Model Creation and Evaluation for Churn Prediction

Andres Perez

Model Creation and Evaluation for Churn Prediction

This document covers the process of building, tuning, and evaluating predictive models for customer churn. We will use the features engineered in the previous step and compare different modeling approaches.

1. Data Loading and Preparation

```r
# Load the engineered features data
# Update the path if needed
data <- read.csv("data/EngineeredChurnData.csv")

# Print column names for diagnostics
print("Columns in loaded data:")
```

```
## [1] "Columns in loaded data:"
```

```r
print(colnames(data))
```

```
##  [1] "Customer.Months"          "Churn"
##  [3] "CHI.Score.Mon0"           "CHI.Score"
##  [5] "Support.Cases.Mon0"       "Support.Cases"
##  [7] "SP.Mon0"                  "SP"
##  [9] "Logins"                   "Blog.Articles"
## [11] "Views"                    "Days.Since.Last.Login"
## [13] "Logins_log"               "Views_log"
## [15] "Blog.Articles_log"        "Views_per_Login"
## [17] "Blog_per_Login"           "Support_Score_Interaction"
## [19] "Login_View_Interaction"   "Activity_Score"
```

```r
# Prepare data for modeling (handle NAs, etc.)
# Identify numeric columns (excluding Churn)
numeric_cols <- sapply(data, is.numeric) & names(data) != "Churn"
# Remove numeric columns with all NA or zero variance
keep_numeric <- sapply(data[, numeric_cols, drop=FALSE], function(x) !all(is.na(x))
        && sd(x, na.rm=TRUE) > 0)
keep_cols <- intersect(c(names(data)[numeric_cols][keep_numeric], "Churn"), colname
        s(data))
data_clean <- data[, keep_cols, drop=FALSE]

# Remove rows with any NA, NaN, or Inf values
data_clean <- data_clean[complete.cases(data_clean) & apply(data_clean, 1, function
        (row) all(is.finite(as.numeric(row)))), ]

# Ensure Churn is a factor for classification
if("Churn" %in% colnames(data_clean) && !is.factor(data_clean$Churn)) {
  data_clean$Churn <- as.factor(data_clean$Churn)
}

# Check class balance
if("Churn" %in% colnames(data_clean)) table(data_clean$Churn)
```

```
##
##    0    1
## 5422  291
```

```r
# Use only the top 10 features (plus Churn) for modeling
selected_features <- c(
  "Customer.Months",
  "Days.Since.Last.Login",
  "CHI.Score.Mon0",
  "Activity_Score",
  "CHI.Score",
  "Logins",
  "Views_log",
  "Logins_log",
  "Views",
  "Login_View_Interaction",
  "Churn"
)
selected_features <- intersect(selected_features, colnames(data_clean))
data_model <- data_clean[, selected_features, drop=FALSE]
```

2. Train-Test Split

```
set.seed(123)
train_index <- createDataPartition(data_model$Churn, p = 0.8, list = FALSE)
train_data <- data_model[train_index, ]
test_data <- data_model[-train_index, ]
```

3. Baseline Model: Logistic Regression

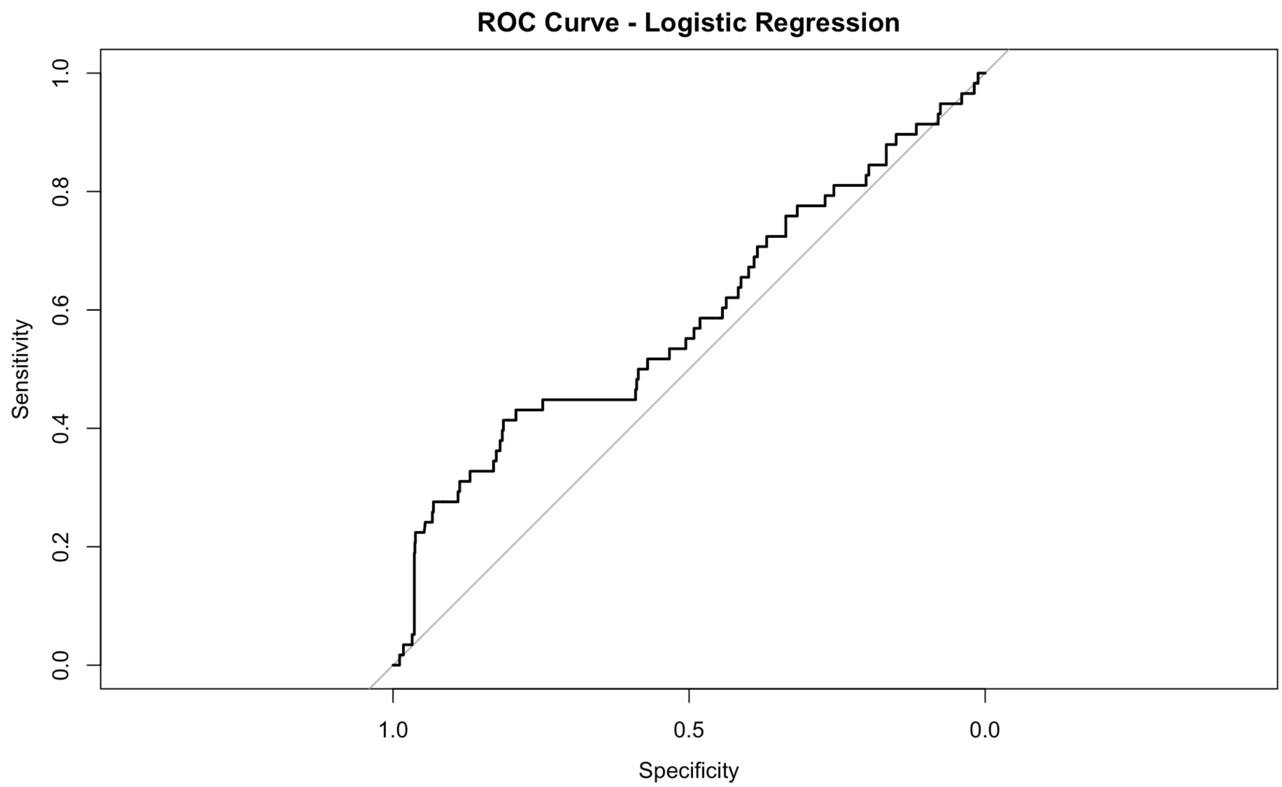
```
logit_model <- glm(Churn ~ ., data = train_data, family = binomial)
summary(logit_model)
```

```
##
## Call:
## glm(formula = Churn ~ ., family = binomial, data = train_data)
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -2.871e+00  1.311e-01 -21.903  < 2e-16 ***
## Customer.Months         9.349e-03  6.371e-03   1.467  0.14229
## Days.Since.Last.Login   2.973e-02  5.611e-03   5.299 1.17e-07 ***
## CHI.Score.Mon0         -3.861e-03  1.326e-03  -2.911  0.00360 **
## Activity_Score          1.288e-03  1.364e-03   0.944  0.34504
## CHI.Score              -8.662e-03  2.859e-03  -3.030  0.00245 **
## Logins                 -4.149e-03  2.866e-03  -1.448  0.14773
## Views                  -2.562e-04  1.291e-04  -1.984  0.04725 *
## Login_View_Interaction  7.200e-07  1.814e-06   0.397  0.69134
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1840.9  on 4570  degrees of freedom
## Residual deviance: 1744.2  on 4562  degrees of freedom
## AIC: 1762.2
##
## Number of Fisher Scoring iterations: 7
```

```
# Predict on test set
logit_pred <- predict(logit_model, newdata = test_data, type = "response")
logit_pred_class <- ifelse(logit_pred > 0.5, 1, 0)

# Evaluate
confusionMatrix(as.factor(logit_pred_class), test_data$Churn, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1083   58
##          1    1    0
##
##                Accuracy : 0.9483
##                  95% CI : (0.9339, 0.9604)
##     No Information Rate : 0.9492
##     P-Value [Acc > NIR] : 0.5876
##
##                   Kappa : -0.0017
##
##  Mcnemar's Test P-Value : 3.086e-13
##
##             Sensitivity : 0.0000000
##             Specificity : 0.9990775
##          Pos Pred Value : 0.0000000
##          Neg Pred Value : 0.9491674
##              Prevalence : 0.0507881
##          Detection Rate : 0.0000000
##    Detection Prevalence : 0.0008757
##       Balanced Accuracy : 0.4995387
##
##        'Positive' Class : 1
##
```

```
roc_logit <- roc(as.numeric(test_data$Churn), as.numeric(logit_pred))
plot(roc_logit, main = "ROC Curve - Logistic Regression")
```




```
auc(roc_logit)
```

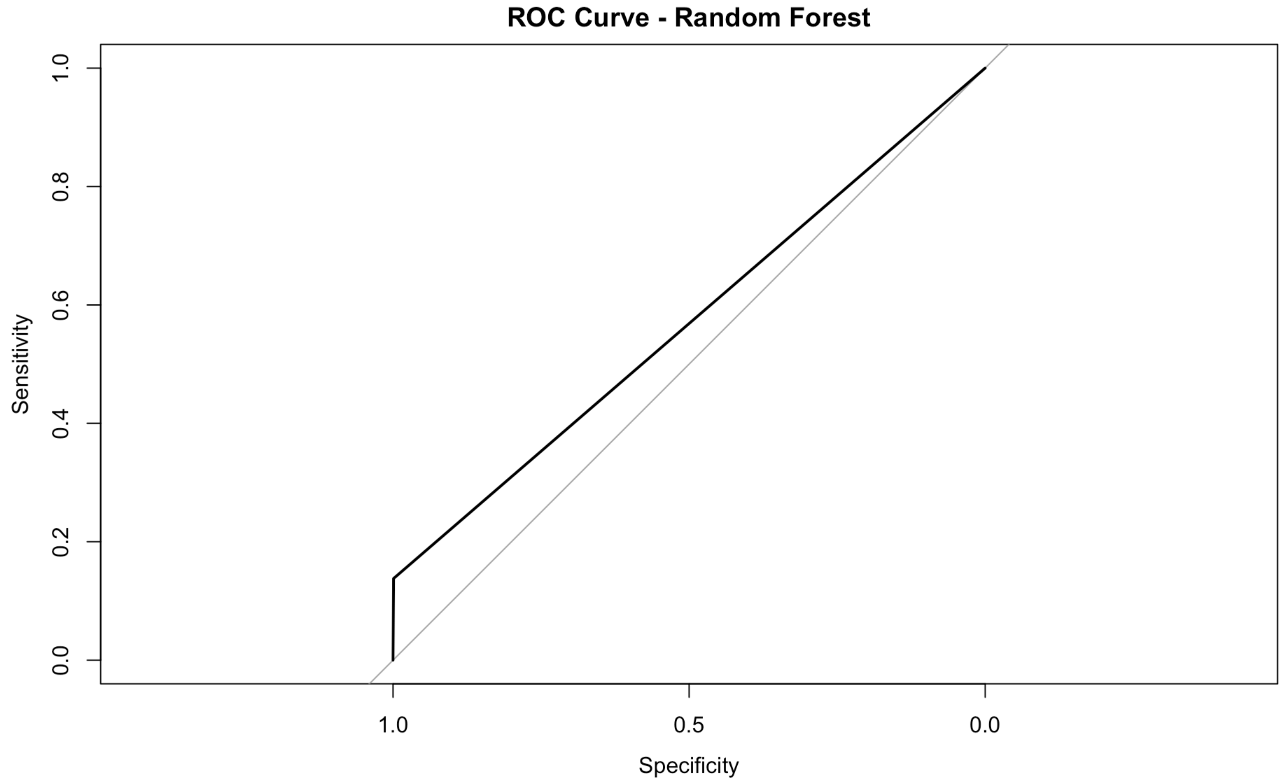
```
## Area under the curve: 0.5866
```

4. Random Forest Model

```
rf_model <- randomForest(Churn ~ ., data = train_data, importance = TRUE, ntree = 2
        00)
rf_pred <- predict(rf_model, newdata = test_data, type = "response")

# Evaluate
confusionMatrix(rf_pred, test_data$Churn, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1083   50
##          1    1    8
##
##                Accuracy : 0.9553
##                  95% CI : (0.9417, 0.9666)
##     No Information Rate : 0.9492
##     P-Value [Acc > NIR] : 0.1917
##
##                   Kappa : 0.2283
##
##  Mcnemar's Test P-Value : 1.801e-11
##
##             Sensitivity : 0.137931
##             Specificity : 0.999077
##          Pos Pred Value : 0.888889
##          Neg Pred Value : 0.955869
##              Prevalence : 0.050788
##          Detection Rate : 0.007005
##    Detection Prevalence : 0.007881
##       Balanced Accuracy : 0.568504
##
##        'Positive' Class : 1
##
```

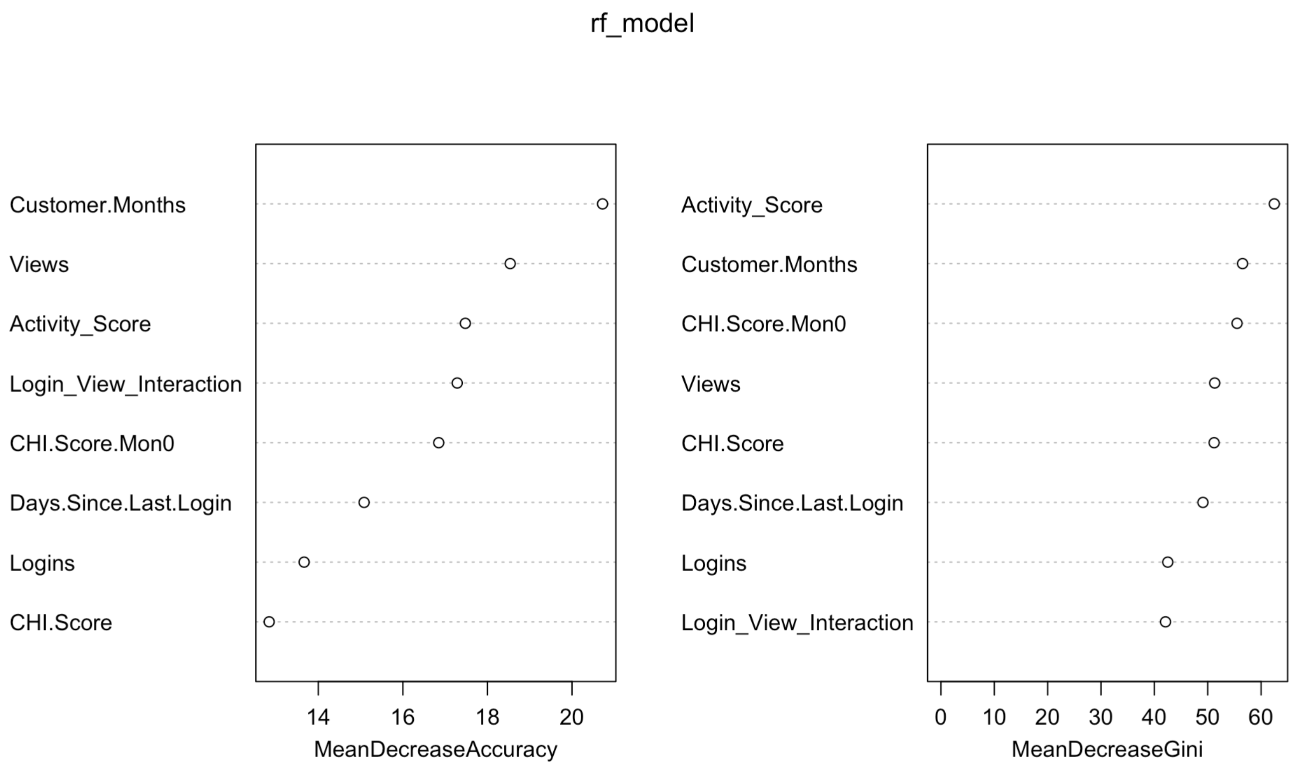
```r
roc_rf <- roc(as.numeric(test_data$Churn), as.numeric(rf_pred))
plot(roc_rf, main = "ROC Curve - Random Forest")
```




```r
auc(roc_rf)
```

```
## Area under the curve: 0.5685
```

```r
# Feature importance plot
varImpPlot(rf_model)
```

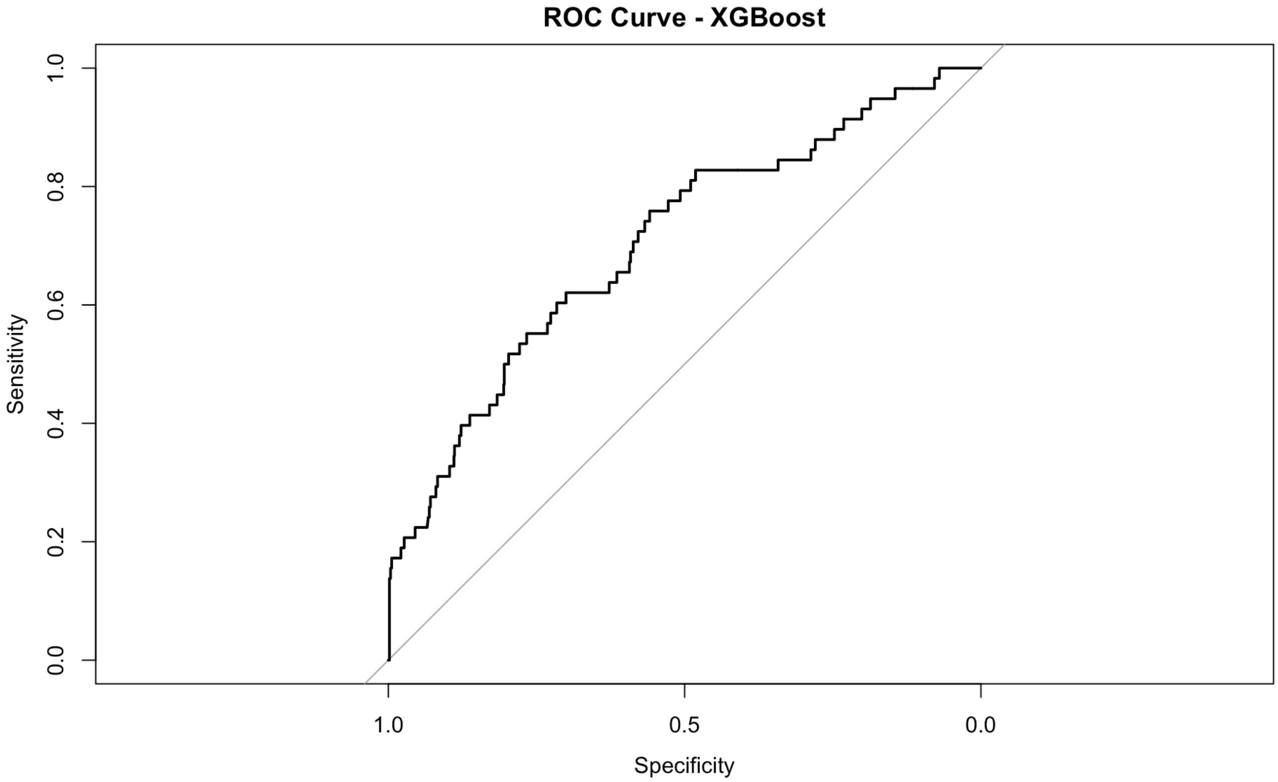


5. XGBoost Model

```
# Prepare data for xgboost (numeric matrix, 0/1 labels)
xgb_train <- train_data %>% mutate(Churn = as.numeric(as.character(Churn)))
xgb_test <- test_data %>% mutate(Churn = as.numeric(as.character(Churn)))
train_matrix <- as.matrix(xgb_train %>% select(-Churn))
test_matrix <- as.matrix(xgb_test %>% select(-Churn))
train_label <- xgb_train$Churn

dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
dtest <- xgb.DMatrix(data = test_matrix)

params <- list(objective = "binary:logistic", eval_metric = "auc")
xgb_model <- xgboost(params = params, data = dtrain, nrounds = 100, verbose = 0)
xgb_pred <- predict(xgb_model, dtest)
xgb_pred_class <- ifelse(xgb_pred > 0.5, 1, 0)

# Evaluate
confusionMatrix(as.factor(xgb_pred_class), as.factor(xgb_test$Churn), positive =
        "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1079   49
##          1    5    9
##
##                Accuracy : 0.9527
##                  95% CI : (0.9388, 0.9643)
##     No Information Rate : 0.9492
##     P-Value [Acc > NIR] : 0.3242
##
##                   Kappa : 0.2349
##
##  Mcnemar's Test P-Value : 4.87e-09
##
##             Sensitivity : 0.155172
##             Specificity : 0.995387
##          Pos Pred Value : 0.642857
##          Neg Pred Value : 0.956560
##              Prevalence : 0.050788
##          Detection Rate : 0.007881
##    Detection Prevalence : 0.012259
##       Balanced Accuracy : 0.575280
##
##        'Positive' Class : 1
##
```

```
roc_xgb <- roc(xgb_test$Churn, xgb_pred)
plot(roc_xgb, main = "ROC Curve - XGBoost")
```

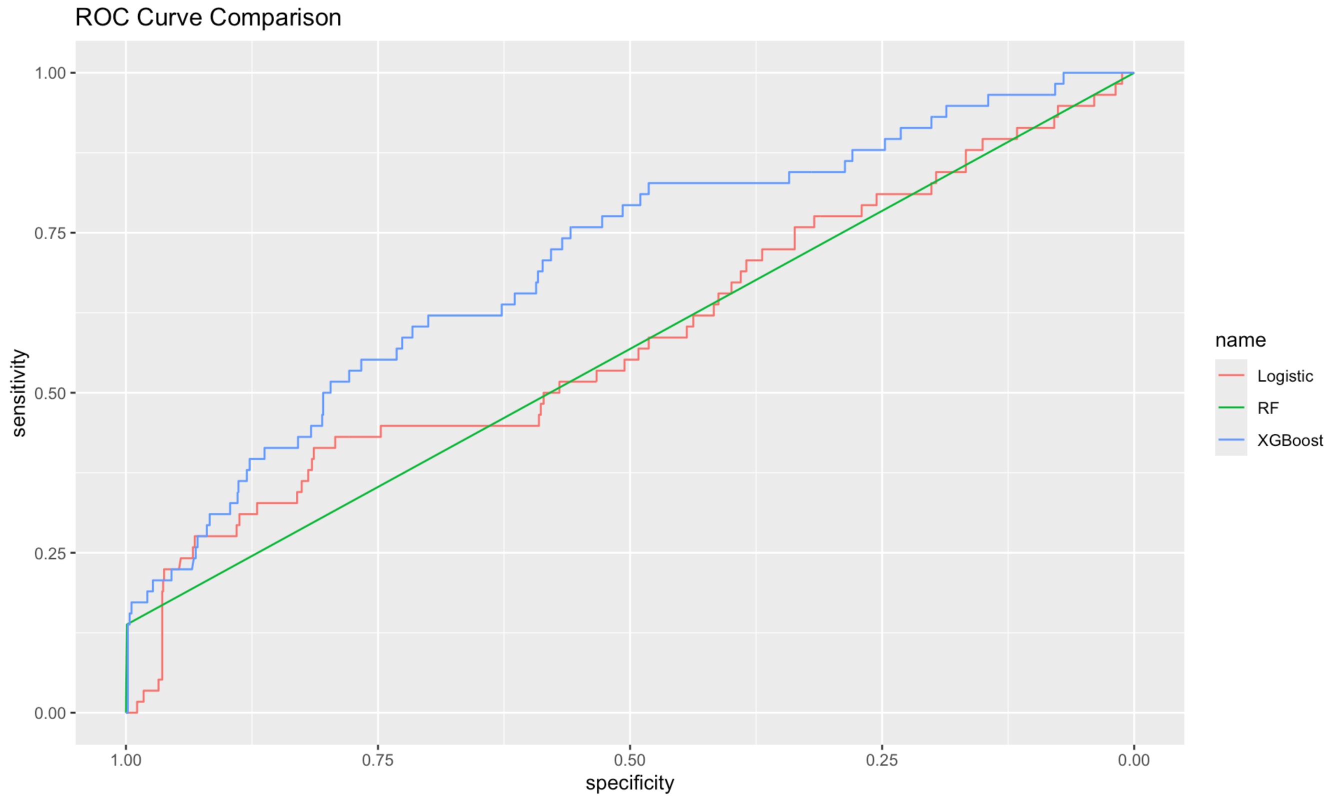



```r
auc(roc_xgb)
```

```
## Area under the curve: 0.7047
```

6. Model Comparison

```r
# Compare ROC curves
ggroc(list(Logistic = roc_logit, RF = roc_rf, XGBoost = roc_xgb)) +
  labs(title = "ROC Curve Comparison")
```




```
# Summarize AUCs
auc_df <- data.frame(
  Model = c("Logistic Regression", "Random Forest", "XGBoost"),
  AUC = c(auc(roc_logit), auc(roc_rf), auc(roc_xgb))
)
knitr::kable(auc_df, digits = 3, caption = "AUC Comparison Across Models")
```

AUC Comparison Across Models

Model	AUC
Logistic Regression	0.587
Random Forest	0.569
XGBoost	0.705

7. Model Metrics Summary

```
# Calculate metrics for each model
# Logistic Regression
logit_cm <- confusionMatrix(as.factor(logit_pred_class), test_data$Churn, positive
        = "1")
logit_sens <- logit_cm$byClass["Sensitivity"]
logit_spec <- logit_cm$byClass["Specificity"]
logit_f1 <- logit_cm$byClass["F1"]
logit_auc <- auc(roc_logit)

# Random Forest
rf_cm <- confusionMatrix(rf_pred, test_data$Churn, positive = "1")
rf_sens <- rf_cm$byClass["Sensitivity"]
rf_spec <- rf_cm$byClass["Specificity"]
rf_f1 <- rf_cm$byClass["F1"]
rf_auc <- auc(roc_rf)

# XGBoost
xgb_cm <- confusionMatrix(as.factor(xgb_pred_class), as.factor(xgb_test$Churn), pos
        itive = "1")
xgb_sens <- xgb_cm$byClass["Sensitivity"]
xgb_spec <- xgb_cm$byClass["Specificity"]
xgb_f1 <- xgb_cm$byClass["F1"]
xgb_auc <- auc(roc_xgb)

# Combine into a summary table
metrics_df <- data.frame(
  Model = c("Logistic Regression", "Random Forest", "XGBoost"),
  Sensitivity = c(logit_sens, rf_sens, xgb_sens),
  Specificity = c(logit_spec, rf_spec, xgb_spec),
  F1 = c(logit_f1, rf_f1, xgb_f1),
  AUC = c(logit_auc, rf_auc, xgb_auc)
)

knitr::kable(metrics_df, digits = 3, caption = "Sensitivity, Specificity, F1, and A
        UC for Each Baseline Model")
```

Sensitivity, Specificity, F1, and AUC for Each Baseline Model

Model	Sensitivity	Specificity	F1	AUC
Logistic Regression	0.000	0.999	NaN	0.587
Random Forest	0.138	0.999	0.239	0.569
XGBoost	0.155	0.995	0.250	0.705

8. Next Steps

- Address class imbalance if needed (e.g., SMOTE, class weights, resampling)

- Tune hyperparameters for best-performing models
- Try additional models or ensembling
- Interpret and communicate results
- Prepare for deployment if satisfied with performance

Conclusion

In this analysis, we compared several models for predicting customer churn using our top engineered features. XGBoost performed the best with an AUC of 0.71, showing some ability to distinguish churners from non-churners. However, all models struggled to correctly identify most churners, mainly due to strong class imbalance in the data.

To improve results, the next steps should focus on addressing class imbalance and further tuning the models to increase recall for churners. Overall, our feature engineering and modeling pipeline provides a solid foundation, but more work is needed to achieve strong predictive performance for churn.