Statto Data Analysis – Uchicago Fission at Regionals 2021

Hiro Schmidt

12/20/2021

Goal:

Calculate summary data for each possession that can be then cumulated over games or tournament. We achieve this by breaking the full throwing data.frames into separate data.frames of each possession.

We can then assign success metrics for each possession (yards (numeric), score (boolean)) and assign these metrics to the states at each throw.

Object structure:

Tournament: Tournament summary Vector of games

Game: Game summary Vector of possessions

Possession: Possession summary Dataframe of throws

```
setClass("Tournament", slots = list(summary = "data.frame", games = "list"))

setClass("Game", slots = list(summary = "data.frame", possessions = "list"))

setClass("Possession", slots = list(summary = "data.frame", throws = "data.frame"))
```

Raw data processing

Path should be the path to the Statto data folder. Automated parsing of files into a tournament object requires a folder for each game, containing the different .csv files for that game, including the Passes and Possessions files.

```
path = 'C:/Users/hiros/Desktop/Storage/documents/fission_data/Fall Regionals Stats'

## given path to tournament data folder return a list. Each index of list represents a game, and has th
parse_directory = function(path) {
  dirs = list.dirs(path)
  game_list = list()
  for (i in dirs) {
    files = list.files(i,full.names = TRUE)

    passes_file = grep('Passes',files,value=TRUE)

    if(!is_empty(passes_file)) {
      passes = read.csv(passes_file)
      possessions_file = grep('Possessions',files,value=TRUE)
      possessions = read.csv(possessions_file)
```

```
      game = list(passes,possessions)
      game_list = append(game_list,list(game))
    }
  }
  return(game_list)
}

## takes in raw game-passes data.frame and returns Game object
game_split = function(passes) {

  ## initialize list of possessions
  poss = list()

  ## rolling index of last turnover
  turn_index = 0

  ## iterate over rows of data.frame and split on each turnover
  for (i in 1:nrow(passes)) {
    if (passes[i,]$Turnover. == 1 | passes[i,]$Throw.to.endzone. == 1) {

      ## generate possession object
      poss_obj = new("Possession",
                     throws = passes[(turn_index + 1):i,])
      poss = append(poss,poss_obj)
      turn_index = i
    }
  }
  ## generate game object
  game_obj = new("Game", possessions = poss)
  return(game_obj)
}

## generates summary for a possession object, uses relevant game-possessions data.frame
poss_summary = function(poss_obj, possessions) {
  #print(poss_obj)
  ## retrieve possession index from object
  point_num = poss_obj@throws[1,]$Point
  poss_num = poss_obj@throws[1,]$Possession
  #print(point_num)
  #print(poss_num)

  ## match rows of possessions df to possession index

  poss_data = suppressWarnings(
    subset(possessions,Point == point_num & Possession == poss_num)
  )

  return(poss_data)
}

## given game-passes and game-possessions data.frames generates Game object with possession objects
game_main = function(passes,possessions) {
  game_obj = game_split(passes)
```

```
  for (i in 1:length(game_obj@possessions)) {
    game_obj@possessions[[i]]@summary = poss_summary(game_obj@possessions[[i]], possessions)
  }
  return(game_obj)
}

## given tournament folder, generate base tournament object
tournament_main = function(path) {
  data = parse_directory(path)

  game = list()
  for (i in data) {
    passes = i[[1]]
    possessions = i[[2]]
    game_obj = game_main(passes,possessions)
    game = append(game, game_obj)
  }
  tournament_obj = new("Tournament", games = game)
}

#IU_game = game_main(IU_pass,IU_poss)

regionals_obj = tournament_main(path)
```

Analysis

We can now run methods on the game object.

Filtering methods

First we create some methods for separating data into categories

Generic data splitting method

```
## Takes in a game object, variable, location of that variable (e.g. Passes or Possessions), and a set
## Currently only set up to handle Possessions
game_split = function(game, var, var_loc, values) {

  ## Create list of lists. Each nested list will contain the possessions at a different value
  ll = rep(0,length(values))
  ll = lapply(ll,function(x) {return(list())})

  for (possession in game@possessions) {

    ## Possessions table handling (summary attribute of Possessions)
    if (var_loc == "Possessions") {

      ## iterate over values and match possessions to the value of the given variable

      for (i in 1:length(values)) {
        if (possession@summary$var == values[i]) {
          ll[[i]] = append(ll[[i]], possession)
```

```
        }
      }
    }

    ## Passes table handling (throws attribute of Possessions)
#    if (var_loc == "Passes") {
#
#      for (i in 1:length(values)) {
#
#      }
#    }
  }

  ## Convert list of lists to a list of Games
  ll = lapply(ll, function(x) {

  })

}


## Takes in a tournament object, variable, location of that variable (e.g. Passes or Possessions), and
tournament_split = function(tournament, var, var_loc, values) {

  ## generate list variable names for each given value
  values_var_names = lapply(x,function(x) {
    assign(paste("var_",x),list())
  })

  for (game in tournament@games) {

  }
}
```

Split data into O-line vs. D-line

```
## splits a game object into 2 game objects, one for O, one for D
game_O_split = function(game){
  game_O_possessions = list()
  game_D_possessions = list()

  for (possession in game@possessions) {
    if (possession@summary$Started.point.on.offense. == 1) {
      game_O_possessions = append(game_O_possessions,possession)
    } else {
      game_D_possessions = append(game_D_possessions,possession)

    }
  }

  game_O = new("Game",possessions=game_O_possessions)
  game_D = new("Game",possessions=game_D_possessions)
```

```
  return(list(game_O,game_D))
}

## splits a tournament object into 2 tournament objects, one for O-line, one for D-line
tournament_O_split = function(tournament) {
  tournament_O_games = list()
  tournament_D_games = list()

  for (game in tournament@games) {
    games = game_O_split(game)
    tournament_O_games = append(tournament_O_games,games[[1]])
    tournament_D_games = append(tournament_D_games,games[[2]])
  }
  tournament_O = new("Tournament",games = tournament_O_games)
  tournament_D = new("Tournament",games = tournament_D_games)

  return(list(tournament_O,tournament_D))
}

regionals_O_split = tournament_O_split(regionals_obj)
regionals_O = regionals_O_split[[1]]
regionals_D = regionals_O_split[[2]]
```

Split data into huck possessions vs. no-huck-possessions

```
## splits a game object into 2 game objects, one for hucks, one for no huck
game_huck_split = function(game){
  game_huck_possessions = list()
  game_no_huck_possessions = list()

  for (possession in game@possessions) {
    ## check if possession was eligible for huck
    max_dist = max(possession@throws$Start.Y..0....1...back.of.opponent.endzone....back.of.own.endzone.)
    if (max_dist >= 26) {
      huck = 0
      for (i in 1:nrow(possession@throws)) {
        row = possession@throws[i,]
        if (row$Huck. == 1) {
          huck = 1
        }
      }
      if (huck == 1) {
        game_huck_possessions = append(game_huck_possessions,possession)
      } else {
        game_no_huck_possessions = append(game_no_huck_possessions,possession)

      }
    }
  }

  game_huck = new("Game",possessions=game_huck_possessions)
  game_no_huck = new("Game",possessions=game_no_huck_possessions)
```

```
  return(list(game_huck,game_no_huck))
}

## splits a tournament object into 2 tournament objects, one for huck possessions, one for no huck poss
tournament_huck_split = function(tournament) {
  tournament_huck_games = list()
  tournament_no_huck_games = list()

  for (game in tournament@games) {
    games = game_huck_split(game)
    tournament_huck_games = append(tournament_huck_games,games[[1]])
    tournament_no_huck_games = append(tournament_no_huck_games,games[[2]])
  }
  tournament_huck = new("Tournament",games = tournament_huck_games)
  tournament_no_huck = new("Tournament",games = tournament_no_huck_games)

  return(list(tournament_huck,tournament_no_huck))
}

regionals_huck_split = tournament_huck_split(regionals_obj)
regionals_huck = regionals_huck_split[[1]]
regionals_no_huck = regionals_huck_split[[2]]
```

Success rate calculations

Generic tournament success rate

We first create a generic method to return the success rate of possessions in a given tournament object.

```
## given a game object return its score total and possession total
game_score_data = function(game) {
  possession_count = 0
  score_count = 0

  for (possession in game@possessions) {
    possession_count = possession_count + 1
    if (possession@summary$Scored. == 1) {
      score_count = score_count + 1
    }
  }

  return(list(possession_count,score_count))
}

## given a tournament object return its score rate, and number of opportunities
tournament_score_rate = function(tournament) {
  possession_count = 0
  score_count = 0

  for (game in tournament@games) {
    game_counts = game_score_data(game)
    possession_count = possession_count + game_counts[[1]]
    score_count = score_count + game_counts[[2]]
  }
```

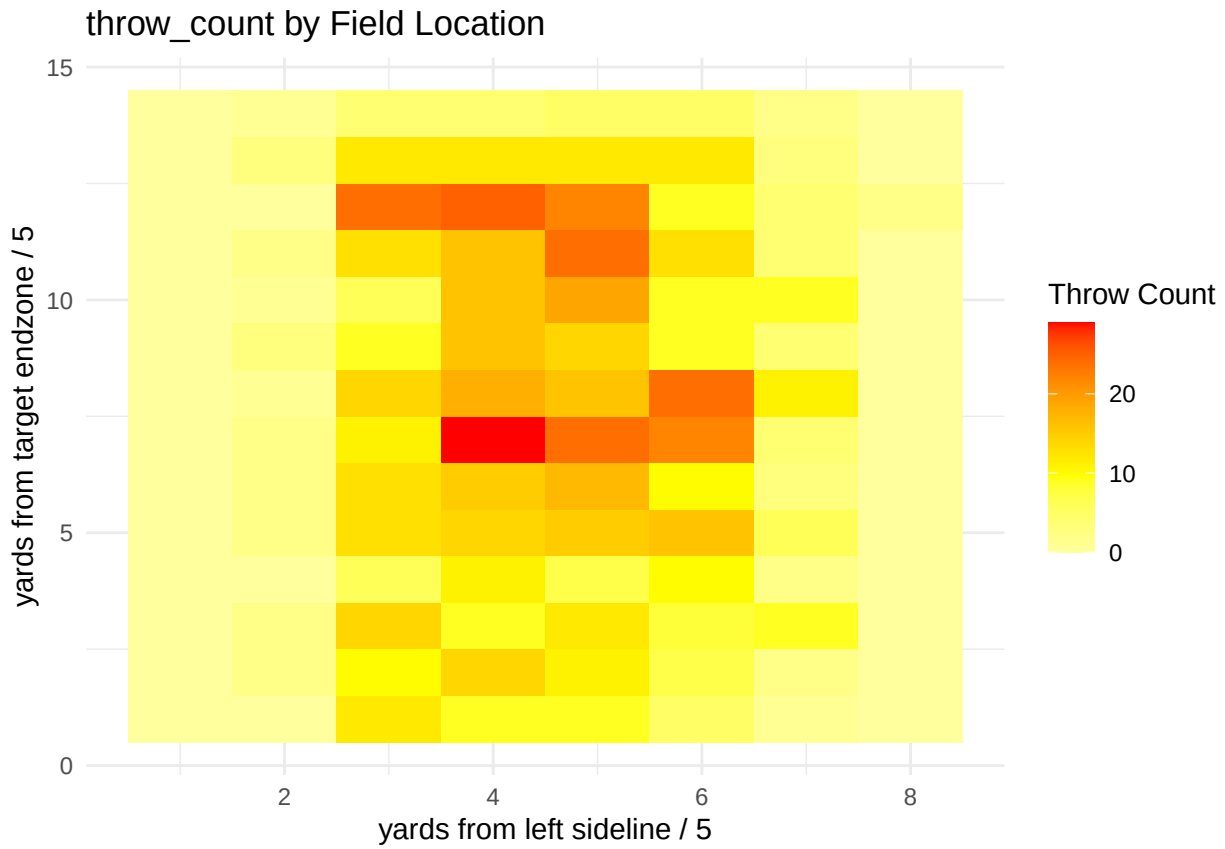
```
  return(list(score_count/possession_count,possession_count))
}
```

Scoring rate by throwing location

Here we break the field down into a matrix of bins and calculate the rate at which throws originating from each zone on the field eventually result in a goal at the end of the possession.

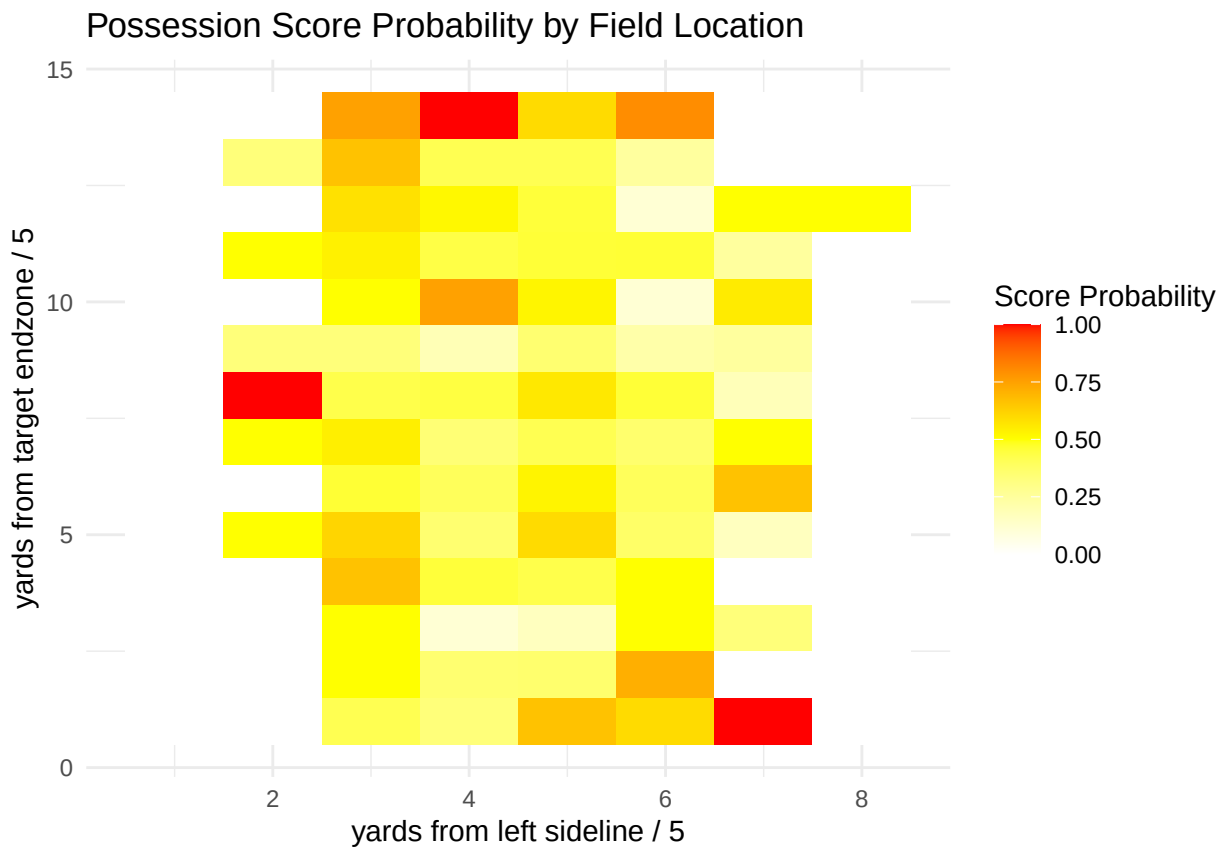
```
## generate 2 matrices based on field location of throws. 1st matrix is number of throws from each loc.
game_loc_data = function(game_obj, row=14, col=8) {

  x_axis = seq(0,1,1/row)
  y_axis = seq(0,1,1/col)

  ## matrix of number of throws from each index
  throw_count = matrix(0,row,col)

  ## matrix of number of scores on possessions from each index
  score_count = matrix(0,row,col)

  ## iterate over possessions
  for (poss in game_obj@possessions) {

    ## iterate over throws
    for (i in 1:nrow(poss@throws)) {

      throw = poss@throws[i,]

      x_pos = throw$Start.X..0....1...left.sideline....right.sideline.
      y_pos = throw$Start.Y..0....1...back.of.opponent.endzone....back.of.own.endzone.
      ## calculate matrix index of throw
      for (i in 1:row) {

        if ((x_pos >= ((i-1)/row)) & (x_pos < (i/row))) {
          x_index = i
        }
      }

      for (j in 1:col) {
        if (y_pos >= (j-1)/col & y_pos < j/col) {
          y_index = j
        }
      }
      ## update matrices
      throw_count[x_index,y_index] = throw_count[x_index,y_index] + 1
      scored = poss@summary$Scored.
      score_count[x_index,y_index] = score_count[x_index,y_index] + scored
    }
  }
  return(list(throw_count,score_count))
}

## generates score count and throw count matrices for tournament
tournament_loc_data = function(tournament, row=14, col=8) {
```

```
  throw_count = matrix(0,row,col)
  score_count = matrix(0,row,col)

  for (game in tournament@games) {
    game_counts = game_loc_data(game, row, col)

    throw_count = throw_count + game_counts[[1]]
    score_count = score_count + game_counts[[2]]

  }
  return(list(throw_count,score_count))
}

## generates heat-map and score rate matrix given 2 matrices as created in game_loc_data
score_rate_map = function(throw_count, score_count) {
  ## combined matrices into probability matrix
  score_rate = score_count/throw_count
  score_rate[is.na(score_rate)] = 0
  melted_score_rate = melt(score_rate)
  print(
    ggplot(data = melted_score_rate, aes(y=Var1, x=Var2, fill=value)) +
      labs(x="yards from left sideline / 5",y="yards from target endzone / 5",
           title="Possession Score Probability by Field Location") +
      geom_tile() +
      scale_fill_gradient2(low = "white", high = "red", mid = "yellow",
      midpoint = .5, limit = c(0,1), space = "Lab",
      name="Score Probability") +
      theme_minimal()
  )
  return(score_rate)
}

## generates heat-map for number of possessions for each zone
throw_count_map = function(throw_count) {
  melted_throw_count = melt(throw_count)
  print(
    ggplot(data = melted_throw_count, aes(y=Var1, x=Var2, fill=value)) +
      labs(x="yards from left sideline / 5",y="yards from target endzone / 5",
           title="throw_count by Field Location") +
      geom_tile() +
      scale_fill_gradient2(low = "white", high = "red", mid = "yellow",
      midpoint = max(throw_count)/3, limit = c(0,max(throw_count)), space = "Lab",
      name="Throw Count") +
      theme_minimal()
  )
  return(throw_count)
}

## running on full data
regionals_loc_data = tournament_loc_data(regionals_obj)
throw_count_map(regionals_loc_data[[1]])
```



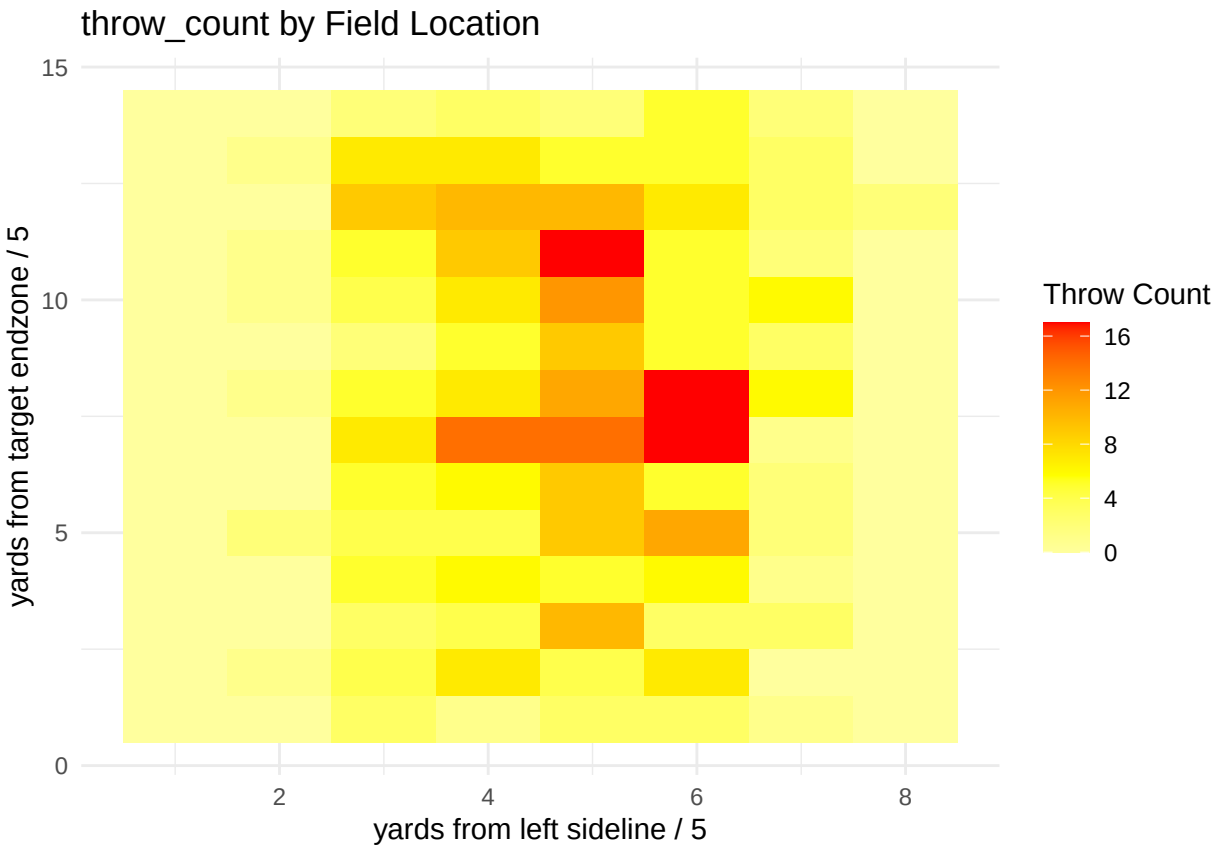

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
##  [1,]    0    0   12    9    9    5    1    0
##  [2,]    0    2   10   14   11    7    2    0
##  [3,]    0    2   14    9   12    8    9    0
##  [4,]    0    0    6   11    7   10    2    0
##  [5,]    0    2   13   14   15   16    6    0
##  [6,]    0    2   13   15   17   10    3    0
##  [7,]    0    2   11   29   24   22    4    0
##  [8,]    0    1   14   18   16   24   11    0
##  [9,]    0    3    9   16   14    9    4    0
## [10,]    0    1    6   16   19    9    9    0
## [11,]    0    2   13   16   24   13    4    0
## [12,]    0    0   24   25   22    9    4    2
## [13,]    0    3   12   12   12   12    3    0
## [14,]    0    1    4    4    5    5    2    0
```

```
score_rate_map(regionals_loc_data[[1]],regionals_loc_data[[2]])
```



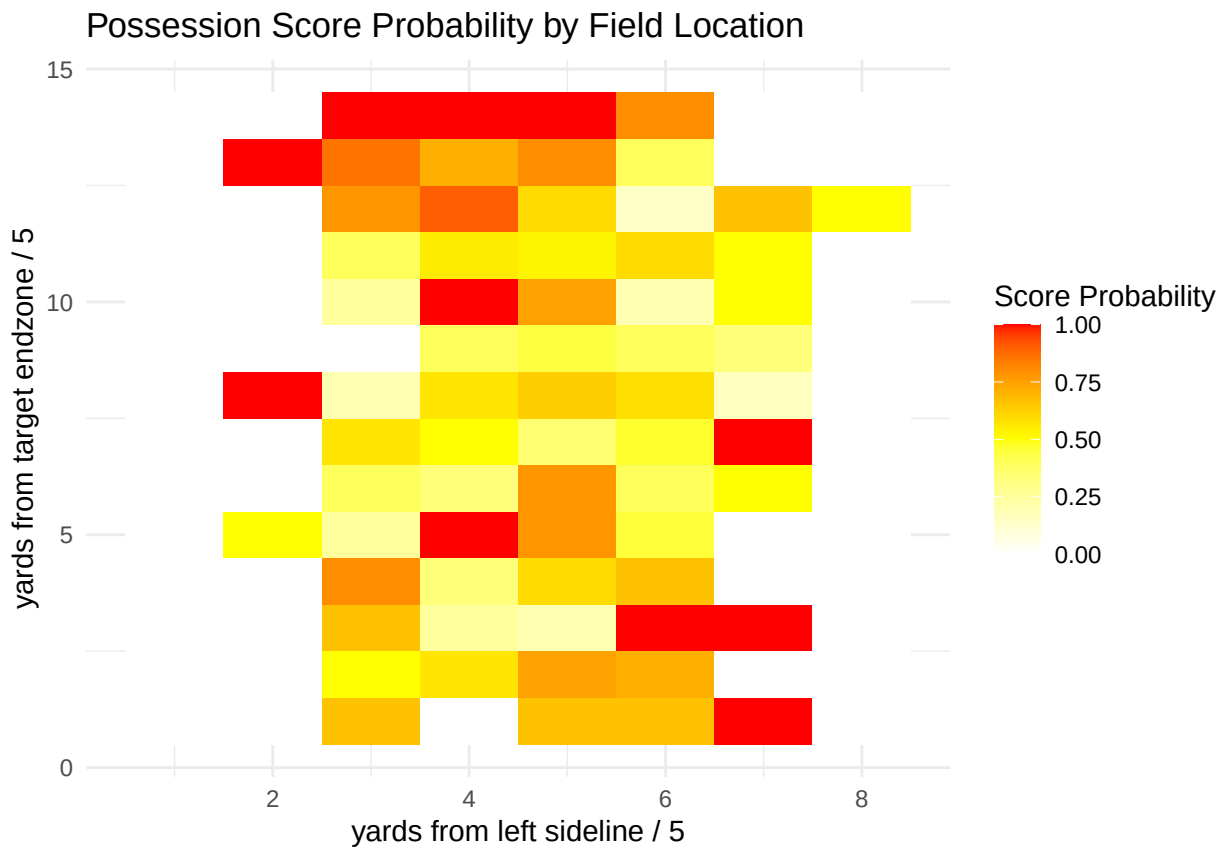

```
##       [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7] [,8]
##  [1,]    0 0.0000000 0.4166667 0.3333333 0.6666667 0.6000000 1.0000000  0.0
##  [2,]    0 0.0000000 0.5000000 0.3571429 0.3636364 0.7142857 0.0000000  0.0
##  [3,]    0 0.0000000 0.5000000 0.1111111 0.1666667 0.5000000 0.3333333  0.0
##  [4,]    0 0.0000000 0.6666667 0.4545455 0.4285714 0.5000000 0.0000000  0.0
##  [5,]    0 0.5000000 0.6153846 0.3571429 0.6000000 0.3750000 0.1666667  0.0
##  [6,]    0 0.0000000 0.4615385 0.4000000 0.5294118 0.4000000 0.6666667  0.0
##  [7,]    0 0.5000000 0.5454545 0.3448276 0.4166667 0.3636364 0.5000000  0.0
##  [8,]    0 1.0000000 0.4285714 0.4444444 0.5625000 0.4583333 0.1818182  0.0
##  [9,]    0 0.3333333 0.3333333 0.1875000 0.3571429 0.2222222 0.2500000  0.0
## [10,]    0 0.0000000 0.5000000 0.7500000 0.5263158 0.1111111 0.5555556  0.0
## [11,]    0 0.5000000 0.5384615 0.4375000 0.4583333 0.4615385 0.2500000  0.0
## [12,]    0 0.0000000 0.5833333 0.5200000 0.4545455 0.1111111 0.5000000  0.5
## [13,]    0 0.3333333 0.6666667 0.4166667 0.4166667 0.2500000 0.0000000  0.0
## [14,]    0 0.0000000 0.7500000 1.0000000 0.6000000 0.8000000 0.0000000  0.0
```

```
## running on O-line data
regionals_O_loc_data = tournament_loc_data(regionals_O)
throw_count_map(regionals_O_loc_data[[1]])
```



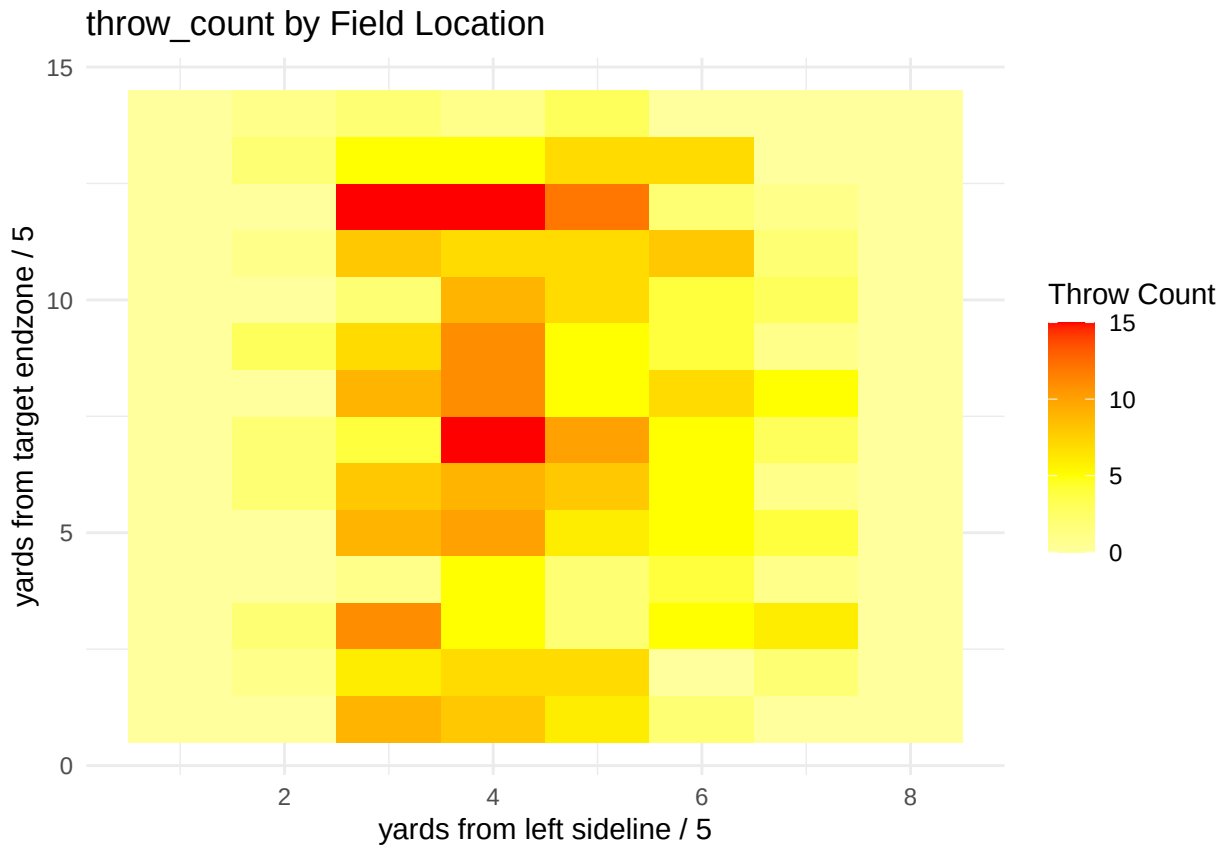

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
##  [1,]    0    0    3    1    3    3    1    0
##  [2,]    0    1    4    7    4    7    0    0
##  [3,]    0    0    3    4   10    3    3    0
##  [4,]    0    0    5    6    5    6    1    0
##  [5,]    0    2    4    4    9   11    2    0
##  [6,]    0    0    5    6    9    5    2    0
##  [7,]    0    0    7   14   14   17    1    0
##  [8,]    0    1    5    7   11   17    6    0
##  [9,]    0    0    2    5    9    5    3    0
## [10,]    0    1    4    7   12    5    6    0
## [11,]    0    1    5    9   17    5    2    0
## [12,]    0    0    9   10   10    7    3    2
## [13,]    0    1    7    7    5    5    3    0
## [14,]    0    0    2    3    2    5    2    0
```

```
score_rate_map(regionals_0_loc_data[[1]],regionals_0_loc_data[[2]])
```



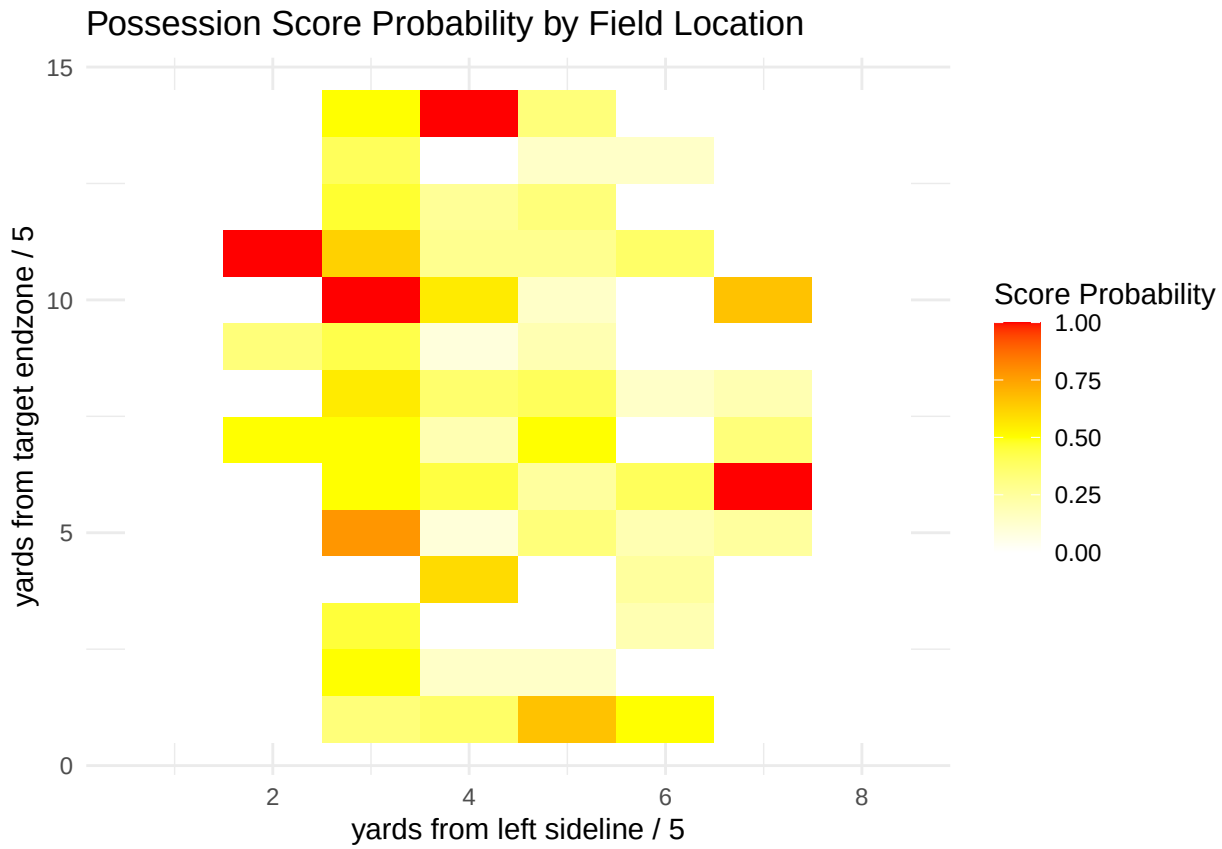

```
##       [,1] [,2]      [,3]      [,4]      [,5]      [,6]      [,7] [,8]
##  [1,]    0  0.0 0.6666667 0.0000000 0.6666667 0.6666667 1.0000000  0.0
##  [2,]    0  0.0 0.5000000 0.5714286 0.7500000 0.7142857 0.0000000  0.0
##  [3,]    0  0.0 0.6666667 0.2500000 0.2000000 1.0000000 1.0000000  0.0
##  [4,]    0  0.0 0.8000000 0.3333333 0.6000000 0.6666667 0.0000000  0.0
##  [5,]    0  0.5 0.2500000 1.0000000 0.7777778 0.4545455 0.0000000  0.0
##  [6,]    0  0.0 0.4000000 0.3333333 0.7777778 0.4000000 0.5000000  0.0
##  [7,]    0  0.0 0.5714286 0.5000000 0.3571429 0.4705882 1.0000000  0.0
##  [8,]    0  1.0 0.2000000 0.5714286 0.6363636 0.5882353 0.1666667  0.0
##  [9,]    0  0.0 0.0000000 0.4000000 0.4444444 0.4000000 0.3333333  0.0
## [10,]    0  0.0 0.2500000 1.0000000 0.7500000 0.2000000 0.5000000  0.0
## [11,]    0  0.0 0.4000000 0.5555556 0.5294118 0.6000000 0.5000000  0.0
## [12,]    0  0.0 0.7777778 0.9000000 0.6000000 0.1428571 0.6666667  0.5
## [13,]    0  1.0 0.8571429 0.7142857 0.8000000 0.4000000 0.0000000  0.0
## [14,]    0  0.0 1.0000000 1.0000000 1.0000000 0.8000000 0.0000000  0.0
```

```r
## running on D-line data
regionals_D_loc_data = tournament_loc_data(regionals_D)
throw_count_map(regionals_D_loc_data[[1]])
```




```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
##  [1,]    0    0    9    8    6    2    0    0
##  [2,]    0    1    6    7    7    0    2    0
##  [3,]    0    2   11    5    2    5    6    0
##  [4,]    0    0    1    5    2    4    1    0
##  [5,]    0    0    9   10    6    5    4    0
##  [6,]    0    2    8    9    8    5    1    0
##  [7,]    0    2    4   15   10    5    3    0
##  [8,]    0    0    9   11    5    7    5    0
##  [9,]    0    3    7   11    5    4    1    0
## [10,]    0    0    2    9    7    4    3    0
## [11,]    0    1    8    7    7    8    2    0
## [12,]    0    0   15   15   12    2    1    0
## [13,]    0    2    5    5    7    7    0    0
## [14,]    0    1    2    1    3    0    0    0
```

```
score_rate_map(regionals_D_loc_data[[1]],regionals_D_loc_data[[2]])
```




```
##       [,1]      [,2]      [,3]       [,4]      [,5]      [,6]      [,7] [,8]
##  [1,]    0 0.0000000 0.3333333 0.37500000 0.6666667 0.5000000 0.0000000    0
##  [2,]    0 0.0000000 0.5000000 0.14285714 0.1428571 0.0000000 0.0000000    0
##  [3,]    0 0.0000000 0.4545455 0.00000000 0.0000000 0.2000000 0.0000000    0
##  [4,]    0 0.0000000 0.0000000 0.60000000 0.0000000 0.2500000 0.0000000    0
##  [5,]    0 0.0000000 0.7777778 0.10000000 0.3333333 0.2000000 0.2500000    0
##  [6,]    0 0.0000000 0.5000000 0.44444444 0.2500000 0.4000000 1.0000000    0
##  [7,]    0 0.5000000 0.5000000 0.20000000 0.5000000 0.0000000 0.3333333    0
##  [8,]    0 0.0000000 0.5555556 0.36363636 0.4000000 0.1428571 0.2000000    0
##  [9,]    0 0.3333333 0.4285714 0.09090909 0.2000000 0.0000000 0.0000000    0
## [10,]    0 0.0000000 1.0000000 0.55555556 0.1428571 0.0000000 0.6666667    0
## [11,]    0 1.0000000 0.6250000 0.28571429 0.2857143 0.3750000 0.0000000    0
## [12,]    0 0.0000000 0.4666667 0.26666667 0.3333333 0.0000000 0.0000000    0
## [13,]    0 0.0000000 0.4000000 0.00000000 0.1428571 0.1428571 0.0000000    0
## [14,]    0 0.0000000 0.5000000 1.00000000 0.3333333 0.0000000 0.0000000    0
```

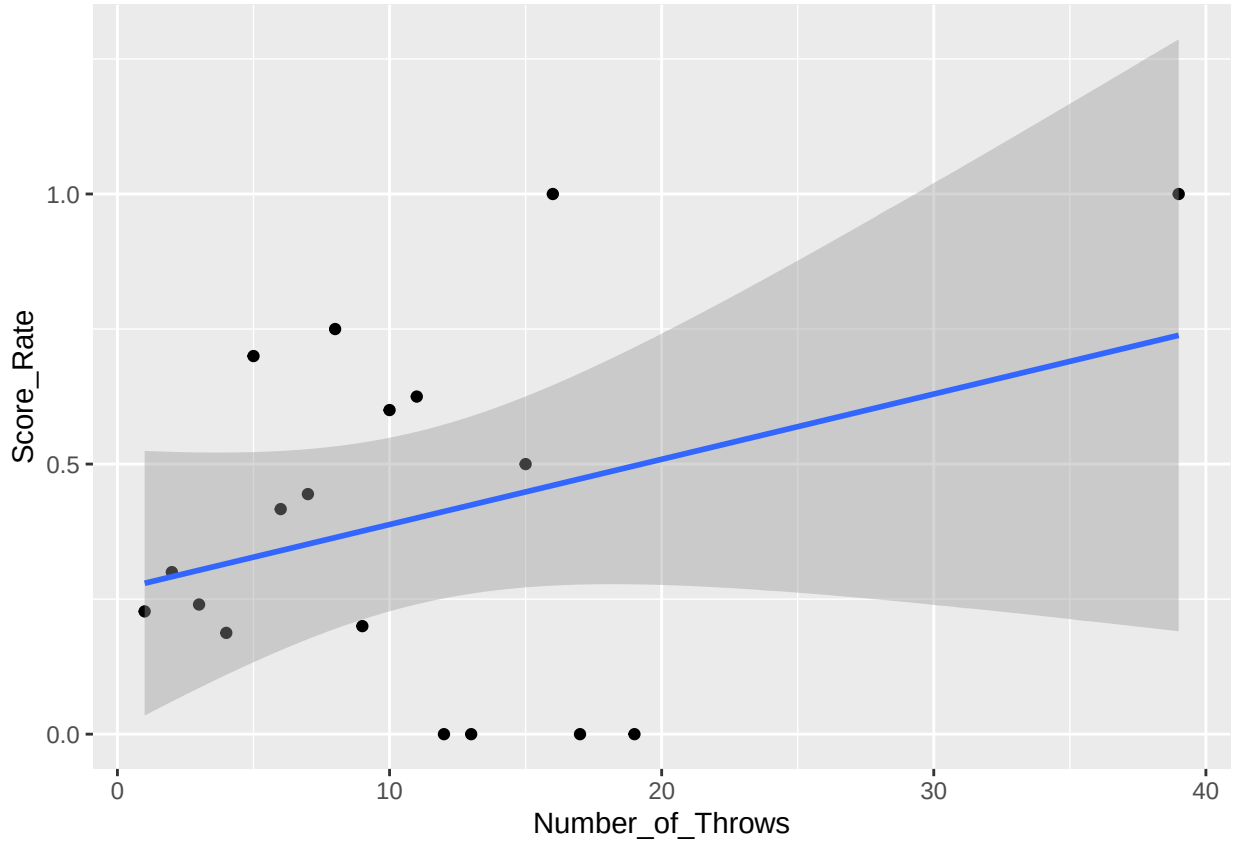
This sample size is low across the board, but especially on the sidelines. We need more data in order to make any significancy claims.

Scoring rate against number of throws taken on a possession

```
## given a game object, returns possession throw count, success count, with possessions broken down by
game_throw_count_success_rate = function(game) {
  max_throws = 100
  throw_count_list = rep(0,max_throws)
  score_count_list = rep(0,max_throws)
```

```
  for (possession in game@possessions) {
    throw_count = possession@summary$Passes

    scored = possession@summary$Scored.
    throw_count_list[throw_count] = throw_count_list[throw_count] + 1
    score_count_list[throw_count] = score_count_list[throw_count] + as.numeric(scored)
  }
  return(list(throw_count_list,score_count_list))
}
## given a tournament object, returns possession success rate with possessions broken into bins of numb
tournament_throw_count_success_rate = function(tournament) {
  max_throws = 100
  throw_count_list = rep(0,max_throws)
  score_count_list = rep(0,max_throws)

  for (game in tournament@games) {
    game_counts = game_throw_count_success_rate(game)
    throw_count_list = throw_count_list + game_counts[[1]]
    score_count_list = score_count_list + game_counts[[2]]
  }
  score_rate_list = score_count_list/throw_count_list

  df = na.omit(as.data.frame(cbind(1:100, score_rate_list)))
  colnames(df) = c("Number_of_Throws","Score_Rate")

  print(
    ggplot(df,aes(x=Number_of_Throws,y=Score_Rate)) +
      geom_point() +
      geom_smooth(method=lm)
  )
  return(df)
}

tournament_throw_count_success_rate(regionals_obj)
```

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




```
##    Number_of_Throws Score_Rate
## 1                 1  0.2272727
## 2                 2  0.3000000
## 3                 3  0.2400000
## 4                 4  0.1875000
## 5                 5  0.7000000
## 6                 6  0.4166667
## 7                 7  0.4444444
## 8                 8  0.7500000
## 9                 9  0.2000000
## 10               10  0.6000000
## 11               11  0.6250000
## 12               12  0.0000000
## 13               13  0.0000000
## 15               15  0.5000000
## 16               16  1.0000000
## 17               17  0.0000000
## 19               19  0.0000000
## 39               39  1.0000000
```

Scoring rate of possessions involving hucks

Here we divide our sample of possessions into 4 categories along 2 axes: O-points vs. D-points, and possessions involving hucks and possessions not involving hucks. In the regionals data set, hucks are defined as throws 26 yards or over

```
O_huck_split = tournament_huck_split(regionals_O)
D_huck_split = tournament_huck_split(regionals_D)

O_huck = O_huck_split[[1]]
O_no_huck = O_huck_split[[2]]

D_huck = D_huck_split[[1]]
D_no_huck = D_huck_split[[2]]

O_huck_data = tournament_score_rate(O_huck)
O_huck_rate = O_huck_data[[1]]
O_huck_n = O_huck_data[[2]]

O_no_huck_data = tournament_score_rate(O_no_huck)
O_no_huck_rate = O_no_huck_data[[1]]
O_no_huck_n = O_no_huck_data[[2]]

D_huck_data = tournament_score_rate(D_huck)
D_huck_rate = D_huck_data[[1]]
D_huck_n = D_huck_data[[2]]

D_no_huck_data = tournament_score_rate(D_no_huck)
D_no_huck_rate = D_no_huck_data[[1]]
D_no_huck_n = D_no_huck_data[[2]]

print("Score % on possessions where -- ")
```

```
## [1] "Score % on possessions where -- "
```

```
print(paste("O-line hucks:",O_huck_rate,". n = ",O_huck_n))
```

```
## [1] "O-line hucks: 0.479166666666667 . n =  48"
```

```
print(paste("O-line does not huck:",O_no_huck_rate,". n = ",O_no_huck_n))
```

```
## [1] "O-line does not huck: 0.321428571428571 . n =  28"
```

```
print(paste("D-line hucks:",D_huck_rate,". n = ",D_huck_n))
```

```
## [1] "D-line hucks: 0.6 . n =  10"
```

```
print(paste("D-line does not huck:",D_no_huck_rate,". n = ",D_no_huck_n))
```

```
## [1] "D-line does not huck: 0.153846153846154 . n =  52"
```

```
## two sample t-test

## O-line H0: huck_p > no_huck_p

# O-line x_bar (p in binomial model) is equal to score %
O_no_huck_p = O_no_huck_rate
O_huck_p = O_huck_rate

# binomial model: V = n*p*(1-p)
# O-line sample variances
O_no_huck_var = O_no_huck_p*(1-O_no_huck_p)/O_no_huck_n
O_huck_var = O_huck_p*(1-O_huck_p)/O_huck_n
```

```r
# combined sample variance, df, and t-statistic
O_S_p = sqrt(((O_huck_n-1)*O_huck_var+(O_no_huck_n-1)*O_no_huck_var) / (O_huck_n + O_no_huck_n - 2))
O_t = (O_huck_p - O_no_huck_p)/O_S_p
O_df = (O_huck_n + O_no_huck_n - 2)

# p- value
O_p = pt(O_t,O_df,lower.tail = FALSE)

print("Testing H0: O-line score % is independent of whether they huck, H1: O-line score % is higher when
```

```
## [1] "Testing H0: O-line score % is independent of whether they huck, H1: O-line score % is higher whe
```

```r
print(paste("p-value =",O_p))
```

```
## [1] "p-value = 0.0239134956881"
```

```r
## D-line H0: huck_p > no_huck_p

D_no_huck_p = D_no_huck_rate
D_huck_p = D_huck_rate

D_no_huck_var = D_no_huck_p*(1-D_no_huck_p)/D_no_huck_n
D_huck_var = D_huck_p*(1-D_huck_p)/D_huck_n

D_S_p = sqrt(((D_huck_n-1)*D_huck_var+(D_no_huck_n-1)*D_no_huck_var) / (D_huck_n + D_no_huck_n - 2))
D_t = (D_huck_p - D_no_huck_p)/D_S_p
D_df = (D_huck_n + D_no_huck_n - 2)

D_p = pt(D_t,D_df,lower.tail = FALSE)

print("Testing H0: D-line score % is independent of whether they huck, H1: D-line score % is higher when
```

```
## [1] "Testing H0: D-line score % is independent of whether they huck, H1: D-line score % is higher whe
```

```r
print(paste("p-value =",D_p))
```

```
## [1] "p-value = 9.19203914806773e-08"
```

This is statistically significant evidence that both O-line and D-line, and especially D-line, score at a higher rate on possessions where they huck. That said, this does not mean each line should only huck, as defenses adjust in real-time to offensive looks. Additionally, this could be improved by separating each line into upwind and downwind. A potential hypothesis to test would be that while each line scores at a higher rate hucking downwind than not hucking downwind, they score at the same or lower rate when hucking upwind compared to not hucking upwind.