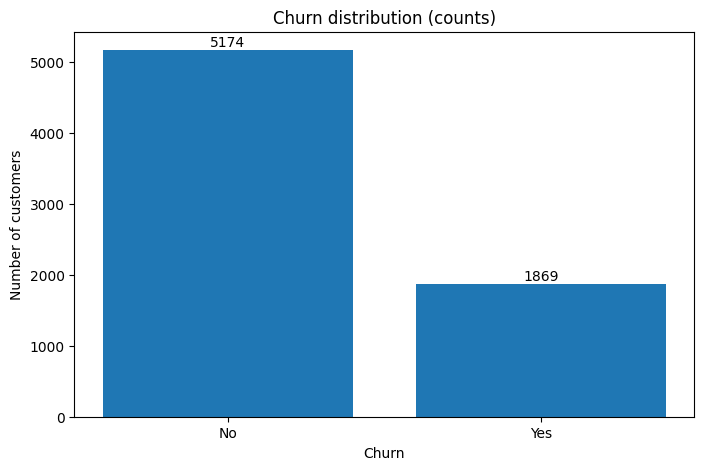
**Customer Churn Prediction**

# **Project goal**

I built an end-to-end churn prediction pipeline to identify customers likely to leave so the business can run targeted retention actions.

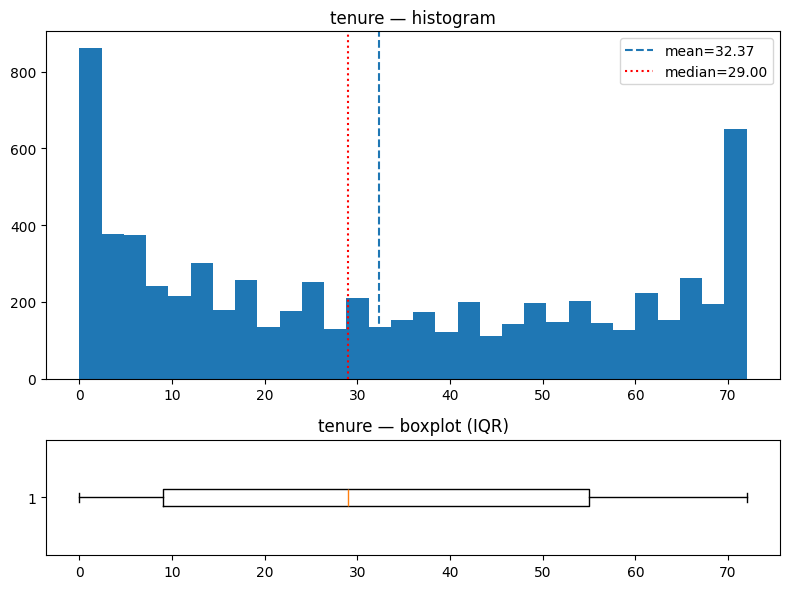
# **Dataset summary**

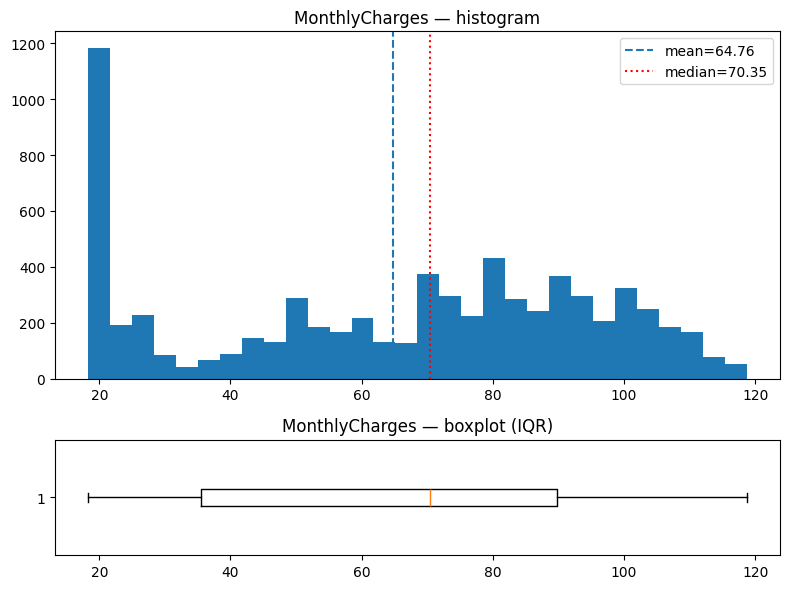
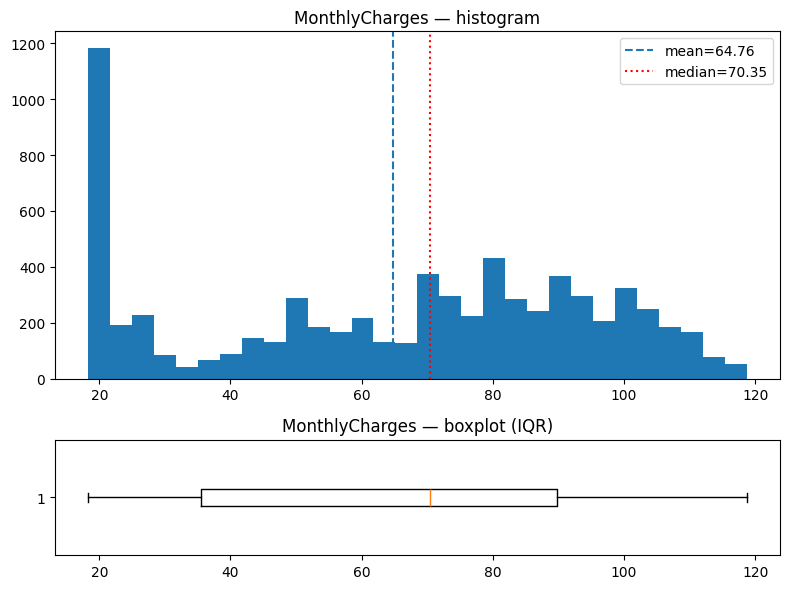
* **Raw rows:** 7,043 customers.
* **Key columns:** demographics, service flags, account info, MonthlyCharges, TotalCharges, tenure, and target Churn.
* **Churn distribution:** No = 5,174, Yes = 1,869 (≈ 73.5% No, 26.5% Yes).
* **Data issues found:** TotalCharges initially read as object; after coercion to numeric there were 11 NaNs (all had tenure == 0).



# **Exploratory Data Analysis (EDA)**

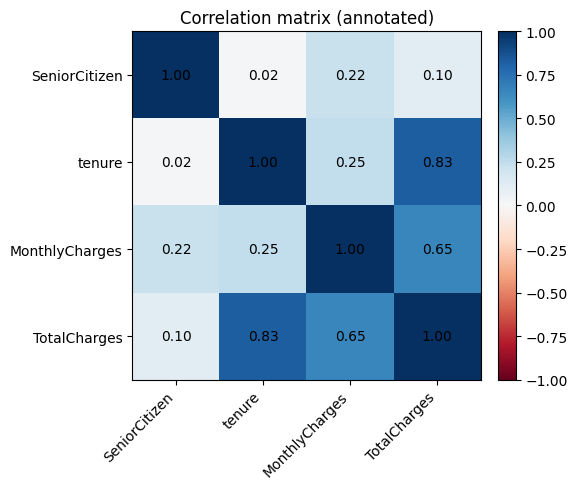
* 1. **Missing values & problematic rows**
* TotalCharges converted to numeric produced 11 NaN values; all these rows had tenure == 0. I kept them and treated them as new-customer cases (impute or handle explicitly).
  1. **Numeric summary**
* Numeric columns used in analysis: SeniorCitizen (binary), tenure, MonthlyCharges, TotalCharges (converted to numeric).
  1. **Distributions & boxplots**
* **Tenure:** many customers with low tenure, right tail to 72 months. No extreme outliers detected by IQR rule.
* **MonthlyCharges:** multi-modal distribution (distinct plan clusters). No extreme outliers detected by IQR rule.
* **TotalCharges:** skewed, wide range (explained by tenure × monthly); visually consistent with business logic.





* 1. **Outlier check (IQR rule)**
* I computed outlier counts using the IQR rule:
* **SeniorCitizen:** flagged (binary behavior causes IQR rule to misreport)
* **tenure:** 0 outliers
* **MonthlyCharges:** 0 outliers
  1. **Correlation analysis**
* Correlation (numeric features):
* **tenure ↔ TotalCharges:** strong positive correlation (≈ 0.83)
* **MonthlyCharges ↔ TotalCharges:** moderate positive correlation (≈ 0.65)
* **tenure ↔ MonthlyCharges:** weak correlation (≈ 0.25)

**Interpretation:** TotalCharges is largely redundant with tenure and MonthlyCharges (expected since TotalCharges ≈ tenure × MonthlyCharges). I kept TotalCharges but treated its scaling cautiously.



# **Feature engineering & preprocessing**

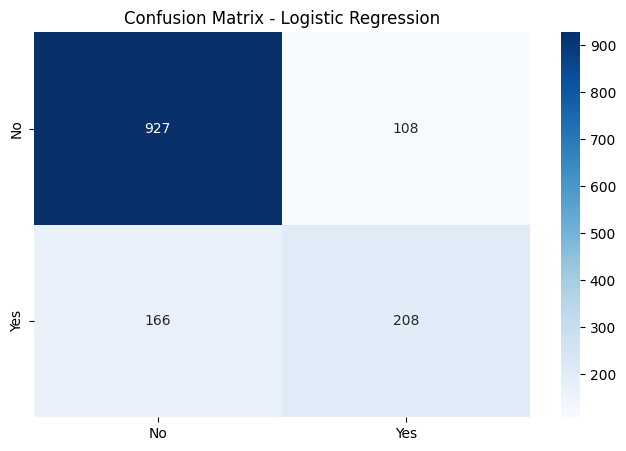
* 1. **Column grouping and transformers**
* **Numeric (StandardScaler):** tenure, MonthlyCharges
* **Rationale:** these columns are on different scales; StandardScaler (z-score) centers and scales for models that are scale-sensitive.
* **Numeric (RobustScaler):** TotalCharges
* **Rationale:** TotalCharges has wide spread; RobustScaler (median & IQR) reduces sensitivity to possible extreme values.
* **Binary passthrough:** SeniorCitizen (0/1) — no scaling applied.
* **Categorical → OneHotEncoder(drop='first', handle\_unknown='ignore'):** gender, Partner, Dependents, PhoneService, MultipleLines, InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, Contract, PaperlessBilling, PaymentMethod.
* drop='first' reduces redundancy. handle\_unknown='ignore' avoids crashes at inference when unseen categories appear.
  1. **Pipeline & output**
* Implemented ColumnTransformer inside an sklearn.Pipeline.
* X\_processed shape: (7,043, 23) features after scaling + one-hot encoding.
* Saved artifacts:
* preprocessor.joblib (the fitted ColumnTransformer pipeline)
* preprocessor\_raw\_columns.json (raw input column order expected by the preprocessor)
* preprocessor\_output\_columns.json (final processed feature names)

# **Dataset splits & SageMaker formatting**

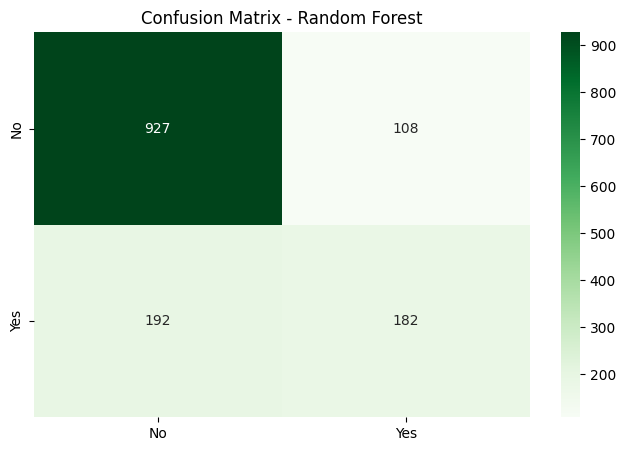
* Locally for sklearn experiments: I used 80% train / 20% test (stratified by y).
* For SageMaker XGBoost HPO: I created 70% train / 15% validation / 15% test CSVs in the format expected by SageMaker XGBoost (label as the first column, no header).
* Saved processed data files:
* churn\_processed.csv (final processed dataset)
* SageMaker CSVs: train\_for\_sagemaker.csv, validation\_for\_sagemaker.csv, test\_for\_sagemaker.csv (label-first format).

# **Modeling — models trained, tuning, and evaluation**

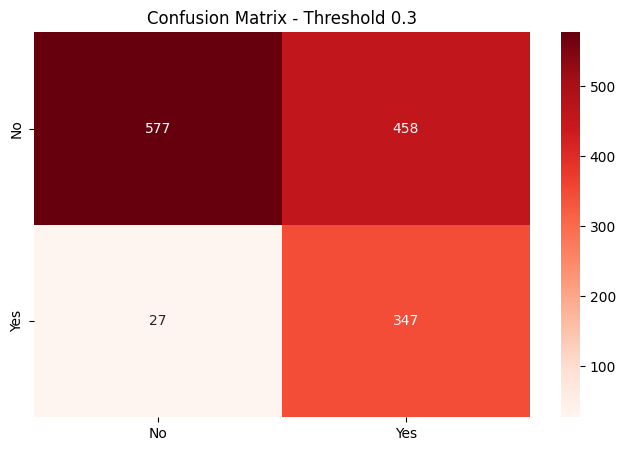
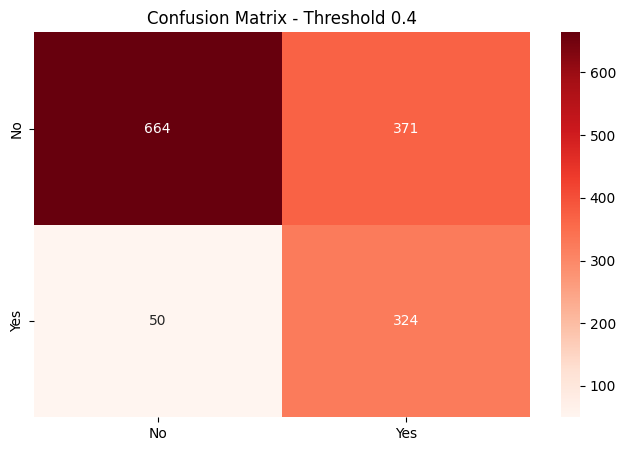
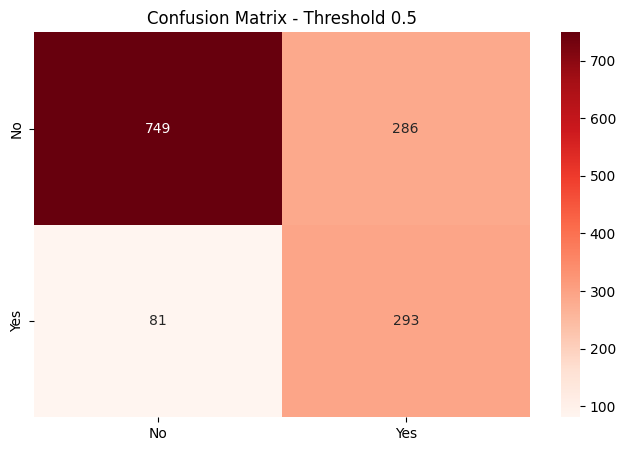
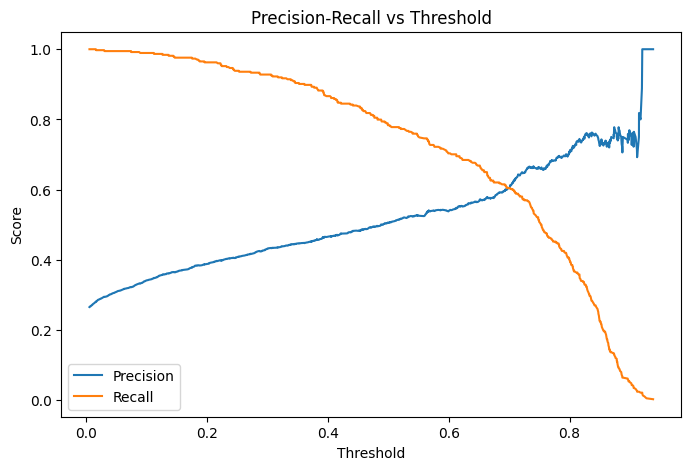
* 1. **Baseline models (local)**
* Logistic Regression (baseline)
* Test set (n ≈ 1,409):
* Class 0 (No): precision 0.85, recall 0.90, f1 0.87 (support 1,035)
* Class 1 (Yes): precision 0.66, recall 0.56, f1 0.60 (support 374)
* Accuracy: 0.81, ROC-AUC: ≈ 0.8422
* Confusion matrix:



* Random Forest (baseline)
* Test set (n ≈ 1,409):
* Class 0: precision 0.83, recall 0.90, f1 0.86
* Class 1: precision 0.63, recall 0.49, f1 0.55
* Accuracy: 0.79, ROC-AUC: ≈ 0.8258
* Confusion matrix:



* 1. **Hyperparameter tuning (sklearn)**
* Logistic Regression (GridSearchCV, scoring='recall', cv=5): best params C=1, penalty='l2', solver='lbfgs' — CV recall ≈ 0.548.
* Random Forest (RandomizedSearchCV, scoring='recall', cv=5): best params n\_estimators=200, max\_depth=10, min\_samples\_split=5, min\_samples\_leaf=1 — CV recall ≈ 0.495.
* After tuning, test set metrics were similar; logistic remained stronger on recall under this objective.
  1. **Addressing imbalance (class weight & threshold tuning)**
* LogisticRegression with class\_weight='balanced':
* At default threshold 0.5:
* Class 1 recall increased to ~0.78, precision decreased to ~0.51, accuracy ≈ 0.74, ROC-AUC ≈ 0.842.
* Precision–Recall vs Threshold: lowering threshold improves recall and reduces precision. Example results:
* threshold 0.5 → recall ≈ 0.78, precision ≈ 0.51
* threshold 0.4 → recall ≈ 0.87, precision ≈ 0.47
* threshold 0.3 → recall ≈ 0.93, precision ≈ 0.43



* 1. **SageMaker XGBoost with Automatic Model Tuning (HPO)**
* **Estimator:** SageMaker XGBoost built-in container. Fixed hyperparams: objective="binary:logistic", eval\_metric="auc", num\_round=200.
* **Hyperparameter search space (Bayesian):**
* eta (0.01–0.3), max\_depth (3–10), min\_child\_weight (1–10), subsample (0.5–1.0), colsample\_bytree (0.5–1.0).
* **Tuner configuration:** objective\_metric\_name="validation:auc", max\_jobs=12, max\_parallel\_jobs=2, strategy="Bayesian".
* **Important fix made:** SageMaker XGBoost requires label as the first column and labels in {0,1}; I updated CSVs accordingly.
  1. **Final XGBoost results (best model from SageMaker HPO, evaluated locally on test.csv)**
* **Test set (n = 1,057):**
* ROC-AUC (test): 0.8552514248942821
* Classification report:
* Class 0: precision 0.92, recall 0.72, f1 0.81 (support 777)
* Class 1: precision 0.51, recall 0.82, f1 0.63 (support 280)
* **Confusion matrix:**

[[561 216]

[ 51 229]]

* TN = 561, FP = 216, FN = 51, TP = 229

# **Comparative summary**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Test rows** | **ROC-AUC** | **Precision (churn)** | **Recall (churn)** | **F1 (churn)** | **Key observation** |
| Logistic (baseline) | 1,409 | 0.8422 | 0.66 | 0.56 | 0.60 | Interpretable baseline |
| Random Forest (baseline) | 1,409 | 0.8258 | 0.63 | 0.49 | 0.55 | Slightly lower recall |
| Logistic (class\_weight='balanced') | 1,409 | 0.8419 | 0.51 | 0.78 | 0.61 | Higher recall via weighting |
| Logistic (threshold 0.4) | 1,409 | — | 0.47 | 0.87 | 0.61 | Threshold tuned for recall |
| XGBoost (SageMaker HPO, best) | 1,057 | **0.8553** | 0.51 | **0.82** | 0.63 | Best ROC-AUC and high recall |

# **Artifacts & files produced**

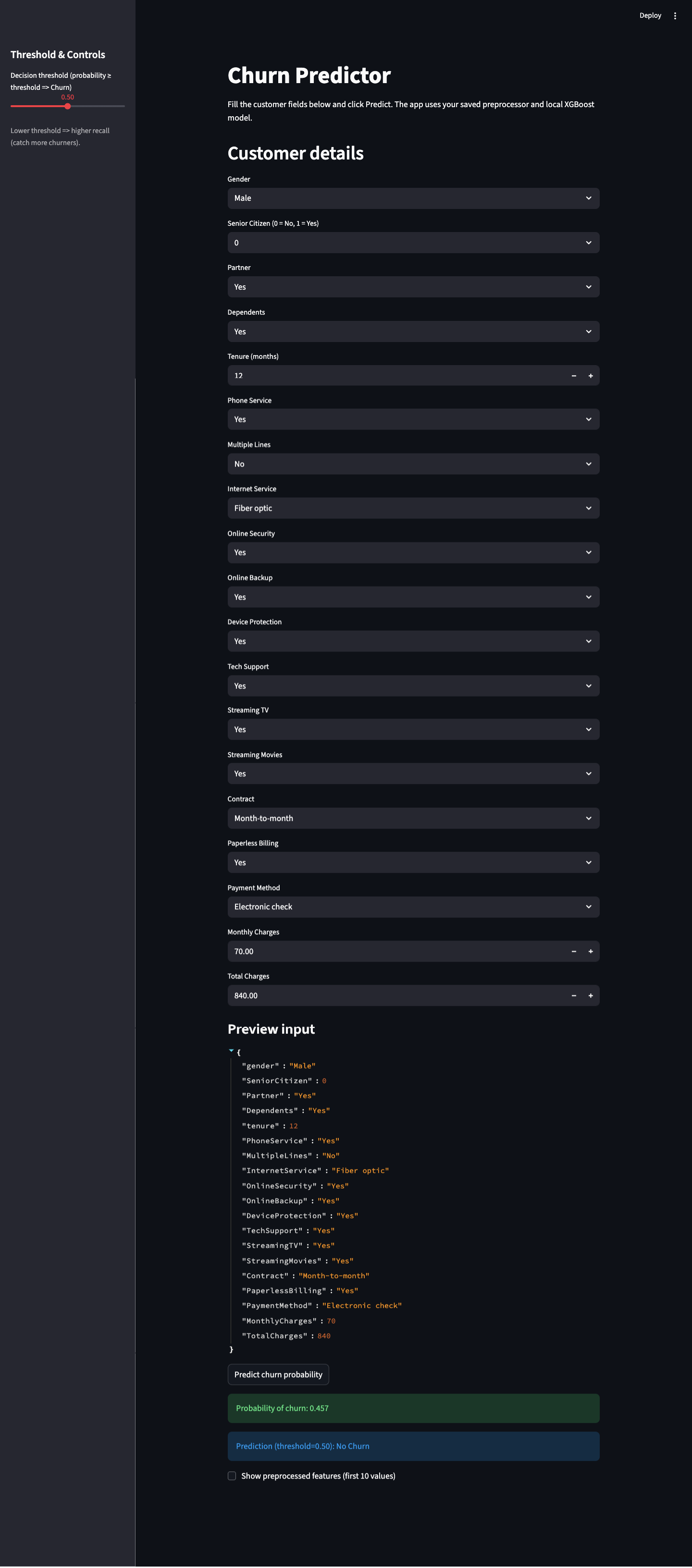
* preprocessor.joblib — saved sklearn ColumnTransformer pipeline (use for inference).
* preprocessor\_raw\_columns.json — raw column order expected by the preprocessor (must be used to build input rows).
* preprocessor\_output\_columns.json — final processed feature names (for debugging).
* churn\_processed.csv — processed dataset (features + label) used in experiments.
* SageMaker artifacts: train.csv, validation.csv, test.csv (label-first format for XGBoost).
* model.tar.gz in S3 (best model path printed by tuner) — contains the XGBoost model file; I downloaded and loaded it locally for evaluation.
* Notebook files: fe.ipynb (feature engineering notebook), SageMaker notebook (HPO + training notebook).

# **Inference & deployment approach**

* Implemented a local Streamlit app (app.py) to perform inference using the saved preprocessing pipeline and the downloaded XGBoost model. The app loads three artifacts that must be placed next to app.py:

preprocessor.joblib, preprocessor\_raw\_columns.json, and the extracted XGBoost model file (xgboost-model). If any of these files are missing the app stops and reports the missing files.

* The Streamlit app workflow (what the app does):
* Presents an input form for all raw customer fields (categoricals as selectboxes, numerics as number inputs).
* Uses preprocessor\_raw\_columns.json to reindex the user input DataFrame into the exact raw column order expected by the preprocessor.
* Fills any missing numeric values with 0 and missing string values with "No" as a safety measure.
* Transforms the reindexed raw DataFrame with the loaded preprocessor.joblib to produce the numeric feature vector.
* Converts the processed vector to an xgboost.DMatrix and calls Booster.predict(...) to obtain a probability of churn.
* Applies an interactive threshold slider (user-controlled in the sidebar) to convert probability → label (Churn / No Churn), and displays both the probability and the thresholded prediction.
* Optionally shows the preprocessed feature values for debugging.



# **Conclusion**

I built and validated a complete churn prediction pipeline—data cleaning, robust preprocessing, baseline models, hyperparameter tuning on SageMaker, and local inference-ready deployment artifacts—resulting in an XGBoost model that achieved the best ROC-AUC (~0.855) and a high churn recall (≈0.82) at the evaluated threshold.

**(END OF REPORT)**