The leaky bucket: predicting customer churn

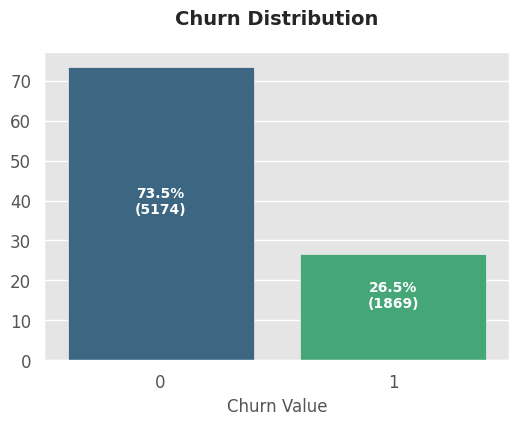
In this project, we will look at a (fictional) IBM dataset of telco customers, and try to beat customer churn score which uses predictive IBM (International Business Machines) SPSS (Statistical Product and Service Solutions) Modeler!

Because acquiring customers is much more costly than keeping them (~ 5-10 times as much), attrition rate is closely watched by executives, and knowing which customers are likely to leave in advance would allow retention efforts to focus on them. Normal rate of churn for a telecom company is less than 1% per month.

## Data exploration

Dataset has 7043 rows and 33 columns, however some are redundant: Lat and Long are the same as Latitude and Longitude; Country (US), Count (1) and State (CA) are identical across the entire data set.

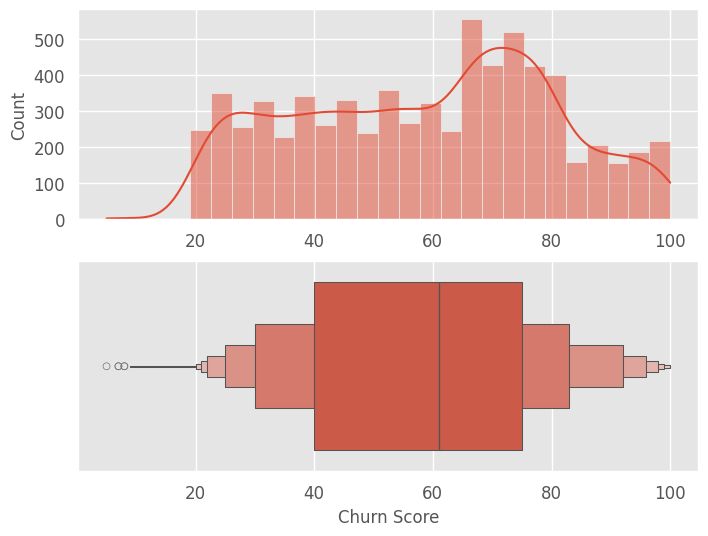
The variable we are interested in – customer churn – is binary: either customer left the carrier during the quarter (1) or not (0). Let us see how balanced two classes are:



With a 2.77:1 ratio, the classes are well balanced, and there is no need for tricks (or is there?). This would not normally be the case if you have a full dataset, as customers of a company like this would have a half-life of only 7 months!

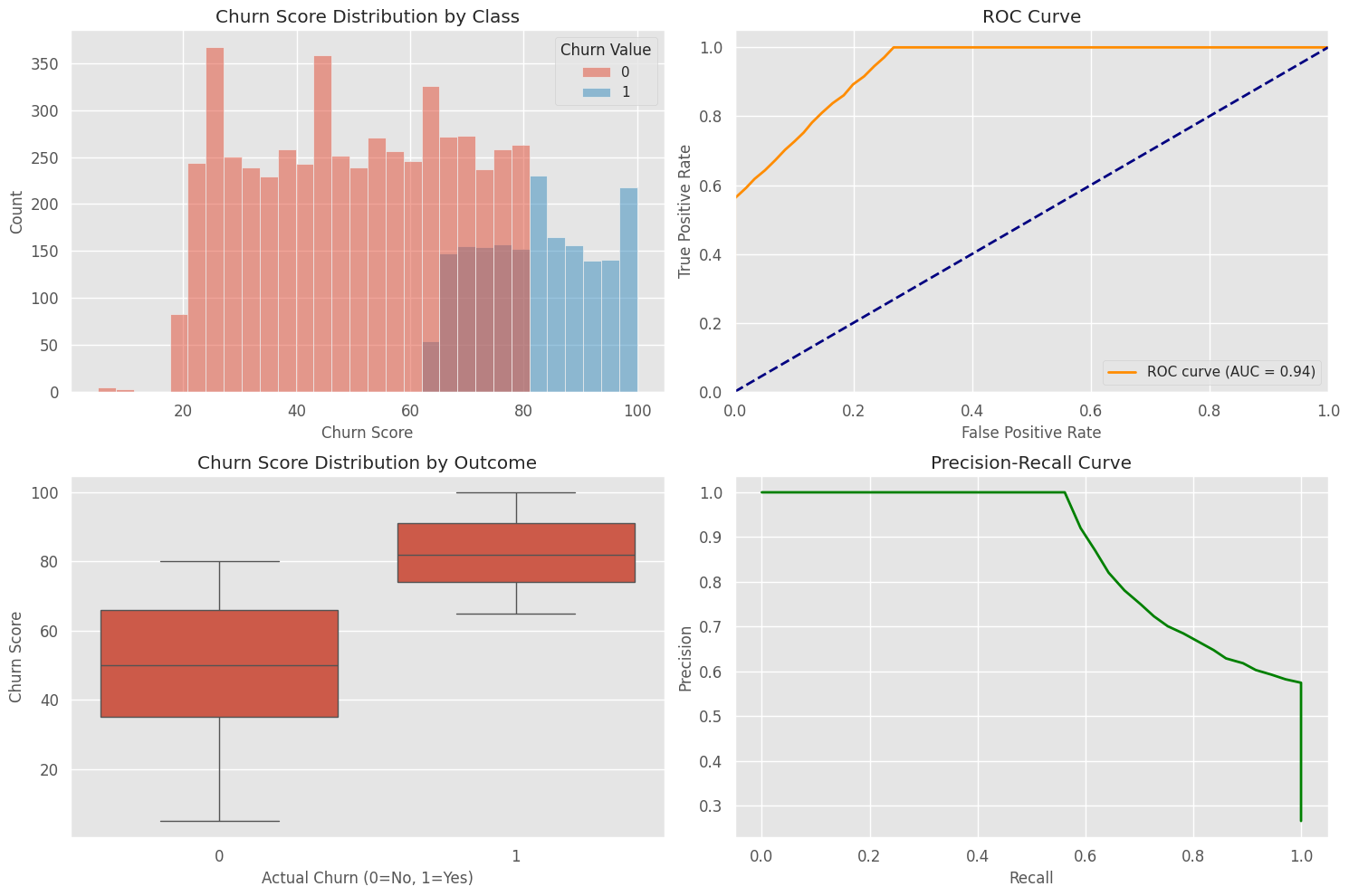
## Churn score model

IBM has kindly provided a churn score meant to predict a likelihood of costumer leaving:



The reason the score is used instead of a simple classifier is because it allows the resources to be targeted towards high-values customer most likely to leave. Moreover, there is often a “point of no return”, when customer has already decided to switch, and retention costs would be high. Therefore, the interventions are usually most effective at “warm” range.

Let us see how churn score performs. We immediately see the difference in scores between churners and non-churners.



The performance is not too bad.

Classification Report using optimal threshold:

precision recall f1-score support

0 0.93 0.85 0.89 5174

1 0.67 0.81 0.73 1869

accuracy 0.84 7043

macro avg 0.80 0.83 0.81 7043

weighted avg 0.86 0.84 0.85 7043

Churn Score is an excellent predictor of actual churn:

-0.665 correlation, AUC 0.94 suggests the a strong scoring system that predicts churn, and we should aim to beat that.

- significant proportion of customers with moderate churn risk.

- around 65–80, represent customers with higher churn risk.

- fewer customers with low (below 20) or high (close to 100) churn scores.

- The kernel density estimate (KDE) curve is fairly evenly distributed between 25 and 65, indicating a wide range of churn risk among customers.

- boxplot indicates the interquartile range (IQR) is roughly between 40 and 75, representing the middle 50% of customers.

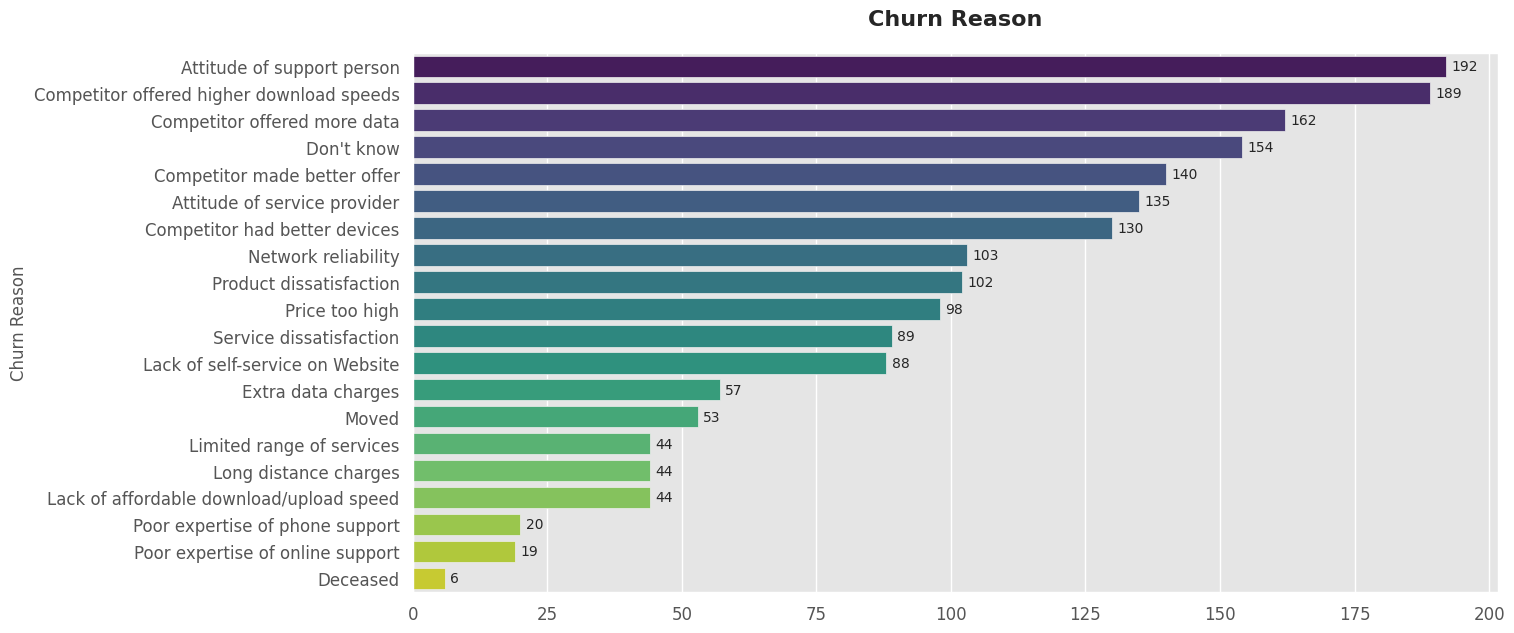
- median churn score is around 60, suggesting a moderate overall churn risk.

- few outliers exist on the lower side (below 20), representing customers with low churn risk.

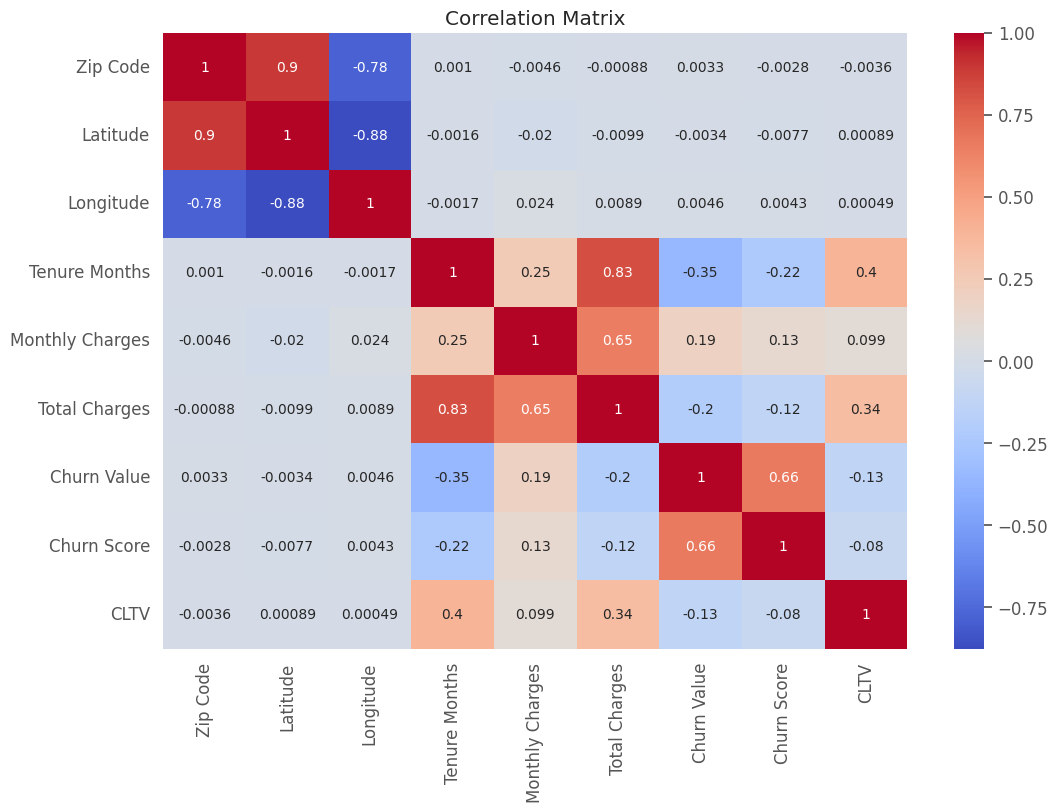
- The cluster of customers with churn scores between 70 and 80 is a high-risk group that should be prioritized for retention strategies.

- Customers with scores below 40 represent loyal customers with a lower churn risk, which is a positive sign.

Churn reasons are exactly what you would expect (“bad” customer service and better offers from competitors):



Now move on to correlation between variables to see which provide no additional value:



Latitude and Longitude: high negative correlation (-0.88) - when one increases, the other tends to decrease (CA geography).

Tenure Months and Total Charges: high positive correlation (0.83) - customers who stay longer tend to have higher total charges.

Churn Value and Churn Score: moderate positive correlation (0.66) - indicates higher churn scores are associated with higher churn values.

Churn Value and Tenure Months: A moderate negative correlation (-0.35) suggests that customers who stay longer are less likely to churn.

Because of high correlation with latitude and longitude, we decided to drop zip code. Also Total and monthly charges were cut into 10 bins each. Additionally, Churn reason was dropped as it could not be known in advance.

When training our own models, churn score was removed, as it is in principle calculated from the data.

# Churn score model

As a baseline, we built a simple logistic regression model using only the churn score

precision recall f1-score support

0 0.90 0.92 0.91 1035

1 0.77 0.71 0.74 374

accuracy 0.87 1409

macro avg 0.83 0.82 0.82 1409

weighted avg 0.86 0.87 0.86 1409

Overall performance:

Accuracy: 87%

ROC AUC: 0.94 (matches the original Churn Score's performance)

Class-specific performance:

Non-churners:

Precision: 90% - When model predicts non-churn, it's right 90% of the time

Recall: 92% - Model correctly identifies 92% of actual non-churners

F1-score: 91%

Churners:

Precision: 77% - When model predicts churn, it's right 77% of the time

Recall: 71% - Model correctly identifies 71% of actual churners

F1-score: 74%

As you can see, the performance is a bit unbalanced – it does worse with churn class.

Now we try to do better than IBM.

# Training other models

After putting churn score aside, we tried 8 different models: Logistic regression (very simple), Random forest, Decision Trees, (usual choices for this type of problems), Gradeint Boosting, XGBoost, LightGBM, SVC, AdaBoost, and KNN. We used standard scaler and mean value imputer for numerical features, and one hot encoder (with drop first) for categorical ones. To deal with class imbalance, stratified sampling was used.

Model Performance Comparison:

Model CV ROC-AUC Mean CV ROC-AUC Std Test Accuracy \

6 GradientBoosting 0.8603 0.0142 0.7913

0 LogisticRegression 0.8568 0.0127 0.7977

3 LightGBM 0.8536 0.0126 0.7892

5 AdaBoost 0.8523 0.0159 0.8013

1 RandomForest 0.8495 0.0096 0.7935

2 XGBoost 0.8391 0.0123 0.7630

4 SVC 0.8369 0.0128 0.7977

7 KNN 0.7933 0.0125 0.7615

8 DecisionTree 0.6673 0.0118 0.7374

Test ROC-AUC Precision Recall F1-Score

6 0.8517 0.7796 0.7913 0.7819

0 0.8454 0.7912 0.7977 0.7936

3 0.8427 0.7812 0.7892 0.7840

5 0.8499 0.7936 0.8013 0.7961

1 0.8368 0.7825 0.7935 0.7848

2 0.8242 0.7561 0.7630 0.7590

4 0.8154 0.7868 0.7977 0.7888

7 0.7863 0.7619 0.7615 0.7617

8 0.6693 0.7406 0.7374 0.7389

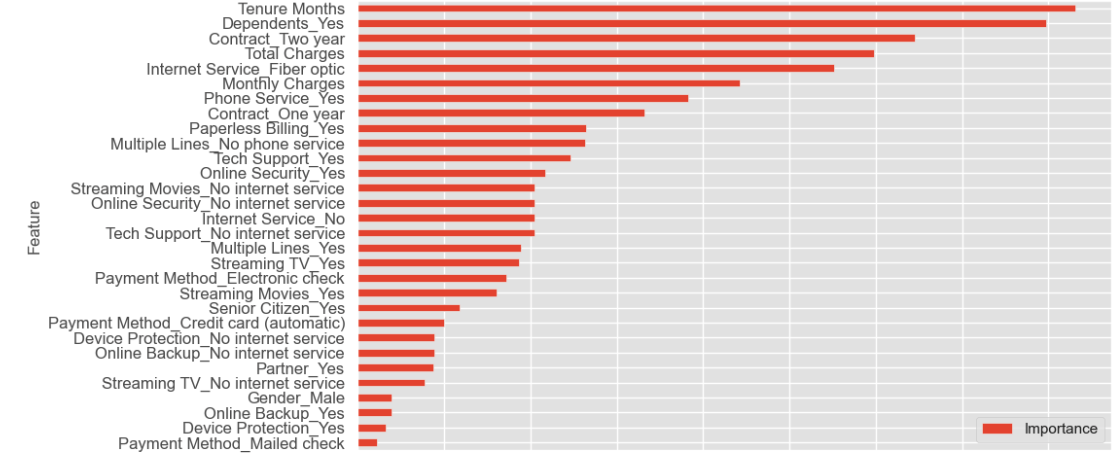
The top 3 performers (by CV ROC-AUC Mean):

1. GradientBoosting (0.8603)
   1. Strong performance across all metrics, suggesting effective handling of complex patterns in the data
   2. High ROC-AUC suggests it balances sensitivity and specificity well
2. Logistic Regression (0.8568)
   1. Super simple, yet effective.
   2. High F1-Score suggests it balances precision and recall well
3. LightGBM (0.8536)
   1. Shows decent accuracy and ROC-AUC

Based on this, will do some fine tuning on:

1. *GradientBoosting*, *LightGBM* (show potential with high ROC-AUC and balanced metrics)
2. AdaBoost (high accuracy, will need to watch for overfitting)
3. LogisticRegression (it's just so simple and it works... must give it a chance, even though tuning likely won't change much, but…)

## Features importance for LR



Long-term customers tend to stay that way. Also 1 or 2-year contracts obviously work for their duration.

## Logistic Regression with class weights

If we prioritise identifying potential churners, e.g. bias the weights 85/15 in favour of churn class, we get pretty good result:

**Precision churners** 48%

**Recall churners** 92%

Not bad for a “linear” model that has almost no tuning parameters! (Synthetic Minority Over-sampling looked like a good idea, but didn’t significantly improve results)

## Fine-turning models

Let us try fine-tuning the 4 models.

We created simple grids with 3-4 different values of each parameter.

The results are below:

Tuned Model Performance Comparison:

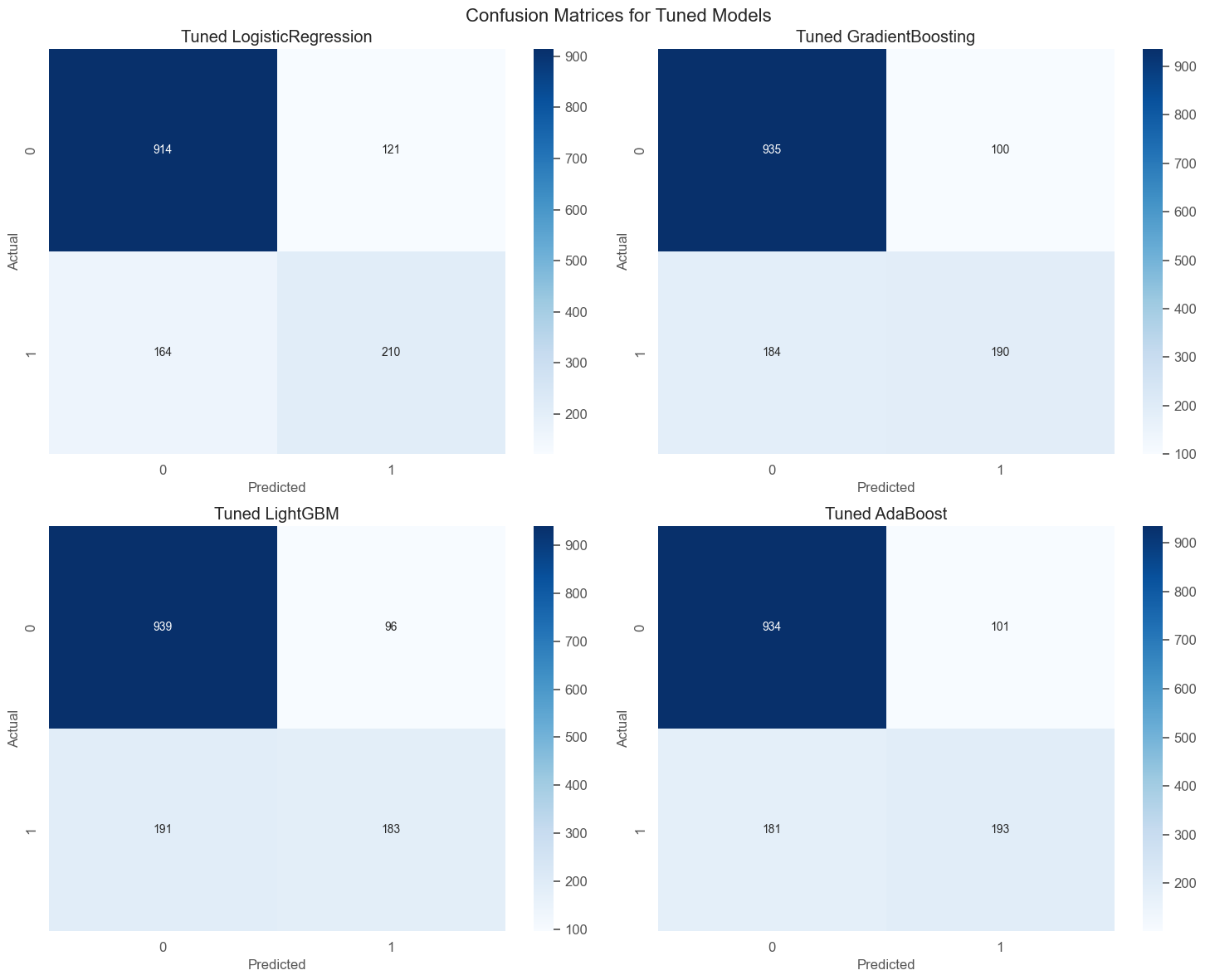
Model Test Accuracy Test ROC-AUC Precision Recall F1-Score

Tuned LR 0.7977 0.8443 0.7912 0.7977 0.7936

Tuned GB 0.7984 0.8547 0.7877 0.7984 0.7896

Tuned LightGBM 0.7963 0.8574 0.7845 0.7963 0.7860

Tuned AdaBoost 0.7999 0.8524 0.7896 0.7999 0.7916



All models still suffer from worse performance in the churn class(1)

Improvement over baseline:

LogisticRegression: -0.13% improvement

GradientBoosting: 0.36% improvement

LightGBM: 1.74% improvement

AdaBoost: 0.30% improvement

Fine Tuning Conclusion

* All tuned models performed very similarly, with test accuracy (0.7963-0.7999)
* *AdaBoost* achieved highest accuracy and F1-Score
* *LogisticRegression* remained highly performant despite simplicity, and as suspected, tuning didn't change much
* *LightGBM* benefited the most from tuning

Single Model Recommendation

1. First choice: **AdaBoost**
   * Highest accuracy and F1-Score
   * Simple parameter set
   * Good balance of metrics
2. Second choice: **GradientBoosting**
   * More robust to different types of data
   * Better handling of complex patterns

## Ensemble Model

We tried to improve performance by building ensemble of 4 models (LR, GB, AdaBoost, LightGBM). Improvement over “best” single model (AdaBoost) is marginal:

| Simple Voting: | Weighted Voting: | Stacking: |
| --- | --- | --- |
| Test Accuracy: +0.27% | Test Accuracy: +0.18% | Test Accuracy: +0.09% |
| Test ROC-AUC: -0.13% | Test ROC-AUC: -0.12% | Test ROC-AUC: -0.08% |
| Test F1-Score: +0.10% | Test F1-Score: -0.01% | Test F1-Score: +0.12% |

Simple Voting does as well as more complex methods, so stick with it.

## Recommendations

* First option: **Simple Voting Ensemble**
  + Slight accuracy improvement over AdaBoost
  + Simple to implement
  + Stable performance
* Second option: stick with AdaBoost to save resources.

## Questions for further study and discussion:

* Why is minority class performance almost always worse even after stratified sampling, SMOTE, etc.?
* Is Churn Score calculated from the full dataset, that’s why it is better?
* How in the world some people got 99% precision and recall on this dataset?