The Impact of Play Clock Timing on NFL Play Outcomes

Dominic Hoar-Weiler, Nathan Yao, and Justin Fung

January 6, 2025

Contents

1 Introduction

One moment etched in the hearts of Minnesota Vikings fans is the 2018 NFL playoff game against the New Orleans Saints. With just 5 seconds left on the

play clock, Case Keenum launches a high, spiraling pass to Stefon Diggs, who streaks down the right sideline. The crowd erupts as Diggs makes the catch and sprints into the end zone, completing one of the most iconic touchdowns in NFL history. This unforgettable play, now famously known as the Minneapolis Miracle, highlights the critical role of the play clock. In football, the play clock isn't just a countdown, it's a pressure cooker that influences decision making, timing, and ultimately, the outcome of a play.

The focus of this research is to examine how the play clock at the snap, along with other variables, impacts play success. By investigating this relationship, we aim to develop a predictive model that forecasts play outcomes based on the timing of the snap.

Tracking data mutation is below. Here, we extract the width of each team's formation before the snap.

```
# Initialize an empty data frame to store alignment data
alignment_data <- data.frame()

for (file in files) {
  # Process each file and calculate alignment data
  file_alignment_data <- tracking_list[[file]] |>
    # Add the `offense` column based on possessionTeam or other indicators
    mutate(offense = (club == possessionTeam)) |>
    filter(frameType == "BEFORE_SNAP") |>
    mutate(
      total_motion = ifelse(offense,
                            sum(sqrt((x - lag(x, default = first(x)))^2 +
                                       (y - lag(y, default = first(y)))^2), na.rm =
```

```
                              NA_real_)
    ) |>
    ungroup() |>
    group_by(gameId, playId) |>
    summarize(
      # Width calculations
      offensive_width = max(x[offense], na.rm = TRUE) - min(x[offense], na.rm = TR
      defensive_width = max(x[!offense], na.rm = TRUE) - min(x[!offense], na.rm =
      # Total offensive motion (sum of offensive player motion)
      total_offensive_motion = sum(total_motion, na.rm = TRUE),
      .groups = "drop"  # Drop grouping after summarizing
    )

  # Append the results from the current file to the overall data
  alignment_data <- bind_rows(alignment_data, file_alignment_data)
}

plays_raw_data <- plays_raw_data |>
  left_join(alignment_data, by = c("gameId", "playId"))
```

2 Exploratory Analysis:

2.1 Background and Variables

We analyze multiple datasets, including player statistics, play-by-play data, and tracking data, sourced from the 2024–2025 NFL season as part of the Big Data Bowl competition.

The data that was used for most of analysis came from the "plays.csv" data set, A number of exploratory variables were recorded:
- passResult:Dropback outcome of the play (C: Complete pass, I: Incomplete pass , S: Quarterback sack, IN: Intercepted pass, R: Scramble, text)
- yardsGained:Net yards gained by the offense, including penalty yardage (numeric)
- playClockAtSnap:What the play clock value was at time of snap (numeric)
- yardsToGo:Distance needed for a first down (numeric)
- down: Down (numeric)
- passLength: The distance beyond the LOS that the ball traveled not including yards into the endzone. If thrown behind LOS, the value is negative. (numeric)
- pff_passCoverage: The pass coverage concept employed by the defense on the play (text)
- dropbackType: The type of drop back after the snap by the QB (Traditional, Designed Rollout, Scramble, Scramble Rollout, Designed Rollout Left, Designed Rollout Right, Scramble Rollout Left, Scramble Rollout Right, Designed Run, QB Draw, Rollout, text)

and many more.

2.2 Some Exploratory Data Analysis with Play Clock at Snap

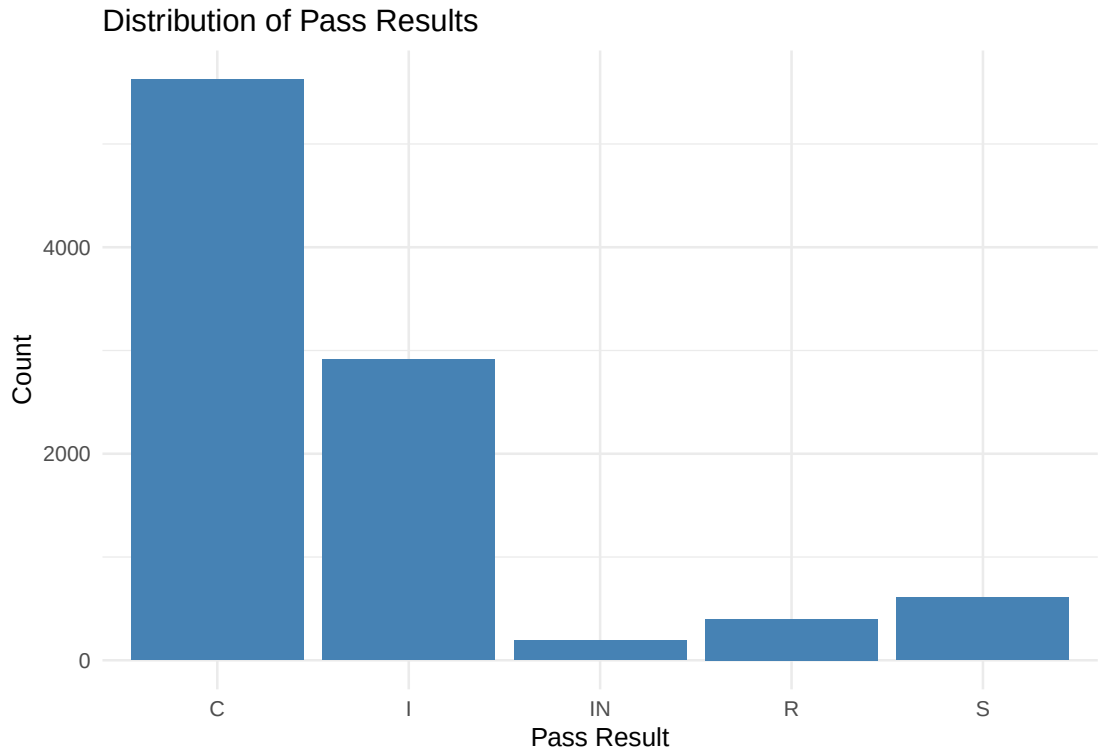
2.2.1 Distributions of Important Variables

There were a variety of possible variables that along with playClockAtSnap that would be important in our analysis. This section explores distributions of impor-

tant variables and the relationships between them. Below are some of the more important variables and their distributions that we looked at in our analysis:

passResult (Categorical): We have filtered out the NA values which indicate non-pass plays, as we are only trying to explore passing plays. (C: Complete pass, I: Incomplete pass , S: Quarterback sack, IN: Intercepted pass, R: Scramble)

```
plays_filtered <- plays_raw_data %>%
  filter(!is.na(passResult))


ggplot(plays_filtered, aes(x = passResult)) +
  geom_bar(fill = "steelblue") +
  labs(
    title = "Distribution of Pass Results",
    x = "Pass Result",
    y = "Count"
  ) +
  theme_minimal()
```

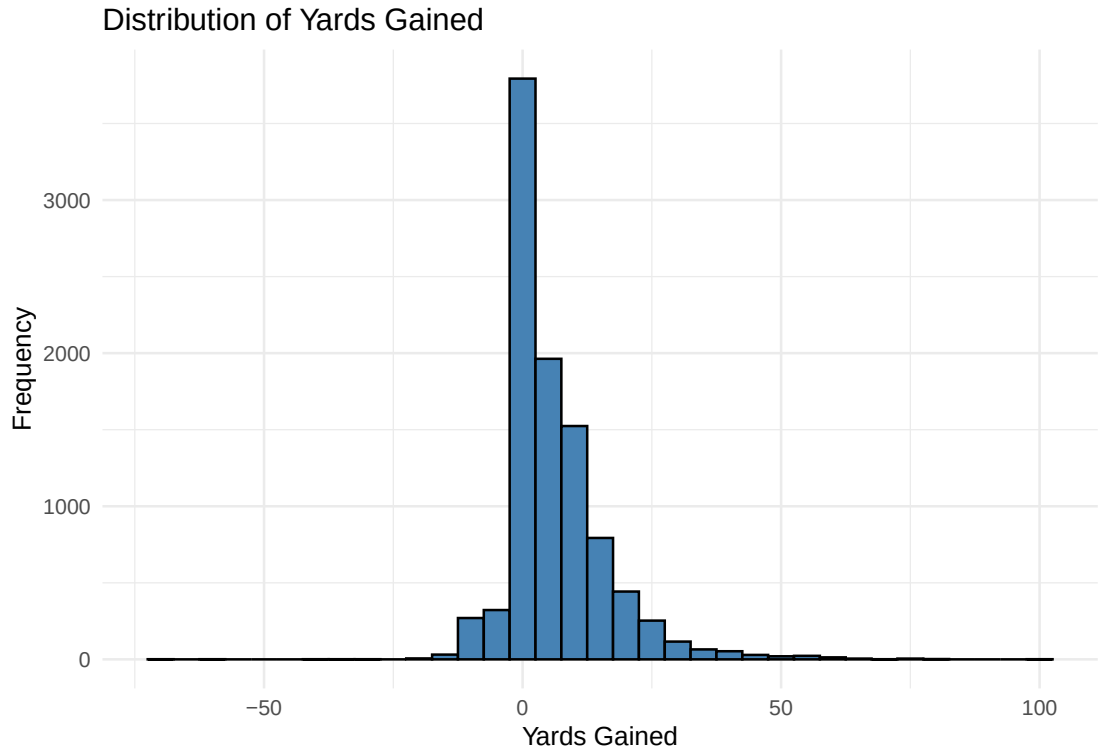


The most frequent pass result is C (completed pass). This makes sense as a lot of the time the quarterback will complete his pass to a receiver. The second most frequent category is I (incomplete pass), which is also expected. After that, the next most frequent categories are in order as follows: S (quarterback sack), R (scramble–quarterback improvised run), and IN (intercepted pass).

yardsGained (Quantitative):

```
ggplot(plays_filtered, aes(x = yardsGained)) +
  geom_histogram(binwidth = 5, fill = "steelblue", color = "black") +
  labs(
    title = "Distribution of Yards Gained",
    x = "Yards Gained",
    y = "Frequency"
  ) +
```

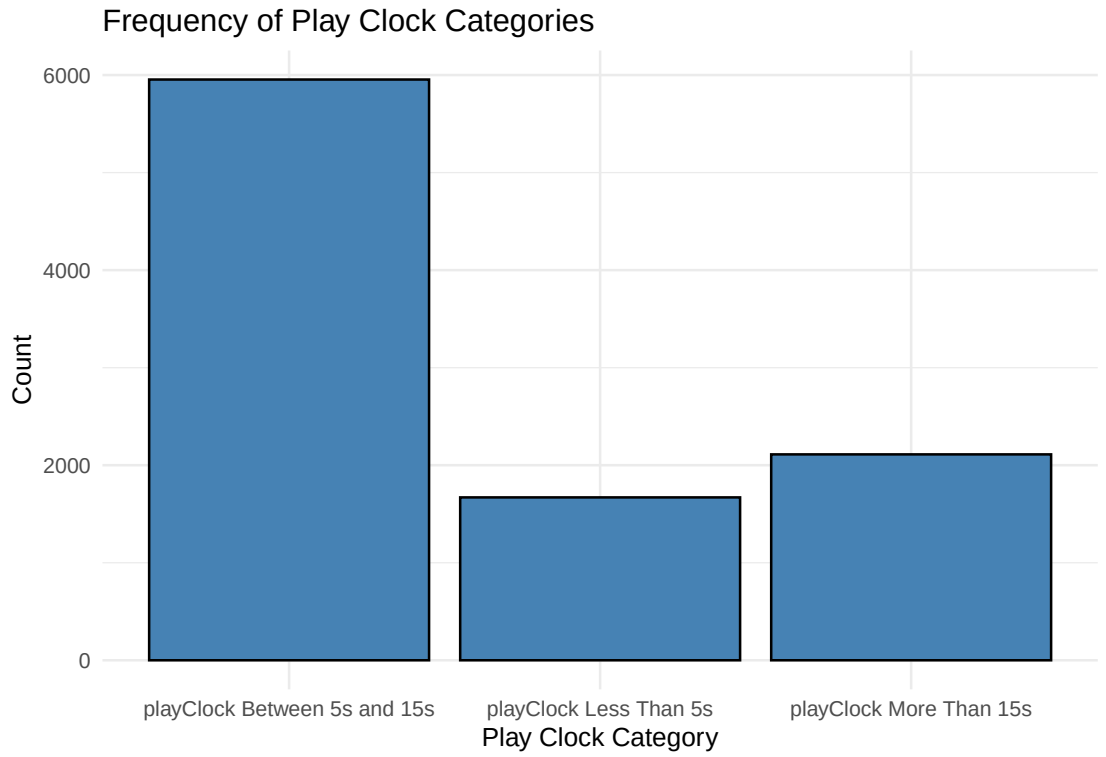
```
theme_minimal()
```



Yards gained looks like it is slightly normally distributed and is right skewed. The overwhelming majority of plays result in 0 yards gained according to the graphic. This makes sense as all incomplete passes result in 0 yards gained, assuming no penalties.

playClockAtSnap (Quantitative): For the play clock at snap, we split the quantitative variable into 3 buckets: High (>=15 seconds), Medium (5–15 seconds), and Low (<5 seconds). These thresholds were chosen to reflect typical game scenarios: early snaps, moderate decision-making time, and rushed snaps. In this case, we created a new variable called playClockCategory which is a categorical variable that has counts of plays in each of the 3 categories. Below is the distribution:

```r
plays_raw_data <- plays_raw_data %>%
  mutate(playClockCategory =
           case_when(playClockAtSnap >= 15 ~ "playClock More Than 15s",
                     playClockAtSnap >= 5 &
                     playClockAtSnap < 15 ~ "playClock Between 5s and 15s",
                     playClockAtSnap < 5 ~ "playClock Less Than 5s",
    )
  )

plays_filtered <- plays_raw_data %>%
  filter(!is.na(playClockCategory), !is.na(passResult))

ggplot(plays_filtered, aes(x = playClockCategory)) +
  geom_bar(fill = "steelblue", color = "black") +
  labs(
    title = "Frequency of Play Clock Categories",
    x = "Play Clock Category",
    y = "Count"
  ) +
  theme_minimal()
```

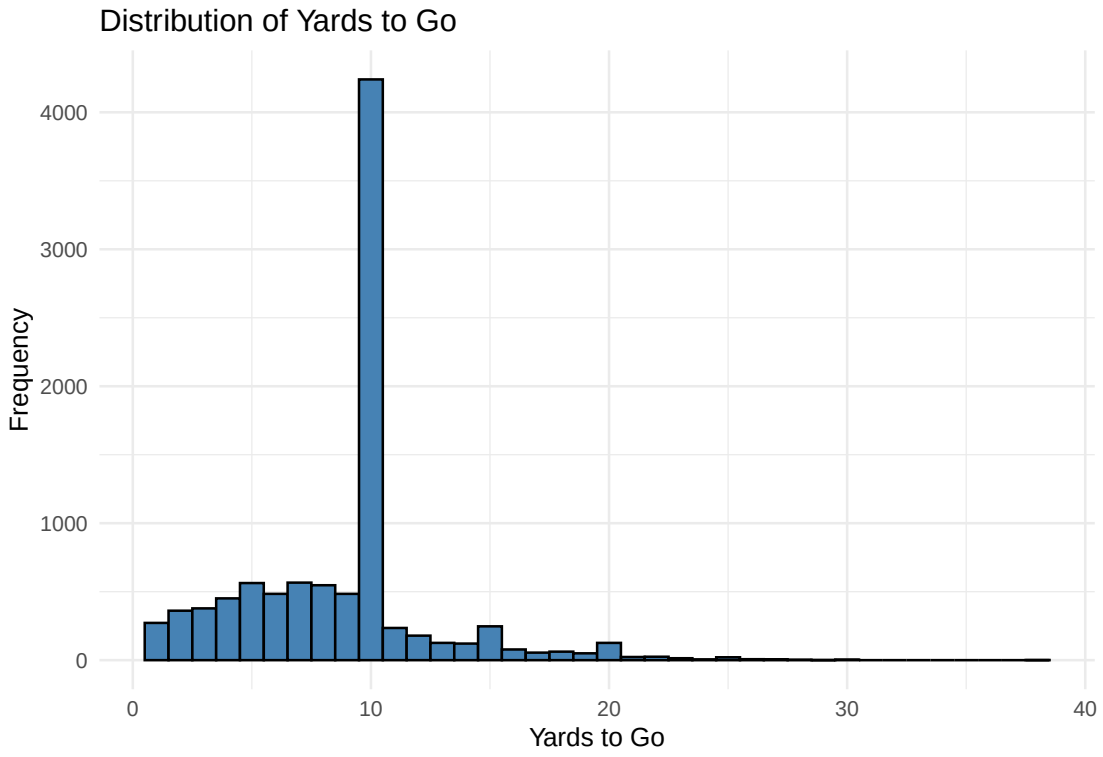


Based on the above graph, we see that the vast majority of the time the ball is snapped in the moderate play clock zone with between 5 and 15 seconds left on the play clock. This makes sense as most of the time teams want to make sure they have time to huddle up, relay the play to everyone and get set before snapping the ball. The second most frequent category is when the play clock is greater than 15 seconds. This category expresses when teams may be in a hurry-up offense where they might need to get down the field quicker or just want to get the defense on their heels. The least frequent category is when the play clock is less than 5 seconds. This category includes plays where teams may take a little longer to get set. This can be for a variety of reasons; teams may be sending players in motion before the snap or the quarterback may be making audibles or adjustments at the line of scrimmage. The significant parts that are relevant in the analysis below will be the more extreme buckets of less than 5 seconds and

more than 15 seconds.

yardsToGo (Quantitative):

```
ggplot(plays_filtered, aes(x = yardsToGo)) +
  geom_histogram(binwidth = 1, fill = "steelblue", color = "black") +
  labs(
    title = "Distribution of Yards to Go",
    x = "Yards to Go",
    y = "Frequency"
  ) +
  theme_minimal()
```



Yards to Go seems to be slightly normally distributed with a significant outlier with the vast majority of plays being at 10 yards to go. This makes sense as every

play defaults to 10 yards to go for a first down. The distribution is slightly right skewed with more plays with less than 10 yards to go.

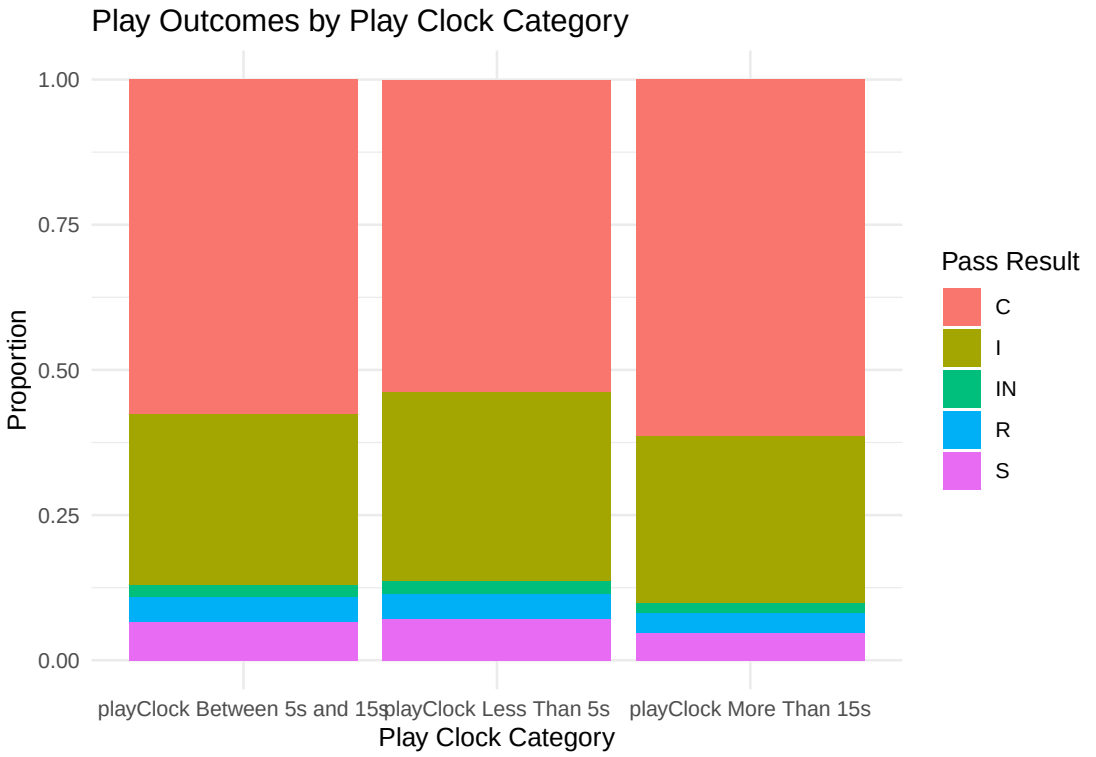
2.2.2 Relationships Between Variables

Before building predictive models, it's important to investigate potential relationships between variables that could impact play outcomes. In this section, we focus on the playClockAtSnap variable, which records the play clock value at the time of the snap, and explore its relationship with pass results and yards gained during a play.

The play clock is a critical element in football, as it dictates when a play must begin. A snap taken with a lower play clock may suggest hurried decision making, potentially leading to rushed plays or errors.

Below are some visuals and descriptions exploring the relationships between variables:

```
ggplot(plays_filtered, aes(x = playClockCategory, fill = passResult)) +
  geom_bar(position = "fill") +
  labs(
    title = "Play Outcomes by Play Clock Category",
    x = "Play Clock Category",
    y = "Proportion",
    fill = "Pass Result"
  ) +
  theme_minimal()
```



We can see that when the playClock is less than or equal to 5s (lower), there is a more varied distribution of pass results, including incomplete passes, sacks, and interceptions. On the other hand (higher), it looks like high play clock snaps tend to result in more complete passes, potentially indicating that the defense had less time to organize and execute a well prepared play.

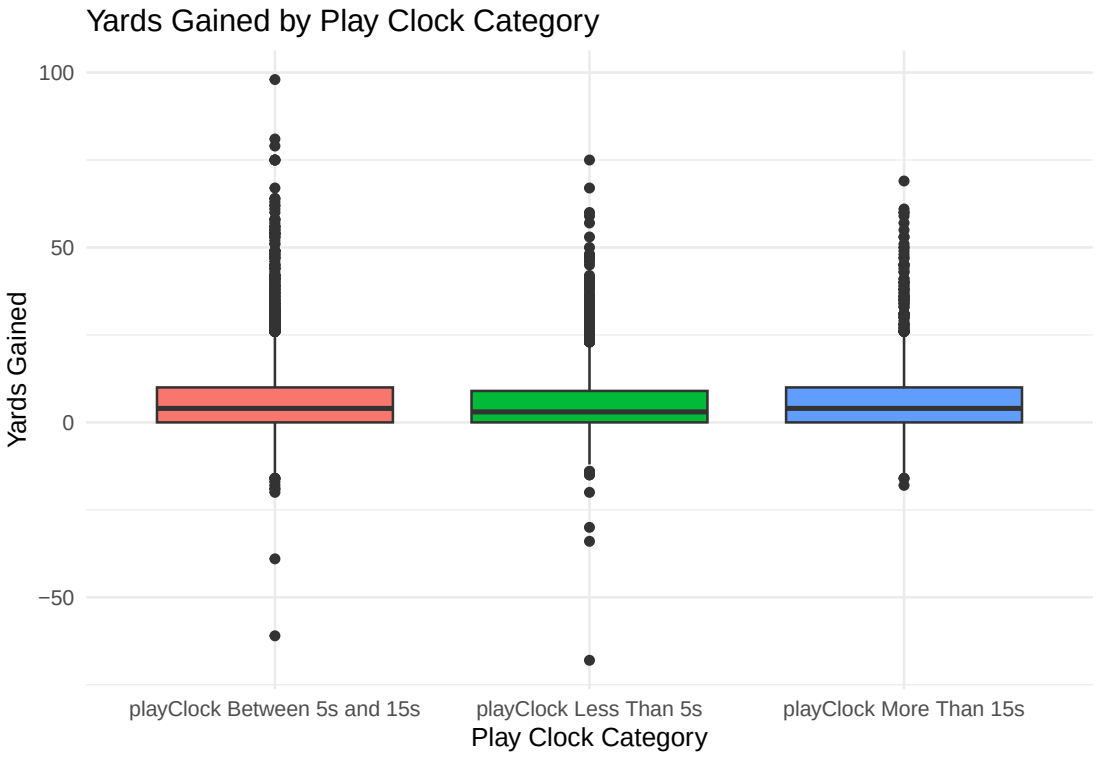
```
contingency_table <- table(plays_filtered$playClockCategory,
                           plays_filtered$passResult)

chi_sq_test <- chisq.test(contingency_table)
print(chi_sq_test)
```

```
##
##  Pearson's Chi-squared test
```

```
##
## data:  contingency_table
## X-squared = 29.539, df = 8, p-value = 0.0002549
```

And we can see here, the playClock is indeed statistically significant with a value of $0.0002549 < 0.05$. The next step would be to find a model to potentially predict results.

```
ggplot(plays_filtered, aes(x = playClockCategory, y = yardsGained,
                           fill = playClockCategory)) +
  geom_boxplot() +
  labs(
    title = "Yards Gained by Play Clock Category",
    x = "Play Clock Category",
    y = "Yards Gained"
  ) +
  theme_minimal() +
  theme(legend.position = "none")
```

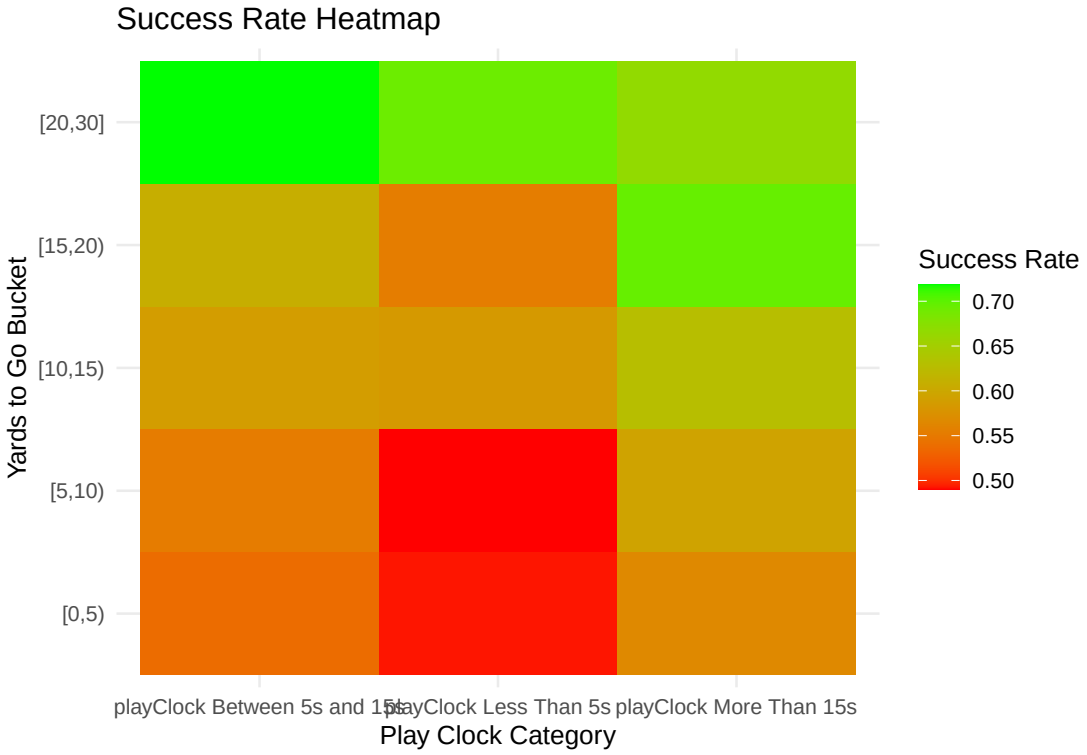


We also investigated the relationship that the play block at snap might have with the yards gained that play. However, as we can see the above visualization, there is no clear correlation between the two. That being said, we do notice that there seems to be slightly more variation in yards gained when the play clock is less than 15 seconds compared to when it is more than 15 seconds.

Further exploration of how the play clock affects other variables yields the following heat map. This heat map illustrates the play success rate based on the play clock and yards to go during that play.

```
heatmap_data <- plays_filtered %>%
  filter(!is.na(playClockCategory), !is.na(yardsToGo), yardsToGo >= 0, yardsToGo <
  mutate(
    yardsToGoBucket = cut(
      yardsToGo, breaks = c(0, 5, 10, 15, 20, 30),
```

```
      right = FALSE, include.lowest = TRUE
    )
  ) %>%
  group_by(playClockCategory, yardsToGoBucket) %>%
  summarize(success_rate = mean(passResult == "C", na.rm = TRUE), .groups = "drop"

ggplot(heatmap_data, aes(x = playClockCategory, y = yardsToGoBucket, fill = succes
  geom_tile() +
  labs(
    title = "Success Rate Heatmap",
    x = "Play Clock Category",
    y = "Yards to Go Bucket",
    fill = "Success Rate"
  ) +
  theme_minimal() +
  scale_fill_gradient(low = "red", high = "green")
```



The heatmap above shows the success rate of plays based on whether the pass was complete (passResult == "C"), categorized by play clock and yards-to-go buckets. Notably, the highest success rate occurs when the play clock exceeds 15 seconds, especially in long-yardage situations (20 to 30 yards), where the success rate is over 70%. This suggests that more time on the play clock leads to better play execution, even when facing tougher distances.

Interestingly, short-yardage situations (0 to 10 yards) do not yield higher success rates. In fact, success rates in the [0, 5] and [5, 10] yard buckets are lower than longer-yardage plays, indicating that defensive anticipation may play a role.

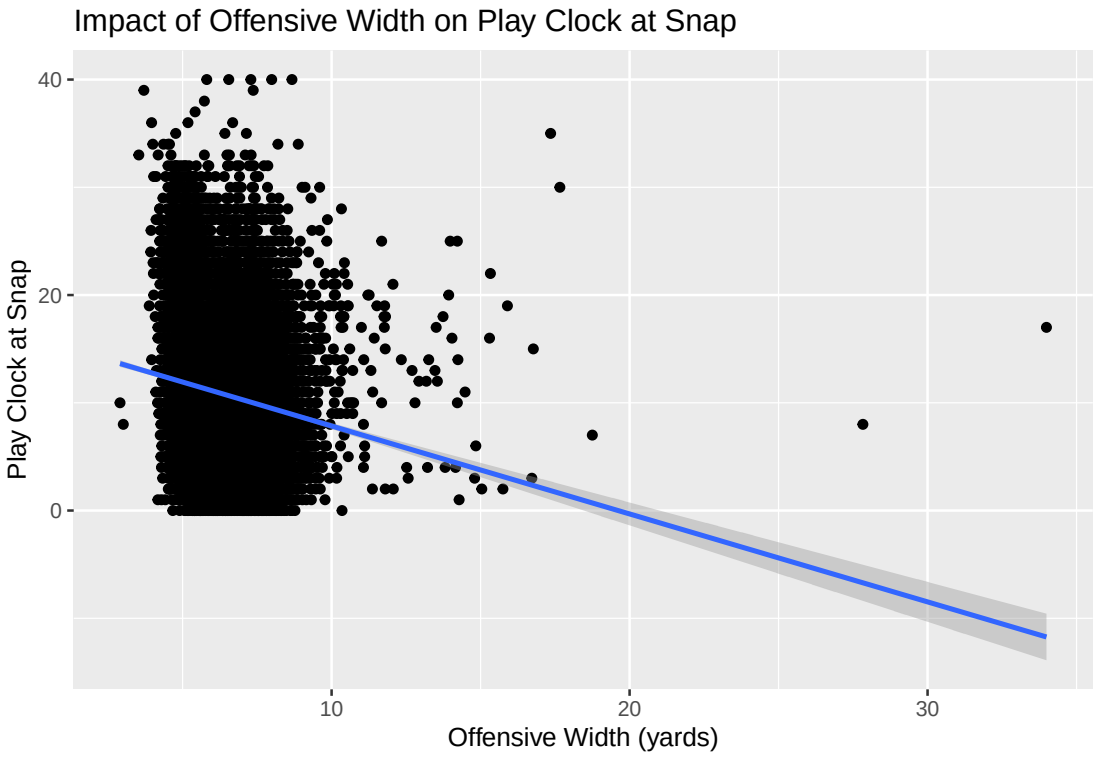
Further, we can explore the tracking data to see if offensive width, defensive width, and total offensive movement may have an impact on how fast the ball is snapped.

```r
# Offensive width x play clock at snap
ggplot(plays_raw_data, aes(x = offensive_width, y = playClockAtSnap)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(
    title = "Impact of Offensive Width on Play Clock at Snap",
    x = "Offensive Width (yards)",
    y = "Play Clock at Snap"
  )
```

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

```
## Warning: Removed 6 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 6 rows containing missing values or values outside the scale ra
## (`geom_point()`).
```

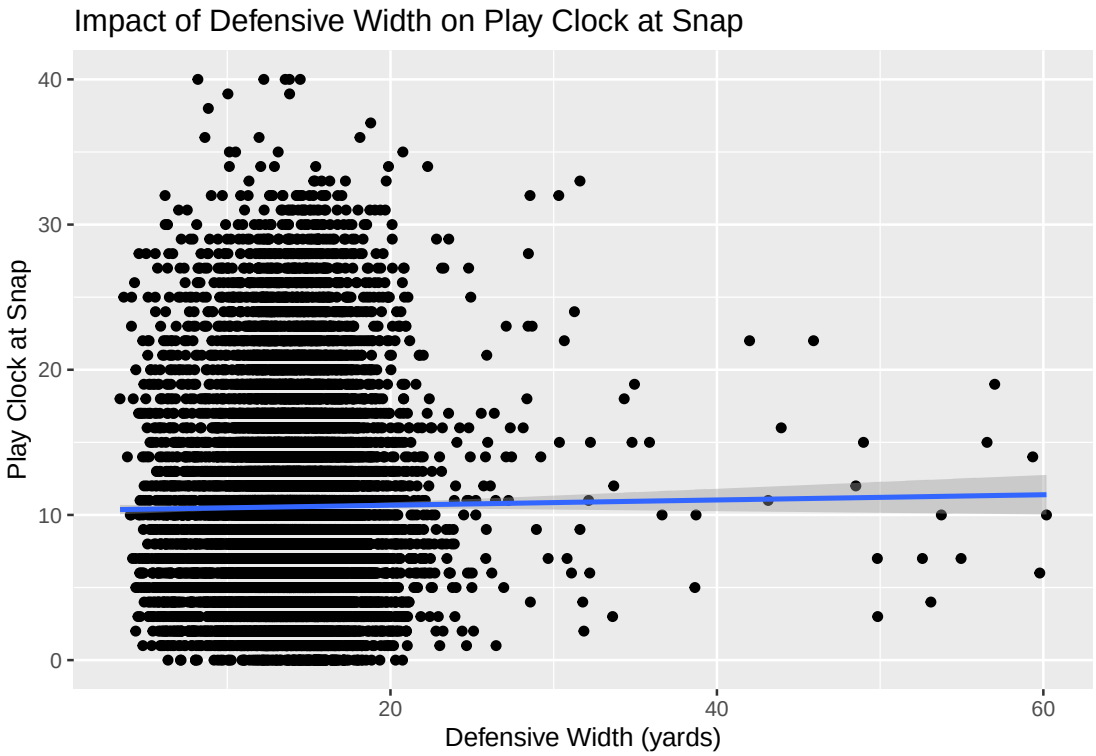



```
# Defensive width x play clock at snap
ggplot(plays_raw_data, aes(x = defensive_width, y = playClockAtSnap)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(
    title = "Impact of Defensive Width on Play Clock at Snap",
    x = "Defensive Width (yards)",
    y = "Play Clock at Snap"
  )
```

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

```
## Warning: Removed 6 rows containing non-finite outside the scale range (`stat_smo
```

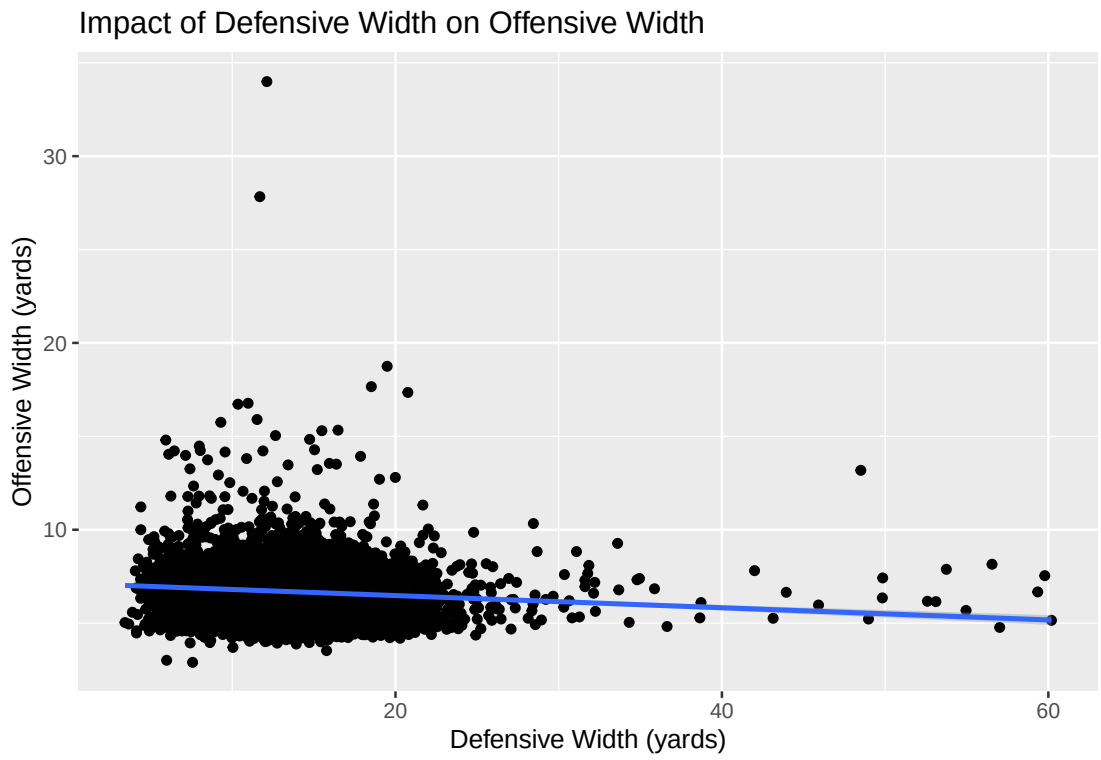
```
## Removed 6 rows containing missing values or values outside the scale range
## (`geom_point()`).
```




```
# Defensive width x offensive width
ggplot(plays_raw_data, aes(x = defensive_width, y = offensive_width)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(
    title = "Impact of Defensive Width on Offensive Width",
    x = "Defensive Width (yards)",
    y = "Offensive Width (yards)"
  )
```

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

```
## Warning: Removed 5 rows containing non-finite outside the scale range
## (`stat_smooth()`).

## Warning: Removed 5 rows containing missing values or values outside the scale ra
## (`geom_point()`).
```



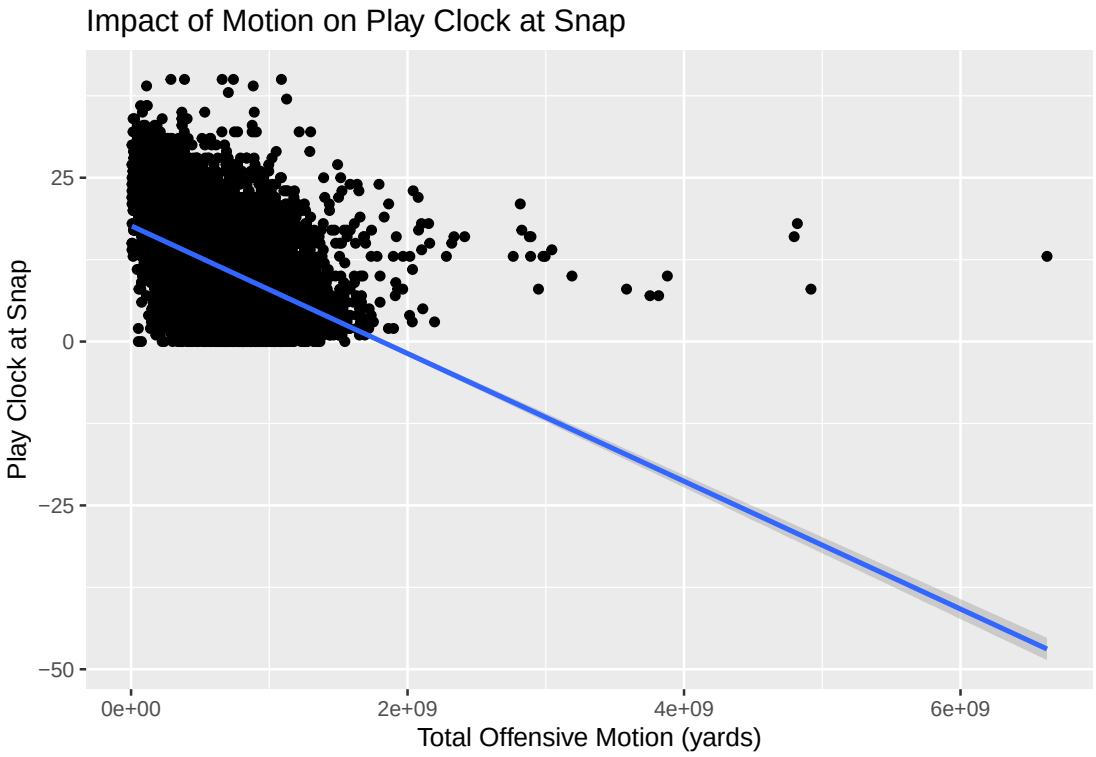

```
# Motion x play clock at snap
ggplot(plays_raw_data, aes(x = total_offensive_motion, y = playClockAtSnap)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(
    title = "Impact of Motion on Play Clock at Snap",
    x = "Total Offensive Motion (yards)",
```

```
    y = "Play Clock at Snap"
  )
```

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

```
## Warning: Removed 6 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 6 rows containing missing values or values outside the scale ra
## (`geom_point()`).
```



3 Modeling

3.1 Multinomial Model

Here for the predictive model, we implemented a multinomial regression model because the prediction outcome is nominal with multiple categories which is perfect for this kind of model. We removed the possibilities of R and S, scrambles and sacks, because the data did not record any results and were unused in our model. We discovered that playClockAtSnap, down, yardsToGo, passLength, pff_passCoverage, dropbackType are all correlated with passResult to some degree. The model predicts the pass result using these variables as well as some of the player tracking data from above. To train the model, we have partitioned the data to a train_data and a test_data in which we will train the model on the train_data and test the model's success on the test_data.
A common ratio for training and testing is 80 to 20 percent of the whole model which is used below. The code block below splits the plays_filtered data and trains the model named passResult_model. Below are the results:

```
library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(nnet)

set.seed(123)

plays_filtered <- plays_filtered %>%
  filter(!passResult %in% c("R", "S"))

plays_filtered$passResult <- factor(plays_filtered$passResult)

train_index <- createDataPartition(plays_filtered$passResult, p = 0.80, list = FAL
train_data <- plays_filtered[train_index, ]
test_data <- plays_filtered[-train_index, ]

train_data$passResult <- droplevels(train_data$passResult)
test_data$passResult <- droplevels(test_data$passResult)

passResult_model <- multinom(
  passResult ~ playClockAtSnap + down + total_offensive_motion + (offensive_width
    pff_passCoverage + dropbackType,
  data = train_data
)
```

```
## # weights:  93 (60 variable)
## initial  value 7650.735978
## iter  10 value 4941.870338
## iter  20 value 4827.366392
## iter  30 value 4754.176568
```

```
## iter  40 value 4716.725860
## iter  50 value 4711.545877
## iter  60 value 4710.287511
## iter  70 value 4710.178983
## final  value 4710.178344
## converged
```

If we were to look at the summary of the mode, it would give a variety of statistics including coefficients and standard errors of all the different variables and relative subcategories. Instead of analyzing these statistics, we analyze the contingency table below that summarizes the success of the model on the test_data:

```
predictions <- predict(passResult_model, test_data)

predictions <- as.factor(predictions)

library(caret)

conf_matrix <- confusionMatrix(predictions, as.factor(test_data$passResult))

print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    C    I   IN
##          C  1027  399   27
##          I    97  174   11
```

```
##         IN    0    0    0
##
## Overall Statistics
##
##                Accuracy : 0.6922
##                  95% CI : (0.6699, 0.7139)
##     No Information Rate : 0.6478
##     P-Value [Acc > NIR] : 5.162e-05
##
##                   Kappa : 0.2377
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: C Class: I Class: IN
## Sensitivity            0.9137   0.3037    0.0000
## Specificity            0.3028   0.9071    1.0000
## Pos Pred Value         0.7068   0.6170       NaN
## Neg Pred Value         0.6560   0.7254    0.9781
## Prevalence             0.6478   0.3303    0.0219
## Detection Rate         0.5919   0.1003    0.0000
## Detection Prevalence   0.8375   0.1625    0.0000
## Balanced Accuracy      0.6082   0.6054    0.5000
```

The results of the model are as follows from the confusion matrix: - 1027 plays were correctly predicted as Complete Passes.

- 174 plays were correctly predicted as Incomplete Passes. - 399 plays were

misclassified as complete when they were actually incomplete.
- 27 plays were misclassified as complete when they were actually interceptions.
- 97 plays were misclassified as incomplete when they were actually complete.
- The model did failed to predict any results as I

When we look at the overall statistics we can see that the accuracy of the model is almost 70% at approximately 69.22% accuracy. This is an improvement to the No Information rate of 64.78% which would be the accuracy if the model predicted passResult purely off of chance. Furthermore, we can see the confidence interval says with 95% confidence that the true prediction accuracy is between 66.99% and and 71.39%.

3.2 Neural Network Model

```
library(torch)
library(dplyr)
library(ggplot2)
library(caret)

# ------------------------------------------------------
# 1) Data Preparation
# ------------------------------------------------------

# Filter data to relevant passResult values and drop rows with NA
# Also filter out rows where total_offensive_motion, offensive_width, or defensiv
plays_filtered <- plays_filtered %>%
  filter(passResult %in% c("C", "I", "IN")) %>%
```

```
  filter(
    !is.na(playClockAtSnap),
    !is.na(down),
    !is.na(yardsToGo),
    !is.na(passLength),
    !is.na(pff_passCoverage),
    !is.na(dropbackType),
    !is.na(passResult),
    !is.na(total_offensive_motion),
    !is.na(offensive_width),
    !is.na(defensive_width)
  ) %>%
  # Create a new column for ratio: offensive_width / defensive_width
  mutate(width_ratio = offensive_width / defensive_width)

# Re-factor passResult in the desired order: "C" -> 1, "I" -> 2, "IN" -> 3
plays_filtered$passResult <- factor(plays_filtered$passResult, levels = c("C", "I"
plays_filtered$passResult <- as.numeric(plays_filtered$passResult)

# ------------------------------------------------------
# 2) Create Dummy Columns (pff_passCoverage + dropbackType)
# ------------------------------------------------------
plays_filtered$pff_passCoverage <- factor(plays_filtered$pff_passCoverage)
plays_filtered$dropbackType     <- factor(plays_filtered$dropbackType)

categorical_dummies <- model.matrix(
  ~ pff_passCoverage + dropbackType - 1,
```

```
  data = plays_filtered
)

# ----------------------------------------------------
# 3) Scale Numeric Columns
# ----------------------------------------------------
# Include total_offensive_motion + width_ratio in numeric columns
num_cols <- c("playClockAtSnap", "down", "yardsToGo", "passLength",
              "total_offensive_motion", "width_ratio")

plays_filtered[num_cols] <- scale(plays_filtered[num_cols])

# ----------------------------------------------------
# 4) Build Final Feature Matrix X
# ----------------------------------------------------
X <- cbind(
  # The scaled numeric columns
  plays_filtered[, num_cols],
  # The dummy columns
  categorical_dummies
)
X <- as.matrix(X)  # convert to numeric matrix

# ----------------------------------------------------
# 5) Define Target
# ----------------------------------------------------
y <- plays_filtered$passResult
```

```r
# ------------------------------------------------------
# Convert R objects to Torch Tensors
# ------------------------------------------------------
X_tensor <- torch_tensor(X, dtype = torch_float())
y_tensor <- torch_tensor(y, dtype = torch_long())

# ------------------------------------------------------
# 6) Train-Test Split
# ------------------------------------------------------
set.seed(1)
train_indices <- sample(seq_len(nrow(X)), size = 0.8 * nrow(X))
test_indices  <- setdiff(seq_len(nrow(X)), train_indices)

X_train <- X_tensor[train_indices, ]
y_train <- y_tensor[train_indices]
X_test  <- X_tensor[test_indices, ]
y_test  <- y_tensor[test_indices]

# ------------------------------------------------------
# 7) Define a Neural Network Model
# ------------------------------------------------------
model <- nn_module(
  initialize = function(num_features) {
    self$fc1    <- nn_linear(num_features, 8)
    self$drop1  <- nn_dropout(p = 0.3)
    self$fc2    <- nn_linear(8, 8)
```

```
    self$drop2  <- nn_dropout(p = 0.3)
    self$fc3    <- nn_linear(8, 8)
    self$output <- nn_linear(8, 3)
  },
  forward = function(x) {
    x <- torch_relu(self$fc1(x))
    x <- self$drop1(x)
    x <- torch_relu(self$fc2(x))
    x <- self$drop2(x)
    x <- torch_relu(self$fc3(x))
    x <- self$output(x)
    x
  }
)

torch_model <- model(ncol(X))

# ------------------------------------------------------
# 8) Define Loss Function (unweighted) & Optimizer
# ------------------------------------------------------
loss_fn  <- nn_cross_entropy_loss()
optimizer <- optim_adam(torch_model$parameters, lr = 0.0005, weight_decay = 1e-4)

# ------------------------------------------------------
# 9) Training Loop
# ------------------------------------------------------
num_epochs <- 50
```

```
batch_size <- 50

train_losses <- numeric(num_epochs)
test_losses  <- numeric(num_epochs)

for (epoch in seq_len(num_epochs)) {

  torch_model$train()
  total_train_loss   <- 0
  shuffled_indices   <- sample(seq_len(nrow(X_train)))
  num_batches        <- ceiling(nrow(X_train) / batch_size)

  for (i in seq_len(num_batches)) {
    start_idx <- (i - 1) * batch_size + 1
    end_idx   <- min(i * batch_size, nrow(X_train))
    batch_ids <- shuffled_indices[start_idx:end_idx]

    batch_X <- X_train[batch_ids, ]
    batch_y <- y_train[batch_ids]

    optimizer$zero_grad()
    predictions <- torch_model(batch_X)
    loss <- loss_fn(predictions, batch_y)
    loss$backward()
    optimizer$step()

    total_train_loss <- total_train_loss + loss$item()
```

```
  }

  # Evaluate on test set
  torch_model$eval()
  with_no_grad({
    test_predictions <- torch_model(X_test)
    test_loss_val <- loss_fn(test_predictions, y_test)$item()
  })

  avg_train_loss <- total_train_loss / num_batches
  train_losses[epoch] <- avg_train_loss
  test_losses[epoch]  <- test_loss_val

  if (epoch %% 10 == 0) {
    cat(sprintf("Epoch %d | Train Loss: %.4f | Test Loss: %.4f\n",
                epoch, avg_train_loss, test_loss_val))
  }
}
```

```
## Epoch 10 | Train Loss: 0.7113 | Test Loss: 0.7071
## Epoch 20 | Train Loss: 0.6945 | Test Loss: 0.6915
## Epoch 30 | Train Loss: 0.6864 | Test Loss: 0.6893
## Epoch 40 | Train Loss: 0.6834 | Test Loss: 0.6881
## Epoch 50 | Train Loss: 0.6821 | Test Loss: 0.6881
```

```
# ------------------------------------------------------
# 10) Evaluate Overall Accuracy
```

```r
# ------------------------------------------------------
torch_model$eval()
with_no_grad({
  full_predictions  <- torch_model(X_tensor)
  predicted_classes <- full_predictions$argmax(dim = 2)
  correct           <- (predicted_classes == y_tensor)$to(dtype = torch_float())
  overall_accuracy  <- torch_mean(correct)$item()
})
cat(sprintf("Overall Classification Accuracy: %.2f%%\n", 100 * overall_accuracy))
```

```
## Overall Classification Accuracy: 68.73%
```

```r
# Create a data frame of true vs. predicted results
results_df <- data.frame(
  TruePassResult      = as_array(y_tensor),
  PredictedPassResult = as_array(predicted_classes)
)

# ------------------------------------------------------
# 11) Evaluate on Test Set & Build Confusion Matrix
# ------------------------------------------------------
torch_model$eval()
with_no_grad({
  test_preds <- torch_model(X_test)
  predicted_test_classes <- test_preds$argmax(dim = 2)
})
```

```r
true_test <- as_array(y_test)
pred_test <- as_array(predicted_test_classes)

cm <- confusionMatrix(
  factor(pred_test, levels = c(1,2,3)),
  factor(true_test, levels = c(1,2,3))
)
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 983 380  22
##          2 125 210  20
##          3   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6856
##                  95% CI : (0.6632, 0.7074)
##     No Information Rate : 0.6368
##     P-Value [Acc > NIR] : 1.046e-05
##
##                   Kappa : 0.2585
##
##  Mcnemar's Test P-Value : < 2.2e-16
```

```
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8872   0.3559  0.00000
## Specificity            0.3639   0.8739  1.00000
## Pos Pred Value         0.7097   0.5915      NaN
## Neg Pred Value         0.6479   0.7256  0.97586
## Prevalence             0.6368   0.3391  0.02414
## Detection Rate         0.5649   0.1207  0.00000
## Detection Prevalence   0.7960   0.2040  0.00000
## Balanced Accuracy      0.6256   0.6149  0.50000
```

4 Discussion

4.1 Play Clock

We found that higher play clock (i.e. quicker snaps) correlates with better outcomes for the offensive teams, particularly in long yardage situations. We suppose that this is because it gives the defense less time to get set and read the offensive team. Additionally, we found that more offensive motion correlates with lower play clock values, which follows logically given that motion would delay the snap time.

4.2 Model Insights

The multinomial model predicted a pass' result using the play clock at snap, down, offensive motion, offensive width, defensive width, yards to go for first down, pass

length, pass coverage and dropback type. This model was able to successfully predict the pass result 69.2% of the time.

The neural network predicted a pass' result using the play clock at snap, down, yards to go for first down, pass length, pass coverage and dropback type. This model had a fairly comparable prediction accuracy of 68.7%.

While these models aren't perfect, they do offer some relatively accurate insights. These models can be employed by teams to model game situations to find the ideal time to snap the ball each down (i.e. hurry-up offense vs taking longer).

4.3 Future Work

In the future, we want to optimize the models to improve prediction accuracy. We could also include rushing plays to potentially help predict the success of more than just passing plays.

It would also be interesting to explore specific situations to come up with some situational rules on when to snap the ball in the play clock.