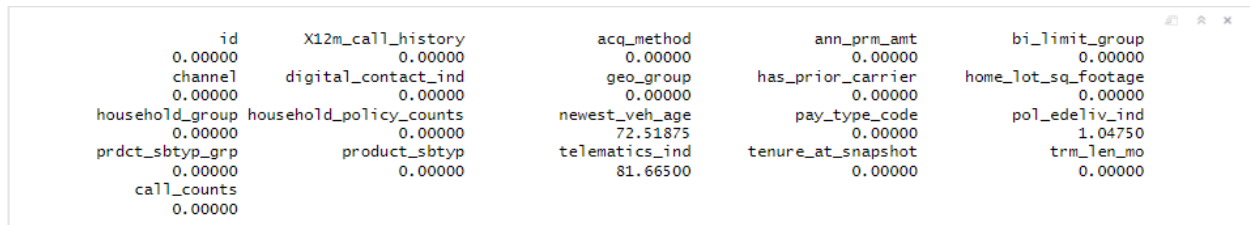


## Augustine Kena Adjei and Oluwafunmibi Omotayo Fasanya

### 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.



id	X12m_call_history	acq_method	ann_prm_amt	bi_limit_group
0.00000	0.00000	0.00000	0.00000	0.00000
channel	digital_contact_ind	geo_group	has_prior_carrier	home_lot_sq_footage
0.00000	0.00000	0.00000	0.00000	0.00000
household_group	household_policy_counts	newest_veh_age	pay_type_code	pol_edeliv_ind
0.00000	0.00000	72.51875	0.00000	1.04750
prdct_sbtyp_grp	product_sbtyp	telematics_ind	tenure_at_snapshot	trm_len_mo
0.00000	0.00000	81.66500	0.00000	0.00000
call_counts				
0.00000				

### Percentage of Missing data in the Training Set

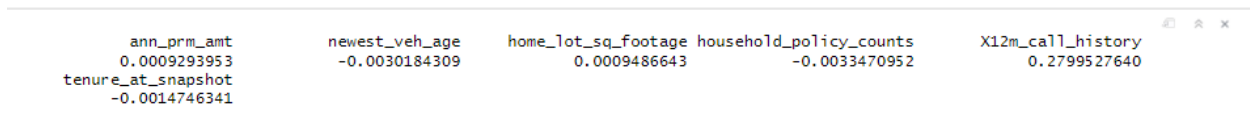
After this replacement was done, we Perform Random Forest imputation using missForest for the training dataset and since the response variable is not known for the test dataset, we Perform KNN imputation on that

### 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

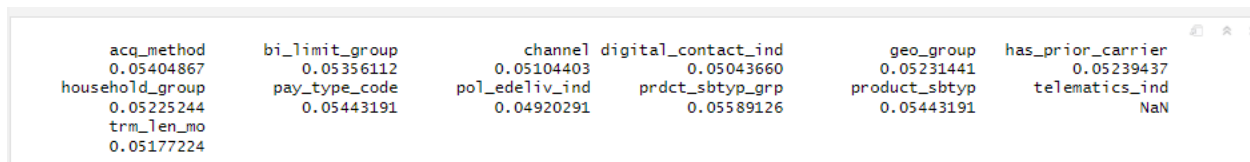
Most of the continuous variables shows a small associations with call counts, except for 12-month call history, which shows a moderate positive relationship. This suggests that prior call activity is likely the most relevant predictor among the variables analyzed.



ann_prm_amt	newest_veh_age	home_lot_sq_footage	household_policy_counts	X12m_call_history
0.0009293953	-0.0030184309	0.0009486643	-0.0033470952	0.2799527640
tenure_at_snapshot	-0.0014746341			

### 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.



acq_method	bi_limit_group	channel	digital_contact_ind	geo_group	has_prior_carrier
0.05404867	0.05356112	0.05104403	0.05043660	0.05231441	0.05239437
household_group	pay_type_code	pol_edeliv_ind	prdct_sbtyp_grp	product_sbtyp	telematics_ind
0.05225244	0.05443191	0.04920291	0.05589126	0.05443191	NaN
trm_len_mo					
0.05177224					

## 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 ( $R^2$ ) for each dataset. These metrics helped us assess model accuracy and generalization across the datasets, with RMSE and MAE reflecting prediction error and  $R^2$  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  $R^2$  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  $R^2$  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  $R^2$  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.

```
> metrics
  Dataset    RMSE    MAE    R2
1 Training 34.93214 26.92330 0.15091643
2 Validation 35.85009 27.59413 0.10391778
3 Test     36.15275 27.87545 0.09813499
< |
```

### 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 ( $R^2$ ), 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.

```
> metrics_df
      Model    RMSE    MAE    R2    MSE
metrics_original Original 35.85009 27.59413 0.10391778 1285.229
metrics_zip      ZIP 37.08797 27.43954 0.09682138 1375.517
metrics_zinb     ZINB 36.74595 27.94074 0.09702563 1350.265
metrics_hurdle   Hurdle 36.37993 27.66777 0.10280761 1323.499
> |

> aic_values
      ZIP      ZINB      Hurdle
304023.27 102354.23  98494.72
> |
```

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  $R^2$  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)
```