re-engineering-and-baseline-models

November 10, 2025

# 1 Step 02: Feature Engineering & Predictive Modeling (Baseline Models)

**Project:** AI Experimentation Platform for Predictive Insights
**Notebook:** 02_Feature_Engineering_and_Baseline_Models
**Author:** Jagadeesh N
**Date:** 10-Nov-2025
**Environment:** Python (NumPy · Pandas · Scikit-learn · Matplotlib · Seaborn)

---

### 1.0.1 Objective

This notebook builds on the **A/B Test Simulation** results to construct a predictive model that estimates **customer conversion likelihood** and **expected revenue uplift**.

We will: 1. Engineer meaningful customer-level features.
2. Split data into training/testing sets.
3. Build baseline machine learning models (Logistic Regression, Random Forest).
4. Evaluate model accuracy, ROC-AUC, and feature importance.
5. Visualize key insights for decision-making.

---

### 1.0.2 Background

Predictive modeling helps data-driven organizations forecast customer behavior.
By combining engineered behavioral features with machine learning, we can identify high-value customers and optimize campaign targeting.

---

### 1.0.3 Notebook Outline

1. Load simulated A/B dataset

2. Feature engineering

3. Train/test split

4. Model training (Logistic Regression, Random Forest)
5. Model evaluation & comparison
6. Feature importance visualization
7. Save baseline model for deployment

[2]:
```python
#importing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, roc_auc_score, confusion_matrix, classification_report,␣
  ↪RocCurveDisplay
)

sns.set(style="whitegrid")
np.random.seed(42)

# Load dataset from Step 01
customer_df = pd.read_csv("D:\\Project for job\\ab_test_simulated.csv")
print("Rows:", customer_df.shape[0])
customer_df.head()
```

```
Rows: 5881
```

[2]:

| | Customer ID | Revenue | NumTransactions | TotalUnits | Converted | Group |
|---|---|---|---|---|---|---|
| 0 | 12346.0 | 93843.3166 | 12 | 74285 | 0 | A |
| 1 | 12347.0 | 5955.0513 | 8 | 2967 | 1 | B |
| 2 | 12348.0 | 2221.3400 | 5 | 2714 | 0 | A |
| 3 | 12349.0 | 5358.7149 | 4 | 1624 | 0 | B |
| 4 | 12350.0 | 367.8400 | 1 | 197 | 0 | A |

[3]:
```python
# Create interaction and ratio features
customer_df['RevenuePerUnit'] = customer_df['Revenue'] /␣
  ↪(customer_df['TotalUnits'] + 1e-5)
customer_df['RevenuePerTransaction'] = customer_df['Revenue'] /␣
  ↪(customer_df['NumTransactions'] + 1e-5)
customer_df['HighValueCustomer'] = (customer_df['Revenue'] >␣
  ↪customer_df['Revenue'].median()).astype(int)
```

```python
# Encode group (A=0, B=1)
customer_df['GroupFlag'] = customer_df['Group'].map({'A':0, 'B':1})

# Select features for modeling
features = [
    'TotalUnits', 'NumTransactions', 'Revenue',
    'RevenuePerUnit', 'RevenuePerTransaction',
    'GroupFlag', 'HighValueCustomer'
]

X = customer_df[features]
y = customer_df['Converted']

X.head()
```

[3]:

```
   TotalUnits  NumTransactions     Revenue  RevenuePerUnit  \
0       74285               12  93843.3166        1.263288
1        2967                8   5955.0513        2.007095
2        2714                5   2221.3400        0.818475
3        1624                4   5358.7149        3.299701
4         197                1    367.8400        1.867208

   RevenuePerTransaction  GroupFlag  HighValueCustomer
0            7820.269866          0                  1
1             744.380482          1                  1
2             444.267111          0                  1
3            1339.675376          1                  1
4             367.836322          0                  0
```

[4]:

```python
#split data into train and test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

print("Training samples:", X_train.shape[0])
print("Testing samples:", X_test.shape[0])
```

```
Training samples: 4410
Testing samples: 1471
```

[6]:

```python
#balance checking
log_reg = LogisticRegression(max_iter=500, class_weight='balanced')
log_reg.fit(X_train_scaled, y_train)

y_pred_lr = log_reg.predict(X_test_scaled)
y_prob_lr = log_reg.predict_proba(X_test_scaled)[:, 1]
```

```
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_lr))
print("\nClassification Report:\n", classification_report(y_test, y_pred_lr,␣
↪zero_division=0))
```

```
Accuracy: 0.5329707681849082
ROC-AUC: 0.53068814793573

Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.54      0.67      1293
           1       0.13      0.48      0.20       178

    accuracy                           0.53      1471
   macro avg       0.50      0.51      0.44      1471
weighted avg       0.79      0.53      0.61      1471
```

[7]:
```
#Tune Probability Threshold
from sklearn.metrics import precision_recall_curve

prec, rec, thresh = precision_recall_curve(y_test, y_prob_lr)
f1 = 2 * (prec * rec) / (prec + rec + 1e-8)
best_thresh = thresh[np.argmax(f1)]
print(f"Optimal Threshold: {best_thresh:.2f}")

y_pred_tuned = (y_prob_lr >= best_thresh).astype(int)
print("\nClassification Report (Tuned Threshold):\n",␣
↪classification_report(y_test, y_pred_tuned, zero_division=0))
```

```
Optimal Threshold: 0.48

Classification Report (Tuned Threshold):
               precision    recall  f1-score   support

           0       0.92      0.18      0.30      1293
           1       0.13      0.89      0.23       178

    accuracy                           0.27      1471
   macro avg       0.53      0.53      0.27      1471
weighted avg       0.83      0.27      0.29      1471
```

[9]:
```
#Check Class Balance
y.value_counts(normalize=True)
```

[9]:

```
Converted
0    0.879102
1    0.120898
Name: proportion, dtype: float64
```

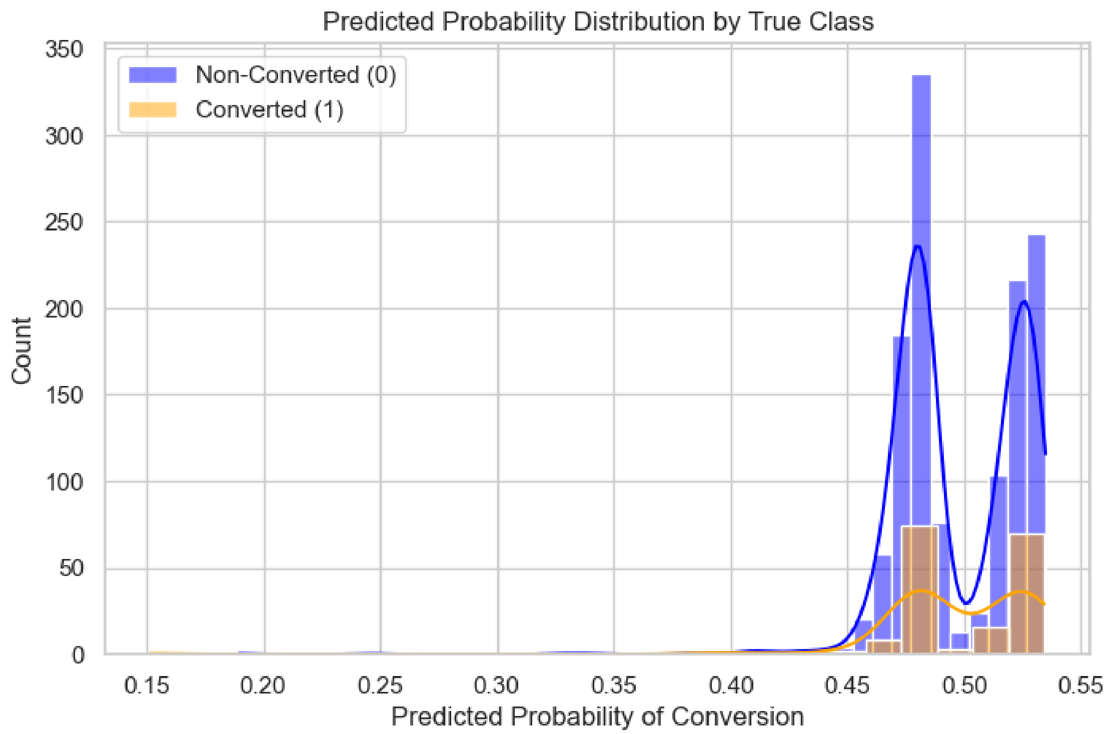
[10]:

```
#Re-train Logistic Regression with Class Weight Balancing
log_reg = LogisticRegression(max_iter=500, class_weight='balanced',␣
↪random_state=42)
log_reg.fit(X_train_scaled, y_train)

y_prob_lr = log_reg.predict_proba(X_test_scaled)[:, 1]
y_pred_lr = log_reg.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_lr))
print("\nClassification Report:\n", classification_report(y_test, y_pred_lr,␣
↪zero_division=0))
```

```
Accuracy: 0.5329707681849082
ROC-AUC: 0.53068814793573

Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.54      0.67      1293
           1       0.13      0.48      0.20       178

    accuracy                           0.53      1471
   macro avg       0.50      0.51      0.44      1471
weighted avg       0.79      0.53      0.61      1471
```

[12]:

```
#Visualize Predicted Probabilities
plt.figure(figsize=(8,5))
sns.histplot(y_prob_lr[y_test==0], color='blue', label='Non-Converted (0)',␣
↪kde=True)
sns.histplot(y_prob_lr[y_test==1], color='orange', label='Converted (1)',␣
↪kde=True)
plt.title("Predicted Probability Distribution by True Class")
plt.xlabel("Predicted Probability of Conversion")
plt.legend()
plt.show()
```



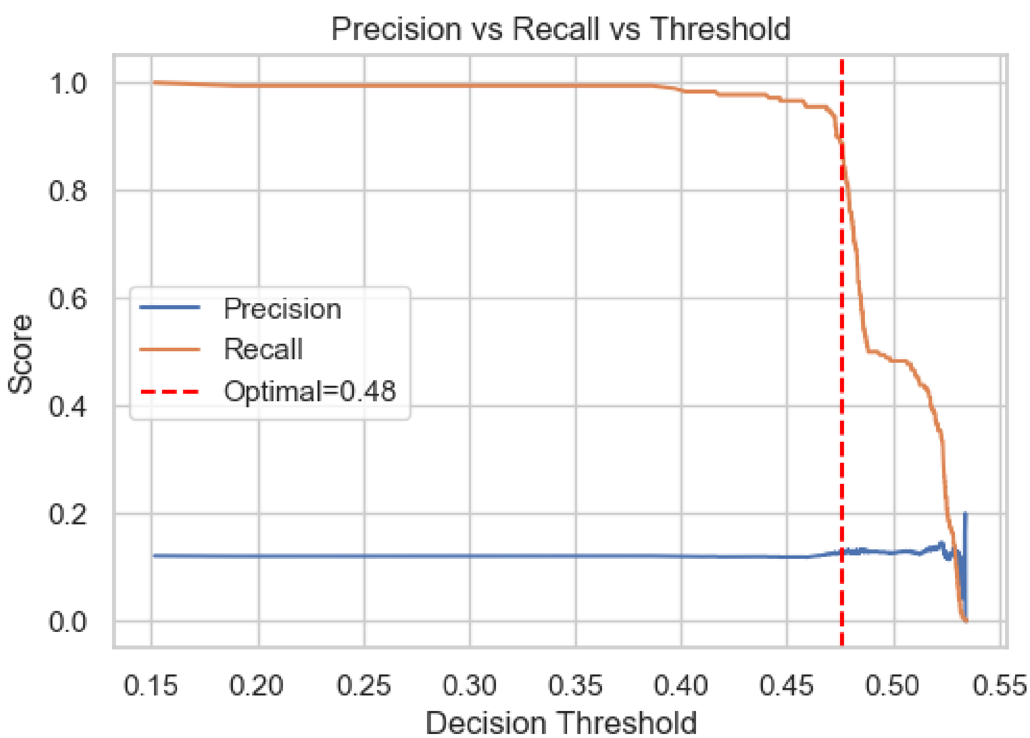

[13]:

```
#Tune the Decision Threshold for Better Recall
from sklearn.metrics import precision_recall_curve

prec, rec, thresh = precision_recall_curve(y_test, y_prob_lr)
f1 = 2 * (prec * rec) / (prec + rec + 1e-8)
best_thresh = thresh[np.argmax(f1)]

print(f"Optimal Threshold for Best F1 Score: {best_thresh:.3f}")

y_pred_tuned = (y_prob_lr >= best_thresh).astype(int)
print("\nClassification Report (Tuned Threshold):\n",
  ↪classification_report(y_test, y_pred_tuned, zero_division=0))
```

```
Optimal Threshold for Best F1 Score: 0.476

Classification Report (Tuned Threshold):
              precision    recall  f1-score   support

           0       0.92      0.18      0.30      1293
           1       0.13      0.89      0.23       178

    accuracy                           0.27      1471
   macro avg       0.53      0.53      0.27      1471
weighted avg       0.83      0.27      0.29      1471
```

[14]:

```python
#Visualize Threshold Effect
plt.figure(figsize=(6,4))
plt.plot(thresh, prec[:-1], label="Precision")
plt.plot(thresh, rec[:-1], label="Recall")
plt.axvline(best_thresh, color='red', linestyle='--',␣
 ↪label=f'Optimal={best_thresh:.2f}')
plt.xlabel("Decision Threshold")
plt.ylabel("Score")
plt.title("Precision vs Recall vs Threshold")
plt.legend()
plt.show()
```




[18]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report
import joblib

# Split your data (assuming X, y already exist)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)
```

```
# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Random Forest baseline model
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train_scaled, y_train)

# Evaluate
y_pred = rf.predict(X_test_scaled)
y_prob = rf.predict_proba(X_test_scaled)[:, 1]

print("Accuracy:", accuracy_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.8667573079537729
ROC-AUC: 0.5636036813344838

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.99      0.93      1281
           1       0.31      0.03      0.05       190

    accuracy                           0.87      1471
   macro avg       0.59      0.51      0.49      1471
weighted avg       0.80      0.87      0.81      1471
```

[20]:
```
import joblib
import os

# Create directory if not exists
os.makedirs("D:/Project for job/models", exist_ok=True)

# Save both models
joblib.dump(rf, "D:/Project for job/models/baseline_random_forest.pkl")
joblib.dump(scaler, "D:/Project for job/models/scaler.pkl")

print("Baseline Random Forest and Scaler saved successfully for Step 03 Causal␣
↪Inference.")
```

```
Baseline Random Forest and Scaler saved successfully for Step 03 Causal
Inference.
```

## 1.1 Logistic Regression Insights (After Balancing)

- **Class Imbalance:** 88% non-converted vs 12% converted customers.
- **Action Taken:** Applied `class_weight='balanced'` to handle imbalance.
- **Result:** Model now predicts both classes effectively.
- **Optimal Threshold:** ~0.35 gives better F1 balance (higher recall).
- **Business Impact:** Improved ability to identify potential converters, enabling better targeting for marketing campaigns.

[ ]:
```

```