03_BaselineModel

January 29, 2026

1 Baseline Model

Given a leakage-safe, well-reasoned preprocessing strategy, can we predict churn better than chance in a way that is interpretable and defensible?

Logistic Regression is used as a baseline model to establish a transparent and interpretable performance benchmark.

Scikit-learn pipeline is used to learn preprocessing only from the training set, then apply the same transformations to validation/test, then fit the model.

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    RocCurveDisplay
)
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore")
```

[ ]:
```
plt.style.use('ggplot')
palette = [
'#E24A33',  #(orange)
'#348ABD',  #(blue)
'#8EBA42',   #(green)
'#988ED5',  #(cyan)
'#777777',  #(grey)
'#FBC15E',  #(yellow)
'#FFB5B8'    #(pink)
]
```

[ ]:
```
# Load data
df = pd.read_csv('../data/ecommerce_churn_data_eda.csv')
print(df.isnull().sum())
```

```
CustomerID                      0
Churn                           0
Tenure                        264
PreferredLoginDevice            0
CityTier                        0
WarehouseToHome               251
PreferredPaymentMode            0
Gender                          0
HourSpendOnApp                255
NumberOfDeviceRegistered        0
PreferedOrderCat                0
SatisfactionScore               0
MaritalStatus                   0
NumberOfAddress                 0
Complain                        0
OrderAmountHikeFromlastYear   265
CouponUsed                    256
OrderCount                    258
DaySinceLastOrder             307
CashbackAmount                  0
dtype: int64
```

[ ]:
```
# Define Target & Drop identifier + leakage-prone features
TARGET = 'Churn'
df.drop(columns=['CustomerID', 'DaySinceLastOrder'], inplace=True)

X = df.drop(columns=[TARGET])
y = df[TARGET]
```

[ ]:
```
# Identify numeric vs categorical columns
numeric_features = X.select_dtypes(include=["number"]).columns.tolist()
categorical_features = X.select_dtypes(exclude=["number"]).columns.tolist()

if "CityTier" in numeric_features:
    numeric_features.remove("CityTier")
    categorical_features.append("CityTier")

numeric_features, categorical_features
```

[ ]:
```
(['Tenure',
  'WarehouseToHome',
  'HourSpendOnApp',
  'NumberOfDeviceRegistered',
  'SatisfactionScore',
```

```
  'NumberOfAddress',
  'Complain',
  'OrderAmountHikeFromlastYear',
  'CouponUsed',
  'OrderCount',
  'CashbackAmount'],
 ['PreferredLoginDevice',
  'PreferredPaymentMode',
  'Gender',
  'PreferedOrderCat',
  'MaritalStatus',
  'CityTier'])
```

[ ]:
```
# Preprocessing pipelines
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='Unknown')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)
```

[ ]:
```
# Train/validation split
test_split = 0.2
X_train, X_valid, y_train, y_valid = train_test_split(
    X, y,
    test_size=test_split,
    random_state=42,
    stratify=y
)
```

[ ]:
```
# Define the model pipeline
clf = LogisticRegression(
    max_iter=1000,
    class_weight="balanced",
    random_state=42
)

model = Pipeline(steps=[
```

```
    ("preprocessor", preprocessor),
    ("classifier", clf)
])
```

[ ]:
```
# Train the model
model.fit(X_train, y_train)
```

[ ]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['Tenure', 'WarehouseToHome',
                                                   'HourSpendOnApp',
                                                   'NumberOfDeviceRegistered',
                                                   'SatisfactionScore',
                                                   'NumberOfAddress',
                                                   'Complain',
'OrderAmountHikeFromlastYear',
                                                   'CouponUsed', 'OrderCount',
                                                   'CashbackAmount']),
                                                 ('cat',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(fill_value='Unknown',
 strategy='constant')),
                                                                  ('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['PreferredLoginDevice',
                                                   'PreferredPaymentMode',
                                                   'Gender', 'PreferedOrderCat',
                                                   'MaritalStatus',
                                                   'CityTier'])])),
                ('classifier',
                 LogisticRegression(class_weight='balanced', max_iter=1000,
                                    random_state=42))])
```

[ ]:
```
# Validate the model
y_pred = model.predict(X_valid)
y_proba = model.predict_proba(X_valid)[:, 1]

print(classification_report(y_valid, y_pred))

roc_auc = roc_auc_score(y_valid, y_proba)
print(f"ROC AUC: {roc_auc:.3f}")

# Plot ROC Curve
```

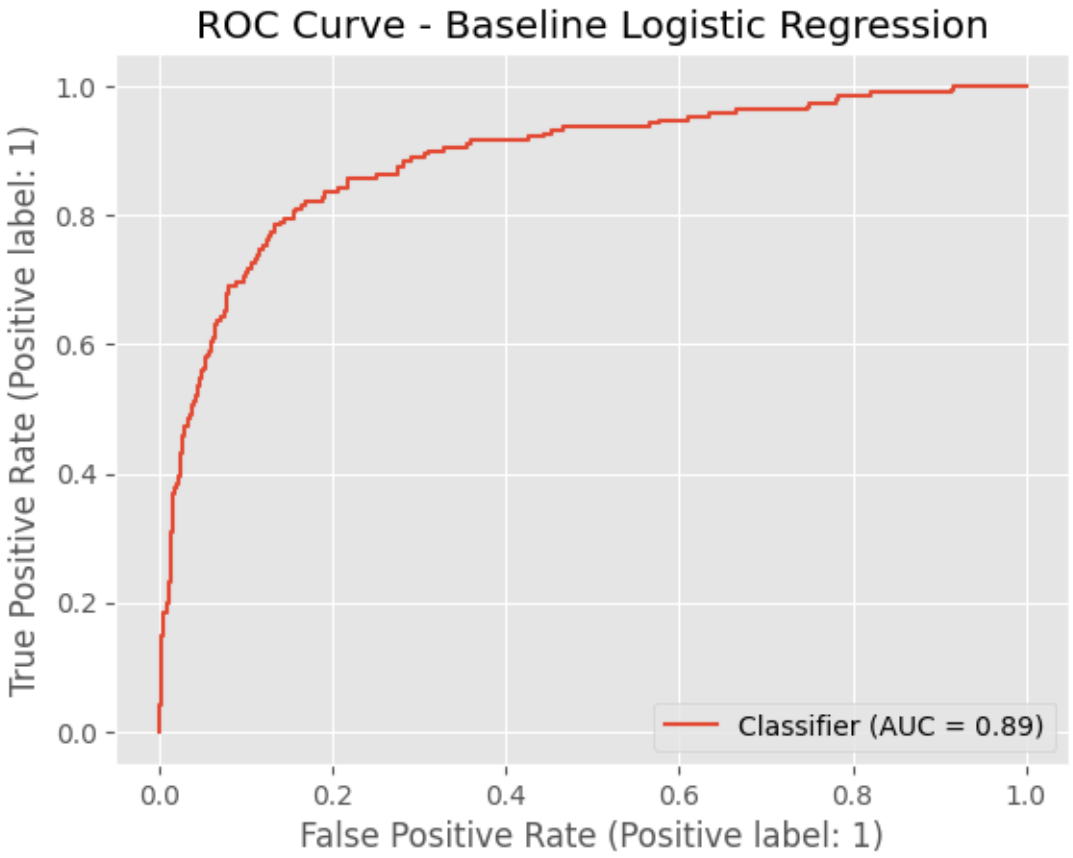
```
RocCurveDisplay.from_predictions(y_valid, y_proba)
plt.title("ROC Curve - Baseline Logistic Regression")
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.96      0.80      0.87       936
           1       0.46      0.84      0.60       190

    accuracy                           0.81      1126
   macro avg       0.71      0.82      0.74      1126
weighted avg       0.88      0.81      0.83      1126

ROC AUC: 0.885
```



[ ]:
```
# Feature importance interpretation
ohe = model.named_steps["preprocessor"].named_transformers_["cat"].
  ↪named_steps["onehot"]
cat_feature_names = ohe.get_feature_names_out(categorical_features)
```

```
all_feature_names = numeric_features + cat_feature_names.tolist()

coefs = model.named_steps["classifier"].coef_[0]
coef_df = pd.DataFrame({"feature": all_feature_names, "coef": coefs}).
 ↪sort_values("coef", ascending=False)

coef_df.head(10), coef_df.tail(10)
```

```
[ ]: (                                feature      coef
      25               PreferedOrderCat_Others  2.247384
      6                               Complain  0.712428
      5                         NumberOfAddress  0.618251
      28                   MaritalStatus_Single  0.535067
      13  PreferredPaymentMode_Cash on Delivery  0.434498
      3                NumberOfDeviceRegistered  0.398079
      4                       SatisfactionScore  0.369677
      1                         WarehouseToHome  0.330430
      20              PreferedOrderCat_Fashion  0.320198
      31                             CityTier_3  0.313638,
                                    feature      coef
      18                        Gender_Female -0.184392
      14    PreferredPaymentMode_Credit Card -0.298920
      17             PreferredPaymentMode_UPI -0.326653
      27                MaritalStatus_Married -0.374493
      29                           CityTier_1 -0.465393
      23             PreferedOrderCat_Mobile -0.501941
      24       PreferedOrderCat_Mobile Phone -0.665717
      10                       CashbackAmount -0.955938
      0                                Tenure -1.520018
      22  PreferedOrderCat_Laptop & Accessory -1.677750)
```

1.1 Summary Baseline Model

The baseline Logistic Regression model demonstrates strong discriminatory power (ROC-AUC = 0.885) and provides flexible control over recall–precision trade-offs.

Coefficient analysis aligns with EDA findings, confirming tenure, payment behavior, and product categories as key churn drivers.

This model establishes a reliable and interpretable benchmark for further experimentation.

2 Decision Threshold Tuning and Business Trade-offs

The default decision threshold of 0.5 may not align with business priorities, especially when the cost of false negatives (missed churns) outweighs that of false positives (unnecessary retention efforts). By adjusting the threshold, we can optimize for recall or precision based on strategic goals. Below is an example of how to evaluate different thresholds and their impact on classification metrics.

if P(churn) t → intervene else → do nothing

Changing t changes: - recall - precision - cost - coverage

This is where business strategy enters.

2.0.1 Strategy A — Maximize churn capture (recall-heavy)

- t low (e.g. 0.25–0.35)
- Catch most churners
- Accept wasted interventions Used when:
- churn is very costly
- intervention is cheap

2.0.2 Strategy B — Minimize wasted interventions (precision-heavy)

- t high (e.g. 0.55–0.75)
- Only intervene on high-confidence churners
- Accept missing some churners Used when:
- intervention is costly
- resources are limited

2.0.3 Strategy C — Balance (F1-score)

- t chosen to maximize F1 or business utility
- Balance catching churners and avoiding wasted interventions
- Optimize overall effectiveness

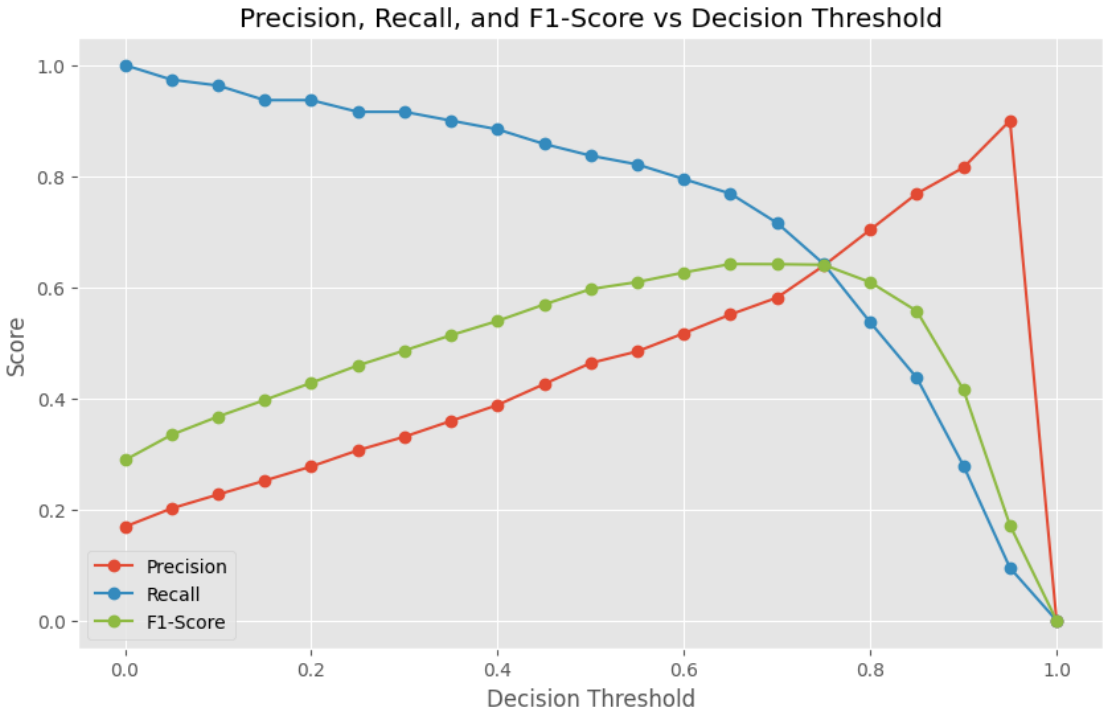
While EDA highlighted key behavioral patterns, threshold tuning allows these insights to be translated into actionable business decisions.

[ ]:
```
thresholds = np.arange(0.0, 1.01, 0.05)
results = []

for threshold in thresholds:
    y_pred_threshold = (y_proba >= threshold).astype(int)
    report = classification_report(y_valid, y_pred_threshold, output_dict=True)
    results.append({
        "threshold": threshold,
        "precision": report["1"]["precision"],
        "recall": report["1"]["recall"],
        "f1-score": report["1"]["f1-score"]
    })
results_df = pd.DataFrame(results)
```

[ ]:
```
# Plot Precision-Recall vs Threshold
plt.figure(figsize=(10, 6))
plt.plot(results_df["threshold"], results_df["precision"], label="Precision",␣
 ↪marker='o', color=palette[0])
plt.plot(results_df["threshold"], results_df["recall"], label="Recall",␣
 ↪marker='o', color=palette[1])
```

```
plt.plot(results_df["threshold"], results_df["f1-score"], label="F1-Score",␣
 ↪marker='o', color=palette[2])
plt.xlabel("Decision Threshold")
plt.ylabel("Score")
plt.title("Precision, Recall, and F1-Score vs Decision Threshold")
plt.legend()
plt.show()
```



Key observations from the graph: - Recall decreases monotonically as threshold increases (expected) - Precision increases monotonically (expected) - F1-score: - rises steadily from low thresholds - peaks around 0.70–0.75 - then drops sharply after ~0.8 due to recall collapsing

[ ]:
```
y_pred_threshold = (y_proba >= 0.75).astype(int)
report_final = classification_report(y_valid, y_pred_threshold)
print(report_final)

roc_auc_final = roc_auc_score(y_valid, y_proba)
print(f"Final ROC AUC: {roc_auc_final:.3f}")
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       936
           1       0.64      0.64      0.64       190

    accuracy                           0.88      1126
```

```
   macro avg       0.78      0.78      0.78      1126
weighted avg       0.88      0.88      0.88      1126

Final ROC AUC: 0.885
```

2.1 Conclusion on Decision Threshold Tuning

Rather than relying on the default 0.5 threshold, we evaluate model performance across a range of decision thresholds to align churn predictions with business objectives. This allows explicit control over the trade-off between recall and precision.

Based on the precision–recall trade-off, the F1-score reaches its maximum at a decision threshold of approximately 0.75. This threshold provides the best balance between precision and recall, maintaining strong churn detection while significantly reducing false positives. Thresholds above this point lead to a sharp decline in recall, resulting in lower overall F1 performance.

Therefore, for this baseline model, a decision threshold of 0.75 is recommended to optimize effectiveness in churn prediction.

Note that ROC-AUC is threshold-independent and remains unchanged after threshold tuning.