

# Creating a Random Forest Classification Model.

## Bottom Line Up Front:

The correlation of the features are too low to get more accurate predictions. Currently predicting at 54% at best, might as well be a coin flip to predict who would churn or not.

We need to collect better data.

### ***Recommendation***

I suggest the following:

- Actual Income amount, not just Low/Medium/High. This would have helped narrow down the tax bracket the customers who churn fall in, and tease out a potential reason for churning.
- Monthly spending to see if it is increasing, decreasing or flat. Might offer insight into predicting who will churn.
- Interaction Quality not just count or resolved. Would answer how the customer felt when it was resolved. Ties in with Monthly Spending, did their spending decrease after an issue was reported, or after it stayed unresolved.
- Check if feedback is positive or negative.
- Consider noSQL to store what the feedback or complaint is, what are they inquiring about to see if customers who churn have the same issue.
- When was the issue reported, when was it resolved. Time to resolution would help me understand better if our customer wait times might be a problem.
- Login dates, not just frequency of login, would be helpful to see if they were logging in a lot at first then suddenly stopped, and maybe we can find out why they stopped by checking the correlation with the above feature recommendations.

The current dataset captures what customers do but not how they feel about the service. I can calculate Customer Satisfaction Scores from the above features and see how that correlates to churn.

With the actual income, I can determine what customers earning churn because from the clustering data, it seems like the lower earners, who spend the most are the ones leaving more so that those who login frequently and spend and complain less. See previous report for more detail.

# Algorithm Selection

Using Random Forest because while accuracy is important, the task specifies that the model must be interpretable to the stakeholders. Random Forest indicates which features mattered most in the decisions and we can show this to stakeholders that this is why the model made a prediction.

XG Boost would be the most accurate, but lacks the interpretable aspect, with each decision tree correcting the one before it, tracing the specific prediction becomes tangled. Logistic Regression will be the easiest to explain but will not be as accurate as random forest.

## HOW TO USE

Customers flagged as high risk (probability  $\geq .30$ ) should be prioritized for proactive outreach by the retention team. The cluster analysis showed that 24% of our most valuable customers churn on average. These are high spenders worth retaining even at the cost of a few false alarms. By combining the cluster membership, the retention team can focus limited resources on high-value customers.

## Implementation:

- I will train the model on 80% of the available Customer\_Data\_Cleaned set and test it on the remaining 20%.
- target (y) is ChurnStatus.
- features (x) are all features except CustomerID because it's not important to the analysis, has no bearing whatsoever.

```
# install libraries
!python -m pip install --upgrade pip -q
!pip install pandas numpy matplotlib seaborn scikit-learn openpyxl -q
!pip install --upgrade openpyxl -q
!pip install xgboost -q

# import libraries

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.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

pd.set_option('display.max_columns', None) # Set option to display all
columns
pd.set_option('display.float_format', '{:,.2f}'.format) # Set float
format to 2 decimal places
```

```
# Load the cleaned customer data we previously created, and verify the
info and the first few rows
```

```
Customer_Data = pd.read_excel('Customer_Data_Cleaned.xlsx')
```

```
Customer_Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 1000 entries, 0 to 999

Data columns (total 27 columns):

#	Column	Non-Null Count		Dtype
---	-----	-----	-----	-----
0	CustomerID	1000	non-null	int64
1	Age	1000	non-null	int64
2	isMale	1000	non-null	int64
3	IncomeLevel	1000	non-null	int64
4	ChurnStatus	1000	non-null	int64
5	TotalSpent	1000	non-null	float64
6	MinTransaction	1000	non-null	float64
7	MaxTransaction	1000	non-null	float64
8	TransactionFrequency	1000	non-null	int64
9	LoyaltyLength	1000	non-null	int64
10	InquiryCount	1000	non-null	int64
11	InquiryResolved	1000	non-null	int64
12	InquiryUnresolved	1000	non-null	int64
13	FeedbackCount	1000	non-null	int64
14	FeedbackResolved	1000	non-null	int64
15	FeedbackUnresolved	1000	non-null	int64
16	ComplaintCount	1000	non-null	int64
17	ComplaintResolved	1000	non-null	int64
18	ComplaintUnresolved	1000	non-null	int64
19	LoginFrequency	1000	non-null	int64
20	MaritalStatus_Divorced	1000	non-null	int64
21	MaritalStatus_Married	1000	non-null	int64
22	MaritalStatus_Single	1000	non-null	int64
23	MaritalStatus_Widowed	1000	non-null	int64
24	ServiceUsage_Mobile App	1000	non-null	int64
25	ServiceUsage_Online Banking	1000	non-null	int64
26	ServiceUsage_Website	1000	non-null	int64

```
dtypes: float64(3), int64(24)
```

```
memory usage: 211.1 KB
```

```
# Separate features and target variable
```

```
X = Customer_Data.drop(columns=['CustomerID', 'ChurnStatus'])
```

```
y = Customer_Data['ChurnStatus']
```

[illegible]

```

makes it reproducible same split every time
churn ratio distributed and balanced in both sets
stratify= y # keep
)

# Scale the X_train and X_test data using StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_train = X_train_scaled
X_test_scaled= scaler.transform(X_test)
X_test = X_test_scaled

```

### **Reasoning:**

When creating the for loop to check for hyperparameters, I could have used GridSearchCV but I wanted to evaluate for false alarms as well.

```

# Create the for loop to tune hyperparameters
best_recall = 0
best_settings = {}

for n_trees in [100, 200, 500]: # select number of trees
    for depth in [5, 10, 20, None]: # select max depth
        for weight in [1, 2, 5, 10]: # select class weight for
churners
            for threshold in [0.3, 0.4, 0.5]: # select probability
threshold
                model = RandomForestClassifier(n_estimators=n_trees,
max_depth=depth,
class_weight={0: 1, 1:
weight},
random_state=42) #
random state for reproducibility
                model.fit(X_train, y_train)
                probs = model.predict_proba(X_test)[: , 1]
                predictions = (probs >= threshold).astype(int)

                caught = confusion_matrix(y_test, predictions)[1, 1]
                false_alarms = confusion_matrix(y_test, predictions)
[0, 1]

                recall = caught / 41 # there are 41 actual churners in
the test set, caught divided by total actual churners

                # Only save if this is the best so far AND false
alarms are reasonable
                if recall > best_recall and false_alarms < 80: # stop
if more than 80 false alarms, store settings
                    best_recall = recall

```

```

        best_settings = {
            'trees': n_trees,
            'depth': depth,
            'weight': weight,
            'threshold': threshold,
            'caught': caught,
            'false_alarms': false_alarms
        }

print("Best settings:", best_settings)
print(f"Recall: {best_recall:.2f}")

Best settings: {'trees': 500, 'depth': 10, 'weight': 5, 'threshold': 0.3, 'caught': np.int64(20), 'false_alarms': np.int64(76)}
Recall: 0.49

```

*# Build the RandomForestClassifier class on the best found parameters*

```

model = RandomForestClassifier(n_estimators=500, max_depth = 10,
class_weight={0:1, 1:5}, random_state=42).fit(X_train, y_train)
probability = model.predict_proba(X_test)[: , 1]
threshold = 0.30
predictions = (probability >= threshold).astype(int)
accuracy = accuracy_score(y_test, predictions)
class_report = classification_report(y_test, predictions)
conf_matrix = confusion_matrix(y_test, predictions)
print(f'Accuracy: {accuracy:.2f}')
print("Classification Report:\n", class_report)
print("Confusion Matrix:\n", conf_matrix)

```

Accuracy: 0.52

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.52	0.63	159
1	0.21	0.49	0.29	41
accuracy			0.52	200
macro avg	0.50	0.50	0.46	200
weighted avg	0.68	0.52	0.56	200

Confusion Matrix:

```

[[83 76]
 [21 20]]

```

*# Check feature importance*

```

importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': model.feature_importances_
})

```

```
}).sort_values(by='Importance', ascending=False)
display(importance)
```

	Feature	Importance
4	MinTransaction	0.12
3	TotalSpent	0.12
17	LoginFrequency	0.12
5	MaxTransaction	0.11
7	LoyaltyLength	0.11
0	Age	0.10
6	TransactionFrequency	0.05
2	IncomeLevel	0.03
11	FeedbackCount	0.02
14	ComplaintCount	0.02
1	isMale	0.02
8	InquiryCount	0.02
18	MaritalStatus_Divorced	0.02
15	ComplaintResolved	0.02
24	ServiceUsage_Website	0.01
22	ServiceUsage_Mobile App	0.01
21	MaritalStatus_Widowed	0.01
10	InquiryUnresolved	0.01
23	ServiceUsage_Online Banking	0.01
16	ComplaintUnresolved	0.01
19	MaritalStatus_Married	0.01
20	MaritalStatus_Single	0.01
13	FeedbackUnresolved	0.01
9	InquiryResolved	0.01
12	FeedbackResolved	0.01

**Observations:** The features have no real weight. It tracks because of the low correlation across the board. **Why it Matters:** The data collection is insufficient, please see recommendations at the top of the report.

Because of how inaccurate the model using RandomForest is, I decided to check it against XGBoost. The results were similar.

```
from xgboost import XGBClassifier

best_recall = 0
best_settings = {}

for n_trees in [100, 200, 500]:
    for depth in [3, 5, 10]:
        for weight in [5, 10, 15, 20]:
            for threshold in [0.2, 0.3, 0.4, 0.5]:
                model = XGBClassifier(n_estimators=n_trees,
                                     max_depth=depth,
                                     scale_pos_weight=weight,
                                     random_state=42)
```

```

model.fit(X_train, y_train)
probs = model.predict_proba(X_test)[: , 1]
predictions = (probs >= threshold).astype(int)

caught = confusion_matrix(y_test, predictions)[1, 1]
false_alarms = confusion_matrix(y_test, predictions)

[0, 1]

recall = caught / 41

if recall > best_recall and false_alarms < 80:
    best_recall = recall
    best_settings = {
        'trees': n_trees,
        'depth': depth,
        'weight': weight,
        'threshold': threshold,
        'caught': caught,
        'false_alarms': false_alarms
    }

```

```

print("Best XGBoost settings:", best_settings)
print(f"Recall: {best_recall:.2f}")

```

```

Best XGBoost settings: {'trees': 200, 'depth': 3, 'weight': 5,
'threshold': 0.2, 'caught': np.int64(20), 'false_alarms':
np.int64(72)}
Recall: 0.49

```

```

model = XGBClassifier(n_estimators=200,
                      max_depth=3,
                      scale_pos_weight=5,
                      random_state=42)

model.fit(X_train, y_train)
probs = model.predict_proba(X_test)[: , 1]
predictions = (probs >= 0.2).astype(int)
accuracy = accuracy_score(y_test, predictions)
print(f'Accuracy: {accuracy:.2f}')
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))

```

```

Accuracy: 0.54

```

```

[[87 72]
 [21 20]]

```

	precision	recall	f1-score	support
0	0.81	0.55	0.65	159
1	0.22	0.49	0.30	41
accuracy			0.54	200
macro avg	0.51	0.52	0.48	200

weighted avg	0.68	0.54	0.58	200
--------------	------	------	------	-----