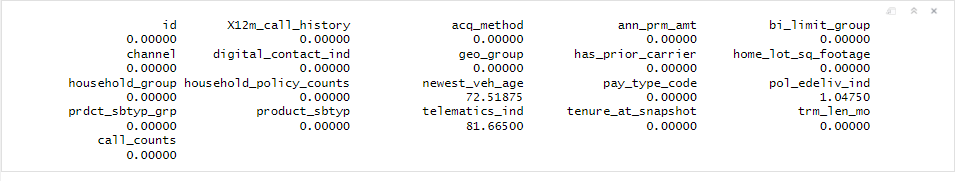
**Augustine Kena Adjei and Oluwafunmibi Omotayo Fasanya**

**Missing Variable Imputation**

To ensure data quality and prepare the dataset for prediction, missing values as described in the variable description on Kaggle were replaced with NA. In the newest\_veh\_age column, values of -20, were converted to NA. Similarly, in the telematics\_ind column, values of -2 and 0, which likely indicated missing or non-auto, were replaced with NA. Lastly, in the pol\_edeliv\_ind column, the placeholder value -2 was also replaced with NA. So this was done in both the test and training data.



**Percentage of Missing data in the Training Set**

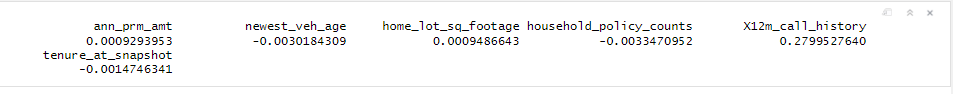
After this replacement was done, we Perform Random Forest imputation using missForest for the training dataset and since the response variable is not known for the test dataset, we Perform KNN imputation on that

**Percentage of zeros for response variable (call\_counts)**

For the response variable (Call Counts), 50.18% of the values are zero.

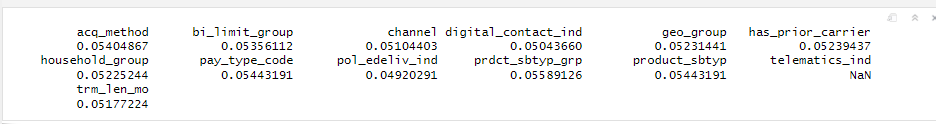
**Association between Call counts and the continuous variable in the dataset**

Most of the continuous variables shows a small associations with call counts, except for 12-month call history, which shows a moderate positive relationship. This suggests that prior call activity is likely the most relevant predictor among the variables analyzed.



**Association between Call counts and the categorical variable in the dataset**

All categorical variables show weak associations with call counts, with the highest being Product Subtype Group (prdct\_sbtyp\_grp) (Cramér's V = 0.05589). This suggest that further investigation or feature engineering may be necessary to enhance the predictive power for this dataset.



**Splitting the training dataset**

We split the dataset into three parts: a training set (approximately 60% of the data), a validation set (about 20%), and a test set (about 20%).

**Training a base learner using LightBoost (LightGBM)**

We trained a base model using LightGBM to predict call counts. The dataset was split into training (60%), validation (20%), and test (20%) sets. Our predictor data included both numerical and categorical variables, requiring preprocessing to prepare it for the LightGBM algorithm, which expects a matrix format. To handle categorical variables, we applied encoding by converting them into numeric factors. This encoding was applied to the training, validation, and test sets to maintain consistency. These encoded datasets were then converted to matrix format for input into LightGBM. We defined a set of parameters for the LightGBM model, including settings for the objective function ("poisson" to address the count nature of the target variable), evaluation metric ("rmse"), and a variety of hyperparameters to balance model complexity and avoid overfitting. Key parameters included a learning rate of 0.05, max depth of 8, and a minimum data requirement per leaf. The model was trained using the LightGBM lgb.train function, incorporating early stopping based on performance on the validation set, which halted training if performance did not improve over 50 rounds. After training, predictions were made on the training, validation, and test sets. To evaluate model performance, we calculated metrics including Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R²) for each dataset. These metrics helped us assess model accuracy and generalization across the datasets, with RMSE and MAE reflecting prediction error and R² indicating the model's explanatory power. The results suggest how well the model is capturing the underlying patterns in the data and its effectiveness for potential future predictions.

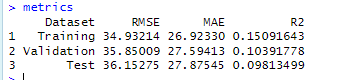
**Model Performance**

We looked at the LightGBM model's performance for the Training, Validation, and Test.

**Training Set:** The model achieved an RMSE of 34.93 and an MAE of 26.92, indicating the average and absolute prediction errors, respectively. The R² value of 0.15 suggests that the model explains only 15% of the variance in the training data, pointing to limited effectiveness in capturing the underlying patterns.

**Validation Set:** On the validation set, the model had an RMSE of 35.85 and an MAE of 27.59, slightly higher than on the training set. The R² of 0.10 suggests that only 10% of the variance is explained, indicating a further decline in performance and potential overfitting.

**Test Set:** The test set results show an RMSE of 36.15, MAE of 27.88, and an R² of 0.10. These metrics indicate that the model’s predictive power did not generalize well beyond the training data, with performance continuing to decrease slightly in the test set.



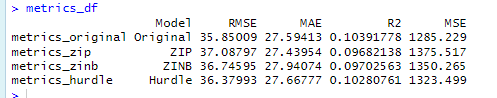
**Use a calibration model (ZIP, ZINB, hurdle) on the validation dataset.**

The following shows the calibration of a LightGBM model’s predictions using zero-inflated and hurdle models to improve the prediction accuracy for the call count data.

Initially, LightGBM predictions on a validation set were rounded and used to create a calibration dataset. Three models were applied for calibration: the Zero-Inflated Poisson (ZIP), Zero-Inflated Negative Binomial (ZINB), and Hurdle model (Negative Binomial). Each model utilized the log-transformed predicted counts as an offset in the model fitting process, accounting for the excess zeros often observed in count data.

After fitting each model, predictions were generated for the validation set, and key performance metrics—Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R-squared (R²), and Mean Squared Error (MSE)—were calculated for each model, including the uncalibrated LightGBM predictions. These metrics were compiled into a table to facilitate comparison across models. Additionally, the Akaike Information Criterion (AIC) was calculated for the ZIP, ZINB, and Hurdle models, offering insight into model fit relative to complexity.

Results indicated slight improvements in accuracy metrics for the calibrated models compared to the original LightGBM predictions, with the ZINB and Hurdle models performing better in handling data overdispersion. The AIC values suggested that the ZINB model offered the best trade-off between fit and complexity. This calibration process demonstrates that applying zero-inflated and hurdle models can enhance predictive performance in datasets with zero-inflated count outcomes, providing a more accurate and nuanced prediction approach.





The performance metrics indicate that the **Hurdle model** outperformed other calibration models, with the lowest RMSE (36.38), MAE (27.67), and MSE (1323.50), and an R2 value close to the original model. The **ZINB model** performed slightly better than ZIP, but its RMSE and MSE were higher than the Hurdle model. AIC values suggest that the Hurdle model (AIC = 98,494.72) is the most parsimonious, outperforming both the ZINB (AIC = 102,354.23) and ZIP (AIC = 304,023.27) models. With regards to this result, the Hurdle model handles excess zeros and non-zero count distributions, making it the preferred calibration approach.

**For the test dataset, first impute the missing values of predictors using Knn-imputation. obtain the predictions from the base learner(s). Take the predictions and pass them thru Calibration/Stacking model to generate final prediction**

**Imputation of Missing Values**:

Missing values in the test dataset were imputed using **KNN imputation**. This involved:

* Converting categorical variables into factors.
* Representing these factors numerically to facilitate KNN imputation.
* Using a weighted mean for continuous variables and the mode for categorical variables based on the nearest neighbors.

**Data Preparation for Modeling**:

After imputation, auxiliary columns created during the process were removed to ensure we have a clean dataset. The imputed dataset was encoded to ensure compatibility with the **LightGBM** model by appropriately handling categorical variables.

**Base Predictions Using LightGBM**:

The base learner (**LightGBM)** model was used to generate base predictions on the prepared test dataset.

**Calibration**:  
The base predictions were passed through a **Hurdle model** (identified as the optimal calibration approach) to adjust for potential biases, excess zeros inherent in the data.

**Appendix**

# Load required libraries

library(dplyr)

library(tidyr)

library(corrplot)

library(ggplot2)

library(lightgbm)

library(caret)

library(Metrics)

library(missForest)

library(vcd) # For Cramer's V

library(GA)

library(pscl) # For zero-inflated models

library(MASS) # For negative binomial

library(performance) # For model comparison

library(ggplot2)

# Read the data

train\_data <- read.csv("train\_data.csv")

test\_data <- read.csv("test\_data.csv")

# Define variable types

continuous\_vars <- c("ann\_prm\_amt", "newest\_veh\_age", "home\_lot\_sq\_footage",

"household\_policy\_counts", "X12m\_call\_history", "tenure\_at\_snapshot")

categorical\_vars <- c("acq\_method", "bi\_limit\_group", "channel", "digital\_contact\_ind",

"geo\_group", "has\_prior\_carrier", "household\_group", "pay\_type\_code",

"pol\_edeliv\_ind", "prdct\_sbtyp\_grp", "product\_sbtyp", "telematics\_ind",

"trm\_len\_mo")

target\_var <- "call\_counts"

# Missing data imputation: First impute all missing values of the predictors in training set.

train\_data$newest\_veh\_age <- ifelse(train\_data$newest\_veh\_age == -20, NA, train\_data$newest\_veh\_age)

train\_data$telematics\_ind <- ifelse(train\_data$telematics\_ind == -2 | train\_data$telematics\_ind == 0, NA, train\_data$telematics\_ind)

train\_data$pol\_edeliv\_ind <- ifelse(train\_data$pol\_edeliv\_ind == -2, NA, train\_data$pol\_edeliv\_ind)

# Handle missing values in Test

test\_data$newest\_veh\_age <- ifelse(test\_data$newest\_veh\_age == -20, NA, test\_data$newest\_veh\_age)

test\_data$telematics\_ind <- ifelse(test\_data$telematics\_ind == -2 | test\_data$telematics\_ind == 0, NA, test\_data$telematics\_ind)

test\_data$pol\_edeliv\_ind <- ifelse(test\_data$pol\_edeliv\_ind == -2, NA, test\_data$pol\_edeliv\_ind)

# Ensure categorical variables are set as factors

train\_data[categorical\_vars] <- lapply(train\_data[categorical\_vars], as.factor)

# Calculate % missing

missing\_percentage <- sapply(train\_data, function(x) mean(is.na(x)) \* 100)

missing\_percentage

continuous\_associations <- sapply(continuous\_vars, function(var) cor(train\_data[[var]], train\_data$call\_counts, use = "complete.obs"))

continuous\_associations

# Chi-square association for each categorical variable

categorical\_associations <- sapply(categorical\_vars, function(var) {

table\_var <- table(train\_data[[var]], train\_data$call\_counts)

cramer\_v <- assocstats(table\_var)$cramer

cramer\_v

})

categorical\_associations

# Perform Random Forest imputation using missForest

imputed\_data <- missForest(train\_data, maxiter = 10, ntree = 10, variablewise = TRUE, decreasing = TRUE)

imputed\_train\_data <- as.data.frame(imputed\_data$ximp)

# Set seed for reproducibility

set.seed(123)

# Split the data into Training (60%) and Remaining (40%)

trainIndex <- createDataPartition(imputed\_train\_data$call\_counts, p = 0.6, list = FALSE)

trainSet <- imputed\_train\_data[trainIndex, ]

remainingSet <- imputed\_train\_data[-trainIndex, ]

# Split the remaining data into Validation (20%) and Test (20%)

set.seed(123) # Reset the seed for consistency

validIndex <- createDataPartition(remainingSet$call\_counts, p = 0.5, list = FALSE)

validSet <- remainingSet[validIndex, ]

testSet <- remainingSet[-validIndex, ]

####################### Train a base learner on the training set

# Prepare data for LightGBM

# LightGBM requires data in matrix format

train\_data <- as.matrix(trainSet[, -which(names(trainSet) == "call\_counts")])

train\_label <- trainSet$call\_counts

valid\_data <- as.matrix(validSet[, -which(names(validSet) == "call\_counts")])

valid\_label <- validSet$call\_counts

test\_data <- as.matrix(testSet[, -which(names(testSet) == "call\_counts")])

test\_label <- testSet$call\_counts

# First, create a function to encode categorical variables

encode\_categorical <- function(data, categorical\_vars) {

# Create a copy of the data

encoded\_data <- data

# Loop through each categorical variable

for(var in categorical\_vars) {

# Convert to numeric, starting from 0

encoded\_data[,var] <- as.numeric(as.factor(data[,var])) - 1

}

return(encoded\_data)

}

# Encode categorical variables in training, validation, and test sets

train\_data\_encoded <- encode\_categorical(train\_data, categorical\_vars)

valid\_data\_encoded <- encode\_categorical(valid\_data, categorical\_vars)

test\_data\_encoded <- encode\_categorical(test\_data, categorical\_vars)

# Convert to matrix format

train\_matrix <- as.matrix(train\_data\_encoded)

valid\_matrix <- as.matrix(valid\_data\_encoded)

test\_matrix <- as.matrix(test\_data\_encoded)

# Create LightGBM datasets with categorical features

dtrain <- lgb.Dataset(

data = train\_matrix,

label = train\_label,

categorical\_feature = categorical\_vars

)

dvalid <- lgb.Dataset(

data = valid\_matrix,

label = valid\_label,

categorical\_feature = categorical\_vars,

reference = dtrain

)

# Set LightGBM parameters

params <- list(

objective = "poisson",

metric = "rmse",

boosting = "gbdt",

num\_leaves = 31,

learning\_rate = 0.05,

feature\_fraction = 0.9,

bagging\_fraction = 0.8,

bagging\_freq = 5,

verbose = -1,

min\_data\_in\_leaf = 20, # Added to prevent overfitting

max\_depth = 8 # Added to control tree depth

)

# Train the model with early stopping

model <- lgb.train(

params = params,

data = dtrain,

valids = list(valid = dvalid),

nrounds = 1000,

early\_stopping\_rounds = 50

)

# Make predictions using encoded matrices

train\_pred <- predict(model, train\_matrix)

valid\_pred <- predict(model, valid\_matrix)

test\_pred <- predict(model, test\_matrix)

# Calculate metrics

metrics <- data.frame(

Dataset = c("Training", "Validation", "Test"),

RMSE = c(

rmse(train\_label, train\_pred),

rmse(valid\_label, valid\_pred),

rmse(test\_label, test\_pred)

),

MAE = c(

mae(train\_label, train\_pred),

mae(valid\_label, valid\_pred),

mae(test\_label, test\_pred)

),

R2 = c(

cor(train\_label, train\_pred)^2,

cor(valid\_label, valid\_pred)^2,

cor(test\_label, test\_pred)^2

)

)

metrics

# Prepare validation data for calibration

valid\_pred\_rounded <- round(valid\_pred)

calibration\_data <- data.frame(

actual = valid\_label,

predicted = valid\_pred\_rounded

)

# Fit ZIP model

zip\_model <- zeroinfl(actual ~ offset(log(predicted + 1)),

data = calibration\_data,

dist = "poisson")

# Fit ZINB model

zinb\_model <- zeroinfl(actual ~ offset(log(predicted + 1)),

data = calibration\_data,

dist = "negbin")

# Fit Hurdle model

hurdle\_model <- hurdle(actual ~ offset(log(predicted + 1)),

data = calibration\_data,

dist = "negbin")

# Get predictions from each model

zip\_pred <- predict(zip\_model, newdata = calibration\_data)

zinb\_pred <- predict(zinb\_model, newdata = calibration\_data)

hurdle\_pred <- predict(hurdle\_model, newdata = calibration\_data)

# Calculate performance metrics for each model

calculate\_metrics <- function(actual, predicted) {

rmse\_val <- sqrt(mean((actual - predicted)^2))

mae\_val <- mean(abs(actual - predicted))

r2\_val <- cor(actual, predicted)^2

# Calculate additional metrics for count data

mse\_val <- mean((actual - predicted)^2)

# Calculate AIC if model object is available

return(c(RMSE = rmse\_val, MAE = mae\_val, R2 = r2\_val, MSE = mse\_val))

}

# Calculate metrics for each model

metrics\_original <- calculate\_metrics(valid\_label, valid\_pred)

metrics\_zip <- calculate\_metrics(valid\_label, zip\_pred)

metrics\_zinb <- calculate\_metrics(valid\_label, zinb\_pred)

metrics\_hurdle <- calculate\_metrics(valid\_label, hurdle\_pred)

# Combine metrics

metrics\_df <- data.frame(

Model = c("Original", "ZIP", "ZINB", "Hurdle"),

rbind(metrics\_original, metrics\_zip, metrics\_zinb, metrics\_hurdle)

)

metrics\_df

# Compare model AICs

aic\_values <- c(

ZIP = AIC(zip\_model),

ZINB = AIC(zinb\_model),

Hurdle = AIC(hurdle\_model)

)

aic\_values

# For the test dataset, first impute the missing values of predictors using Knn-imputation (see the MissForest paper).

# obtain the predictions from the base learner(s). Take the predictions and pass them thru Calibration/Stacking model to generate

#final prediction

library(VIM) # For KNN imputation

library(pscl) # For zero-inflated models

library(ranger) # For random forest imputation comparison

library(dplyr)

# Function to prepare data for KNN imputation

prepare\_data <- function(data, categorical\_vars) {

# Convert specified categorical variables to factors

data[categorical\_vars] <- lapply(data[categorical\_vars], as.factor)

# Convert factors to numeric for KNN imputation

data\_numeric <- data

for(var in categorical\_vars) {

data\_numeric[[var]] <- as.numeric(data\_numeric[[var]])

}

return(data\_numeric)

}

test\_prepared <- prepare\_data(test\_data, categorical\_vars)

# Perform KNN imputation

set.seed(123)

test\_imputed\_knn <- kNN(

test\_prepared,

k = 5, # number of neighbors

numFun = weighted.mean, # weighted mean for continuous variables

catFun = maxCat # mode for categorical variables

)

# Remove auxiliary columns created by kNN

test\_imputed\_knn <- test\_imputed\_knn[, !grepl("\_imp$", names(test\_imputed\_knn))]

# Generate predictions from base learner

# Encode categorical variables for LightGBM

test\_encoded <- encode\_categorical(test\_imputed\_knn, categorical\_vars)

test\_matrix <- as.matrix(test\_encoded)

# Get base predictions from LightGBM model

base\_predictions <- predict(model, test\_matrix)

# Apply calibration model (using the best model from previous analysis)

# Create data frame for calibration

test\_pred\_data <- data.frame(

predicted = round(base\_predictions)

)

# Apply ZINB calibration model

final\_predictions <- predict(hurdle\_model,

newdata = data.frame(predicted = test\_pred\_data$predicted))

# Prediction

test\_predictions\_df <- data.frame(Predictions = final\_predictions)

# Convert test predictions to a data frame with 'id' and 'Predict' columns

submission <- data.frame(

id = seq\_len(nrow(test\_data)), # Sequence of IDs, assuming row order matches

Predict = test\_predictions\_df$Predictions

)

# View the final submission data frame

head(submission)