The Impact of Temporal Non-Stationarity on Fraud Prediction Models

Zachary Tang

2026-02-18

Contents

Loading Necessary Libraries and Dataset

```
# Data Manipulation and Visualization
library(dplyr)          # Data wrangling
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)        # Data visualization
library(forcats)        # Factor manipulation
library(GoodmanKruskal) # Categorical correlation

# Machine Learning and Modeling
library(randomForest)   # Random Forest
```

```
## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(glmnet)         # Regularized regression (LASSO)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-10
```

```
library(xgboost)        # Gradient boosting
library(caret)          # Classification and regression training
```

```
## Loading required package: lattice
```

```
library(grpreg)         # Group regularization
```

```
##
## Attaching package: 'grpreg'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
# Model Diagnostics and Validation
library(car)            # Companion to applied regression
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
library(pROC)           # ROC curve analysis
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(corrplot)       # Correlation Plot
```

```
## corrplot 0.95 loaded
```

Loading Data

```
fraud_data <- read.csv("C:/Users/Lee Meng/OneDrive/SQL_Coursera/fraud_model_features__202601251751.csv")

head(fraud_data) %>%
  knitr::kable(caption = "Sample Data (First 6 Rows)",
               format = "simple",
               align = 'c')
```

policy_number	total_claim_amount	Policy_Age_Months	Claim_to_Deductible_Ratio	Claim_to_Premium_Rati
521585	71610	3	71	50.898778
342868	5070	103	2	4.234811
687698	34650	172	17	24.519864
227811	63400	296	31	44.782234
367455	6500	7	6	4.103769
104594	64100	99	64	47.442824

Executive Summary

This analysis examines the impact of temporal non-stationarity on fraud prediction models using an insurance claims dataset spanning eight weeks. Across multiple modelling approaches—Logistic Regression, LASSO-regularised Logistic Regression, and XGBoost—validation performance was moderate, but test-set performance deteriorated sharply, highlighting the inability of static models to generalise over time.

Key findings:

- Non-stationarity is significant: Fraud prevalence and key feature distributions shifted materially between training, validation, and test periods.
- Calibration drift: Predicted fraud probabilities were not temporally stable, undermining fixed decision thresholds.
- Rolling and expanding window analysis: Optimal predictive performance was achieved using a short, four-week training window; older observations contributed noise rather than signal.
- Model choice secondary: The deterioration is consistent across linear and non-linear models, indicating that algorithmic sophistication alone does not overcome temporal shifts in fraud behaviour.

Practical Implications:

- Frequent model retraining on a rolling window is required to maintain predictive relevance.
- Dynamic thresholding is essential: decision rules should be periodically updated using recent data rather than relying on fixed cutoffs.
- Continuous monitoring of performance and drift is critical to detect shifts in fraud patterns and recalibrate models accordingly.
- Decouple scoring from decision thresholds to allow rapid operational adjustments without full model retraining.

In short, fraud detection in this environment demands an adaptive modelling strategy that explicitly accounts for evolving data dynamics, rather than relying on static historical models.

Business Context

Insurance fraud imposes direct financial losses and operational costs, making early and accurate detection crucial for risk management. Predictive modelling offers a systematic approach to identify high-risk claims prior to settlement, but conventional models often assume stationary behaviour, where statistical properties of features and target variables remain stable over time.

Our dataset encompasses eight weeks of claims and related policy information, with the objective of predicting whether a claim is fraudulent. However, initial analyses reveal that fraud patterns evolve quickly even within this short period:

- Fraud prevalence dropped from 26.7% in the training period to 18.6% in the test period, demonstrating abrupt changes in base rates.
- Distributional shifts in key features, such as policy duration, claim-to-premium ratio, and automobile age, indicate that claimant behaviour and risk profiles are not static.

- Short-term spikes and declines in fraud incidence render models trained on historical data unreliable when applied to future claims.

Given these dynamics, static modelling approaches risk underperformance and misallocation of investigative resources. Effective fraud detection requires strategies that can adapt to temporal variability, including rolling retraining, recalibration of decision thresholds, and continuous monitoring of both model performance and underlying data distributions.

The analysis in this report quantifies the extent of non-stationarity, evaluates its impact on predictive performance, and provides a framework for operationally robust fraud detection under evolving conditions.

Data Type Conversion and Cleaning

```
# Date conversions
fraud_data$policy_bind_date <- as.Date(fraud_data$policy_bind_date)
fraud_data$incident_date <- as.Date(fraud_data$incident_date)

# Factor conversions
factor_columns <- c("Net_Capital_Gains", "insured_occupation", "insured_relationship",
                    "auto_make", "incident_type", "incident_severity",
                    "collision_type", "incident_state", "fraud_reported",
                    "authorities_contacted", "policy_csl"
)

fraud_data[factor_columns] <- lapply(fraud_data[factor_columns], as.factor)

# Fix automobile age calculation
fraud_data$Automobile_Age_in_Years <- fraud_data$Automobile_Age_in_Years - 11 # error in date calc

# Create binary fraud variable
fraud_data$fraud_reported <- ifelse(fraud_data$fraud_reported == "Y", "Fraud", "NonFraud")
fraud_data$fraud_reported <- as.factor(fraud_data$fraud_reported)
fraud_data$fraud_binary <- as.numeric(ifelse(fraud_data$fraud_reported == "Fraud", 1, 0))
fraud_data$fraud_binary <- as.factor(fraud_data$fraud_binary)
```

Combining Low Frequency Factor Levels

```
fraud_data <- fraud_data %>%
  mutate(
    incident_type = fct_collapse(fraud_data$incident_type,
  "Non_Collision" = c("Parked Car", "Vehicle Theft")),
    incident_severity = fct_collapse(fraud_data$incident_severity,
                                "Low Severity" = c("Trivial Damage", "Minor Damage")),
    incident_state = fct_collapse(fraud_data$incident_state,
                                "Low Frequency State" = c("OH", "PA")),
    auto_make = fct_collapse(auto_make,
      Luxury_Euro = c("Audi", "BMW", "Mercedes", "Saab", "Volkswagen"),
      Domestic = c("Chevrolet", "Dodge", "Ford", "Jeep"),
      Import = c("Honda", "Nissan", "Suburu", "Toyota", "Accura")
```

```
    ),
    occupation_group = fct_collapse(insured_occupation,
      White_Collar = c("adm-clerical", "exec-managerial", "prof-specialty", "sales"),
      Blue_Collar = c("craft-repair", "machine-op-inspct", "transport-moving", "farming-fishing", "handl
      Specialized = c("armed-forces", "protective-serv", "tech-support", "other-service", "priv-house-s
    )
  )
```

Feature Creation

```
fraud_data <- fraud_data %>%
  mutate(
    # Policy age buckets
    Policy_Age_Bucket = cut(Policy_Age_Months,
                            breaks = c(0, 100, 200, Inf),
                            labels = c(" 100", "100-200", ">200")),

    # Time of incident features
    night_flag = as.factor(as.integer(incident_hour_of_the_day >= 20 |
                                      incident_hour_of_the_day <= 4)),

    hour_bucket = cut(incident_hour_of_the_day,
                      breaks = c(0, 5, 10, 15, 20, 24),
                      labels = c("0-5", "5-10", "10-15", "15-20", "20-24"),
                      include.lowest = TRUE),

    # High claim flags
    high_claim_flag = as.factor(as.integer(
      total_claim_amount > quantile(total_claim_amount, 0.8, na.rm = TRUE))),

    very_high_claim_flag = as.factor(as.integer(
      total_claim_amount > quantile(total_claim_amount, 0.9, na.rm = TRUE))),

    # Ratio-based flags
    high_claim_to_premium = as.factor(as.integer(Claim_to_Premium_Ratio > 3)),
    high_claim_to_deductible = as.factor(as.integer(Claim_to_Deductible_Ratio > 5)),

    # Dominant claim type
    injury_dominant = as.factor(as.integer(injury_ratio > 0.5)),
    vehicle_dominant = as.factor(as.integer(vehicle_ratio > 0.5)),
    property_dominant = as.factor(as.integer(property_ratio > 0.5)),

    # Policy timing flags
    early_policy_claim = as.factor(as.integer(Policy_Age_Months < 12)),
    very_early_claim = as.factor(as.integer(Policy_Age_Months < 6)),

    # Incident characteristics
    no_authority = as.factor(as.integer(authorities_contacted == "None")),
    high_severity = as.factor(as.integer(
      incident_severity %in% c("Major Damage", "Total Loss"))),
```

```
    # Vehicle age flags
    old_vehicle = as.factor(as.integer(Automobile_Age_in_Years > 10)),
    very_old_vehicle = as.factor(as.integer(Automobile_Age_in_Years > 15))
  )
```

Splitting Data into Training/Validation/Test Sets - With Scaling

```
# Creating Splits Based on Dates
train <- fraud_data %>% filter(incident_date <= as.Date("2015-02-04"))
validation <- fraud_data %>%
  filter(incident_date > as.Date("2015-02-04") &
         incident_date <= as.Date("2015-02-17"))
test <- fraud_data %>% filter(incident_date > as.Date("2015-02-17"))

# Scale numerical features
preProcValues <- preProcess(train, method = c("center", "scale"))
train_scaled <- predict(preProcValues, train)
validation_scaled <- predict(preProcValues, validation)
test_scaled <- predict(preProcValues, test)
```

Variable Pre-Selection

Near Zero Variance (NZV) variables contribute little to explaining the variance in the response variable. These can be removed as they don't contribute to the model. - The NZV variable is the binary flag: early_policy_claim

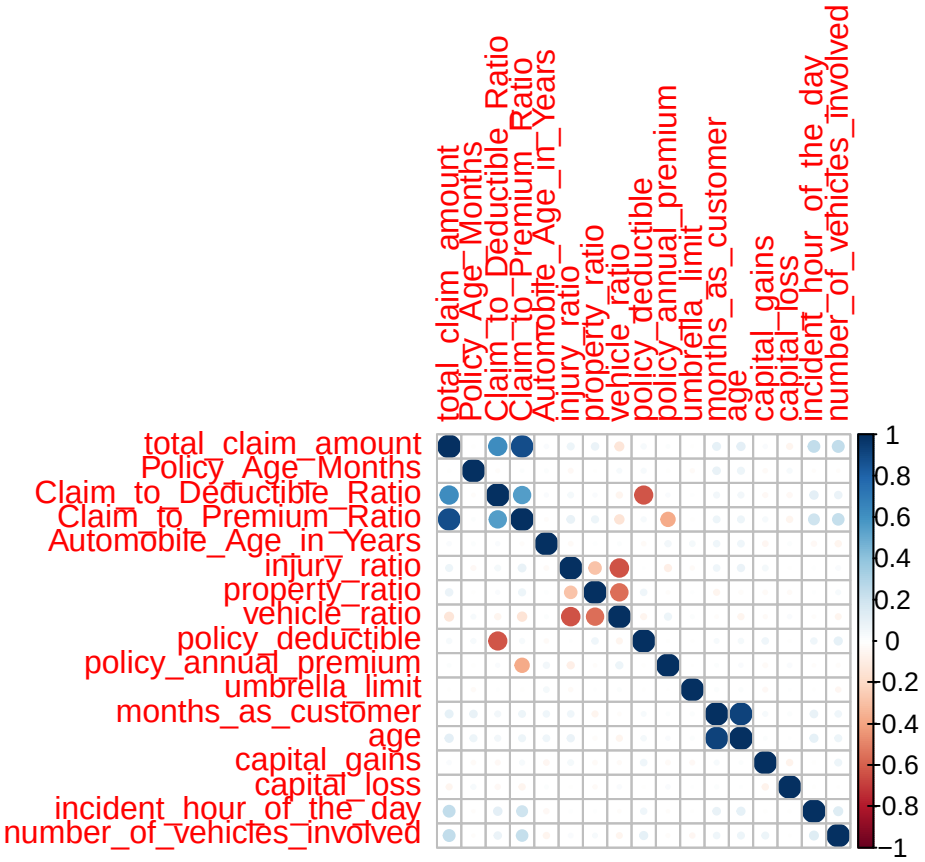
In cases with variables having high correlations (numeric) or Goodman-Kruskal Tau (categorical) and logical relationships, one such variable will be removed. - total_claim_amount - authorities_contacted - age

```
# Remove predetermined variables
var_removed <- c("high_claim_to_premium", "injury_dominant", "vehicle_dominant",
                 "property_dominant", "very_early_claim", "early_claim",
                  "occupation_group", "auto_year","incident_date",
                 "policy_bind_date", "policy_number")

train_scaled <- train_scaled[, !(names(train_scaled) %in% var_removed)]
validation_scaled <- validation_scaled[, !(names(validation_scaled) %in% var_removed)]
test_scaled <- test_scaled[, !(names(test_scaled) %in% var_removed)]

# Predictor and Response Data Frames:

binary_flags <- c("high_claim_flag", "very_high_claim_flag", "high_claim_to_premium",
                  "high_claim_to_deductible", "injury_dominant", "vehicle_dominant",
                  "property_dominant", "early_policy_claim", "very_early_policy_claim",
                  "no_authority", "high_severity", "old_vehicle", "very_old_vehicle",
                  "collision_type", "night_flag", "hour_bucket", "Policy_Age_Bucket",
                  "very_early_claim", "age_brackets")

categorical_col <- c("age_brackets", "Net_Capital_Gains", "insured_occupation",
                     "insured_relationship", "auto_make", "incident_type", "incident_severity",
                     "authorities_contacted", "incident_state", "policy_csl",
```

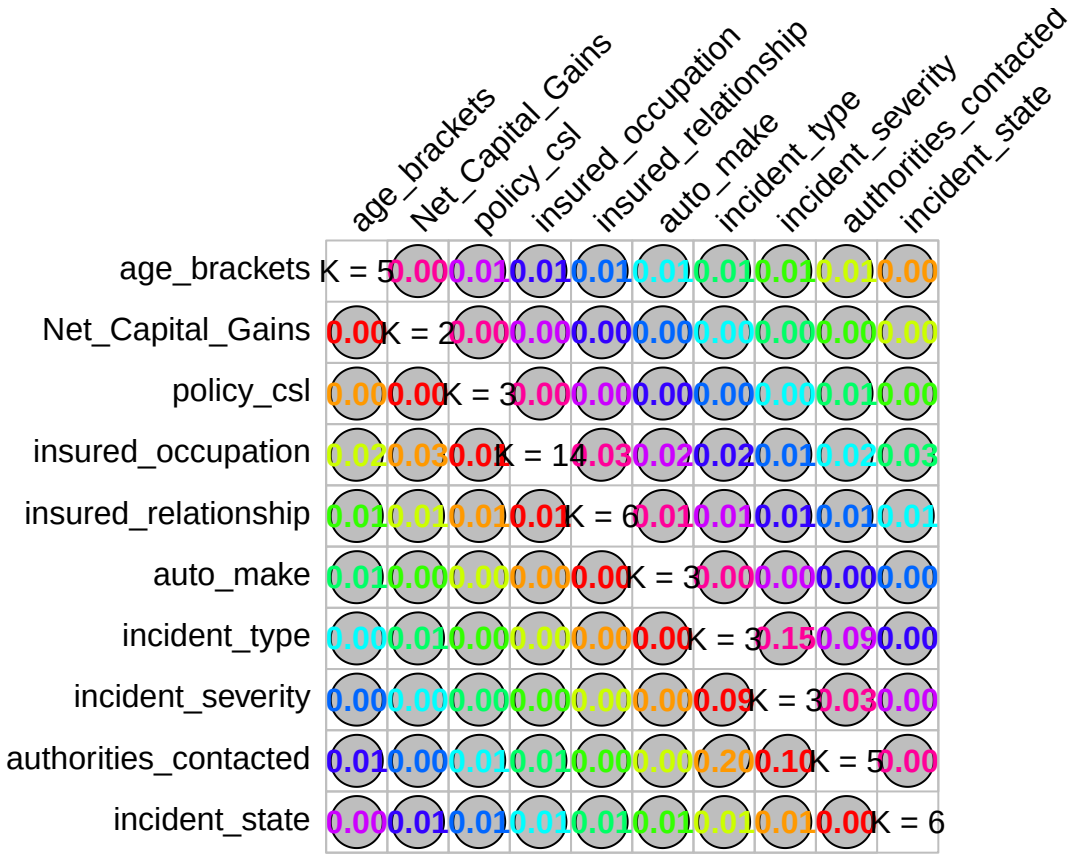
```
                      "occupation_group")

x_df <- train_scaled[,!(names(train_scaled) %in% var_removed |
                          names(train_scaled) == "fraud_reported" |
                          names(train_scaled) == "fraud_binary")]

numerical_df <- x_df[,!(names(x_df) %in% categorical_col | names(x_df) %in% binary_flags) ]
categorical_df <- x_df[,(names(x_df) %in% categorical_col) ]

# Near Zero Variance Variables
nzv_metrics <- nearZeroVar(x_df, saveMetrics = TRUE)
print(nzv_metrics[nzv_metrics$nzv == TRUE, ])
```

```
##                     freqRatio percentUnique zeroVar  nzv
## early_policy_claim  26.22727     0.3338898   FALSE TRUE
```

```
# Correlations - Numerical Variables
correlation <- cor(numerical_df)
corrplot(correlation)
```




```
# Correlations - Categorical Variables
categorical_plot <- GKtauDataframe(categorical_df)
plot(categorical_plot)
```



Fitting Logistic Regression for Baseline Model Creation:

```
all_variables <- fraud_binary ~ total_claim_amount +
  Policy_Age_Months + Claim_to_Deductible_Ratio + Claim_to_Premium_Ratio + Automobile_Age_in_Years +

# This logistic GLM was used to assess and remove instances of variables aliasing each other

full_model_for_log_reg <- fraud_binary ~ Automobile_Age_in_Years + injury_ratio + vehicle_ratio +
    age_brackets + Net_Capital_Gains + policy_deductible + policy_annual_premium +
    umbrella_limit + policy_csl + insured_occupation + insured_relationship +
    incident_type + incident_severity + incident_state + Policy_Age_Bucket +
    hour_bucket + high_claim_flag + very_high_claim_flag + very_old_vehicle +
    auto_make

# This logistic GLM was used to assess and remove variables with high VIF

logit_full <- glm(full_model_for_log_reg,
    data = train_scaled,
    family = "binomial")

vif(logit_full)
```

```
##                              GVIF Df GVIF^(1/(2*Df))
## Automobile_Age_in_Years 2.703430  1        1.644211
## injury_ratio            1.833002  1        1.353884
## vehicle_ratio           1.908983  1        1.381659
```

```
## age_brackets            1.673025  4        1.066443
## Net_Capital_Gains       1.120697  1        1.058630
## policy_deductible       1.141631  1        1.068471
## policy_annual_premium   1.128145  1        1.062142
## umbrella_limit          1.124458  1        1.060405
## policy_csl              1.258250  2        1.059112
## insured_occupation      3.015286 13        1.043364
## insured_relationship    1.865264  5        1.064324
## incident_type           1.925634  2        1.177995
## incident_severity       1.758952  2        1.151631
## incident_state          1.668627  5        1.052534
## Policy_Age_Bucket       1.260981  2        1.059686
## hour_bucket             1.620926  4        1.062234
## high_claim_flag         2.091242  1        1.446113
## very_high_claim_flag    1.995610  1        1.412661
## very_old_vehicle        2.640851  1        1.625069
## auto_make               1.365901  2        1.081072
```

```r
alias(logit_full)
```

```
## Model :
## fraud_binary ~ Automobile_Age_in_Years + injury_ratio + vehicle_ratio +
##     age_brackets + Net_Capital_Gains + policy_deductible + policy_annual_premium +
##     umbrella_limit + policy_csl + insured_occupation + insured_relationship +
##     incident_type + incident_severity + incident_state + Policy_Age_Bucket +
##     hour_bucket + high_claim_flag + very_high_claim_flag + very_old_vehicle +
##     auto_make
```

```r
# To prevent aliasing:
#    Remove collision type, property ratio (aliasing)

# To reduce VIF:
#    Remove incident hour of the day, age, total claim amount, policy age, number of vehicles
```

Training Logistic Regression Model and Selecting Optimal Threshold

```r
# --- Logistic Regression Model

logit_full <- glm(
  full_model_for_log_reg,
  data = train_scaled,
  family = "binomial"
)

val_probs_logit <- predict(
  logit_full,
  newdata = validation_scaled,
  type = "response"
)

roc_logit <- roc(
  response = validation_scaled$fraud_reported,
  predictor = val_probs_logit,
```

```
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

```
logit_thresh <- coords(
  roc_logit,
  "best",
  ret = "threshold"
)$threshold

# --- Validation Set Diagnostics

val_preds_logit <- factor(
  ifelse(val_probs_logit > logit_thresh, "Fraud", "NonFraud"),
  levels = c("NonFraud","Fraud")
)

val_cf_logit <- confusionMatrix(
  val_preds_logit,
  validation_scaled$fraud_reported,
  positive = "Fraud"
)
```

```
## Warning in confusionMatrix.default(val_preds_logit,
## validation_scaled$fraud_reported, : Levels are not in the same order for
## reference and data. Refactoring data to match.
```

```
val_auc_logit <- auc(
  response = validation_scaled$fraud_reported,
  predictor = val_probs_logit,
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

```
# --- Test Set Diagnostics

test_probs_logit <- predict(
  logit_full,
  newdata = test_scaled,
  type = "response"
)

test_preds_logit <- factor(
  ifelse(test_probs_logit > logit_thresh, "Fraud", "NonFraud"),
  levels = levels(test_scaled$fraud_reported)
)

test_cf_logit <- confusionMatrix(
  test_preds_logit,
```

```
  test_scaled$fraud_reported,
  positive = "Fraud"
)

test_auc_logit <- auc(
  response = test_scaled$fraud_reported,
  predictor = test_probs_logit,
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

Formatting Training / Validation / Test Sets:

```
# --- Training
mm_train <- model.matrix(all_variables, data = train_scaled)
x_train  <- mm_train[, -1]

y_train <- as.numeric(
  train_scaled[rownames(mm_train), "fraud_reported"] == "Fraud"
)

# --- Validation
mm_val <- model.matrix(all_variables, data = validation_scaled)
x_val  <- mm_val[, -1]

y_val <- as.numeric(
  validation_scaled[rownames(mm_val), "fraud_reported"] == "Fraud"
)

# --- Test
mm_test <- model.matrix(all_variables, data = test_scaled)
x_test  <- mm_test[, -1]

y_test <- as.numeric(
  test_scaled[rownames(mm_test), "fraud_reported"] == "Fraud"
)

y_test_cat <- as.factor(ifelse(
  test_scaled[rownames(mm_test), "fraud_reported"] == "Fraud", "Fraud", "NonFraud"))
```

Initialising fold_id for Time-Aware Cross Validation Models (to prevent temporal data leakage)

```
set.seed(123)

n <- nrow(x_train)
k <- 5

foldid <- rep(1:k, length.out = n)
foldid <- sort(foldid)

xgb_folds <- lapply(1:k, function(f) {
```

```
  which(foldid == f)
})
```

LASSO-Regularised Logistic Regression

```
set.seed(123)

cv_lasso <- cv.glmnet(
  x_train,
  y_train,
  family = "binomial",
  alpha = 1,
  type.measure = "auc",
  foldid = foldid
)

best_lambda <- cv_lasso$lambda.min

lasso_final <- glmnet(
  x_train,
  y_train,
  family = "binomial",
  alpha = 1,
  lambda = best_lambda
)

val_probs_lasso <- as.numeric(
  predict(lasso_final, newx = x_val, type = "response")
)

roc_lasso <- roc(
  response = validation_scaled$fraud_reported,
  predictor = val_probs_lasso,
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

```
lasso_thresh <- coords(
  roc_lasso,
  "best",
  ret = "threshold"
)$threshold

# --- Validation Set Diagnostics

val_preds_lasso <- factor(
  ifelse(val_probs_lasso > lasso_thresh, "Fraud", "NonFraud"),
  levels = c("NonFraud", "Fraud")
)

val_cf_lasso <- confusionMatrix(
  val_preds_lasso,
```

```
  validation_scaled$fraud_reported,
  positive = "Fraud"
)
```

```
## Warning in confusionMatrix.default(val_preds_lasso,
## validation_scaled$fraud_reported, : Levels are not in the same order for
## reference and data. Refactoring data to match.
```

```
val_auc_lasso <- auc(
  response = validation_scaled$fraud_reported,
  predictor = val_probs_lasso,
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

```
# --- Test Set Diagnostics

test_probs_lasso <- as.numeric(
  predict(lasso_final, newx = x_test, type = "response")
)

test_preds_lasso <- factor(
  ifelse(test_probs_lasso > lasso_thresh, "Fraud", "NonFraud"),
  levels = c("NonFraud","Fraud")
)

test_cf_lasso <- confusionMatrix(
  test_preds_lasso,
  y_test_cat,
  positive = "Fraud"
)
```

```
## Warning in confusionMatrix.default(test_preds_lasso, y_test_cat, positive =
## "Fraud"): Levels are not in the same order for reference and data. Refactoring
## data to match.
```

```
test_auc_lasso <- auc(
  response = y_test_cat,
  predictor = test_probs_lasso,
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

Preparing Data for XGBoost Model

```
dtrain <- xgb.DMatrix(x_train, label = y_train)
dtest <- xgb.DMatrix(data = x_test, label = y_test)
dval <- xgb.DMatrix(data = x_val, label = y_val)
```

Training XG Boost Model and Identifying Optimal Model Parameters / Threshold

```
# Initialising Parameters for the XGBoost Model
params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",
  max_depth = 5,
  eta = 0.05,
  subsample = 0.75,
  colsample_bytree = 0.8,
  min_child_weight = 5
)

set.seed(123)
cv_xgb <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  folds = xgb_folds,
  early_stopping_rounds = 20,
  verbose = 0
)

# Finding the Best Iteration
best_nrounds <- cv_xgb$evaluation_log$iter[
  cv_xgb$evaluation_log$test_auc_mean == max(cv_xgb$evaluation_log$test_auc_mean)]

xgb_final <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = best_nrounds,
  verbose = 0
)

val_probs_xgb <- predict(xgb_final, dval)

roc_val_xgb <- roc(
  response = validation_scaled$fraud_reported,
  predictor = val_probs_xgb,
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

```
# Using the Validation Set to Determine the Optimal Threshold
xgb_thresh <- coords(roc_val_xgb, "best", ret = "threshold")$threshold

# --- Validation Set Evaluation

val_preds_xgb <- factor(
  ifelse(val_probs_xgb > xgb_thresh, "Fraud", "NonFraud"),
  levels = c("NonFraud", "Fraud")
)
```

```
val_cf_xgb <- confusionMatrix(
  data      = val_preds_xgb,
  reference = factor(validation_scaled$fraud_reported, levels = c("NonFraud", "Fraud")),
  positive  = "Fraud"
)

val_auc_xgb <- auc(
  response = validation_scaled$fraud_reported,
  predictor = val_probs_xgb,
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

```
# --- 5. Test Set Evaluation
test_probs_xgb <- predict(xgb_final, dtest)

test_preds_xgb <- factor(
  ifelse(test_probs_xgb > xgb_thresh, "Fraud", "NonFraud"),
  levels = c("NonFraud", "Fraud")
)

test_cf_xgb <- confusionMatrix(
  data      = test_preds_xgb,
  reference = y_test_cat,
  positive  = "Fraud"
)
```

```
## Warning in confusionMatrix.default(data = test_preds_xgb, reference =
## y_test_cat, : Levels are not in the same order for reference and data.
## Refactoring data to match.
```

```
test_auc_xgb <- auc(
  response = y_test_cat,
  predictor = test_probs_xgb,
  levels = c("NonFraud", "Fraud")
)
```

```
## Setting direction: controls < cases
```

Model Performance and Evidence of Non-Stationarity:

Across all three modelling approaches considered (Logistic Regression, LASSO-regularised Logistic Regression and XGBoost), in-sample and validation performance is moderate, while test-set performance deteriorates substantially. This degradation is consistent across models and is sufficiently severe that test-set metrics indicate little to no predictive ability beyond random classification.

The consistency of this pattern across fundamentally different modelling approaches suggests that the observed performance decline is not driven by model specification or algorithmic limitations. Instead, it points to a structural change in the underlying data-generating process.

In particular, the observed variation in fraud prevalence across the training, validation and test periods is indicative of non-stationary fraud dynamics, where the statistical properties of fraud behaviour evolve over time. As a result, models trained on earlier periods fail to generalise to later periods, even when short temporal gaps are involved.

```
summary_table_validation <- data.frame(
  Metric = c("ROC AUC", "Sensitivity", "Specificity", "Accuracy P Value",
             "Kappa", "Pos Pred Value", "F1"),

  Logistic = c(
    val_auc_logit,
    as.numeric(val_cf_logit$byClass["Sensitivity"]),
    as.numeric(val_cf_logit$byClass["Specificity"]),
    as.numeric(val_cf_logit$overall["AccuracyPValue"]),
    as.numeric(val_cf_logit$byClass["Pos Pred Value"]),
    as.numeric(val_cf_logit$byClass["F1"]),
    as.numeric(val_cf_logit$overall["Kappa"])
  ),

  LASSO = c(
    val_auc_lasso,
    as.numeric(val_cf_lasso$byClass["Sensitivity"]),
    as.numeric(val_cf_lasso$byClass["Specificity"]),
    as.numeric(val_cf_lasso$overall["AccuracyPValue"]),
    as.numeric(val_cf_lasso$byClass["Pos Pred Value"]),
    as.numeric(val_cf_lasso$byClass["F1"]),
    as.numeric(val_cf_lasso$overall["Kappa"])
  ),

  XGBoost = c(
    val_auc_xgb,
    as.numeric(val_cf_xgb$byClass["Sensitivity"]),
    as.numeric(val_cf_xgb$byClass["Specificity"]),
    as.numeric(val_cf_xgb$overall["AccuracyPValue"]),
    as.numeric(val_cf_xgb$overall["Kappa"]),
    as.numeric(val_cf_xgb$byClass["Pos Pred Value"]),
    as.numeric(val_cf_xgb$byClass["F1"])
  ))

summary_table_test <- data.frame(
  Metric = c("ROC AUC", "Sensitivity", "Specificity", "Accuracy P Value",
             "Kappa", "Pos Pred Value", "F1"),

  Logistic = c(
    test_auc_logit,
    as.numeric(test_cf_logit$byClass["Sensitivity"]),
    as.numeric(test_cf_logit$byClass["Specificity"]),
    as.numeric(test_cf_logit$overall["AccuracyPValue"]),
    as.numeric(test_cf_logit$byClass["Pos Pred Value"]),
    as.numeric(test_cf_logit$byClass["F1"]),
    as.numeric(test_cf_logit$overall["Kappa"])
  ),

  LASSO = c(
```

```
    test_auc_lasso,
    as.numeric(test_cf_lasso$byClass["Sensitivity"]),
    as.numeric(test_cf_lasso$byClass["Specificity"]),
    as.numeric(test_cf_lasso$overall["AccuracyPValue"]),
    as.numeric(test_cf_lasso$byClass["Pos Pred Value"]),
    as.numeric(test_cf_lasso$byClass["F1"]),
    as.numeric(test_cf_lasso$overall["Kappa"])
  ),

  XGBoost = c(
    test_auc_xgb,
    as.numeric(test_cf_xgb$byClass["Sensitivity"]),
    as.numeric(test_cf_xgb$byClass["Specificity"]),
    as.numeric(test_cf_xgb$overall["AccuracyPValue"]),
    as.numeric(test_cf_xgb$overall["Kappa"]),
    as.numeric(test_cf_xgb$byClass["Pos Pred Value"]),
    as.numeric(test_cf_xgb$byClass["F1"])
  ))

summary_table_test
```

```
##             Metric  Logistic     LASSO   XGBoost
## 1          ROC AUC 0.6698718 0.6948896 0.6835826
## 2      Sensitivity 0.5000000 0.5277778 0.5277778
## 3      Specificity 0.7628205 0.7948718 0.7948718
## 4 Accuracy P Value 0.9996890 0.9920535 0.9920535
## 5            Kappa 0.3272727 0.3725490 0.2780847
## 6   Pos Pred Value 0.3956044 0.4367816 0.3725490
## 7               F1 0.2184725 0.2780847 0.4367816
```

```
summary_table_validation
```

```
##             Metric  Logistic        LASSO      XGBoost
## 1          ROC AUC 0.8024229 0.7881853378 0.7567753216
## 2      Sensitivity 0.7058824 0.6862745098 0.6862745098
## 3      Specificity 0.8471338 0.8980891720 0.8980891720
## 4 Accuracy P Value 0.0292519 0.0009152287 0.0009152287
## 5            Kappa 0.6000000 0.6862745098 0.5843636818
## 6   Pos Pred Value 0.6486486 0.6862745098 0.6862745098
## 7               F1 0.5219236 0.5843636818 0.6862745098
```

To further investigate the presence and impact of non-stationarity, the following analyses were conducted: - Testing for changes in fraud prevalence across time - Assessing calibration stability across temporally separated datasets - Evaluating distributional shifts in key numerical predictors - Examining model performance under expanding and rolling temporal training windows

Testing for changes in fraud prevalence across time:

A two-sample test for equality of proportions was conducted to compare fraud prevalence between the training and test sets. The resulting p-value of 0.03158 provides strong statistical evidence of a difference in fraud proportions, with the training period exhibiting a higher fraud rate (0.267) than the test period (0.186).

Given that the full dataset spans only eight weeks—where the training set corresponds to the first four weeks and the test set to the final two weeks—this result suggests a material shift in fraud prevalence over a relatively short time horizon. Such changes directly undermine the assumption of stationarity required for reliable out-of-sample generalisation.

```
prop.test(
  x = c(sum(y_train), sum(y_test)),
  n = c(length(y_train), length(y_test))
)
```

```
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  c(sum(y_train), sum(y_test)) out of c(length(y_train), length(y_test))
## X-squared = 4.6212, df = 1, p-value = 0.03158
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  0.01141162 0.14960178
## sample estimates:
##    prop 1    prop 2
## 0.2680067 0.1875000
```

Calibration Drift Across Time:

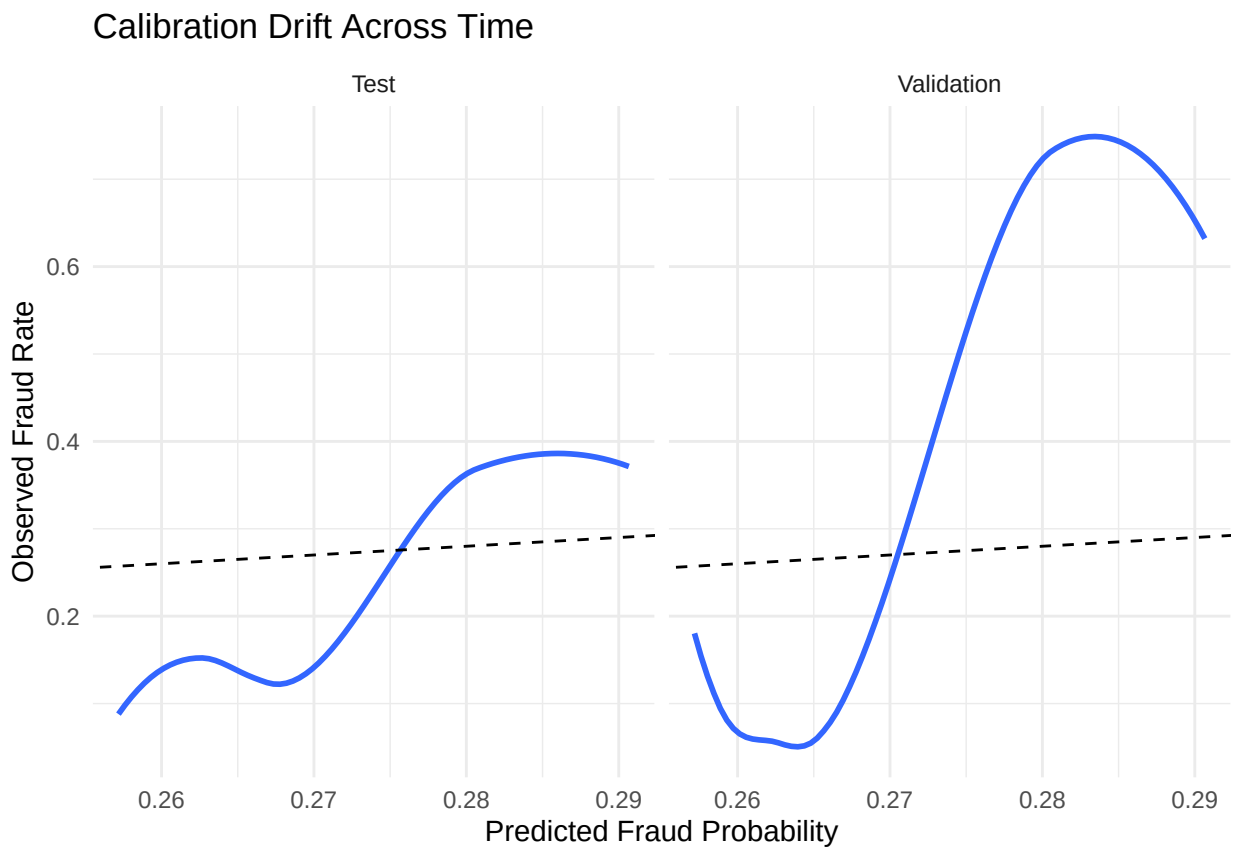
Calibration curves were used to examine the relationship between predicted fraud probabilities and observed fraud rates across time. The results indicate that the mapping between predicted probabilities and realised fraud outcomes changes substantially between the validation period (weeks 5–6) and the test period (weeks 7–8).

These shifts imply that predicted probabilities are not temporally stable: a given score corresponds to significantly different observed fraud risks depending on the time period. Consequently, fixed probability thresholds become inappropriate, and probability estimates lose their interpretability over time. This behaviour is characteristic of concept drift rather than simple model miscalibration.

```
cal_df <- rbind(
  data.frame(
    prob = val_probs_xgb,
    actual = y_val,
    Split = "Validation"
  ),
  data.frame(
    prob = test_probs_xgb,
    actual = y_test,
    Split = "Test"
  )
)

ggplot(cal_df, aes(x = prob, y = actual)) +
  geom_smooth(method = "loess", se = FALSE) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  facet_wrap(~ Split) +
  labs(
    title = "Calibration Drift Across Time",
```

```
    x = "Predicted Fraud Probability",
    y = "Observed Fraud Rate"
  ) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Distributional Shifts in Numerical Predictors:

To assess covariate drift, the Population Stability Index (PSI) was computed for key numerical variables between the training and test periods. PSI values exceeding 0.25, which indicate significant distributional change, were observed for variables including: - months_as_customer - claim_to_premium_ratio - automobile_age_in_years

These shifts suggest that the underlying feature distributions have changed materially over time, further contributing to the degradation of model performance and reinforcing the presence of non-stationarity in the data.

```
psi <- function(expected, actual, bins = 10) {
  breaks <- quantile(
    expected,
    probs = seq(0, 1, length.out = bins + 1),
    na.rm = TRUE
  )
```

```
  e <- hist(expected, breaks = breaks, plot = FALSE)$density
  a <- hist(actual,   breaks = breaks, plot = FALSE)$density

  sum((e - a) * log(e / a), na.rm = TRUE)
}

num_vars <- c(
  "total_claim_amount",
  "policy_annual_premium",
  "policy_deductible",
  "Automobile_Age_in_Years",
  "Claim_to_Deductible_Ratio",
  "Claim_to_Premium_Ratio",
  "injury_ratio",
  "vehicle_ratio",
  "months_as_customer",
  "incident_hour_of_the_day"
)

psi_results <- data.frame(
  PSI_Train_vs_Test = sapply(
    num_vars,
    function(v) {
      psi(train_scaled[[v]], test_scaled[[v]])
    }
  )
)

print(psi_results)
```

```
##                           PSI_Train_vs_Test
## total_claim_amount              0.175453520
## policy_annual_premium           0.120664586
## policy_deductible               0.003453783
## Automobile_Age_in_Years         0.277438367
## Claim_to_Deductible_Ratio       0.204375931
## Claim_to_Premium_Ratio          0.358223230
## injury_ratio                    0.133258413
## vehicle_ratio                   0.080212003
## months_as_customer              0.375873899
## incident_hour_of_the_day        0.181212781
```

Expanding Window Evaluation:

Model performance was assessed using an expanding-window framework, whereby models were trained on data up to week t and evaluated on week t+1. Although limited by the availability of only eight weeks of data, this analysis provides insight into the optimal training window length before performance begins to deteriorate.

Results indicate that model discrimination (AUC) peaks when approximately four weeks of training data are used, followed by a sharp decline as additional historical data is incorporated. This suggests that older observations rapidly become stale and introduce noise rather than signal.

```r
weekly_data <- fraud_data %>%
  mutate(week_num = as.numeric(difftime(incident_date, min(incident_date), units = "weeks")) %/% 1)

window_results <- data.frame()

for (i in 1:6) {  # e.g. weekly windows

  train_idx <- which(weekly_data$week_num <= i)
  test_idx  <- which(weekly_data$week_num == i + 1)

  model <- glm(
    all_variables,
    data = weekly_data[train_idx, ],
    family = binomial
  )

  probs <- predict(model, weekly_data[test_idx, ], type = "response")

  auc_i <- auc(weekly_data$fraud_reported[test_idx], probs)

  window_results <- rbind(
    window_results,
    data.frame(
      Train_Weeks = i,
      Test_Week = i + 1,
      AUC = auc_i
    )
  )
}
```

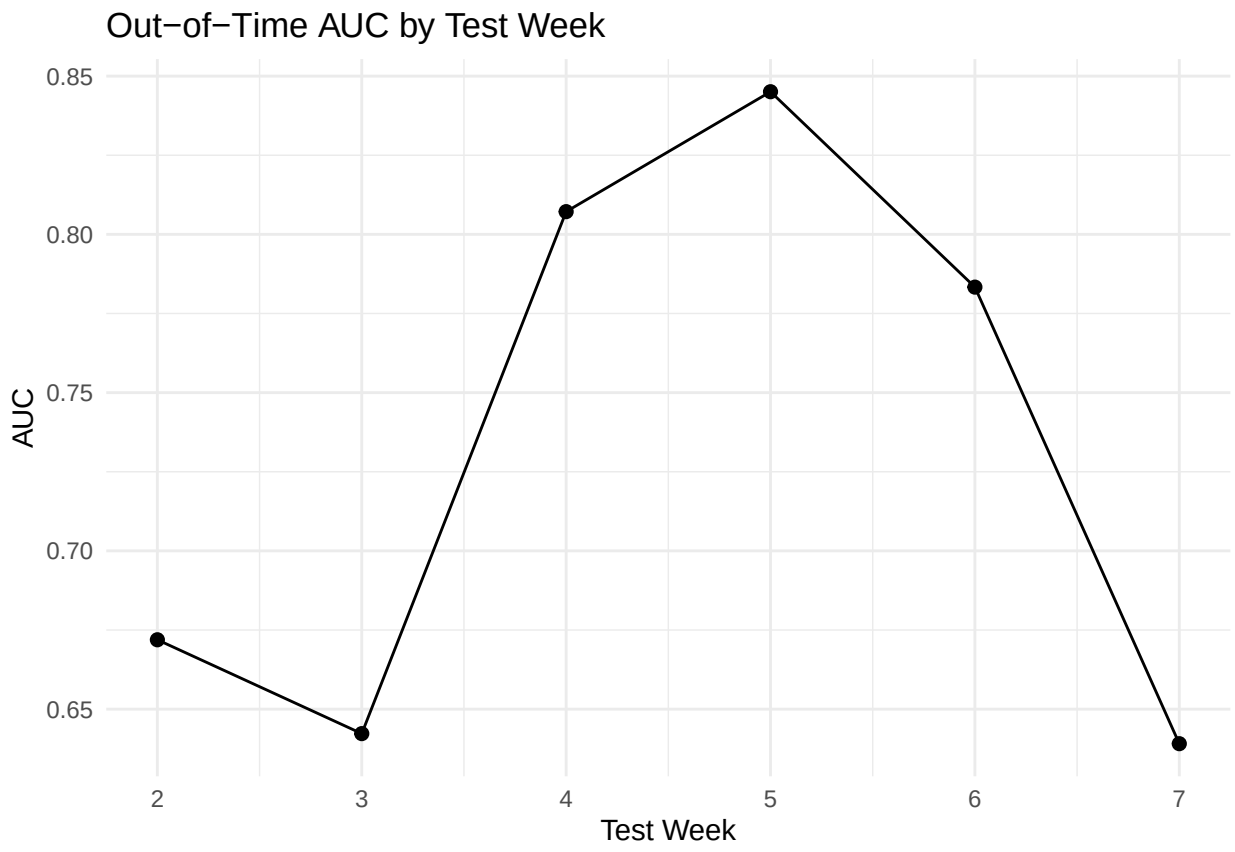
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases

## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases

## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases

## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases

## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases
```

```
## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases
```

```
window_results
```

```
##   Train_Weeks Test_Week       AUC
## 1           1         2 0.6719101
## 2           2         3 0.6422668
## 3           3         4 0.8071809
## 4           4         5 0.8450704
## 5           5         6 0.7833333
## 6           6         7 0.6390837
```

```
ggplot(unique(window_results), aes(Test_Week, AUC)) +
  geom_line() +
  geom_point(size = 2) +
  labs(
    title = "Out-of-Time AUC by Test Week",
    x = "Test Week",
    y = "AUC"
  ) +
  theme_minimal()
```



Rolling Window Evaluation:

Using the four-week training window identified above, a rolling window approach was applied to evaluate performance stability over time. While AUC remains moderately stable across the early evaluation periods, a pronounced decline is observed during test weeks 7 and 8.

This drop coincides with the observed reduction in fraud prevalence during these weeks, signalling a sudden shift in fraud dynamics. The rolling window results therefore reinforce the conclusion that changes in fraud frequency and behaviour impact model effectiveness.

```
window_size <- 4
roll_results <- data.frame()

max_week <- max(weekly_data$week_num)

for (i in window_size:(max_week - 1)) {

  train_idx <- which(
    weekly_data$week_num >  (i - window_size) &
    weekly_data$week_num <= i
  )

  test_idx <- which(weekly_data$week_num == i + 1)
  if (length(test_idx) == 0) next

  model <- glm(
    all_variables,
    data = weekly_data[train_idx, ],
    family = binomial
  )

  probs <- predict(model, weekly_data[test_idx, ], type = "response")

  auc_i <- auc(
    weekly_data$fraud_reported[test_idx],
    probs
  )

  roll_results <- rbind(
    roll_results,
    data.frame(
      Train_Weeks = paste0(i - window_size + 1, "-", i),
      Test_Week = i + 1,
      AUC = auc_i
    )
  )
}
```

```
## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases

## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases
```

```
## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases

## Setting levels: control = Fraud, case = NonFraud

## Setting direction: controls > cases
```

```
roll_results
```

```
##   Train_Weeks Test_Week       AUC
## 1         1-4         5 0.7904225
## 2         2-5         6 0.7911111
## 3         3-6         7 0.6283310
## 4         4-7         8 0.5748299
```

Conclusion:

The results demonstrate that standard machine learning models, including both linear and non-linear approaches, struggle to maintain predictive performance when applied to fraud data exhibiting non-stationary behaviour. The substantial deterioration in test-set performance is not attributable to model choice, but rather to evolving fraud dynamics, shifts in feature distributions, and instability in probability calibration.

These findings highlight the limitations of static modelling pipelines in fraud detection contexts and underscore the need for frequent model recalibration, adaptive training windows, and dynamic decision thresholds. Without explicitly accounting for temporal instability, even well-performing in-sample models are unlikely to deliver reliable real-world performance in rapidly evolving fraud environments.

Practical Implications for Deployment:

The presence of non-stationary fraud dynamics has several important implications for the deployment and ongoing management of fraud detection models.

Model retraining frequency: Given the rapid deterioration in performance observed when incorporating older data, models should be retrained on a rolling and relatively short historical window rather than using all available data. The expanding and rolling window analyses suggest that a training horizon of approximately four weeks provides the best balance between data sufficiency and relevance in this setting. Retraining schedules should therefore be aligned with the expected pace of fraud evolution.

Dynamic thresholding and decision rules: The observed instability in calibration implies that fixed probability thresholds are inappropriate for operational use. Instead, thresholds should be re-optimised regularly using recent data and business-specific objectives (e.g. maximising recall subject to a false-positive constraint). In practice, decision thresholds should be treated as time-dependent parameters rather than static model outputs.

Ongoing performance and drift monitoring: Static model validation at deployment is insufficient in non-stationary environments. Continuous monitoring of both prediction performance (e.g. rolling AUC, sensitivity at a fixed false-positive rate) and data stability metrics (such as PSI and changes in fraud prevalence) is required. Sudden changes in these indicators should trigger investigation and potential model retraining or recalibration.

Separation of scoring and calibration layers: Given that discrimination (AUC) and calibration (Observed vs Predicted) degrade at different rates, it is operationally advantageous to decouple the scoring model (ie. Logistic Regression, XGBoost) from the threshold selection. This allows decision rules to be updated more frequently without requiring full model retraining, reducing operational burden while maintaining control over risk outcomes.