/{processed_prefix}/train.csv"
val_path   = f"s3://{bucket}/{processed_prefix}/validation.csv"
test_path  = f"s3://{bucket}/{processed_prefix}/test.csv"

train_combined.to_csv(train_path, index=False, header=False)
val_combined.to_csv(val_path, index=False, header=False)
X_test.to_csv(test_path, index=False, header=False)

print("SageMaker-compatible datasets saved to S3")

# Shape sanity check (most important)
print("X_train:", X_train.shape)
print("X_val  :", X_val.shape)
print("X_test :", X_test.shape)
```

```
print("y_train:", y_train.shape)
print("y_val  :", y_val.shape)
print("y_test :", y_test.shape)

# NaN check
print("NaNs in X_train:", X_train.isnull().sum().sum())
print("NaNs in X_val  :", X_val.isnull().sum().sum())
print("NaNs in X_test :", X_test.isnull().sum().sum())

# Target balance check (stratification validation)
print("Train target distribution:")
print(y_train.value_counts(normalize=True))

print("\nValidation target distribution:")
print(y_val.value_counts(normalize=True))

print("\nTest target distribution:")
print(y_test.value_counts(normalize=True))

# Feature count consistency
print("Feature count:")
print(X_train.shape[1], X_val.shape[1], X_test.shape[1])

for df in [X_train, X_val, X_test]:
    bool_cols = df.select_dtypes(include=["bool"]).columns
    df[bool_cols] = df[bool_cols].astype(int)

### Summary

- Raw data loaded from Amazon S3
- Missing values handled
- Ordinal, binary, and nominal features encoded
- Data split into train, validation, and test sets
- Numerical features scaled
- Processed datasets saved back to Amazon S3
```

Details of Each Step in Code File 3: Model Training using AWS SageMaker

This notebook trains an XGBoost model on processed data from S3, using SageMaker's managed training. No preprocessing here.

1. **Imports**: Load sagemaker, boto3. Retrieve container URIs.
2. **SageMaker Session Setup**: Initialize session, get role, bucket, region. Print them.
3. **Define S3 Paths**: Set processed prefix and construct train/validation paths. Print them.

4. **Prepare Training Inputs**: Create TrainingInput objects for train and validation data, specifying content_type="text/csv".
5. **Get XGBoost Container**: Retrieve the XGBoost image URI for version 1.7-1 in the region. Print it.
6. **Define XGBoost Estimator**: Create Estimator with image, role, instance details (ml.m5.large, etc.), output path for artifacts.
7. **Set Hyperparameters**: Configure objective (binary:logistic), eval_metric (auc), boosting rounds (200), tree depth (5), learning rate (0.1), subsampling (0.8), colsample (0.8).
8. **Start Training**: Call fit on the estimator with train and validation inputs as a dictionary.

#----------code file 3 ----------

Code File 3:

```
# Employee Attrition Prediction
## Notebook 03: Model Training using AWS SageMaker

This notebook covers:
- Loading processed datasets from Amazon S3
- Training an XGBoost classification model using SageMaker
- Saving trained model artifacts to S3

No data preprocessing or feature engineering is performed here.


Basic imports

import sagemaker
import boto3
from sagemaker.session import Session
from sagemaker.inputs import TrainingInput
from sagemaker.image_uris import retrieve

SageMaker Session Setup

sagemaker_session = sagemaker.Session()
role = sagemaker.get_execution_role()
bucket = sagemaker_session.default_bucket()
region = boto3.Session().region_name

print("Bucket:", bucket)
print("Region:", region)
print("Role:", role)

Defining S3 Paths for Processed Data

processed_prefix = "employee-attrition/processed"
```

```
train_data_path = f"s3://{bucket}/{processed_prefix}/train.csv"
val_data_path   = f"s3://{bucket}/{processed_prefix}/validation.csv"

print("Train path:", train_data_path)
print("Validation path:", val_data_path)
```

Preparing SageMaker Training Inputs

```
train_input = TrainingInput(
    s3_data=train_data_path,
    content_type="text/csv"
)

validation_input = TrainingInput(
    s3_data=val_data_path,
    content_type="text/csv"
)
```

Getting XGBoost Container Image

```
xgb_image = retrieve(
    framework="xgboost",
    region=region,
    version="1.7-1"
)

print(xgb_image)
```

Defining XGBoost Estimator

```
xgb_estimator = sagemaker.estimator.Estimator(
    image_uri=xgb_image,
    role=role,
    instance_count=1,
    instance_type="ml.m5.large",
    volume_size=30,
    max_run=3600,
    output_path=f"s3://{bucket}/employee-attrition/model-artifacts",
    sagemaker_session=sagemaker_session
)
```

Setting up Hyperparameters

```
xgb_estimator.set_hyperparameters(
    objective="binary:logistic",
    eval_metric="auc",
```

```
    num_round=200,
    max_depth=5,
    eta=0.1,
    subsample=0.8,
    colsample_bytree=0.8
)

Starting Training

xgb_estimator.fit(
    {
        "train": train_input,
        "validation": validation_input
    }
)
```

Details of Each Step in Code File 4: Model Evaluation, Registry & Deployment

This notebook evaluates the trained model, registers it, deploys an endpoint, runs predictions, computes metrics, cleans up, and provides a project summary.

1. **Imports**: Load boto3, sagemaker, pandas, numpy, Model, serializers/deserializers, sklearn metrics.
2. **SageMaker Setup**: Initialize session, get role, bucket, region, SageMaker client. Print bucket and region.
3. **Load Test Data from S3**: Construct paths for X_test and y_test CSVs (note: y_test save might be implied or missing in prior code; assume available). Read them and print shapes.
4. **Get Model Artifacts**: Hardcode training job name. Describe job to extract model artifact S3 path and training image. Print them.
5. **Create SageMaker Model**: Instantiate Model with image, artifact, role, session.
6. **Register Model**: Register to a model package group ("EmployeeAttritionModel") with content/response types, instances, and auto-approve.
7. **Create Predictor**: Define endpoint name. Instantiate Predictor with session, serializer (CSV), deserializer (JSON).
8. **Prepare Data for Prediction**: Convert bool columns in X_test to int. (Prediction step assumes a response variable; code shows conditional extraction of probabilities from response.)
9. **Extract Predictions**: Parse response to get probabilities (handling dict/list formats). Threshold at 0.5 for binary labels.
10. **Encode y_test**: Map "Stayed"=0, "Left"=1 consistently.
11. **Compute Metrics**: Print accuracy, precision, recall, F1, ROC-AUC. Display confusion matrix.
12. **Cleanup**: Delete the endpoint.
13. **Project Summary and Conclusion**: Provide a detailed recap including problem statement, dataset, methodology, model/tools, evaluation results (with metrics explanations), confusion matrix, takeaways, limitations, and conclusion.

#----------code file 4----------

Code File 4:

```
# Employee Attrition Prediction
## Notebook 04: Model Evaluation, Registry & Deployment

This notebook covers:
- Loading trained SageMaker XGBoost model
- Evaluating performance on validation and test data
- Registering the model in SageMaker Model Registry
- Deploying a real-time inference endpoint
- Running sample predictions
- Cleaning up AWS resources


import boto3
import sagemaker
import pandas as pd
import numpy as np

from sagemaker.model import Model
from sagemaker.serializers import CSVSerializer
from sagemaker.deserializers import JSONDeserializer

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
    confusion_matrix
)

# sagemaker setup

sagemaker_session = sagemaker.Session()
role = sagemaker.get_execution_role()
bucket = sagemaker_session.default_bucket()
region = boto3.Session().region_name

sm_client = boto3.client("sagemaker")

print("Bucket:", bucket)
print("Region:", region)

# Loading Test Data from S3
```

```
processed_prefix = "employee-attrition/processed"

X_test_path = f"s3://{bucket}/{processed_prefix}/X_test.csv"
y_test_path = f"s3://{bucket}/{processed_prefix}/y_test.csv"

X_test = pd.read_csv(X_test_path)
y_test = pd.read_csv(y_test_path).values.ravel()

print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)


training_job_name = "sagemaker-xgboost-2026-02-19-21-45-42-632"

training_desc = sm_client.describe_training_job(
    TrainingJobName=training_job_name
)

model_artifact = training_desc["ModelArtifacts"]["S3ModelArtifacts"]
training_image = training_desc["AlgorithmSpecification"]["TrainingImage"]

print("Model artifact:", model_artifact)
print("Training image:", training_image)

# Creating SageMaker Model Object

xgb_model = Model(
    image_uri=training_image,
    model_data=model_artifact,
    role=role,
    sagemaker_session=sagemaker_session
)

# Registering Model in Model Registry

model_package = xgb_model.register(
    content_types=["text/csv"],
    response_types=["application/json"],
    inference_instances=["ml.m5.large"],
    transform_instances=["ml.m5.large"],
    model_package_group_name="EmployeeAttritionModel",
    approval_status="Approved"
)

from sagemaker.predictor import Predictor
```

```
endpoint_name = "employee-attrition-xgb-endpoint"

predictor = Predictor(
    endpoint_name=endpoint_name,
    sagemaker_session=sagemaker_session,
    serializer=CSVSerializer(),
    deserializer=JSONDeserializer()
)

print(predictor)

X_test.dtypes[X_test.dtypes == "bool"]

# Converting boolean columns to int (True/False → 1/0)
bool_cols = X_test.select_dtypes(include=["bool"]).columns

X_test[bool_cols] = X_test[bool_cols].astype(int)

print("Converted boolean columns:")
print(bool_cols.tolist())

# Robust extraction of prediction probabilities from SageMaker XGBoost response

if isinstance(response, dict) and "predictions" in response:
    preds = response["predictions"]

    # Case 1: list of dicts
    if isinstance(preds[0], dict):
        # Try common keys
        if "score" in preds[0]:
            y_pred_prob = np.array([p["score"] for p in preds], dtype=float)
        elif "probability" in preds[0]:
            y_pred_prob = np.array([p["probability"] for p in preds], dtype=float)
        else:
            raise ValueError(f"Unknown prediction dict format: {preds[0]}")
    else:
        # Case 2: list of floats
        y_pred_prob = np.array(preds, dtype=float)

else:
    # Fallback: raw list
    y_pred_prob = np.array(response, dtype=float)

# Convert probabilities to class labels
y_pred = (y_pred_prob >= 0.5).astype(int)
```

```
# Ensuring y_test is encoded consistently with training labels
y_test = pd.Series(y_test).map({
   "Stayed": 0,
   "Left": 1
}).values

print("y_test unique:", np.unique(y_test))
print("y_pred unique:", np.unique(y_pred))

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_pred))
confusion_matrix(y_test, y_pred)

predictor.delete_endpoint()
```

Project Summary and Conclusion

In this project, an end-to-end machine learning system was designed and implemented to predict employee attrition using structured HR data. The objective was to identify employees who are likely to leave the organization, enabling data-driven decision-making for employee retention.

Problem Statement
Employee attrition is a critical challenge for organizations, leading to increased hiring costs and loss of experienced talent. This project aims to build a binary classification model that predicts whether an employee is likely to leave the company based on demographic, professional, and organizational attributes.

Dataset
The dataset was sourced from Kaggle and includes features such as age, job role, monthly income, work-life balance, job satisfaction, company tenure, remote work status, leadership opportunities, and company reputation. The target variable, **Attrition**, indicates whether an employee stayed (`0`) or left (`1`) the organization.

Methodology
The project followed a structured machine learning lifecycle:
- Data ingestion and exploratory data analysis
- Data cleaning and preprocessing
- Feature encoding and scaling
- Train, validation, and test split
- Model training using **XGBoost** with **AWS SageMaker managed training**
- Model evaluation using a held-out test set
- Model registration and real-time deployment using **AWS SageMaker**
- Inference and evaluation via a temporary endpoint

- Proper cleanup of cloud resources

Model and Tools
- **Model**: XGBoost Classifier
- **Cloud Platform**: AWS SageMaker
- **Training Mode**: Managed SageMaker training job
- **Deployment**: Real-time inference endpoint
- **Version Control**: GitHub

Evaluation Results
The trained model was evaluated on unseen test data using standard classification metrics:

- **Accuracy**: 65.39%
- **Precision**: 64.22%
- **Recall**: 61.44%
- **F1 Score**: 62.80%
- **ROC-AUC**: 65.21%

Confusion Matrix:

[[3234 1455]
[1639 2612]]

Detailed Model Evaluation Metrics

To evaluate the performance of the employee attrition prediction model, multiple classification metrics were used. Since employee attrition is a binary classification problem with potential class imbalance, relying on a single metric such as accuracy is insufficient. Therefore, a combination of threshold-dependent and threshold-independent metrics was analyzed.

Accuracy (65.39%)
Accuracy represents the proportion of total predictions that were correct.

An accuracy of **65.39%** indicates that the model correctly classified approximately two-thirds of employees. While accuracy provides a general sense of performance, it does not distinguish between different types of classification errors, which is important in attrition prediction.

Precision (64.22%)
Precision measures how many employees predicted to leave the company actually left.

A precision score of **64.22%** suggests that when the model predicts attrition, it is correct most of the time. This is particularly important in HR contexts, as false positives may lead to unnecessary retention interventions for employees who are not actually at risk.

Recall (61.44%)
Recall measures how many employees who actually left the company were correctly identified by the model.

With a recall of **61.44%**, the model is able to identify a majority of employees who are at risk of leaving. In attrition use cases, recall is critical because failing to identify at-risk employees (false negatives) can result in missed retention opportunities.

F1 Score (62.80%)
The F1 score is the harmonic mean of precision and recall.

An F1 score of **62.80%** indicates a balanced trade-off between precision and recall, making the model suitable as a baseline classifier where both false positives and false negatives have practical implications.

ROC-AUC (65.21%)
The Receiver Operating Characteristic – Area Under the Curve (ROC-AUC) measures the model's ability to distinguish between employees who leave and those who stay across all classification thresholds.

A ROC-AUC score of **65.21%** indicates that the model has a good discriminative ability and performs significantly better than random guessing (ROC-AUC = 50%). This metric is especially useful in scenarios where the decision threshold may change based on business requirements.

Confusion Matrix Interpretation

- **True Negatives (3234):** Employees who stayed and were correctly predicted to stay
- **False Positives (1455):** Employees predicted to leave but actually stayed
- **False Negatives (1639):** Employees who left but were predicted to stay
- **True Positives (2612):** Employees who left and were correctly predicted to leave

This confusion matrix shows that the model maintains a reasonable balance between identifying at-risk employees and avoiding excessive false alarms.

Overall Evaluation Summary

The evaluation results demonstrate that the model performs consistently across multiple metrics and provides a reliable baseline for employee attrition prediction. While there is room for improvement, the current model effectively captures important patterns in the data and can support informed HR decision-making when used alongside human judgment.

Key Takeaways
- AWS SageMaker enables scalable and reproducible model training and deployment.
- Proper preprocessing and label consistency are critical for reliable evaluation.
- XGBoost performs well on structured HR datasets with mixed feature types.
- The project successfully demonstrates the full machine learning lifecycle in a cloud environment.

Limitations and Future Improvements
- Model performance can be improved through advanced hyperparameter tuning.
- Additional behavioral and temporal features could enhance predictive power.
- Cross-validation and ensemble methods may further improve robustness.
- In a production system, preprocessing and inference pipelines would be fully automated.

Conclusion
This project successfully demonstrates an end-to-end machine learning workflow for employee attrition prediction using AWS SageMaker. It highlights the practical application of cloud-based ML services, from data preprocessing to deployment and evaluation, and provides a solid foundation for further experimentation and real-world implementation.

#---------- end of python code ----------

---------- End of Document ----------