Q&A/DEMO SESSION OVERVIEW

- **SESSION 1 TOPICS**
  - **Q&A**
    - Project and Analysis
    - Data and Computation
  - **Demo:** Analysis in R
- **UPCOMING SESSION TOPICS (PROJECTED)**
  - **July 5:** Demo Analysis in Python
  - **July 26:** How to write a paper
  - **August 16:** Structuring and commenting code (R markdown etc.)
- **ARCHIVED VIDEOS AVAILABLE AFTER**

Q&A

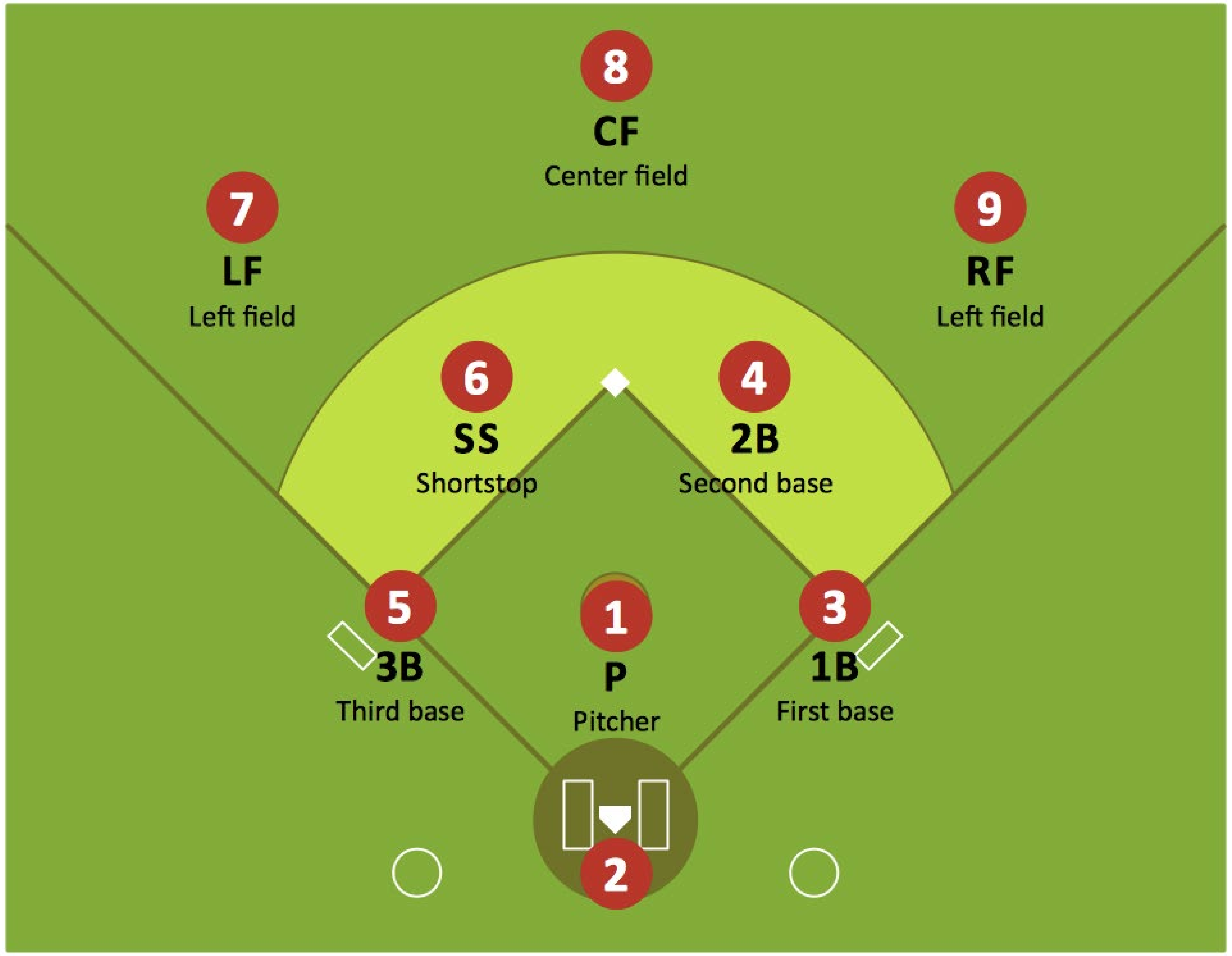
SMT®

PROJECT AND ANALYSIS

1. **Start with an idea**
2. **Turn a bug into a feature**
3. **"Frame it as a story"**
4. **Keep your audience(s) in mind**
    1. Judges
    2. Teams and other employers

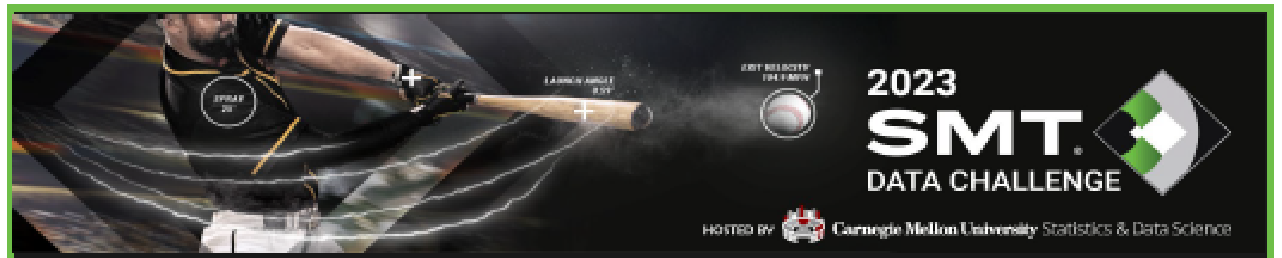
DATA AND COMPUTATION

1. Explaining *player_position* and *event_code*
2. What about missing data?
3. Identifying different plays
4. Do you have to use all the data?
5. Can you include other data?
6. Computer language?



DATA AND COMPUTATION

1. Explaining *player_position* and *event_code*
2. What about missing data?
3. Identifying different plays
4. Do you have to use all the data?
5. Can you include other data?
6. Computer language?



GLOSSARY

VARIABLE	TABLE	CODE	DEFINITION
player_position	game_events player_pos (game_info)	1	pitcher
		2	catcher
		3	first baseman
		4	second baseman
		5	third baseman
		6	shortstop
		7	left field
		8	center field
		9	right field
		10	batter
		11	runner on first base
		12	runner on second base
		13	runner on third base
		255	ball event with no player (e.g., ball bounce)
event_code	game_events	1	pitch
		2	ball acquired
		3	throw (ball-in-play)
		4	ball hit into play
		5	end of play
		6	pickoff throw
		7	ball acquired - unknown field position
		8	throw (ball-in-play) - unknown field position
		9	ball deflection
		10	ball deflection off of wall
		11	home run
		16	ball bounce

DATA AND COMPUTATION

1. Explaining *player_position* and *event_code*
2. What about missing data?
3. Identifying different plays
4. Do you have to use all the data?
5. Can you include other data?
6. Computer language?

game_str	play_id	at_bat	play_per_game	timestamp	player_position	event_code
1900_09_TeamKK_TeamB	154	NA	157	5799556	0	5
1900_09_TeamKK_TeamB	155	NA	158	5822725	1	1
1900_09_TeamKK_TeamB	155	NA	158	5823187	10	4
1900_09_TeamKK_TeamB	155	NA	158	5825102	6	2
1900_09_TeamKK_TeamB	155	NA	158	5826191	6	3
1900_09_TeamKK_TeamB	155	NA	158	5827412	3	2
1900_09_TeamKK_TeamB	155	NA	158	5827775	0	5
1900_09_TeamKK_TeamB	156	NA	159	5858470	1	1
1900_09_TeamKK_TeamB	156	NA	159	5858932	2	2
1900_09_TeamKK_TeamB	156	NA	159	5858932	0	5
1900_09_TeamKK_TeamB	157	NA	160	5876391	1	1
1900_09_TeamKK_TeamB	157	NA	160	5876854	2	2
1900_09_TeamKK_TeamB	157	NA	160	5876854	0	5
1900_09_TeamKK_TeamB	158	NA	161	5915205	1	1
1900_09_TeamKK_TeamB	158	NA	161	5915602	10	4
1900_09_TeamKK_TeamB	158	NA	161	5915734	0	5
1900_09_TeamKK_TeamB	159	NA	162	5945306	1	1
1900_09_TeamKK_TeamB	159	NA	162	5945768	2	2
1900_09_TeamKK_TeamB	159	NA	162	5945768	0	5
1900_09_TeamKK_TeamB	160	NA	163	5968278	1	1
1900_09_TeamKK_TeamB	160	NA	163	5968707	10	4
1900_09_TeamKK_TeamB	160	NA	163	5969268	255	16
1900_09_TeamKK_TeamB	160	NA	163	5970522	4	2
1900_09_TeamKK_TeamB	160	NA	163	5971776	4	3
1900_09_TeamKK_TeamB	160	NA	163	5972766	3	2
1900_09_TeamKK_TeamB	160	NA	163	5973261	0	5

DEMO ANALYSIS IN R

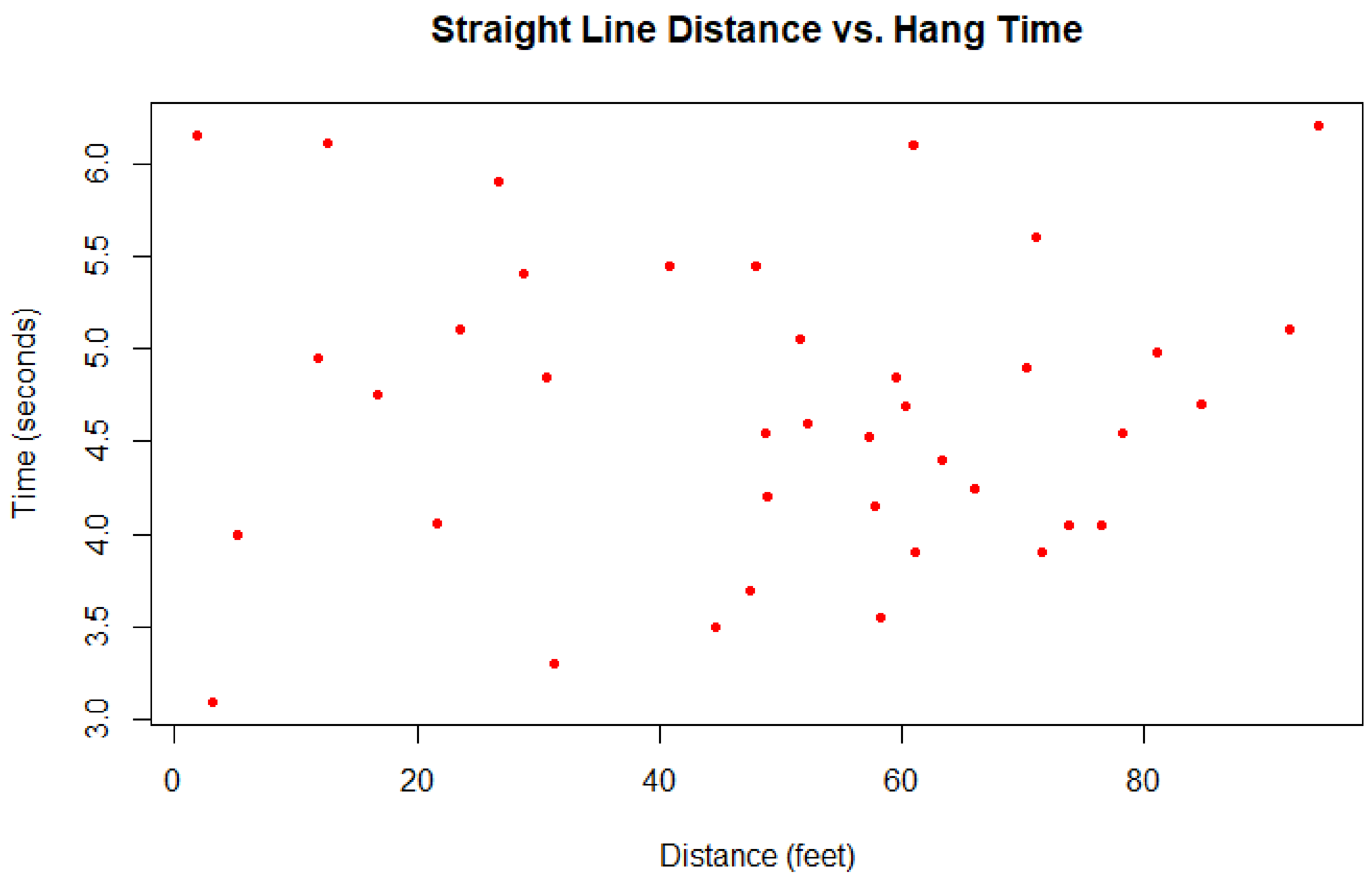
```r
##### Total distance traveled vs. straight line distance to make a fly out

### EXAMPLE: Consider fly ball routes for a single player



### LIBRARIES
library(tidyverse)



### LOAD DATA
game_info <- read.csv('game_info_anon_2.0.csv', header = TRUE)
game_events <- read.csv('game_events_anon_2.0.csv', header = TRUE)
team_info <- read.csv('team_info_anon_2.0.csv', header = TRUE)
ball_pos <- read.csv('ball_pos_anon_2.0.csv', header = TRUE)
player_pos <- read.csv('player_pos_anon_2.0.csv', header = TRUE)



### CHOOSE A PLAYER
# Limit to center fielders
# Choose a player that's made a lot of fly outs

## Using game_info, how many plays for each center fielder?
cf_play_count <- game_info %>% group_by(center_field) %>%
      summarize(count=n()) %>% arrange(desc(count))

player1 <- as.numeric(cf_play_count[1,1])

## In which games did player1 play?
player1_game_info <- game_info %>% filter(center_field == player1)
games_unique <- unique(player1_game_info$game_str)



### FIND THE PLAYS WE WANT
# In which games did player1 make a fly ball catch?
#
#   1)  Pull information from game_events
#
#   2)  Look for sequences where a ball is put in play, then caught by the center fielder
#
#           player_position | event_code
#           ----------------------------
#                   10      |     4
#           ----------------------------
#                   8       |     2
#           ----------------------------
#
#   3)  Are the catches by player1 (not the other team's center fielder)?
#       Check using play_per_game, and keep track of play_per_game values.

## Create a data frame containing games, play_id, and play_per_game
player1_plays_df <- data.frame(game_str = character(0),
                               play_id = integer(0),
                               play_per_game = integer(0),
                               stringsAsFactors = FALSE)

## Loop through player1 games in game_events
for(n1 in 1:length(games_unique)){
# for(n1 in 1:1){
  game <- game_events %>% filter(game_str == games_unique[n1])
```

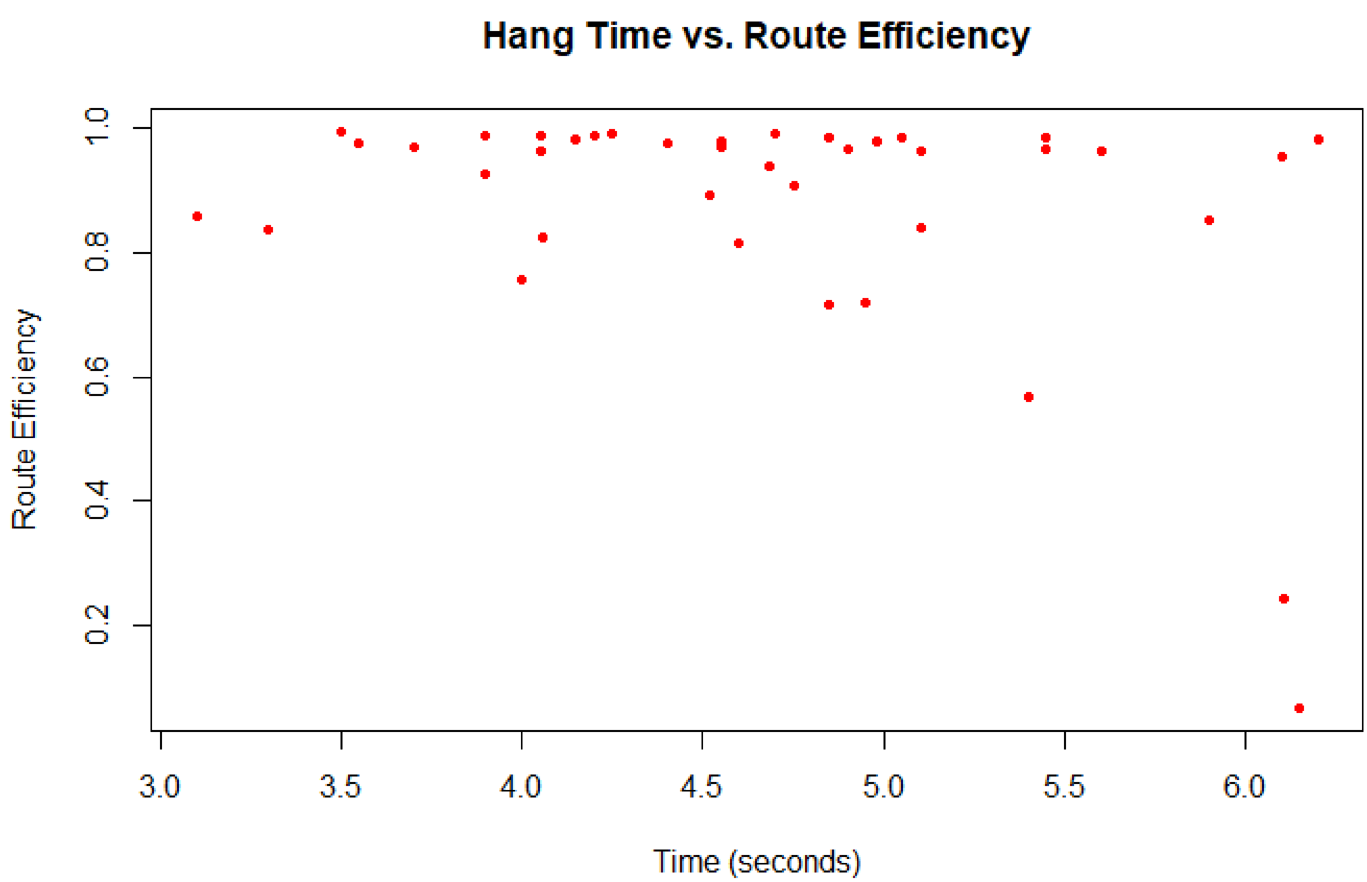
```
  player1_game <- player1_game_info %>% filter(game_str == games_unique[n1])

  ## 1) Loop through an individual game
  for(n2 in 1:(nrow(game)-1)){

    ## 2) Find catches
    if(game$player_position[n2] == 10 & game$event_code[n2] == 4
      & game$player_position[n2+1] == 8 & game$event_code[n2+1] == 2){
      play_id <- game$play_id[n2]
      play_num <- game$play_per_game[n2]
      if(play_num %in% player1_game$play_per_game){
        play_df <- data.frame(game_str = games_unique[n1],
                              play_id = play_id,
                              play_per_game = play_num)
        player1_plays_df <- rbind(player1_plays_df, play_df)
      }
    }
  }
}



### ------------------------------------------------
# How do we determine the start/end times for distance traveled?
#
#   - Get timestamp of hit and timestamp of catch
#
#   - Assumption: player1 starts moving as soon as the ball leaves the bat.
#       Start time = timestamp when event_code == 4
#       End time = timestamp when event_code == 2
#
# Let's see how long each ball is in the air.
#
### (This starts with the same code as above. I'm just adding to it.)
#

## Expand the data frame to include start and end times.
player1_plays_df <- data.frame(game_str = character(0),
                               play_id = integer(0),
                               play_per_game = integer(0),
                               start_time = numeric(0),
                               end_time = numeric(0),
                               del_t = numeric(0),
                               stringsAsFactors = FALSE)

## Loop through player1 games in game_events
for(n1 in 1:length(games_unique)){
  # for(n1 in 1:1){
  game <- game_events %>% filter(game_str == games_unique[n1])
  player1_game <- player1_game_info %>% filter(game_str == games_unique[n1])

  ## 1) Loop through an individual game
  for(n2 in 1:(nrow(game)-1)){

    ## 2) Find catches
    if(game$player_position[n2] == 10 & game$event_code[n2] == 4
      & game$player_position[n2+1] == 8 & game$event_code[n2+1] == 2){
      play_id <- game$play_id[n2]
      play_num <- game$play_per_game[n2]
      if(play_num %in% player1_game$play_per_game){
        start_time <- game$timestamp[n2]
        end_time <- game$timestamp[n2+1]
        del_t <- end_time - start_time
```

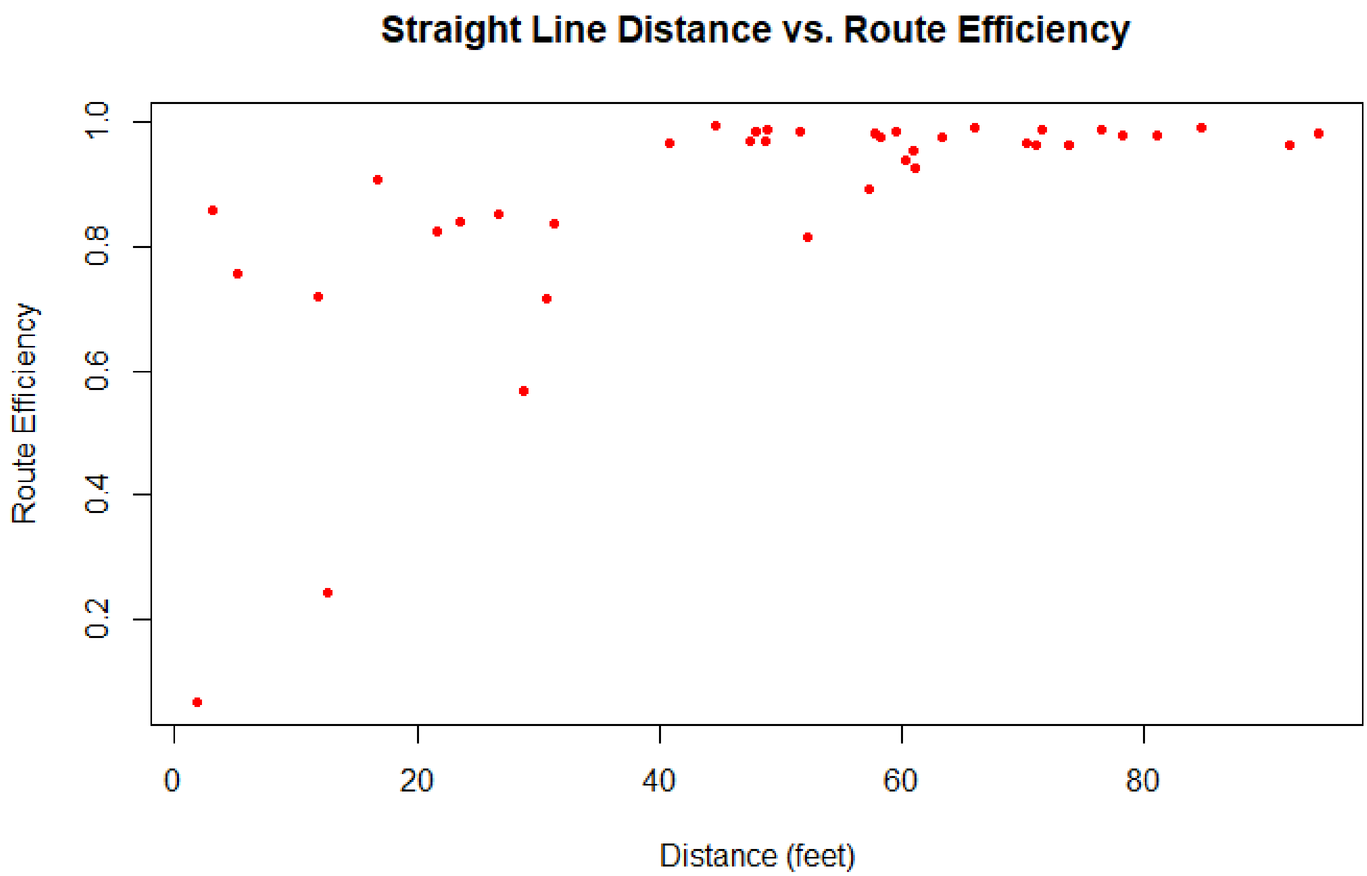
```r
        play_df <- data.frame(game_str = games_unique[n1],
                              play_id = play_id,
                              play_per_game = play_num,
                              start_time = start_time,
                              end_time = end_time,
                              del_t = del_t)
        player1_plays_df <- rbind(player1_plays_df, play_df)
      }
    }
  }
}



### FIND PLAYER DISTANCE INFORMATION
#
#   1)  Pull information from player_pos
#
#   2)  Use info in player1_plays_df to constrain data (it's a big data frame!)
#
#   3)  Add data frame columns for straight line distance and total distance traveled
#         STRAIGHT LINE DISTANCE: [x,y] distance between player1 location
#           at start and end times
#         TOTAL DISTANCE TRAVELED: the sum of [x,y] distances by timestep
#           between start and end times
#         ROUTE EFFICIENCY: (straight line distance)/(total distance traveled)
#           (N.B. - This is considered a suspect metric. We'll see why in a minute.)

## Create a new data frame that includes distance columns
player1_distance_df <- data.frame(game_str = character(0),
                                  play_id = integer(0),
                                  play_per_game = integer(0),
                                  start_time = numeric(0),
                                  end_time = numeric(0),
                                  del_t = numeric(0),
                                  d_straight = numeric(0),
                                  d_total = numeric(0),
                                  route_eff = numeric(0),
                                  stringsAsFactors = FALSE)

# NOTE: Since player_pos is big, we want to filter it down as early as possible
#   - Use player1_plays_df to loop through player_pos
#   - player_pos uses play_id, but not play_per_game

for(n1 in 1:nrow(player1_plays_df)){
  # for(n1 in 1:1){
  # n1 = 1
  player1_pos <- player_pos %>% filter(game_str == player1_plays_df$game_str[n1]
                                        & play_id == player1_plays_df$play_id[n1]
                                        & player_position == 8
                                        & timestamp >= player1_plays_df$start_time[n1]
                                        & timestamp <= player1_plays_df$end_time[n1]
                                       ) %>%
    arrange(timestamp)

  ## Calculate straight line distance
  start_x <- player1_pos$field_x[1]
  end_x <- player1_pos$field_x[nrow(player1_pos)]
  start_y <- player1_pos$field_y[1]
  end_y <- player1_pos$field_y[nrow(player1_pos)]

  d_straight <- sqrt((end_x - start_x)^2 + (end_y - start_y)^2)

```

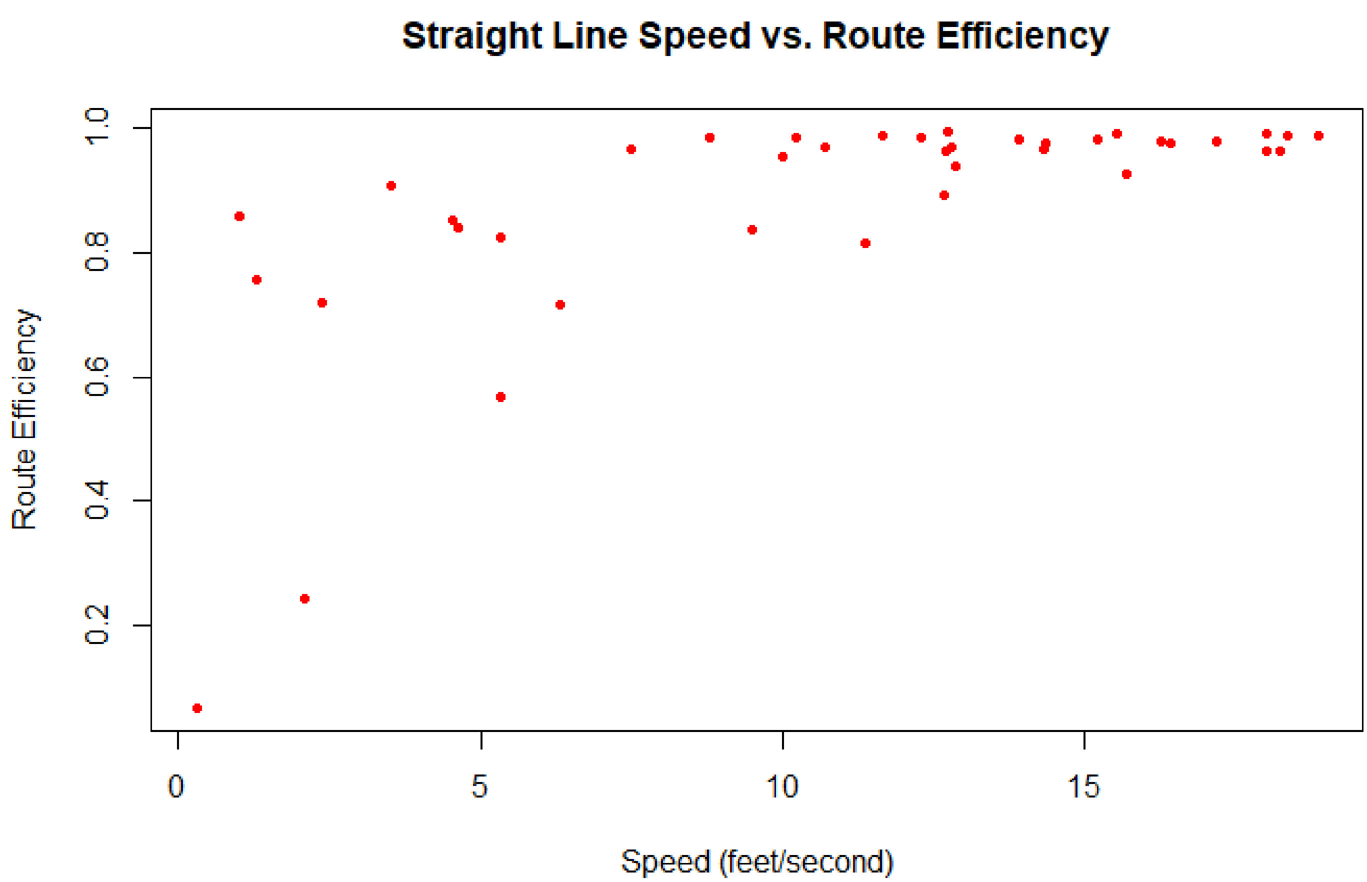
```r
  ## Calculate total distance traveled
  d_total = 0
  for(n2 in 1:(nrow(player1_pos)-1)){
    start_x <- player1_pos$field_x[n2]
    end_x <- player1_pos$field_x[n2+1]
    start_y <- player1_pos$field_y[n2]
    end_y <- player1_pos$field_y[n2+1]

    d_step <- sqrt((end_x - start_x)^2 + (end_y - start_y)^2)
    d_total <- d_total + d_step
  }

  ## Calculate route efficiency
  route_eff <- d_straight/d_total

  distance_df <- data.frame(game_str = player1_plays_df$game_str[n1],
                            play_id = player1_plays_df$play_id[n1],
                            play_per_game = player1_plays_df$play_per_game[n1],
                            start_time = player1_plays_df$start_time[n1],
                            end_time = player1_plays_df$end_time[n1],
                            del_t = player1_plays_df$del_t[n1],
                            d_straight = d_straight,
                            d_total = d_total,
                            route_eff = route_eff)
  player1_distance_df <- rbind(player1_distance_df, distance_df)
}



##### ---------------------------------------
### USING VISUALS TO ANSWER QUESTIONS ABOUT PLAYER1

## 1) How does hang time (del_t) correspond to straight line distance?
plot(player1_distance_df$d_straight, player1_distance_df$del_t/1000,
     main = 'Straight Line Distance vs. Hang Time',
     xlab = 'Distance (feet)', ylab = 'Time (seconds)',
     col = 'red', pch = 20)

#*** The farther a player has to travel to make a catch, the longer the ball has to
#*** be in the air. Otherwise, he wouldn't be able to get to it.


## 2) How does route efficiency correspond to hang time?
plot(player1_distance_df$del_t/1000, player1_distance_df$route_eff,
     main = 'Hang Time vs. Route Efficiency',
     xlab = 'Time (seconds)', ylab = 'Route Efficiency',
     col = 'red', pch = 20)

#*** If the ball is airborne longer, the player has more time to accommodate
#*** an inefficient route and still make the catch.


## 3) How does route efficiency correspond to straight line distance?
plot(player1_distance_df$d_straight, player1_distance_df$route_eff,
     main = 'Straight Line Distance vs. Route Efficiency',
     xlab = 'Distance (feet)', ylab = 'Route Efficiency',
     col = 'red', pch = 20)

#*** The farther a player has to travel to make a catch, the more efficient the route
#*** required. He has less time to cover the distance, and if his route isn't
#*** efficient enough, he can't make the play.
```

```
## 4) How does route efficiency correspond to straight-line speed (d_straight/del_t)?
speed_straight <- (player1_distance_df$d_straight/(player1_distance_df$del_t/1000))
plot(speed_straight, player1_distance_df$route_eff,
     main = 'Straight Line Speed vs. Route Efficiency',
     xlab = 'Speed (feet/second)', ylab = 'Route Efficiency',
     col = 'red', pch = 20)

#*** As a player gets closer to sprint-level speed, it is less likely that he will
#*** change direction, for the simple reason that it generally requires the player
#*** to slow down.

###################################################################
#                                                                 #
#   CONCLUSION: ROUTE EFFICIENCY SUFFERS FROM SELECTION BIAS.     #
#                                                                 #
#         "HARDER" CATCHES ONLY OCCUR WHEN PLAYERS                #
#               TAKE MORE EFFICIENT ROUTES.                       #
#                                                                 #
###################################################################
```


Straight Line Distance vs. Hang Time
Time (seconds)
3.0
3.5
4.0
4.5
5.0
5.5
6.0
0
20
40
60
80
Distance (feet)


Hang Time vs. Route Efficiency
Route Efficiency
0.2
0.4
0.6
0.8
1.0
3.0
3.5
4.0
4.5
5.0
5.5
6.0
Time (seconds)


Straight Line Distance vs. Route Efficiency
Route Efficiency
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
Distance (feet)


Straight Line Speed vs. Route Efficiency
Route Efficiency
0.2
0.4
0.6
0.8
1.0
0
5
10
15
Speed (feet/second)