Baseball Swing Probability Prediction

Our goal is to give a SwingProbability for all pitches in the `year 3` dataset. There is no description column for these pitches for year 3. The CSV files contain information from around 2,000,000 pitches from baseball games over three years. There are 17 columns (definitions can be found in the documentation csv):

```
# Load the data
documentation <- read.csv('Documentation.csv')
year1 <- read.csv('year1.csv')
year2 <- read.csv('year2.csv')
year3 <- read.csv('year3.csv')

# Function to clean and convert numeric columns
clean_and_convert <- function(df) {
  df %>%
    mutate(across(c(release_speed, pfx_x, pfx_z, plate_x, plate_z, sz_top, sz_bot),
                  ~ as.numeric(gsub("[^0-9.-]", "", .))))
}


# Apply the cleaning and conversion function to each dataset
year1 <- clean_and_convert(year1)
year2 <- clean_and_convert(year2)
year3 <- clean_and_convert(year3)

# Combine year 1 and year 2 data
combined_data <- bind_rows(year1, year2)

# Check for missing values
print(colSums(is.na(combined_data)) )
```

```
##        season      pitch_id release_speed        batter       pitcher
##             0             0           779             0             0
##   description         stand      p_throws    pitch_type         balls
##             0             0             0             0             0
##       strikes         pfx_x         pfx_z       plate_x       plate_z
##             0          3460          1477           779           812
##        sz_top        sz_bot
##           779           824
```

```
# Calculate the percentage of missing values for each column
print(colSums(is.na(combined_data)) / nrow(combined_data) * 100)
```

```
##        season      pitch_id release_speed        batter       pitcher
##    0.00000000    0.00000000    0.05492135    0.00000000    0.00000000
##   description         stand      p_throws    pitch_type         balls
##    0.00000000    0.00000000    0.00000000    0.00000000    0.00000000
##       strikes         pfx_x         pfx_z       plate_x       plate_z
##    0.00000000    0.24393821    0.10413200    0.05492135    0.05724793
##        sz_top        sz_bot
##    0.05492135    0.05809395
```

The percentages of missing values are relatively low (all less than 0.25%), which suggests that removing these entries will not reduce the data too much.

```
# Remove rows with any missing data
combined_data <- combined_data[complete.cases(combined_data), ]

# Check the dimensions of the cleaned data
dim(combined_data)
```

```
## [1] 1414163      17
```

```
# Check for any remaining NAs
print(colSums(is.na(combined_data)))
```

```
##        season      pitch_id release_speed        batter       pitcher
##             0             0             0             0             0
##   description         stand      p_throws    pitch_type         balls
##             0             0             0             0             0
##       strikes         pfx_x         pfx_z       plate_x       plate_z
##             0             0             0             0             0
##        sz_top        sz_bot
##             0             0
```

There are no more missing values in the dataset now. Lets proceed with the next steps.

```
# Create swing target variable
combined_data <- combined_data %>%
  mutate(swing = 1)

print(unique(combined_data$description))
```

```
##  [1] "ball"                    "foul"
##  [3] "called_strike"           "blocked_ball"
##  [5] "hit_into_play"           "hit_by_pitch"
##  [7] "swinging_strike"         "foul_tip"
##  [9] "foul_bunt"               "swinging_strike_blocked"
## [11] "missed_bunt"             "pitchout"
## [13] "bunt_foul_tip"           "foul_pitchout"
```

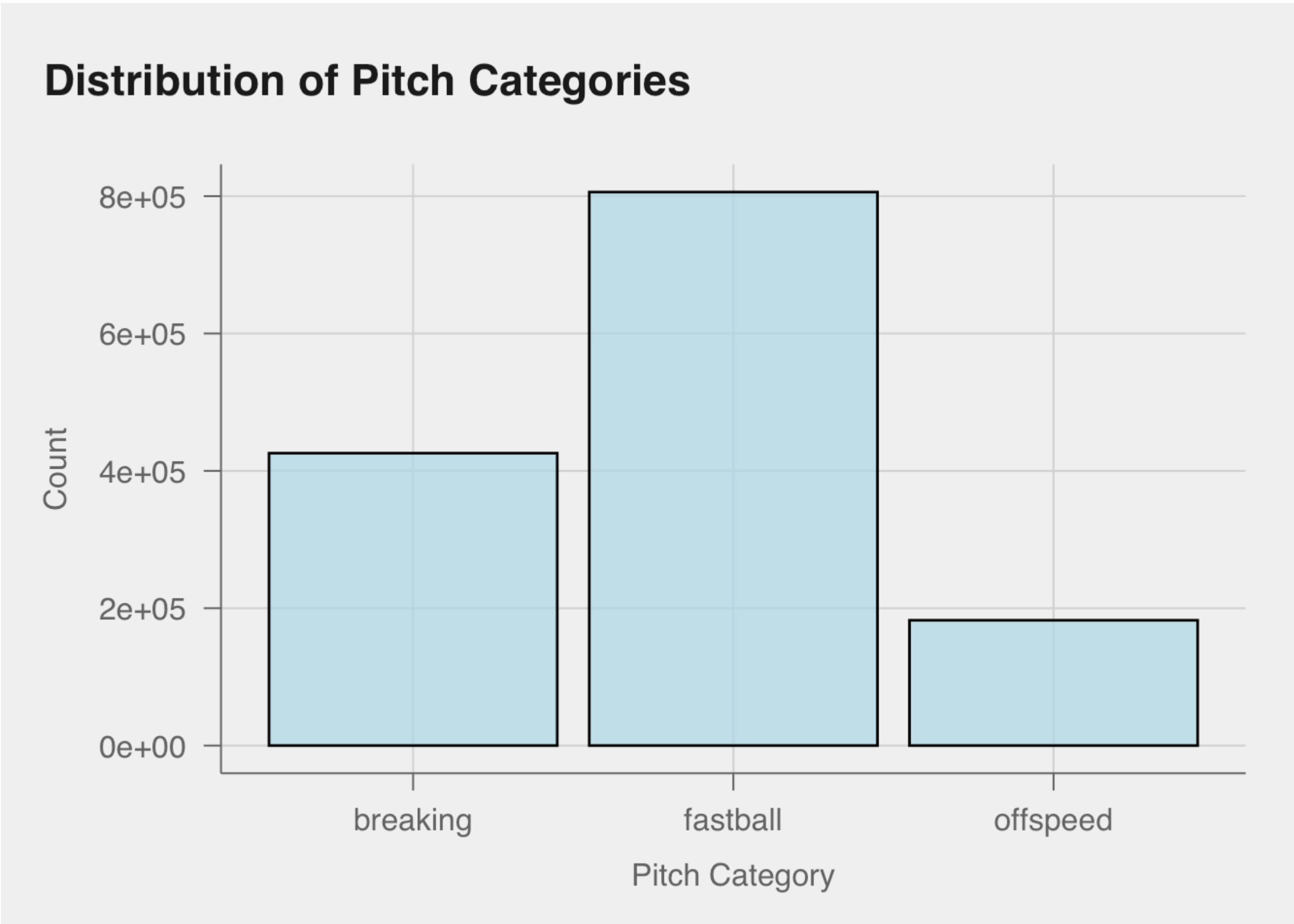
```r
# Convert swing = 0 for non-swing descriptions
combined_data$swing <- ifelse(combined_data$description %in%
                                c('ball', 'called_strike', 'blocked_ball',
                                  'hit_by_pitch', 'pitchout'), 0,
                              combined_data$swing)

# Drop the description column
combined_data <- combined_data %>% select(-description)

# Feature engineering
combined_data <- combined_data %>%
  mutate(ball_strike_ratio = balls / (strikes + 1))

# Categorize pitches in 3 categories to reduce dimensionality during model training
print(unique(combined_data$pitch_type))
```

```
##  [1] "SI" "FF" "SL" "KC" "CH" "CU" "FC" "ST" "FS" "CS" "SV" "FA" "PO" "SC" "EP"
## [16] "KN"
```

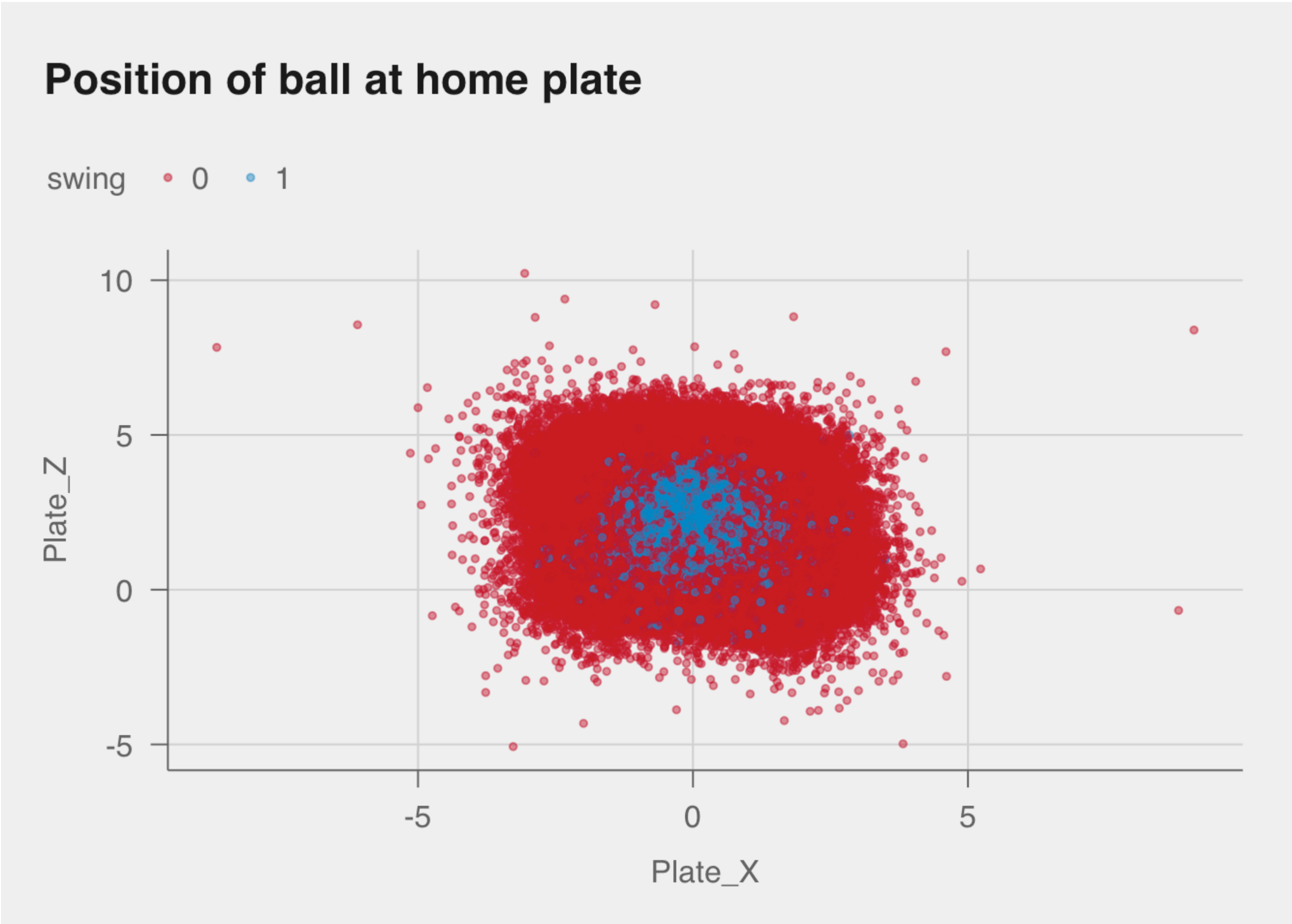
```r
combined_data <- combined_data %>%
  mutate(pitch_type = case_when(
    pitch_type %in% c("FF", "SI", "FC", "FA") ~ "fastball",
    pitch_type %in% c("SL", "CU", "KC", "SC", "SV", "ST") ~ "breaking",
    pitch_type %in% c("CH", "FS", "CS", "EP", "KN", "PO") ~ "offspeed",
    TRUE ~ "Other"
  ))

# Encode categorical variables
combined_data <- combined_data %>%
  mutate(across(c(stand, p_throws, pitch_type, swing), as.factor)) %>%
  mutate(across(c(batter, pitcher, balls, strikes), as.numeric))

# Create a new feature for the interaction between pitch type and release speed
combined_data <- combined_data %>%
  mutate(interaction_pitch_release = as.numeric(as.factor(pitch_type)) * release_speed)
%>%
  select("season", "pitch_id", "release_speed", "batter", "pitcher", "stand",
         "p_throws", "pitch_type", "balls", "strikes", "pfx_x", "pfx_z",
         "plate_x", "plate_z", "sz_top", "sz_bot", "ball_strike_ratio",
         "interaction_pitch_release", "swing")
```

The data is now cleaned, missing values are imputed, and the categorical variables are encoded. The target variable “swing” has been created based on the description column. The data-set is now ready for model training. But first lets visualize some of the features.

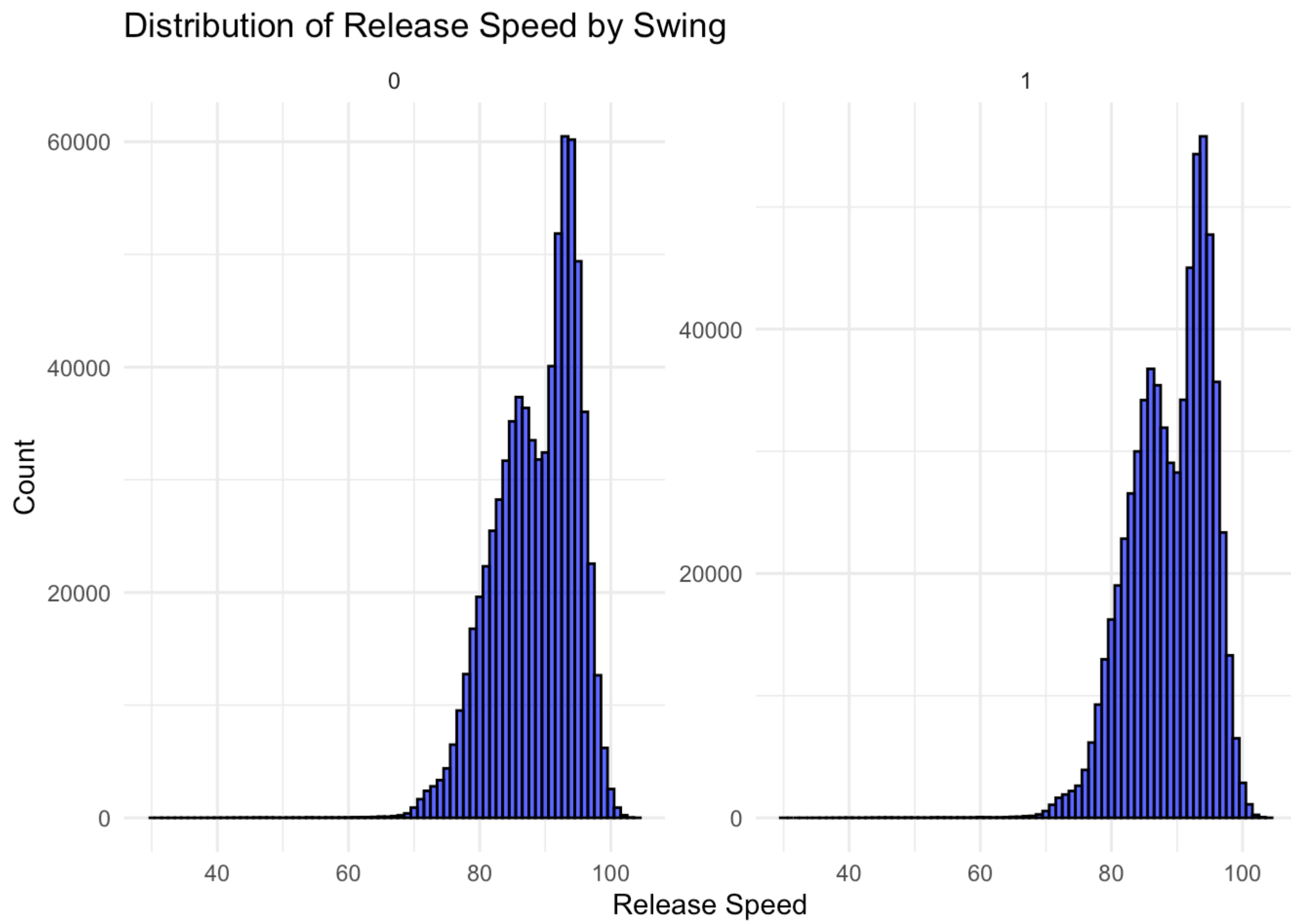
```
# Bar plot for pitch categories
ggplot(combined_data, aes(x = pitch_type)) +
  geom_bar(fill = "lightblue", color = "black", alpha = 0.7) +
  theme_minimal() +
  labs(title = "Distribution of Pitch Categories", x = "Pitch Category", y = "Count") +
  theme_pub()
```



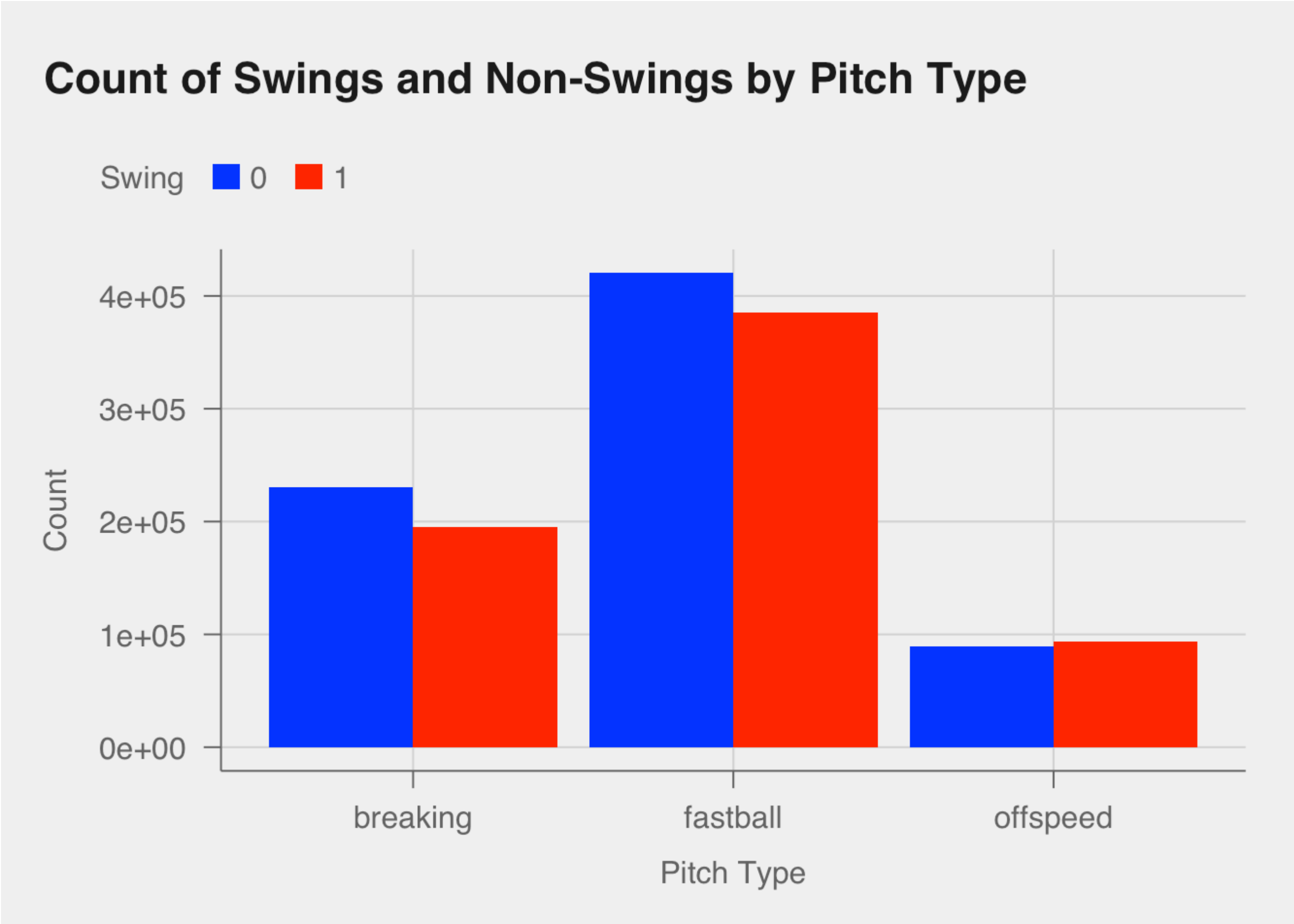

```
# Scatter plot of ball position at home plate by swing
ggplot(combined_data, aes(x = plate_x, y = plate_z, color = swing)) +
  geom_point(alpha = 0.5, size = 1) +
  theme_minimal() +
  labs(title = "Position of ball at home plate", x = "Plate_X", y = "Plate_Z") +
  theme_pub()
```




```
# Distribution of release_speed by swing
ggplot(combined_data, aes(x = release_speed)) +
  geom_histogram(binwidth = 1, fill = "blue", color = "black", alpha = 0.7) +
  theme_minimal() +
  labs(title = "Distribution of Release Speed by Swing", x = "Release Speed", y = "Coun
t") +
  facet_wrap(~ swing, scales = "free_y")
```




```
# Ensure pitch_type and swing are factors
combined_data$pitch_type <- as.factor(combined_data$pitch_type)
combined_data$swing <- as.factor(combined_data$swing)

# Plot the count of swings and non-swings for each pitch type
ggplot(combined_data, aes(x = pitch_type, fill = swing)) +
  geom_bar(position = "dodge") +
  theme_minimal() +
  labs(title = "Count of Swings and Non-Swings by Pitch Type",
       x = "Pitch Type", y = "Count", fill = "Swing") +
  theme_pub() +
  scale_fill_manual(values = c("0" = "blue", "1" = "red"))
```



From the plots above, we make some observations:

- Most of the pitches are fastballs, followed by breaking balls and offspeed pitches.
- The distribution of pitches that result in a swing is similar across the different pitch categories. Offspeed pitches have the highest proportion of swings but only marginally.
- The distribution of release speed is similar for both swings and non-swings.
- Most of the swings are concentrated at the middle of the plate.

Now lets take care of some pre-requesites before training the models.

```
# Function to evaluate model
calculate_metrics <- function(actual, predicted, prob, train_time) {
  actual_factor <- factor(actual, levels = c(0, 1))
  predicted_factor <- factor(predicted, levels = c(0, 1))

  accuracy <- mean(predicted_factor == actual_factor)
  roc_auc <- roc(actual_factor, prob)$auc
  conf_matrix <- confusionMatrix(predicted_factor, actual_factor)

  precision <- conf_matrix$byClass['Pos Pred Value']
  recall <- conf_matrix$byClass['Sensitivity']
  f1_score <- 2 * (precision * recall) / (precision + recall)

  return(list(accuracy = accuracy, roc_auc = roc_auc, precision = precision,
              recall = recall, f1_score = f1_score, training_time = train_time,
              confusion_matrix = conf_matrix$table))
}

# List to store results
results <- list()

# Check if target variable is balanced
table(combined_data$swing) # Fairly balanced
```

```
##
##      0      1
## 739934 674229
```

```
# Drop columns that are not needed
combined_data_sample <- combined_data %>% select(-season, -pitch_id, -batter, -pitcher)

# Sample a subset of the training data (100000 rows) for faster training
set.seed(42)
combined_data_sample <- combined_data_sample %>%
  sample_n(100000)

# Define features and target
features <- combined_data_sample %>% select(-swing)
target <- combined_data_sample$swing

# Split the data into training and validation sets
set.seed(42)
train_index <- createDataPartition(target, p = 0.8, list = FALSE)
train_data <- combined_data_sample[train_index, ]
val_data <- combined_data_sample[-train_index, ]
```

Lets train some models now. Starting with a simple logistic regression.

```r
# Logistic Regression
start_time <- Sys.time()
log_model <- glm(as.factor(swing) ~ ., data = train_data, family = binomial)
end_time <- Sys.time()
train_time <- end_time - start_time

# Evaluate the model
val_prob_log <- predict(log_model, val_data, type = "response")
val_pred_log <- ifelse(val_prob_log > 0.5, 1, 0)

# Calculate results
results$logistic_regression <- calculate_metrics(val_data$swing, val_pred_log, val_prob_
log, train_time)
```

```
## Setting levels: control = 0, case = 1
```

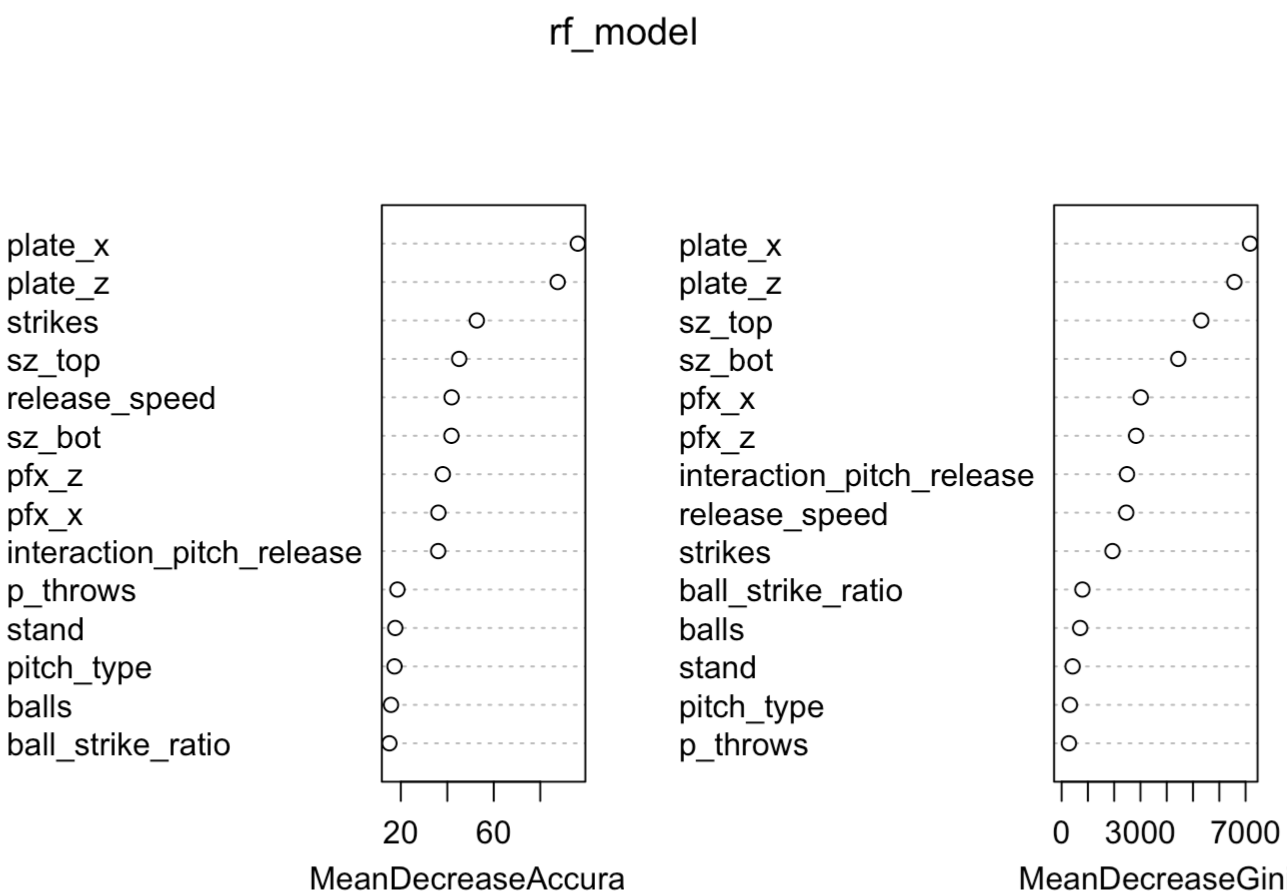
```
## Setting direction: controls < cases
```

```r
print(results$logistic_regression)
```

```
## $accuracy
## [1] 0.6337817
##
## $roc_auc
## Area under the curve: 0.6908
##
## $precision
## Pos Pred Value
##      0.6398505
##
## $recall
## Sensitivity
##   0.6869208
##
## $f1_score
## Pos Pred Value
##      0.6625507
##
## $training_time
## Time difference of 0.5506501 secs
##
## $confusion_matrix
##           Reference
## Prediction    0    1
##          0 7190 4047
##          1 3277 5485
```

It was quick but not the best performance. An accuracy of only 0.63 and an ROC AUC of 0.69. Lets try some more complex models to see if we can do better. Moving on to random forests so we can also extract important features.

```
# Random Forest
start_time <- Sys.time()
set.seed(42)
rf_model <- randomForest(as.factor(swing) ~ ., data = train_data, ntree = 100, importanc
e = TRUE)
end_time <- Sys.time()
train_time <- end_time - start_time

# Extracting important features to only use those
importance <- importance(rf_model)
important_features <- rownames(importance[order(importance[, 1], decreasing = TRUE), ])
[1:10]

# Plot feature importance
varImpPlot(rf_model)
```



```
# Evaluate the model
val_pred_rf <- predict(rf_model, val_data, type = "response")
val_prob_rf <- predict(rf_model, val_data, type = "prob")[, 2]

# Calculate results
results$random_forest <- calculate_metrics(val_data$swing, val_pred_rf, val_prob_rf, tra
in_time)
```

```
## Setting levels: control = 0, case = 1
```

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

```
print(results$random_forest)
```

```
## $accuracy
## [1] 0.8180909
##
## $roc_auc
## Area under the curve: 0.901
##
## $precision
## Pos Pred Value
##      0.8147295
##
## $recall
## Sensitivity
##   0.8444636
##
## $f1_score
## Pos Pred Value
##      0.8293301
##
## $training_time
## Time difference of 59.97665 secs
##
## $confusion_matrix
##           Reference
## Prediction    0    1
##          0 8839 2010
##          1 1628 7522
```

The random forest model achieved an accuracy of 0.82 and an ROC AUC of 0.90 on the validation set, indicating decently good performance. We can see from the feature importance plots that the features related to the ball's position at home plate are the most important in predicting the swing outcome. We also see that strikes, release speed, and batter's strike zone features are also important. From here, we can see that the random forest model is a better choice than the logistic regression model.

Lets move on to XGBoost to see if we can top the accuracy of the random forest model.

```r
# XGBoost
# Combine train and validation data to ensure consistent dummy variable encoding
combine <- rbind(train_data, val_data)

# Create dummy variables for combined data
dummies <- dummyVars(~ ., data = combine, fullRank = TRUE)

# Encode train data
train_data_encoded <- predict(dummies, newdata = train_data)
train_data_encoded <- as.data.frame(train_data_encoded)

# Encode validation data
val_data_encoded <- predict(dummies, newdata = val_data)
val_data_encoded <- as.data.frame(val_data_encoded)

# Ensure labels are 0 or 1 for binary classification
train_label <- ifelse(train_data[, ncol(train_data)] == "1", 1, 0)
val_label <- ifelse(val_data[, ncol(val_data)] == "1", 1, 0)

# Separate features and target variable
train_matrix <- as.matrix(train_data_encoded[, -ncol(train_data_encoded)])
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)

# Set parameters for binary classification
params <- list(
  objective = "binary:logistic",
  eta = 0.3,
  max_depth = 6,
  eval_metric = "error"
)

# Train the model
start_time <- Sys.time()
set.seed(42)
num_rounds <- 100
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = num_rounds)
end_time <- Sys.time()
train_time <- end_time - start_time

# Convert validation data to matrix
val_matrix <- as.matrix(val_data_encoded[, -ncol(val_data_encoded)])
dval <- xgb.DMatrix(data = val_matrix)

# Make predictions
val_prob_xgb <- predict(xgb_model, dval)
val_pred_xgb <- ifelse(val_prob_xgb > 0.5, 1, 0)

# Calculate results
results$xgb <- calculate_metrics(val_label, val_pred_xgb, val_prob_xgb, train_time)
```

```
## Setting levels: control = 0, case = 1
```

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

```
print(results$xgb)
```

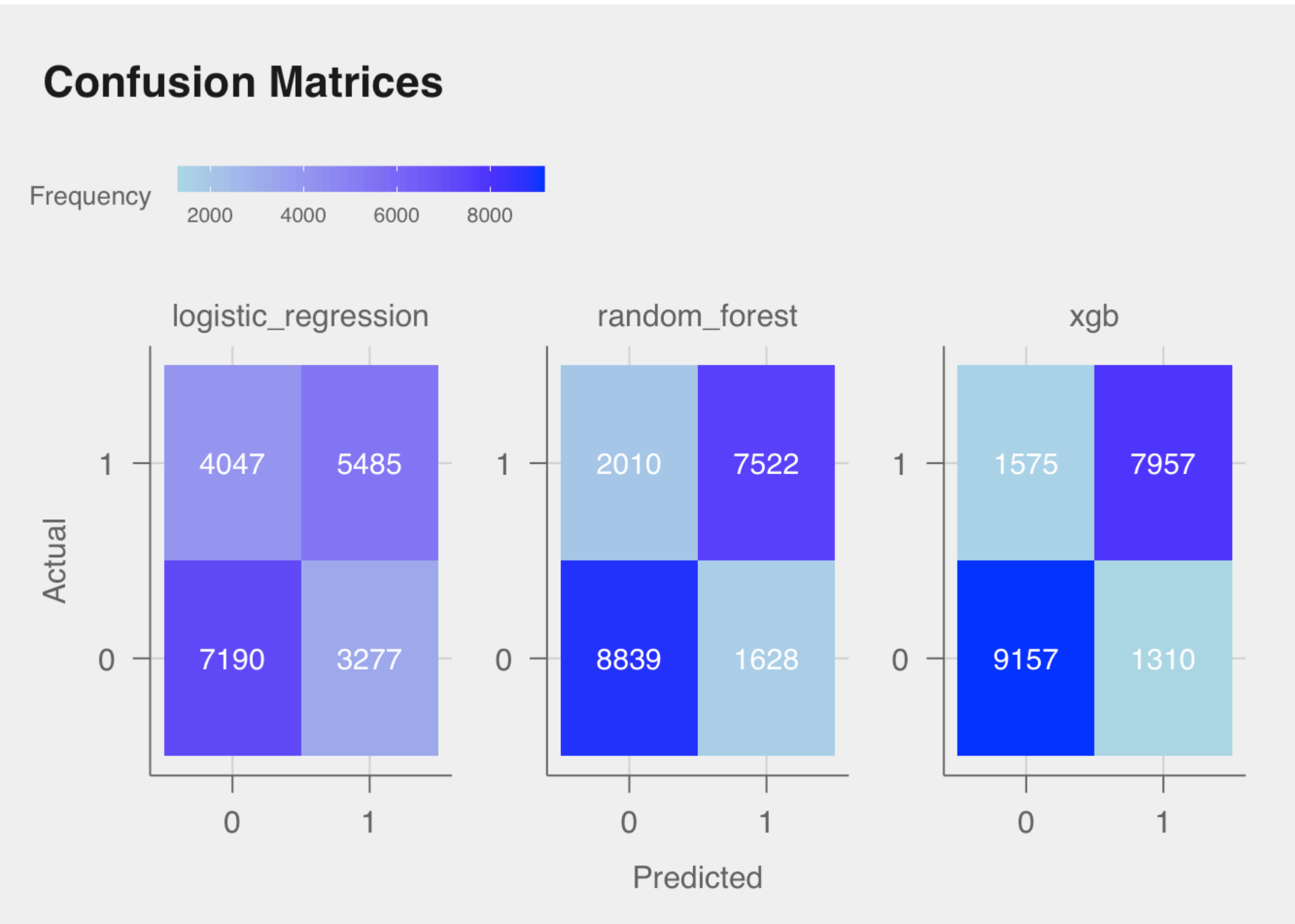
```
## $accuracy
## [1] 0.8557428
##
## $roc_auc
## Area under the curve: 0.9362
##
## $precision
## Pos Pred Value
##      0.8532426
##
## $recall
## Sensitivity
##   0.8748448
##
## $f1_score
## Pos Pred Value
##      0.8639087
##
## $training_time
## Time difference of 8.098759 secs
##
## $confusion_matrix
##           Reference
## Prediction    0    1
##          0 9157 1575
##          1 1310 7957
```

We have gotten the best results with the XGBoost model. An accuracy of 0.85 and an ROC AUC of 0.94. We can see that the XGBoost model outperforms the random forest and logistic regression models. The XGBoost model is the best choice for predicting swing outcomes in this case.

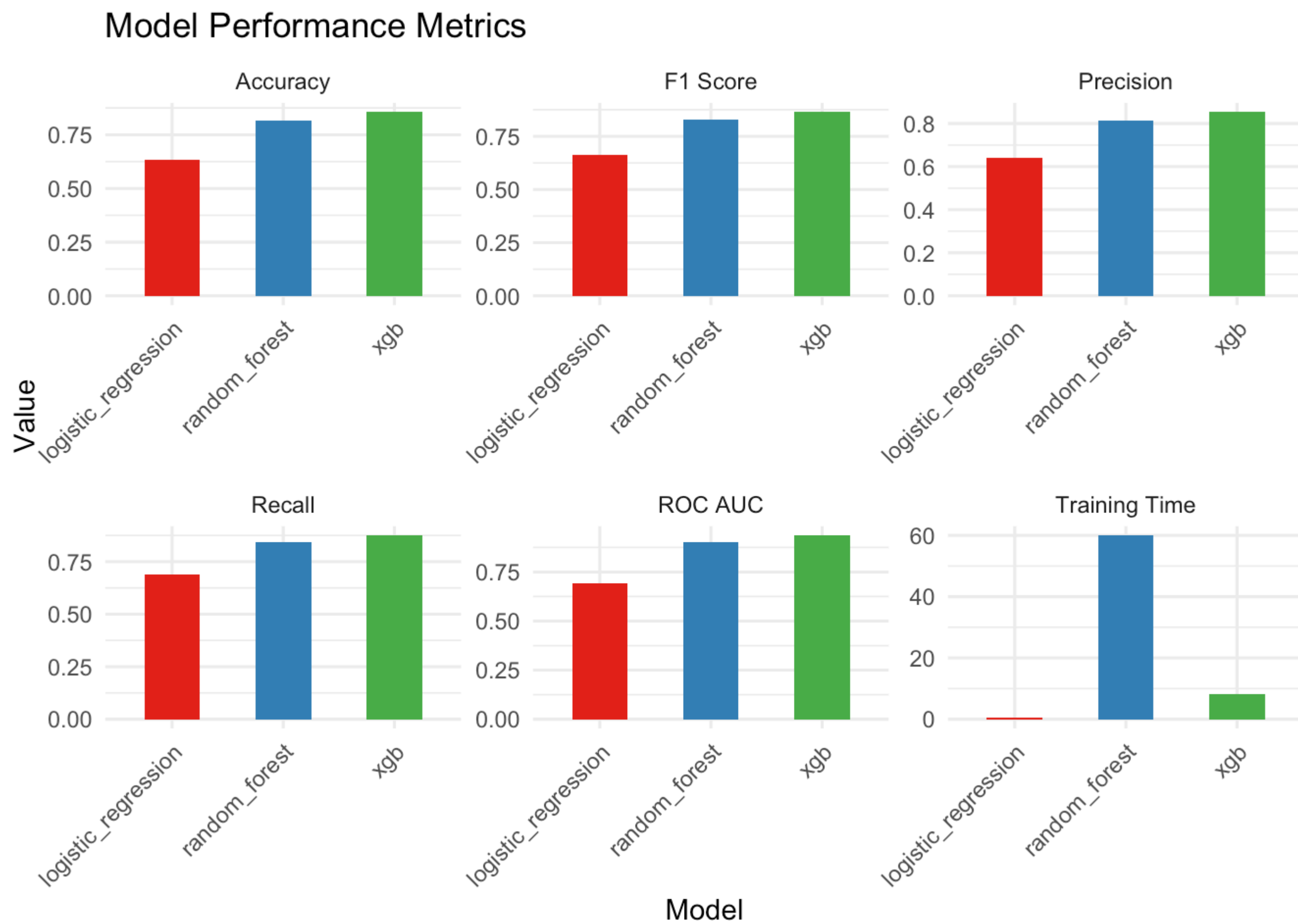
We can see this by also visualizing the results of all our three models.

```
# Function to extract metrics
# Function to extract metrics including training time
extract_metrics <- function(results) {
  data.frame(
    Model = rep(names(results), each = 6),
    Metric = rep(c("Accuracy", "ROC AUC", "Precision", "Recall", "F1 Score", "Training T
ime"), times = length(results)),
    Value = unlist(lapply(results, function(x) {
      # Handle potential structure differences
      accuracy <- if ("accuracy" %in% names(x)) x$accuracy else NA
      roc_auc <- if ("roc_auc" %in% names(x)) as.numeric(x$roc_auc) else NA
      precision <- if ("precision" %in% names(x)) x$precision[1] else NA
      recall <- if ("recall" %in% names(x)) x$recall[1] else NA
      f1_score <- if ("f1_score" %in% names(x)) x$f1_score[1] else NA
      training_time <- if ("training_time" %in% names(x)) {
        # Convert training time to seconds if necessary
        if (attr(x$training_time, "units") == "mins") {
          as.numeric(x$training_time) * 60
        } else {
          as.numeric(x$training_time)
        }
      } else NA
      c(accuracy, roc_auc, precision, recall, f1_score, training_time)
    }))
  )
}

# Prepare the data for metrics
metrics <- extract_metrics(results)

# Separate the training time data from the metrics
training_time_data <- metrics %>% filter(Metric == "Training Time")

# Extract confusion matrices into a data frame
extract_conf_matrices <- function(results) {
  do.call(rbind, lapply(names(results), function(name) {
    cm <- results[[name]]$confusion_matrix
    data.frame(Model = name,
               Actual = rep(rownames(cm), each = ncol(cm)),
               Predicted = rep(colnames(cm), times = nrow(cm)),
               Freq = as.vector(cm))
  }))
}

# Prepare the data for confusion matrices
conf_matrix_df <- extract_conf_matrices(results)

# Plot the confusion matrices
ggplot(conf_matrix_df, aes(x = Predicted, y = Actual, fill = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), color = "white", size = 4) +
  facet_wrap(~ Model, scales = "free") +
```

```
  scale_fill_gradient(low = "lightblue", high = "blue",
                      breaks = pretty(conf_matrix_df$Freq, n = 5),
                      labels = pretty(conf_matrix_df$Freq, n = 5)) +
  theme_minimal() +
  labs(title = "Confusion Matrices", fill = "Frequency") +
  theme_pub() +
  theme(
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8),
    legend.key.size = unit(0.8, "cm"),
    legend.key.width = unit(1, "cm")
  )
```




```
# All metrics visualized
ggplot(metrics, aes(x = Model, y = as.numeric(Value), fill = Model)) +
  geom_bar(stat = "identity", position = position_dodge(), width = 0.5) +
  theme_minimal() +
  facet_wrap(~ Metric, scales = "free") +
  labs(title = "Model Performance Metrics", y = "Value") +
  scale_fill_brewer(palette = "Set1") +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    legend.position = "none"
  )
```



We can see above that in all metrics, the XGBoost model outperforms the other models. The accuracy, ROC AUC, precision, recall, and F1 score are all higher for the XGBoost model. The training time for the XGBoost model is quite low when compared to random_forests, and only a little bit higher than logistic_regression. So it is the best model at our disposal. The same can be seen from the confusion matrix plots.

We will now train the XBGoost model on the complete data so we can use it to predict the swing probabilities for the year 3 data.

```
# Drop columns that are not needed
combined_data_complete <- combined_data %>% select(-season, -pitch_id, -batter, -pitche
r)

# Define features and target
features <- combined_data_complete %>% select(-swing)
target <- combined_data_complete$swing

# Split the data into training and validation sets
set.seed(42)
train_index <- createDataPartition(target, p = 0.8, list = FALSE)
train_data_complete <- combined_data_complete[train_index, ]
val_data_complete <- combined_data_complete[-train_index, ]

# XGBoost
# Combine train and validation data to ensure consistent dummy variable encoding
combine <- rbind(train_data_complete, val_data_complete)

# Create dummy variables for combined data
dummies <- dummyVars(~ ., data = combine, fullRank = TRUE)

# Encode train data
train_data_complete_encoded <- predict(dummies, newdata = train_data_complete)
train_data_complete_encoded <- as.data.frame(train_data_complete_encoded)

# Encode validation data
val_data_complete_encoded <- predict(dummies, newdata = val_data_complete)
val_data_complete_encoded <- as.data.frame(val_data_complete_encoded)

# Ensure labels are 0 or 1 for binary classification
train_label <- ifelse(train_data_complete[, ncol(train_data_complete)] == "1", 1, 0)
val_label <- ifelse(val_data_complete[, ncol(val_data_complete)] == "1", 1, 0)

# Separate features and target variable
train_matrix <- as.matrix(train_data_complete_encoded[, -ncol(train_data_complete_encode
d)])
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)

# Set parameters for binary classification
params <- list(
  objective = "binary:logistic",
  eta = 0.3,
  max_depth = 6,
  eval_metric = "error"
)

# Train the model
start_time <- Sys.time()
set.seed(42)
num_rounds <- 100
xgbf_model <- xgb.train(params = params, data = dtrain, nrounds = num_rounds)
end_time <- Sys.time()
```

```
train_time <- end_time - start_time

# Convert validation data to matrix
val_matrix <- as.matrix(val_data_complete_encoded[, -ncol(val_data_complete_encoded)])
dval <- xgb.DMatrix(data = val_matrix)

# Make predictions
val_prob_xgbf <- predict(xgbf_model, dval)
val_pred_xgbf <- ifelse(val_prob_xgbf > 0.5, 1, 0)

# Calculate results
results$xgbf <- calculate_metrics(val_label, val_pred_xgbf, val_prob_xgbf, train_time)
```

```
## Setting levels: control = 0, case = 1
```

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

```
print(results$xgbf)
```

```
## $accuracy
## [1] 0.8634379
##
## $roc_auc
## Area under the curve: 0.9434
##
## $precision
## Pos Pred Value
##      0.8566835
##
## $recall
## Sensitivity
##   0.8874691
##
## $f1_score
## Pos Pred Value
##      0.8718046
##
## $training_time
## Time difference of 3.145638 mins
##
## $confusion_matrix
##           Reference
## Prediction      0      1
##          0 131333  21971
##          1  16653 112874
```

So we finally have our model which has an accuracy of 0.86 and an ROC AUC of 0.94.

Now, let's prepare the year 3 data for prediction and predict the swing probabilities using the trained model.

```r
# Prepare year 3 data for prediction
# Check the structure
str(year3)
```

```
## 'data.frame':    717945 obs. of  16 variables:
##  $ season       : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ pitch_id     : chr  "4135978" "4135989" "4135993" "4131576" ...
##  $ release_speed: num  78.8 93.7 94.2 91.2 84.5 ...
##  $ batter       : int  5464 5464 5464 6446 5667 5327 6826 6826 6826 5327 ...
##  $ pitcher      : int  6936 6936 6936 6727 6727 6727 6727 6727 6727 6727 ...
##  $ stand        : chr  "L" "L" "L" "R" ...
##  $ p_throws     : chr  "R" "R" "R" "R" ...
##  $ pitch_type   : chr  "ST" "FF" "FF" "FF" ...
##  $ balls        : int  1 2 3 0 3 0 0 1 2 0 ...
##  $ strikes      : int  1 2 2 0 2 1 0 1 1 1 ...
##  $ pfx_x        : num  1.11 -1.16 -1.24 -1.03 0.2 ...
##  $ pfx_z        : num  0.3 1.36 1.26 1.38 0.12 ...
##  $ plate_x      : num  -0.33 -1.57 -1.31 1.02 0.61 ...
##  $ plate_z      : num  0.49 2.49 3.48 2.31 1.36 ...
##  $ sz_top       : num  3.58 3.58 3.68 3.29 3.41 ...
##  $ sz_bot       : num  1.66 1.69 1.69 1.58 1.63 ...
```

```r
# Check for missing values
print(colSums(is.na(year3)))
```

```
##        season      pitch_id release_speed        batter       pitcher
##             0             0           270             0             0
##         stand      p_throws    pitch_type         balls       strikes
##             0             0             0             0             0
##         pfx_x         pfx_z       plate_x       plate_z        sz_top
##          2835           769          5726           296          2539
##        sz_bot
##           315
```

```r
# Calculate the percentage of missing values for each column
print(colSums(is.na(year3)) / nrow(year3) * 100)
```

```
##        season      pitch_id release_speed        batter       pitcher
##    0.00000000    0.00000000    0.03760734    0.00000000    0.00000000
##         stand      p_throws    pitch_type         balls       strikes
##    0.00000000    0.00000000    0.00000000    0.00000000    0.00000000
##         pfx_x         pfx_z       plate_x       plate_z        sz_top
##    0.39487704    0.10711127    0.79755413    0.04122878    0.35364826
##        sz_bot
##    0.04387523
```

```
# The percentages of missing values are relatively low (all less than 0.8%),
# Imputing the mean for the missing values
year3 <- year3 %>%
  mutate_if(is.numeric, ~ifelse(is.na(.), mean(., na.rm = TRUE), .))

# Check the dimensions of the cleaned data
dim(year3)
```

```
## [1] 717945     16
```

```
# Check for any remaining NAs
print(colSums(is.na(year3)))
```

```
##        season      pitch_id release_speed        batter       pitcher 
##             0             0             0             0             0 
##         stand      p_throws    pitch_type         balls       strikes 
##             0             0             0             0             0 
##         pfx_x         pfx_z       plate_x       plate_z        sz_top 
##             0             0             0             0             0 
##        sz_bot 
##             0
```

```
# Feature engineering
year3 <- year3 %>%
  mutate(ball_strike_ratio = balls / (strikes + 1))

# Categorize pitches in 3 categories to reduce dimensionality during model training
print(table(year3$pitch_type))
```

```
## 
##     CH     CS     CU     EP     FA     FC     FF     FO     FS     KC     KN 
##  78464     53  51675    523   1140  55594 230936    778  15600  12154    190 
##   NULL     PO     SC     SI     SL     ST     SV 
##    269     46     74 110871 126294  30954   2330
```

```
year3 <- year3 %>%
  mutate(pitch_type = case_when(
    pitch_type %in% c("FF", "SI", "FC", "FA", "FO") ~ "fastball",
    pitch_type %in% c("SL", "CU", "KC", "SC", "SV", "ST") ~ "breaking",
    pitch_type %in% c("CH", "FS", "CS", "EP", "KN", "PO") ~ "offspeed",
    TRUE ~ "Other"
  ))

print(table(year3$pitch_type)) # Most common - fastball so replacing `others`
```

```
##
## breaking fastball offspeed    Other
##   223481   399319    94876      269
```

```
year3 <- year3 %>% mutate(pitch_type = ifelse(pitch_type == "Other", "fastball", pitch_t
ype))

# Encode categorical variables
year3 <- year3 %>%
  mutate(across(c(stand, p_throws, pitch_type), as.factor)) %>%
  mutate(across(c(batter, pitcher, balls, strikes), as.numeric))

# Create a new feature for the interaction between pitch type and release speed
year3 <- year3 %>%
  mutate(interaction_pitch_release = as.numeric(as.factor(pitch_type)) * release_speed)
%>%
  select("season", "pitch_id", "release_speed", "batter", "pitcher", "stand",
         "p_throws", "pitch_type", "balls", "strikes", "pfx_x", "pfx_z",
         "plate_x", "plate_z", "sz_top", "sz_bot", "ball_strike_ratio",
         "interaction_pitch_release")

# Drop columns that are not needed
year3 <- year3 %>% select(-season, -pitch_id, -batter, -pitcher)

# Ensure the columns match
missing_cols <- setdiff(names(features), names(year3))
missing_cols # Empty, hence columns match
```

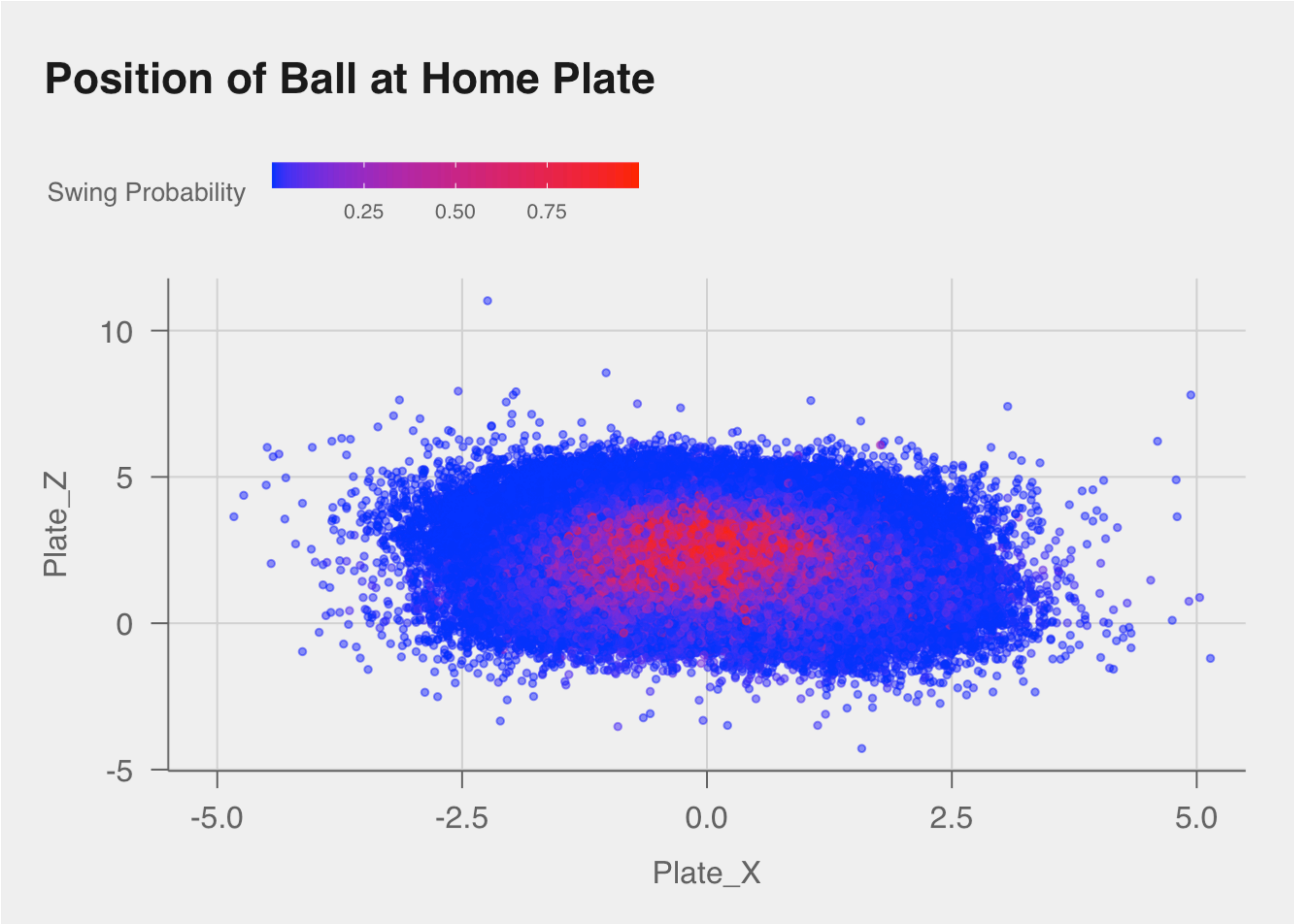
```
## character(0)
```

```
# Prepare for XGBoost
# Create dummy variables for combined data
dummies <- dummyVars(~ ., data = year3, fullRank = TRUE)

# Encode test data
test_year3_encoded <- predict(dummies, newdata = year3)
test_year3_encoded <- as.data.frame(test_year3_encoded)

# Convert validation data to matrix
test_year3_matrix <- as.matrix(test_year3_encoded)
dtest <- xgb.DMatrix(data = test_year3_matrix)

# Make predictions for probability
year3_prob <- predict(xgbf_model, dtest)

# Read year 3 again to get original dataset
year3og <- read.csv("year3.csv")

# Append swing probabilities to the original dataset
year3og <- cbind(year3og, SwingProbability = year3_prob)

# Save the result
write_csv(year3og, 'validation.csv')
```

Done! We have successfully predicted the swing probabilities for the year 3 data and saved the results in a CSV file named `validation.csv`.

Now, just for fun, lets visualize the swinging probabilities on a plot with respect to the horizontal and vertical position of the ball as it reaches the home plate.

```
# Append swing probabilities to the mutated dataset
year3plot <- cbind(year3, SwingProbability = year3_prob)

ggplot(year3plot, aes(x = plate_x, y = plate_z, color = SwingProbability)) +
  geom_point(alpha = 0.5, size = 1) +
  coord_cartesian(xlim = c(-5, 5)) +
  theme_minimal() +
  labs(title = "Position of Ball at Home Plate", x = "Plate_X", y = "Plate_Z", color =
"Swing Probability") +
  scale_color_gradient(low = "blue", high = "red") +
  theme_pub() +
  theme(
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8),
    legend.key.size = unit(0.8, "cm"),
    legend.key.width = unit(1, "cm")
  )
```



As a fascinating visual aid, we can see how the probability of swinging increases when the ball is aimed at the middle of the plate. The red color indicates a higher probability of swinging, while the blue color indicates a lower probability of swinging.