hockeyR: Easy access to detailed NHL play-by-play data

Dan Morse

2022-09-02

Introduction

The sports world seems to delve deeper into statistical analysis with each passing year. Not only are team front offices across major sports hiring more analytics staffers than ever, but fans and beat reporters are becoming more fluent and intrigued by the underlying numbers behind the sport that they love. The key to growing this community of sports-loving statisticians is through easy access to as much data as these leagues will allow. In American football, there is no better example of easy access to data than the `nflfastR` package (Carl and Baldwin 2021) (and its predecessor, `nflscrapr` (Yurko, Ventura, and Horowitz 2018)). But when it comes to hockey, accessing raw, detailed play-by-play data isn't nearly as straightforward. There was once an `nhlscrapr` package, but it was lost when authors Samuel Ventura and Andrew C. Thomas were hired by NHL teams. Both Emmanuel Perry and Josh & Luke Younggren have published their own scrapers in the past as well, but have since taken them down. This package, `hockeyR`, seeks to fill the space with access to not only bare scraping functions, but already compiled play-by-play data that anyone can use.

Loading data with hockeyR

Play-by-play data from the NHL has long been available to the general public through the NHL's Real Time Scoring System (RTSS), but it was up to the end user to scrape that data themselves. With hockeyR, it's already been scraped so the analyst can load the data with a single, simple function call.

```
# install hockeyR 1.0.0 from CRAN
# install.packages("hockeyR")

library(hockeyR)
library(tidyverse)
```

```r
library(ggalluvial)
library(sportyR)
```

```r
pbp <- load_pbp(season = "2021-22")

pbp |>
  select(event, event_team, event_player_1_name, description) |>
  head()
```

```
# A tibble: 6 x 4
  event          event_team          event_player_1_name description
  <chr>          <chr>               <chr>               <chr>
1 Game Scheduled <NA>                <NA>                Game Scheduled
2 Faceoff        Pittsburgh Penguins Jeff.Carter         Jeff Carter faceoff wo~
3 Hit            Tampa Bay Lightning Ondrej.Palat        Ondrej Palat hit Jeff ~
4 Stoppage       <NA>                <NA>                Puck in Netting
5 Faceoff        Tampa Bay Lightning Anthony.Cirelli     Anthony Cirelli faceof~
6 Hit            Tampa Bay Lightning Anthony.Cirelli     Anthony Cirelli hit Ma~
```

Play-by-play data going back to the 2010-11 season is stored in a public GitHub repository. The `load_pbp` function is just an easy way to read the data in from that repository, and it has the advantage of accepting multiple seasons in the `season` argument as well as accepting either the full name of the season (e.g. "2021-22") or just the end-year of the season (e.g. 2022).

The data

The loaded data is event-based and contains 107 variables, including the type of event, the player(s) involved in the event, the time of the game, and the players on the ice during the event. Every "event" during a game is recorded as its own row in the data. There are nine different on-ice events in the data, plus a handful of other game state events (start of period, end of game, etc.).

```r
unique(pbp$event)
```

```
 [1] "Game Scheduled"          "Faceoff"
 [3] "Hit"                     "Stoppage"
 [5] "Shot"                    "Takeaway"
 [7] "Blocked Shot"            "Missed Shot"
 [9] "Giveaway"                "Period End"
```

```
[11] "Goal"                       "Penalty"
[13] "Game End"                   "Official Challenge"
[15] "Shootout Complete"          "Early Intermission Start"
[17] "Early Intermission End"     "Emergency Goaltender"
```

There is also an option when loading the play-by-play data to include shift change events. This nearly doubles the size of the data, and for most analyses isn't necessary, so the default is to exclude those for faster loading and easier manipulating.

A single season's worth of play-by-play data generally includes records of over 100,000 unblocked shots and their locations. A heat map of the shots shows how shooters frequently look to take their shots from as close to the net as possible.

```
shots <- filter(pbp, event_type %in% c("SHOT","MISSED_SHOT","GOAL"))

geom_hockey("nhl") +
  geom_hex(data = shots, aes(x, y), alpha = .7, binwidth = c(5,5), show.legend = FALSE) +
  geom_text(aes(0,-40, label = paste("n shots:",nrow(shots)))) +
  scale_fill_gradient2(low = "white", mid = "#dff5f7", high = "darkred")
```

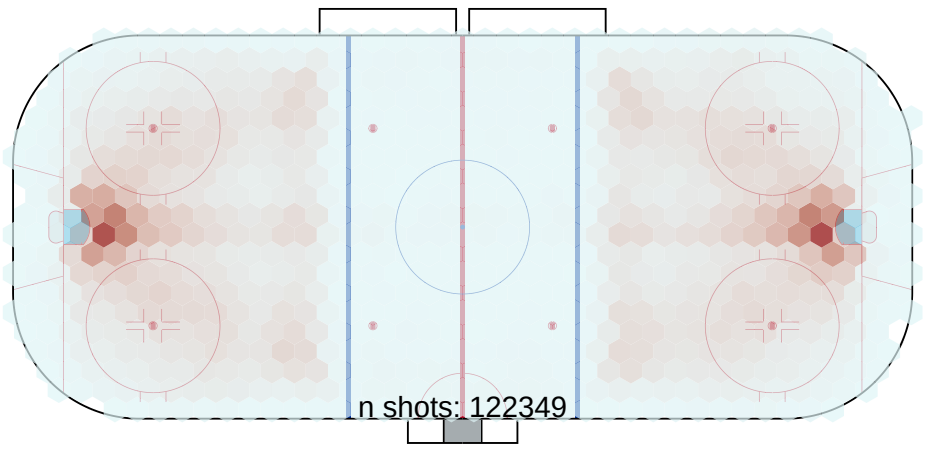

Figure 1: Heatmap of all unblocked shot attempts in the NHL in 2021-22

Investigating player stats

The full play-by-play can be used to calculate basic counting stats, like goals and shots.

```
leaders <- pbp |>
  # get regular season stats, excluding shootout goals
  filter(season_type == "R" & period < 5) |>
  group_by(scorer = event_player_1_name, id = event_player_1_id) |>
  summarize(
    goals = sum(event_type == "GOAL"),
    shot_attempts = sum(event_type %in% c("SHOT","MISSED_SHOT","BLOCKED_SHOT","GOAL")),
    shot_percentage = round(goals/shot_attempts,3),
    .groups = "drop"
  ) |>
  arrange(-goals)

head(leaders)
```

```
# A tibble: 6 x 5
  scorer               id goals shot_attempts shot_percentage
  <chr>             <int> <int>         <int>           <dbl>
1 Auston.Matthews 8479318    60           599           0.1
2 Leon.Draisaitl  8477934    55           482           0.114
3 Chris.Kreider   8475184    52           412           0.126
4 Alex.Ovechkin   8471214    50           642           0.078
5 Kirill.Kaprizov 8478864    47           515           0.091
6 Kyle.Connor     8478398    47           499           0.094
```

Each goal event includes information about who scored both the primary and secondary assists. That information can be leveraged to look at who contributed most often to the leading goal scorers' production. For example, Auston Matthews frequently scored with help from Mitchell Marner.

```
matthews <- pbp |>
  filter(season_type == "R" & period < 5 & event_type == "GOAL") |>
  filter(event_player_1_name == "Auston.Matthews") |>
    group_by(
      a2 = event_player_3_name, a1 = event_player_2_name, g = event_player_1_name
      ) |>
  summarize(n = n(), .groups = "drop")

matthews |>
```

```r
ggplot(aes(y = n, axis1 = a2, axis2 = a1, axis3 = g)) +
ggalluvial::geom_alluvium(width = 1/12, aes(fill = a1), show.legend = FALSE) +
ggalluvial::geom_stratum(width = 1/12, fill = "black", color = "grey") +
geom_label(stat = "stratum", aes(label = after_stat(stratum))) +
scale_x_continuous(
  breaks = 1:3, labels = c("A2","A1","G"), limits = c(.8,3.2)) +
theme_bw() +
theme(
  plot.title = element_text(hjust = 0.5),
  axis.line = element_blank(),
  axis.ticks = element_blank(),
  axis.title = element_blank(),
  axis.text.y = element_blank(),
  panel.grid = element_blank(),
  panel.border = element_blank()
) +
labs(title = "Auston Matthews 2021-22 goal paths")
```

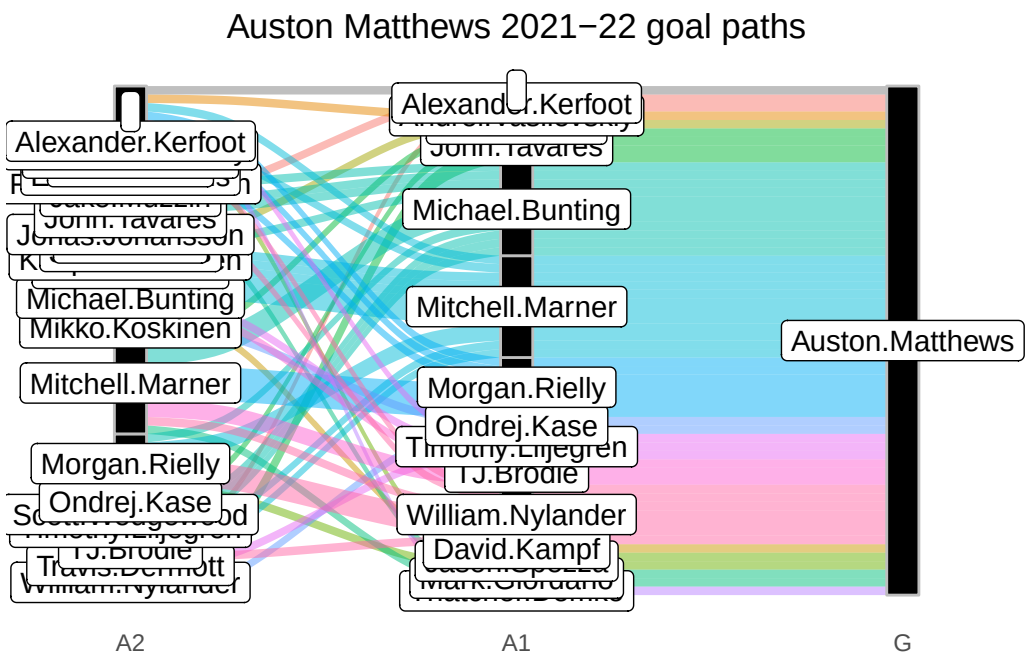

Figure 2: Plot showing who assisted on every Auston Matthews goal in 2021-22

Single game charts

Single games can be pulled out of the full season play-by-play data to make shot charts. These can be filtered using the game ID, if it is known, or as is more often the case by using the game date and the name of either the home or away team. The simplest way to create a shot chart is by leveraging the `sportyR` (Drucker 2021) package. This allows for easy plotting of an NHL ice. The `ggimage` (Yu 2020) package can also be used along with the included team logos data set in `hockeyR` to make a detailed shot chart for any game.

```
#get single game
game <- filter(pbp, game_date == "2021-12-01" & home_abbreviation == "TOR")

# grab team logos & colors
team_logos <- hockeyR::team_logos_colors |>
  filter(team_abbr == unique(game$home_abbreviation) |
           team_abbr == unique(game$away_abbreviation)) |>
  # add in dummy variables to put logos on the ice
  mutate(x = ifelse(full_team_name == unique(game$home_name), 50, -50),
         y = 0)

shots <- game |> filter(event_type %in% c("MISSED_SHOT","SHOT","GOAL")) |>
  # adding team colors
  left_join(team_logos, by = c("event_team_abbr" = "team_abbr"))

# add transparency to logo
transparent <- function(img) {
  magick::image_fx(img, expression = "0.3*a", channel = "alpha")
}

away_abbr <- unique(shots$away_abbreviation)
away_final <- unique(shots$away_final)
home_abbr <- unique(shots$home_abbreviation)
home_final <- unique(shots$home_final)
# create shot plot
geom_hockey("nhl") +
  ggimage::geom_image(
    data = team_logos,
    aes(x = x, y = y, image = team_logo_espn),
    image_fun = transparent, size = 0.22, asp = 2.35
    ) +
  geom_point(
    data = shots, aes(x_fixed, y_fixed), size = 6,
```

```
  color = shots$team_color1, shape = ifelse(shots$event_type == "GOAL", 19, 1)
  ) +
labs(
  title = glue::glue("{unique(game$away_name)} @ {unique(game$home_name)}"),
  subtitle = glue::glue("{unique(game$game_date)}\n
   {away_abbr} {away_final} - {home_final} {home_abbr}")
  ) +
theme(plot.title = element_text(hjust = 0.5),
      plot.subtitle = element_text(hjust = 0.5),
      plot.caption = element_text(hjust = .9))
```

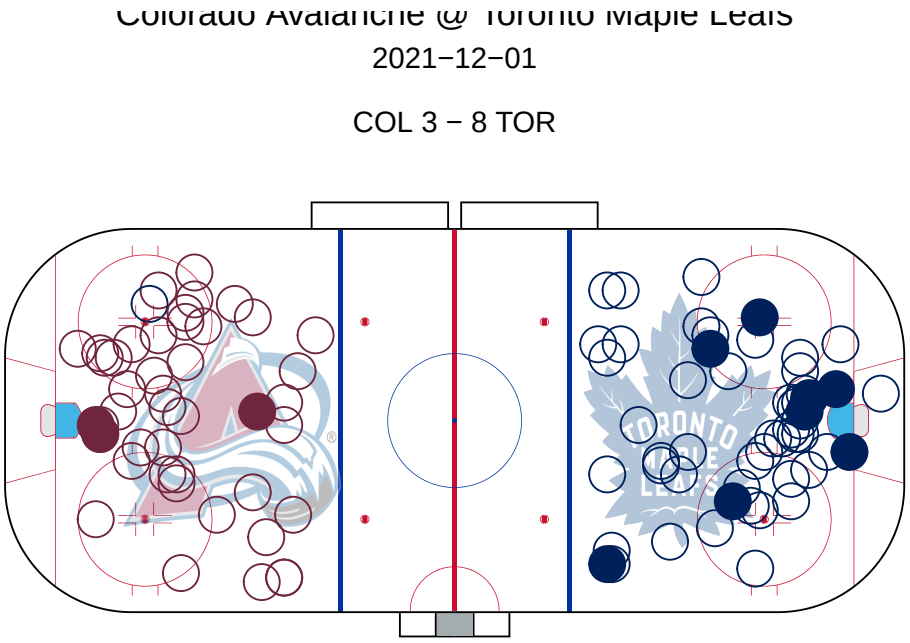

Figure 3: Plot showing unblocked shot locations for a single game

Scraping functions

There are three main scraping functions in `hockeyR` that deal with scraping play-by-play data:

- `scrape_day`: gets all play-by-play data for a given day; this is what is used to update the data repository each night.

- `scrape_season`: gets all play-by-play data for a given season; this is what built the initial database
- `scrape_game`: gets all play-by-play data for a given game; this is where the real meat of all three scraping functions lies

The `scrape_game` function takes a game ID as an argument. Game IDs can be found using the `get_game_ids` function, which returns all game IDs for either a single day or an entire season. With these two functions, it becomes quite simple to scrape play-by-play data for a live game, without the need to wait for the data repository to update at the end of the night.

```
ids <- get_game_ids(day = "2021-10-17")

game_pbp <- scrape_game(ids$game_id[1])

head(game_pbp)
```

```
# A tibble: 6 x 111
     xg event_id event_type event secon~1 event~2 event~3 descr~4 period perio~5
  <dbl>    <dbl> <chr>      <chr> <chr>   <chr>   <chr>   <chr>    <int>   <dbl>
1    NA  2.02e13 GAME_SCHE~ Game~ <NA>    <NA>    <NA>    Game S~      1       0
2    NA  2.02e13 CHANGE     Chan~ <NA>    Dallas~ away    ON: Ry~      1       0
3    NA  2.02e13 CHANGE     Chan~ Line c~ Ottawa~ home    ON: Th~      1       0
4    NA  2.02e13 FACEOFF    Face~ <NA>    Ottawa~ home    Josh N~      1       0
5    NA  2.02e13 STOP       Stop~ <NA>    <NA>    <NA>    Hand P~      1       8
6    NA  2.02e13 FACEOFF    Face~ <NA>    Dallas~ away    Luke G~      1       8
# ... with 101 more variables: period_seconds_remaining <dbl>,
#   game_seconds <dbl>, game_seconds_remaining <dbl>, home_score <dbl>,
#   away_score <dbl>, event_player_1_name <chr>, event_player_1_type <chr>,
#   event_player_2_name <chr>, event_player_2_type <chr>,
#   event_player_3_name <chr>, event_player_3_type <chr>,
#   event_goalie_name <chr>, strength_state <glue>, strength_code <chr>,
#   strength <chr>, game_winning_goal <lgl>, empty_net <lgl>, ...
```

In addition to play-by-play data, hockeyR also provides functions to scrape two other details regarding NHL rosters from the NHL API: `get_draft_class` and `get_current_rosters`. With the `get_current_rosters` function, the user can scrape an up-to-the minute data frame of the current rosters for all 32 NHL teams, as listed on NHL.com.

```
rosters <- get_current_rosters()

rosters |>
```

```
    select(player, player_id, jersey_number, position, full_team_name, everything())
```

```
# A tibble: 845 x 8
   player           player_id jersey_n~1 posit~2 full_~3 posit~4 team_id team_~5
   <chr>                <int>      <int> <chr>   <chr>   <chr>     <int> <chr>
 1 Jonathan Bernier   8473541         45 G       New Je~ G             1 NJD
 2 Reilly Walsh       8480054          8 D       New Je~ D             1 NJD
 3 Brendan Smith      8474090          2 D       New Je~ D             1 NJD
 4 Tomas Tatar        8475193         90 LW      New Je~ F             1 NJD
 5 Erik Haula         8475287         56 LW      New Je~ F             1 NJD
 6 Ondrej Palat       8476292         18 LW      New Je~ F             1 NJD
 7 Dougie Hamilton    8476462          7 D       New Je~ D             1 NJD
 8 Damon Severson     8476923         28 D       New Je~ D             1 NJD
 9 Andreas Johnsson   8477341         11 LW      New Je~ F             1 NJD
10 Mason Geertsen     8477419         55 D       New Je~ D             1 NJD
# ... with 835 more rows, and abbreviated variable names 1: jersey_number,
#   2: position, 3: full_team_name, 4: position_type, 5: team_abbr
```

The `get_draft_class` function allows the user to load the draft selections for every team for a single draft class. The returned data includes just the basics – player name, drafting team, and round and pick number – but it can also return more details such as amateur league, height and weight, and birthplace by setting the `player_details` argument to `TRUE`.[1] This step is necessary in order to get the proper NHL player ID for each player, which then allows for easier joining to previously calculated player stats. For example, here's the top goal scorers in the 2021-22 season among players selected in the 2016 NHL Entry Draft:

```
draft <- get_draft_class(2016, player_details = TRUE)

leaders_2016 <- leaders |>
  left_join(draft, by = c("id" = "player_id")) |>
  filter(!is.na(draft_year))

leaders_2016 |>
  select(scorer, id, goals_2022 = goals, draft_year, round, pick_overall) |>
  head(n = 10)
```

```
# A tibble: 10 x 6
   scorer                id goals_2022 draft_year round pick_overall
```

[1] By default, the `player_details` argument is set to `FALSE` simply because adding the details makes the scrape take ~45 seconds instead of ~1 second.

	<chr>	<int>	<int>	<int>	<chr>	<int>
1	Auston.Matthews	8479318	60	2016	1	1
2	Matthew.Tkachuk	8479314	42	2016	1	6
3	Alex.DeBrincat	8479337	41	2016	2	39
4	Tage.Thompson	8479420	38	2016	1	26
5	Clayton.Keller	8479343	28	2016	1	7
6	Pierre.Luc.Dubois	8479400	28	2016	1	3
7	Jordan.Kyrou	8479385	27	2016	2	35
8	Jesper.Bratt	8479407	26	2016	6	162
9	Patrik.Laine	8479339	26	2016	1	2
10	Brandon.Hagel	8479542	25	2016	6	159

What's next

The next steps for `hockeyR` will be to include the package's own expected goals model – a common model in the NHL statistics world to evaluate how likely any given shot is to result in a goal. With an included expected goals model, the play-by-play data could then have an extra column appended to it to include an expected goal value for every shot attempt. This would allow analysts to calculate expected goals in different situations and time frames without requiring the construction of their own model.

There is also work to be done to scrape games prior to the 2010-11 season. While the RTSS data existed as far back as 2007, the JSON files utilized by `hockeyR` didn't exist until 2010 – so a new HTML scraper is necessary to go back further.

References

Carl, Sebastian, and Ben Baldwin. 2021. "nflfastR: Functions to Efficiently Access NFL Play by Play Data."

Drucker, Ross. 2021. "sportyR: Plot Scaled 'Ggplot' Representations of Sports Playing Surfaces." https://github.com/rossdrucker/sportyR.

Yu, Guangchuang. 2020. "Ggimage: Use Image in 'Ggplot2'." https://CRAN.R-project.org/package=ggimage.

Yurko, Ronald, Samuel Ventura, and Maksim Horowitz. 2018. "nflWAR: A Reproducible Method for Offensive Player Evaluation in Football." https://doi.org/10.48550/ARXIV.1802.00998.