Customer Churn Prediction & Retention Decision Engine

(Online Retail Dataset)

Prepared by
Sachin shanbhogue k

Project Objective

Business Problem

Customer churn directly impacts revenue in retail businesses. Retaining an existing customer is significantly cheaper than acquiring a new one, but retention budgets are limited.

Objective

- Predict which customers are likely to churn
- Quantify churn risk using probabilities
- Design a **data-driven retention strategy** to optimize marketing spend

Dataset Overview

Dataset

•Online Retail transactional dataset

•Time period: **Dec 2010 – Dec 2011**

•Invoice-level purchase and return data

Key Fields

•CustomerID

•InvoiceDate

•Quantity

•UnitPrice

•InvoiceNo

•Description

•Country

•StockCode

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 401604 entries, 0 to 401603
Data columns (total 11 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   InvoiceNo      401604 non-null  object
 1   StockCode      401604 non-null  object
 2   Description    401604 non-null  object
 3   Quantity       401604 non-null  int64
 4   InvoiceDate    401604 non-null  datetime64[ns]
 5   UnitPrice      401604 non-null  float64
 6   CustomerID     401604 non-null  int64
 7   Country        401604 non-null  object
 8   quantity_flag  401604 non-null  object
 9   price_flag     401604 non-null  object
 10  invoice_flag   401604 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(2), object(7)
memory usage: 33.7+ MB
```

Your model **is**:

“As of the end of August, will this customer return in the NEXT 3 months?”

Data Cleaning & Preparation

Key Cleaning Steps

•Removed missing CustomerIDs

•Removed invalid transactions (zero or negative prices)

•Separated:

- **Sales transactions** (positive quantities)
- **Returns transactions** (negative quantities)

Why This Matters

•Accurate revenue and return calculations

•Prevents misleading churn signals

```
df_returns = df[df["Quantity"] < 0]
df_sales = df[df["Quantity"] >0]
```

```
df_sales = df_sales[df_sales["UnitPrice"] > 0]
```

```
df_sales["Revenue"] = df_sales["Quantity"] * df_sales["UnitPrice"]
```

```
df_sales.drop({'invoice_flag', 'price_flag','quantity_flag' }, axis=1, inplace=True)
```

Critical Challenge: Data Leakage

Initial Issue Identified

•Unrealistically high model performance (ROC-AUC ≈ 0.99)

•Indicated **temporal target leakage**

Root Cause

•Features and churn labels were derived from the same time window

•Model indirectly “knew the future”

Fix Implemented

•Introduced **time-based splitting**

- Observation window → feature creation (Period: **Before September 2011)**
- Prediction window → churn labeling (Period: **September to December 2011)**

This correction significantly improved model validity and realism.

```
cutoff_date = pd.Timestamp("2011-10-01")

sales_obs = df_sales[df_sales["InvoiceDate"] < cutoff_date]
sales_future = df_sales[df_sales["InvoiceDate"] >= cutoff_date]

returns_obs = df_returns[df_returns["InvoiceDate"] < cutoff_date]
```

“Initially, the model showed near-perfect performance, which indicated temporal data leakage. I fixed this by introducing observation and prediction windows so that features were computed strictly from past data, while churn labels were derived from future behavior. After this correction, the model performance became realistic and deployable.”

Churn Definition

Churn Logic

A customer is considered **churned** if:

- They made purchases during the observation window
- But made **no purchases** during the following 3-month prediction window (sept – dec 2011)

Why This Definition

- Realistic for retail behavior
- Avoids future data leakage
- Easy to explain to business stakeholders

```
future_customers = set(sales_future["CustomerID"].unique())

customer_churn = (
    sales_obs[["CustomerID"]]
    .drop_duplicates()
    .assign(
        Churn=lambda x: (~x["CustomerID"].isin(future_customers)).astype(int)
    )
)
```

Feature Engineering

Customer-Level Features (RFM-based)

Feature	Description
Recency	Days since last purchase
Frequency	Number of distinct invoices
Monetary	Total spend
Tenure	Duration between first and last purchase
ReturnRate	Returned quantity / total quantity

Design Principle

- All features computed **only from observation window**
- Ensures fair prediction setup

```
rfm = (
    sales_obs
    .groupby("CustomerID")
    .agg(
        Recency=("InvoiceDate", lambda x: (snapshot_date - x.max()).days),
        Frequency=("InvoiceNo", "nunique"),
        Monetary=("TotalPrice", "sum"),
        Tenure=("InvoiceDate", lambda x: (x.max() - x.min()).days)
    )
    .reset_index()
)
```

```python
returns_agg = (
    returns_obs
    .groupby("CustomerID")["Quantity"]
    .sum()
    .abs()
    .reset_index(name="ReturnedQty")
)

sales_qty = (
    sales_obs
    .groupby("CustomerID")["Quantity"]
    .sum()
    .reset_index(name="SoldQty")
)

return_rate = sales_qty.merge(
    returns_agg, on="CustomerID", how="left"
).fillna(0)

return_rate["ReturnRate"] = (
    return_rate["ReturnedQty"] / return_rate["SoldQty"]
).clip(0, 1)
```

```python
customer_features = (
    rfm
    .merge(return_rate[["CustomerID", "ReturnRate"]], on="CustomerID", how="left")
    .merge(customer_churn, on="CustomerID", how="left")
)
```

Modeling Approach

Model Used

- Logistic Regression (baseline)

Why Logistic Regression

- Interpretable coefficients
- Produces churn probabilities
- Well-suited for business decision-making

Handling Class Balance

- Stratified train-test split
- Balanced class weights

```
X = customer_features.drop(columns=["CustomerID", "Churn"])
y = customer_features["Churn"]
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.25,
    random_state=42,
    stratify=y
)
```

```
from sklearn.preprocessing import StandardScaler

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

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(
    max_iter=1000,
    class_weight="balanced",
    random_state=42
)

log_reg.fit(X_train_scaled, y_train)
```

```
y_pred = log_reg.predict(X_test_scaled)
y_prob = log_reg.predict_proba(X_test_scaled)[:, 1]
```

Model Performance

Evaluation Metrics

- ROC-AUC: **0.75**
- Recall (Churn): **0.72** at default threshold
- Precision (Churn): **0.65**

Key Insight

- Model performance is realistic and stable
- Indicates strong separation between churners and non-churners

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.70      0.63      0.67       460
           1       0.65      0.72      0.68       444

    accuracy                           0.67       904
   macro avg       0.68      0.68      0.67       904
weighted avg       0.68      0.67      0.67       904
```

```
roc_auc_score(y_test, y_prob)
```

```
0.7554494712103408
```

```
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

confusion_matrix(y_test, y_pred)
```

```
array([[292, 168],
       [126, 318]])
```

Threshold Tuning

Why Threshold Tuning

•Default threshold (0.5) is arbitrary

•Business prefers **catching churners** over missing them

Final Decision

•Selected threshold = **0.40**

•Recall (Churn) ≈ **0.85**

•Balanced precision to control retention costs

This aligns the model with business objectives rather than pure accuracy.

```
Threshold 0.30 | Recall (Churn): 0.95 | Precision (Churn): 0.58
Threshold 0.35 | Recall (Churn): 0.91 | Precision (Churn): 0.60
Threshold 0.40 | Recall (Churn): 0.85 | Precision (Churn): 0.62
Threshold 0.45 | Recall (Churn): 0.79 | Precision (Churn): 0.64
Threshold 0.50 | Recall (Churn): 0.72 | Precision (Churn): 0.65
Threshold 0.55 | Recall (Churn): 0.65 | Precision (Churn): 0.68
Threshold 0.60 | Recall (Churn): 0.54 | Precision (Churn): 0.73
Threshold 0.65 | Recall (Churn): 0.41 | Precision (Churn): 0.76
Threshold 0.70 | Recall (Churn): 0.27 | Precision (Churn): 0.80
Threshold 0.75 | Recall (Churn): 0.09 | Precision (Churn): 0.75
```

```python
import numpy as np
from sklearn.metrics import classification_report

thresholds = np.arange(0.3, 0.8, 0.05)

for t in thresholds:
    y_pred_t = (y_prob >= t).astype(int)
    report = classification_report(y_test, y_pred_t, output_dict=True)
    recall_churn = report['1']['recall']
    precision_churn = report['1']['precision']
    print(
        f"Threshold {t:.2f} | "
        f"Recall (Churn): {recall_churn:.2f} | "
        f"Precision (Churn): {precision_churn:.2f}"
    )
```

```python
best_threshold = 0.40
y_pred_final = (y_prob >= best_threshold).astype(int)

from sklearn.metrics import confusion_matrix, classification_report

confusion_matrix(y_test, y_pred_final)
print(classification_report(y_test, y_pred_final))
```

```
              precision    recall  f1-score   support

           0       0.77      0.49      0.60       460
           1       0.62      0.85      0.71       444

    accuracy                           0.67       904
   macro avg       0.69      0.67      0.66       904
weighted avg       0.70      0.67      0.66       904
```

Retention Decision Engine

From Prediction to Action

Churn probabilities were converted into actionable strategies:

Churn Probability	Retention Action
≥ 0.80	Personal call + 30% discount
0.60 – 0.80	15% discount email
0.40 – 0.60	Engagement campaign
< 0.40	No action

Why This Matters

- Focuses spending on high-risk customers
- Avoids unnecessary discounts to loyal customers

```python
def retention_action(p):
    if p >= 0.80:
        return "High Risk – Personal Call + 30% Discount"
    elif p >= 0.60:
        return "Medium Risk – 15% Discount Email"
    elif p >= 0.40:
        return "Low Risk – Engagement Campaign"
    else:
        return "No Action"

customer_features["Retention_Action"] = (
    customer_features["Churn_Probability"].apply(retention_action)
)
```

The model calculates the churn probabilities and upon which retention action is taken

```python
X_all = customer_features.drop(columns=["CustomerID", "Churn"])

customer_features["Churn_Probability"] = log_reg.predict_proba(
    scaler.transform(X_all)
)[:, 1]
```

Business Value & Impact

Value Delivered

•Identified high-risk customers before churn
•Enabled targeted, cost-efficient retention
•Reduced blanket discounting

Business Benefits

•Higher customer lifetime value
•Better ROI on marketing campaigns
•Scalable decision framework

```
customer_features.head(5)
```

	CustomerID	Recency	Frequency	Monetary	Tenure	ReturnRate	Churn	Churn_Probability	Retention_Action
0	12346	256	1	77183.60	0	1.000000	1	0.822214	High Risk – Personal Call + 30% Discount
1	12347	60	5	2790.86	237	0.000000	0	0.294118	No Action
2	12348	6	4	1797.24	282	0.000000	1	0.303081	No Action
3	12350	240	1	334.40	0	0.000000	1	0.739069	Medium Risk – 15% Discount Email
4	12352	3	7	2194.31	224	0.161369	0	0.187436	No Action

OUTPUT :

CustomerID	Recency	Frequency	Monetary	Tenure	ReturnRate	Churn	Churn_Probability	Retention_Action
12346	256	1	77183.6	0	1	1	0.822213876	High Risk â€“ Personal Call + 30% Discount
12347	60	5	2790.86	237	0	0	0.29411801	No Action
12348	6	4	1797.24	282	0	1	0.303080689	No Action
12350	240	1	334.4	0	0	1	0.739068891	Medium Risk â€“ 15% Discount Email
12352	3	7	2194.31	224	0.161369193	0	0.187436358	No Action
12353	134	1	89	0	0	1	0.672947334	Medium Risk â€“ 15% Discount Email
12354	163	1	1079.4	0	0	1	0.691589452	Medium Risk â€“ 15% Discount Email
12355	145	1	459.4	0	0	1	0.680088874	Medium Risk â€“ 15% Discount Email
12356	176	2	2753.08	80	0	0	0.616417277	Medium Risk â€“ 15% Discount Email
12358	81	1	484.86	0	0	0	0.636673232	Medium Risk â€“ 15% Discount Email
12359	120	3	3495.73	141	0.004145078	0	0.49045626	Low Risk â€“ Engagement Campaign
12360	43	2	1618.28	88	0	0	0.516006838	Low Risk â€“ Engagement Campaign
12361	218	1	189.9	0	0	1	0.726104809	Medium Risk â€“ 15% Discount Email
12362	3	5	2577.34	223	0.007968127	0	0.264212736	No Action
12363	40	2	552	132	0	1	0.499586523	Low Risk â€“ Engagement Campaign
12364	8	2	703.72	34	0	0	0.507834598	Low Risk â€“ Engagement Campaign
12365	222	2	641.38	0	0.005747126	1	0.673853498	Medium Risk â€“ 15% Discount Email
12370	205	3	2802.66	85	0	0	0.572320461	Low Risk â€“ Engagement Campaign
12372	2	3	1298.04	224	0	1	0.376774466	No Action
12373	242	1	364.6	0	0	1	0.740222509	Medium Risk â€“ 15% Discount Email
12375	29	1	230.3	0	0	0	0.599739747	Low Risk â€“ Engagement Campaign
12377	246	2	1628.12	39	0	1	0.677369799	Medium Risk â€“ 15% Discount Email
12378	60	1	4008.62	0	0	1	0.620611655	Medium Risk â€“ 15% Discount Email
12379	12	2	852.24	88	0.002457002	1	0.49325757	Low Risk â€“ Engagement Campaign
12380	8	2	1233.56	107	0	0	0.483461569	Low Risk â€“ Engagement Campaign
12381	50	1	1268.74	0	0.044014085	0	0.620301684	Medium Risk â€“ 15% Discount Email
12383	115	5	1850.56	167	0.001958225	1	0.351107669	No Action

Key Learnings

Temporal data leakage can invalidate ML models

- Model performance must be **realistic**, not perfect
- Threshold tuning is critical in churn problems
- ML is most valuable when paired with decision logic

Final Outcome

Deliverables

- Valid churn prediction model
- Tuned decision threshold
- Retention decision engine
- Business-ready insights

End Result

A complete **end-to-end churn analytics solution**, not just a model.

Final Note “This project demonstrates how to build a realistic churn prediction system, avoid data leakage, and convert ML predictions into business-driven retention strategies.”