NHL Goalie and Skater Heat Maps

Sasank Vishnubhatla

January 10, 2019

Last Update: 2019-01-10 22:03:58

GitHub Repository: sv4u/goalie-and-skater-heat-maps

Introduction

In the NHL, players have their *sweet-spots*. For skaters, it can be the top of a certain circle, or right in-front of the net. For goalies, it could be where their vision is best and where they have the best angle to cut down a shot. All players have these sweet-spots, but it is difficult to analytically say where they are. By using shot location data, we can determine these locations and create models show where goalies and skaters need improvement and where they succeed.

Before we jump in, let's clean up our R environment and also load in some libraries we will be using.

```
rm(list = ls())

library(purrr)
library(ggplot2)
```

Data Formatting

To start, we need to read in our data. Our data is formatted nicely in CSV format. We have data from the 2016-2017 season, 2017-2018 season, and 2018-2019 season (up to 1/7/19). This data was downloaded from MoneyPuck. Let's first start by loading in all three seasons of data:

```
data.2016 = read.csv("data/2016.csv")
data.2017 = read.csv("data/2017.csv")
data.2018 = read.csv("data/2018.csv")
```

> Note: this will take a *relatively* long time to compute as the datasets are large. Each dataset contains all shot data (**including** playoffs).

We'll only look at regular season data. The playoffs in the NHL are a beast of their own.

```
get.regular.season = function(data) {
    subset(data, isPlayoffGame == 0)
}

season.2016 = get.regular.season(data.2016)
season.2017 = get.regular.season(data.2017)
season.2018 = get.regular.season(data.2018)
```

Now that we have our data, we can remove extraneous columns. Here is a table of what columns we are keeping, and what we are renaming them to:

Old Column	New Column
xCordAdjusted	x

Old Column	New Column
yCordAdjusted	y
goal	goal
shotAngleAdjusted	angle
goalieNameForShot	goalie_name
shooterName	skater_name
game_id	game

Now, here is the R code to do this subsetting of the original dataset.

```
get.helpful.data = function(data) {
    data.frame(x = data$xCordAdjusted,
           y = data$yCordAdjusted,
           goal = data$goal,
           angle = data$shotAngleAdjusted,
           goalie_name = data$goalieNameForShot,
           skater_name = data$shooterName,
           game = data$game_id)
}

analysis.2016 = get.helpful.data(season.2016)
analysis.2017 = get.helpful.data(season.2017)
analysis.2018 = get.helpful.data(season.2018)
```

Now, we have all the data we need.

Function Definitions

Generic

From our data, we can calculate some very important statistics like the following:

- Goal Percent: goals per total shots
- Save Percent: saves (total shots - goals) per total shots
- Shots per Goal: total shots per goal

Additionally, we can break up our data by game. There are some generic functions we can write to help for both goalies and skaters. Let's write them now!

```
get.goal.percent = function(data) {
    shots = length(data$goal)
    temp = subset(data, goal == 1)
    goals = length(temp$goal)
    goals / shots
}

get.save.percent = function(data) {
    shots = length(data$goal)
    temp = subset(data, goal == 1)
    goals = length(temp$goal)
    (shots - goals) / shots
}
```

```
get.shots.per.goal = function(data) {
    shots = length(data$goal)
    temp = subset(data, goal == 1)
    goals = length(temp$goal)
    shots / goal
}
```

> Note: when using `get.shots.per.goal`, if there were no goals scored, R will handle the division by zero by returning infinity. This will be problematic when graphing data. I am still working on a good solution to this problem. Earlier, I used 200 as a substitute value. However, 200 still skews graphs, which is unideal.

```
get.games = function(data) {
    unique(data$game)
}

get.single.game = function(data, game_id) {
    subset(data, game == game_id)
}

get.all.games = function(data) {
    games = get.games(data)
    Map(function(x) get.single.game(data, x), games)
}
```

Now, we can create our game by game statistic functions:

```
get.game.goal.percent = function(data) {
    gameframe = get.all.games(data)
    games.gp = map(gameframe, function(x) get.goal.percent(x))
    unlist(games.gp, use.names = FALSE)
}

get.game.save.percent = function(data) {
    gameframe = get.all.games(data)
    games.sp = map(gameframe, function(x) get.save.percent(x))
    unlist(games.sp, use.names = FALSE)
}

get.game.shots.per.goal = function(data) {
    gameframe = get.all.games(data)
    games.spg = map(gameframe, function(x) get.shots.per.goal(x))
    unlist(games.spg, use.names = FALSE)
}
```

Also, we'll need a function to get match-ups between a specific goalie and skater. Let's write that here, instead of in our goalies *and* our skaters sections.

```
get.matchup.data = function(data, goalie, skater) {
    subset(data, goalie_name == goalie & skater_name == skater)
}
```

We've now written our generic data handling functions.

Goalies

Let's first start with a function to get data for a specific goalie.

```
get.goalie.data = function(data, name) {
    subset(data, goalie_name == name)
}
```

Skaters

Let's first start with a function to get data for a specific skater.

```
get.skater.data = function(data, name) {
    subset(data, skater_name == name)
}
```

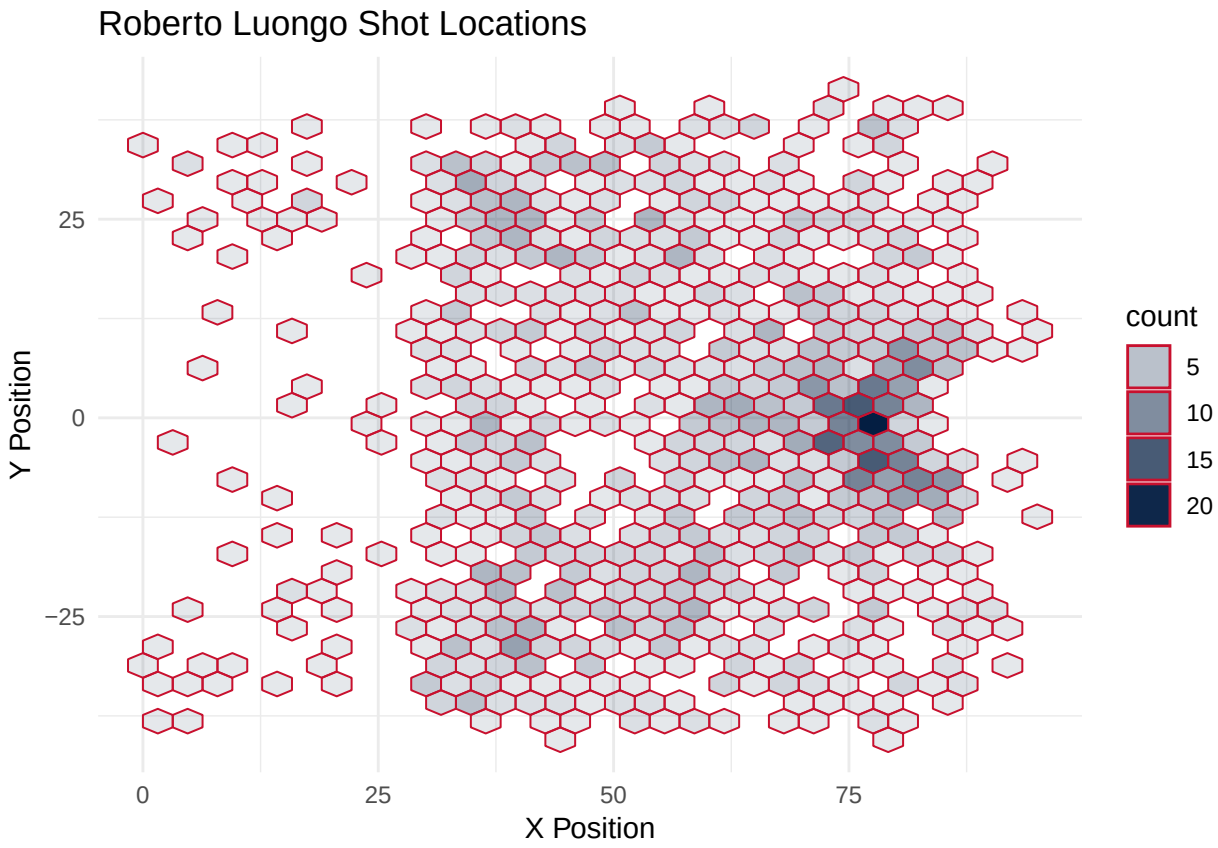
Graphing

Given specific data, we should be able to graph the location of shots. Let's write a function that uses ggplot to do so.

```
graph.shot.locations = function(data, primary, secondary, name) {
    plot = ggplot(data) +
        geom_hex(aes(x = x, y = y, alpha = ..count..), fill = primary, color = secondary) +
        labs(title = paste(name, "Shot Locations", sep = " "), x = "X Position", y = "Y Position") +
        theme_minimal()
    plot
}
```

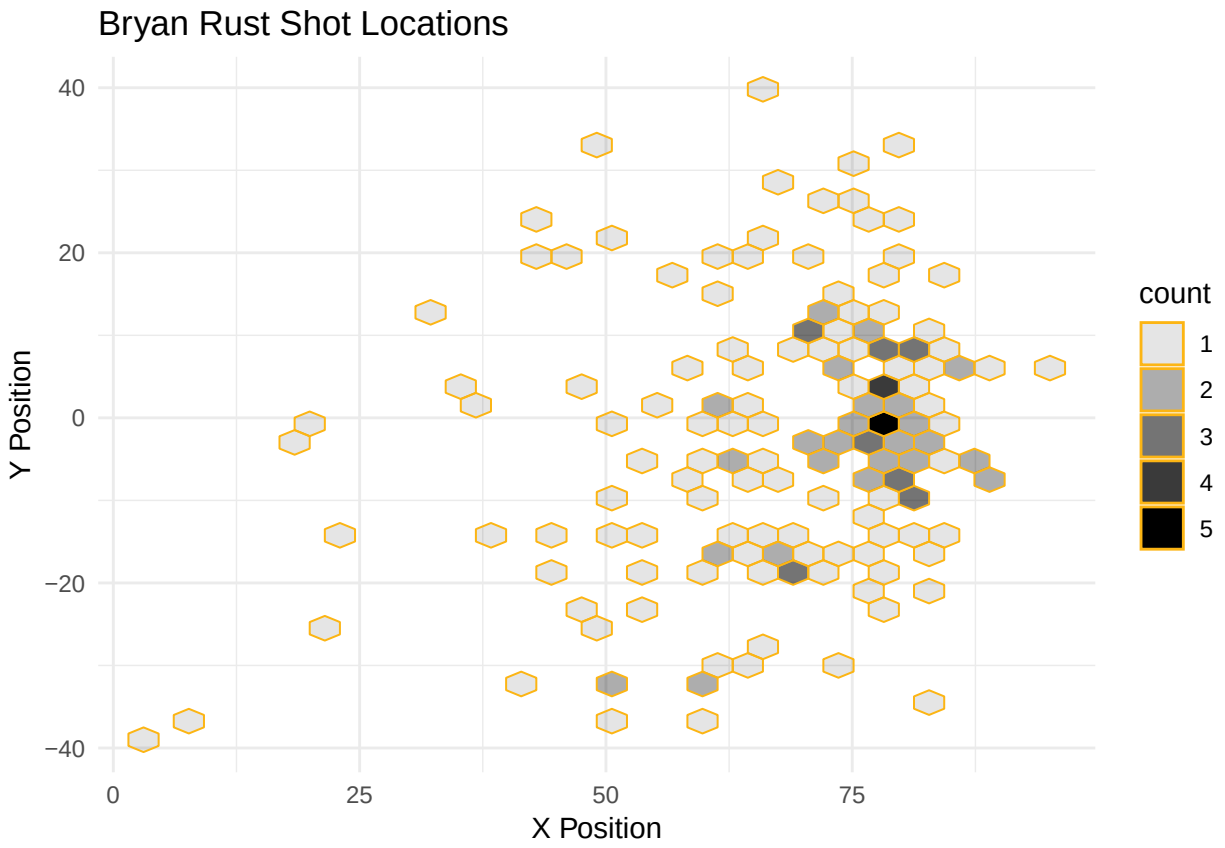
To see a test of what this does, let's quickly make a graph of Roberto Luongo's shots against him.

```
luongo = get.goalie.data(analysis.2017, "Roberto Luongo")
plot = graph.shot.locations(luongo, "#041E42", "#C8102E", "Roberto Luongo")

plot
```



Now, let's see how it looks for a skater. Let's look at Bryan Rust.

```
rust = get.skater.data(analysis.2017, "Bryan Rust")
plot = graph.shot.locations(rust, "#000000", "#FCB514", "Bryan Rust")

plot
```

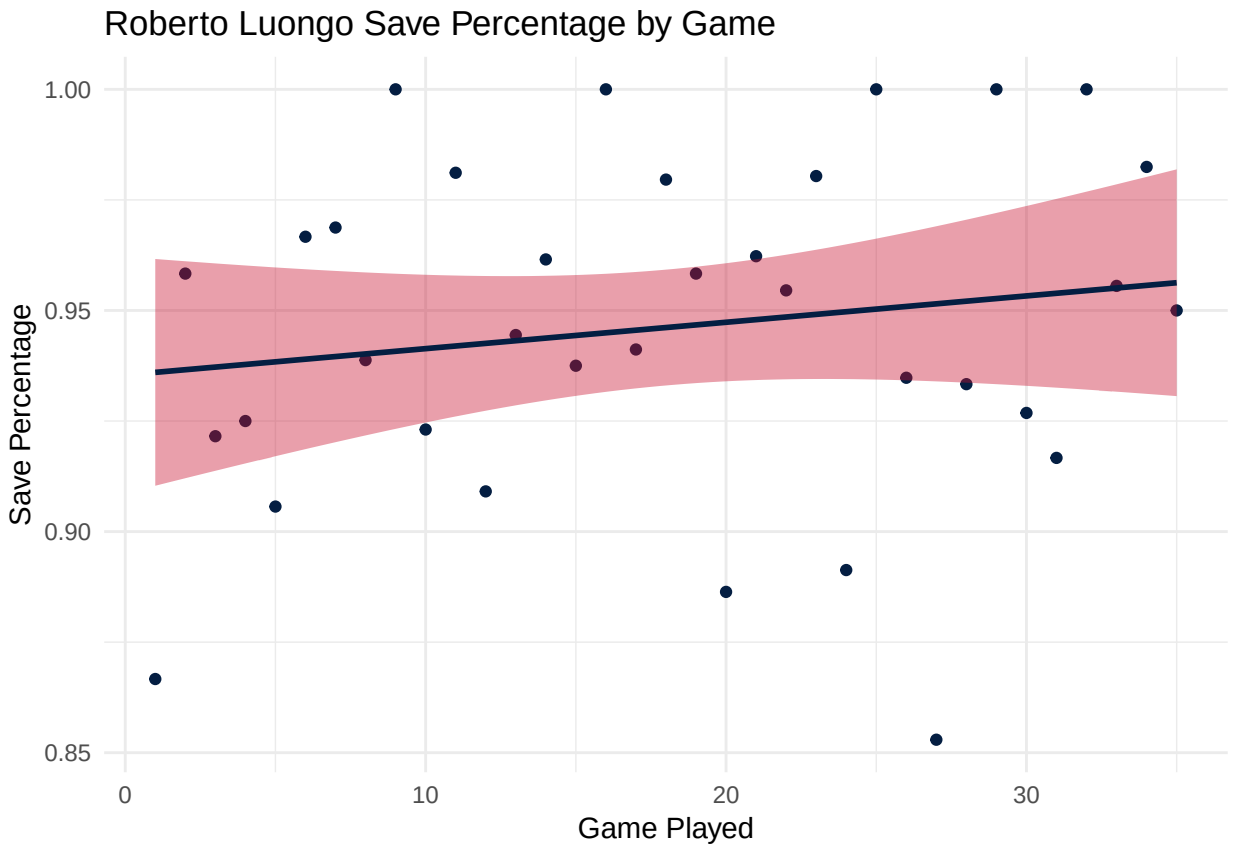


Furthermore, we should be able to graph trends in certain statistics.

```
graph.trend = function(trend, type, primary, secondary, name) {
    frame = data.frame(x = c(1:length(trend)), y = trend)
    disp = paste(name, type, "by Game", sep = " ")
    plot = ggplot(frame) +
        geom_point(aes(x = x, y = y), color = primary) +
        geom_smooth(aes(x = x, y = y), method = "lm", color = primary, fill = secondary) +
        labs(title = disp, x = "Game Played", y = type) +
        theme_minimal()
    plot
}
```

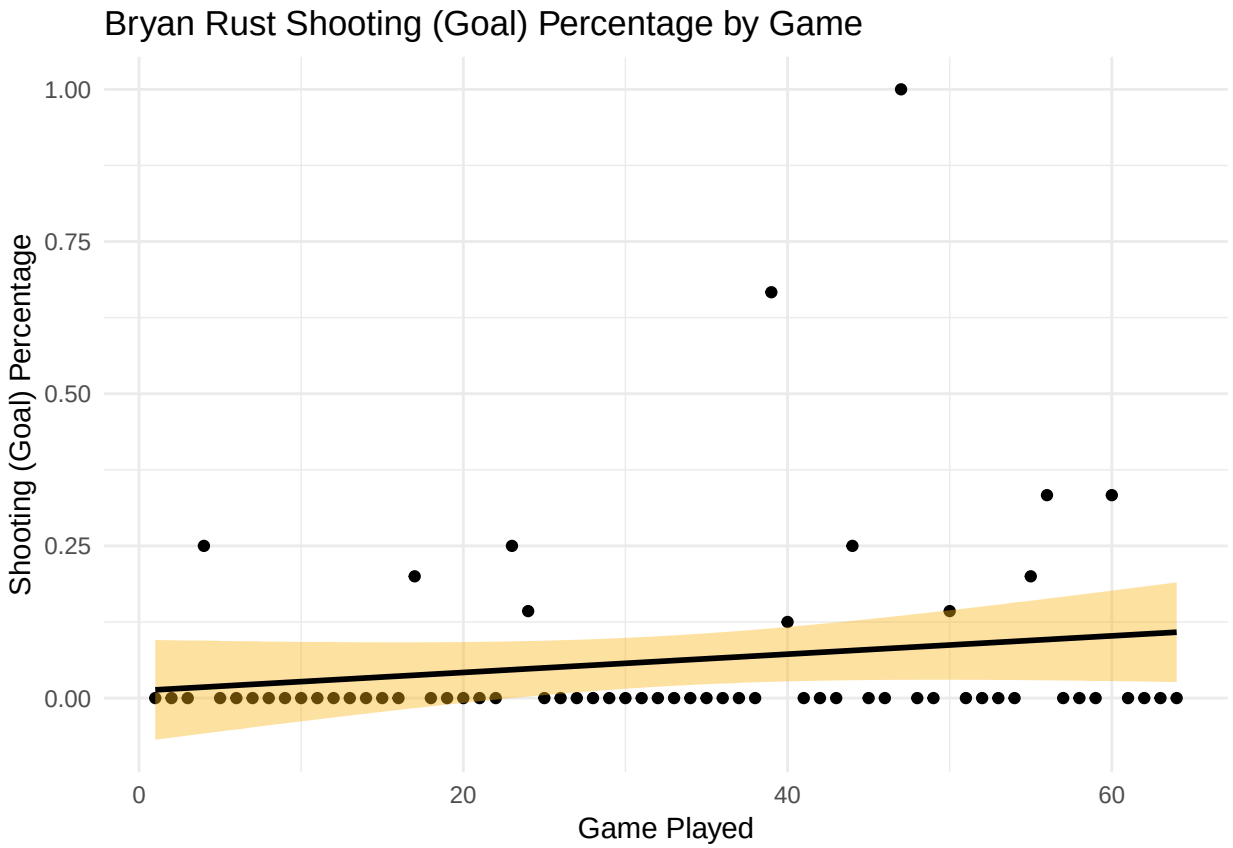
So, let's go back to Luongo and look at his save percentage per game.

```
luongo.game.sp = get.game.save.percent(luongo)
plot = graph.trend(luongo.game.sp, "Save Percentage", "#041E42", "#C8102E", "Roberto Luongo")

plot
```



We can also see the shooter's perspective. Let's look at Rust's shooting percentage (goal percent).

```
rust.game.gp = get.game.goal.percent(rust)
plot = graph.trend(rust.game.gp, "Shooting (Goal) Percentage", "#000000", "#FCB514", "Bryan Rust")

plot
```


Bryan Rust Shooting (Goal) Percentage by Game
Shooting (Goal) Percentage
1.00
0.75
0.50
0.25
0.00
0
20
40
60
Game Played

Analysis

Goalies

Matt Murray

Casey DeSmith

Marc-Andre Fleury

Andrei Vasilevskiy

Carey Price

Braden Holtby

John Gibson

Skaters

Sidney Crosby

Evgeni Malkin

Alex Ovechkin

Nikita Kucherov

Jake Guentzel

Erik Karlsson

Kris Letang