# Duke Sports Analtyics

Jack Lichtenstein

2021-02-02

## Introduction

The goal of this post is to familiarize R programming beginners with some public sports data sources. Specifically, we will look at scraping sports-reference.com 's college basketball data and then making some visualizations. Let's get started!

## Scraping sports-reference.com

We are going to be making use of a bunch of R packages, but we will be using `tidyverse_style()` coding. To begin, let's scrape the Duke basketball game logs off of sports-reference.com using the `rvest` package.

```
# find url of website
url <- "https://www.sports-reference.com/cbb/schools/duke/2021-schedule.html"

xml2::read_html(url) %>%
  # grab the tables
  rvest::html_table(fill = T)
```

```
## [[1]]
##   School Pre 11/30 12/7 12/14 12/21 12/28 1/4 1/11 1/18 1/25 2/1
## 1   Duke   9     6   10    21    20    20  21   19    -    -   -
##
## [[2]]
##     G              Date   Time Type              Opponent    Conf   Tm Opp OT
## 1   1 Sat, Nov 28, 2020  2:00p  REG          Coppin State    MEAC W 81  71 NA
## 2   2  Tue, Dec 1, 2020  7:30p  REG    Michigan State (8) Big Ten L 69  75 NA
## 3   3  Fri, Dec 4, 2020  7:00p  REG            Bellarmine   A-Sun W 76  54 NA
## 4   4  Tue, Dec 8, 2020  9:30p  REG          Illinois (6) Big Ten L 68  83 NA
## 5   5 Wed, Dec 16, 2020  9:00p  REG @          Notre Dame     ACC W 75  65 NA
## 6   6  Wed, Jan 6, 2021  8:30p  REG        Boston College     ACC W 83  82 NA
## 7   7  Sat, Jan 9, 2021 12:00p  REG          Wake Forest     ACC W 79  68 NA
## 8   8 Tue, Jan 12, 2021  7:00p  REG @  Virginia Tech (20)     ACC L 67  74 NA
## 9   9 Tue, Jan 19, 2021  9:00p  REG @          Pittsburgh     ACC L 73  79 NA
## 10 10 Sat, Jan 23, 2021  4:00p  REG @          Louisville     ACC L 65  70 NA
## 11 11 Tue, Jan 26, 2021  9:00p  REG          Georgia Tech     ACC W 75  68 NA
## 12 12 Sat, Jan 30, 2021 12:00p  REG              Clemson     ACC W 79  53 NA
## 13 13  Mon, Feb 1, 2021  7:00p  REG @          Miami (FL)     ACC L 75  77 NA
## 14 14  Sat, Feb 6, 2021  6:00p  REG        North Carolina     ACC   NA  NA NA
## 15 15  Tue, Feb 9, 2021  4:00p  REG            Notre Dame     ACC   NA  NA NA
```

```
## 16 16 Sat, Feb 13, 2021  4:00p  REG @ North Carolina State     ACC   NA  NA NA
## 17 17 Wed, Feb 17, 2021  8:30p  REG @          Wake Forest     ACC   NA  NA NA
## 18 18 Sat, Feb 20, 2021  8:00p  REG          Virginia (14)     ACC   NA  NA NA
## 19 19 Mon, Feb 22, 2021  7:00p  REG              Syracuse     ACC   NA  NA NA
## 20 20 Sat, Feb 27, 2021  6:00p  REG            Louisville     ACC   NA  NA NA
## 21 21  Tue, Mar 2, 2021  7:00p  REG @         Georgia Tech     ACC   NA  NA NA
## 22 22  Sat, Mar 6, 2021  6:00p  REG @       North Carolina     ACC   NA  NA NA
##     W  L Streak                                 Arena
## 1   1  0    W 1               Cameron Indoor Stadium
## 2   1  1    L 1               Cameron Indoor Stadium
## 3   2  1    W 1               Cameron Indoor Stadium
## 4   2  2    L 1               Cameron Indoor Stadium
## 5   3  2    W 1 Purcell Pavilion at the Joyce Center
## 6   4  2    W 2               Cameron Indoor Stadium
## 7   5  2    W 3               Cameron Indoor Stadium
## 8   5  3    L 1                     Cassell Coliseum
## 9   5  4    L 2               Petersen Events Center
## 10  5  5    L 3                      KFC Yum! Center
## 11  6  5    W 1               Cameron Indoor Stadium
## 12  7  5    W 2               Cameron Indoor Stadium
## 13  7  6    L 1                    BankUnited Center
## 14 NA NA
## 15 NA NA
## 16 NA NA
## 17 NA NA
## 18 NA NA
## 19 NA NA
## 20 NA NA
## 21 NA NA
## 22 NA NA
```

When we scrape the tables off the site, notice that we actually receive a list of two tables. We are interested in the second table, so we will use `purrr::pluck()` to "pluck" the second table from the list. After some additional cleaning, we are left with the following dataframe from the 2020-21 season:

```
xml2::read_html(url) %>%
  # grab the tables
  rvest::html_table(fill = T) %>%
  # pluck the second table from the list of tables
  purrr::pluck(2) %>%
  # clean column names to snake_case
  janitor::clean_names() %>%
  # convert to tibble
  dplyr::as_tibble() %>%
  # grab only what we want
  dplyr::transmute(
    game_number = g,
    date,
    result = x_2,
    opponent,
    team_score = tm,
    opp_score = opp
  ) %>%
  # get rid of mid-table headings
```

```
  dplyr::filter(date != "Date", opponent != "Opponent") %>%
  # convert to numeric values
  dplyr::mutate(dplyr::across(
    c(game_number, team_score, opp_score),
    suppressWarnings(as.numeric)
  ))
```

```
## # A tibble: 22 x 6
##    game_number date              result opponent           team_score opp_score
##          <dbl> <chr>             <chr>  <chr>                   <dbl>     <dbl>
##  1           1 Sat, Nov 28, 2020 W      Coppin State               81        71
##  2           2 Tue, Dec 1, 2020  L      Michigan State (8)         69        75
##  3           3 Fri, Dec 4, 2020  W      Bellarmine                 76        54
##  4           4 Tue, Dec 8, 2020  L      Illinois (6)               68        83
##  5           5 Wed, Dec 16, 2020 W      Notre Dame                 75        65
##  6           6 Wed, Jan 6, 2021  W      Boston College             83        82
##  7           7 Sat, Jan 9, 2021  W      Wake Forest                79        68
##  8           8 Tue, Jan 12, 2021 L      Virginia Tech (20)         67        74
##  9           9 Tue, Jan 19, 2021 L      Pittsburgh                 73        79
## 10          10 Sat, Jan 23, 2021 L      Louisville                 65        70
## # ... with 12 more rows
```

What if we wanted data dating all the way back to Coach K's first season? We can make a function for this! Using the workflow above, all we need to do is pass in a url for each season we want data from and extract the same information.

```
# function to pull game logs from sports-reference.com
scrape_game_logs <- function(url) {
  xml2::read_html(url) %>%
    # grab the tables
    rvest::html_table(fill = T) %>%
    # pluck the second table
    purrr::pluck(2) %>%
    # clean column names
    janitor::clean_names() %>%
    # convert to tibble
    dplyr::as_tibble() %>%
    # grab only what we want
    dplyr::transmute(
      game_number = g,
      date,
      result = x_2,
      opponent,
      team_score = tm,
      opp_score = opp
    ) %>%
    # get rid of mid-table headings
    dplyr::filter(date != "Date", opponent != "Opponent") %>%
    # convert to numeric values
    dplyr::mutate(dplyr::across(
      c(game_number, team_score, opp_score),
      suppressWarnings(as.numeric)
```

```
    ))
}
```

Next, we will use `purrr::map()` to scrape for all seasons dating back to 1980-81.

```
# dataframe of seasons from 1981-2021
game_logs <- dplyr::tibble(
  season_url = 1981:2021,
  season = paste0(season_url - 1, "-", stringr::str_sub(season_url, start = 3)),
  url = paste0("https://www.sports-reference.com/cbb/schools/duke/", season_url, "-schedule.html")
) %>%
  dplyr::mutate(data = purrr::map(url, purrr::possibly(scrape_game_logs, otherwise = NA)))

head(game_logs)
```

```
## # A tibble: 6 x 4
##   season_url season  url                                              data
##        <int> <chr>   <chr>                                            <list>
## 1       1981 1980-81 https://www.sports-reference.com/cbb/schools/~ <tibble [30~
## 2       1982 1981-82 https://www.sports-reference.com/cbb/schools/~ <tibble [27~
## 3       1983 1982-83 https://www.sports-reference.com/cbb/schools/~ <tibble [28~
## 4       1984 1983-84 https://www.sports-reference.com/cbb/schools/~ <tibble [34~
## 5       1985 1984-85 https://www.sports-reference.com/cbb/schools/~ <tibble [31~
## 6       1986 1985-86 https://www.sports-reference.com/cbb/schools/~ <tibble [40~
```

Great! Now we have each seasons data nested as a list inside our dataframe. In order to pull the game logs out of the list, we will use the `tidyr::unnest()` function.

```
game_logs_unnest <- game_logs %>%
  # don't need the url anymore
  dplyr::select(-url, -season_url) %>%
  # unnest the data
  tidyr::unnest(data) %>%
  # keep only the games that have completed
  dplyr::filter(!is.na(team_score))

head(game_logs_unnest)
```

```
## # A tibble: 6 x 7
##   season  game_number date          result opponent         team_score opp_score
##   <chr>         <dbl> <chr>         <chr>  <chr>                 <dbl>     <dbl>
## 1 1980-81           1 Sat, Nov 29,~ W      Stetson                  67        49
## 2 1980-81           2 Tue, Dec 2, ~ W      South Florida            83        72
## 3 1980-81           3 Fri, Dec 5, ~ L      North Carolina ~         76        78
## 4 1980-81           4 Sat, Dec 6, ~ L      North Carolina ~         60        74
## 5 1980-81           5 Tue, Dec 9, ~ W      Vanderbilt               72        69
## 6 1980-81           6 Fri, Dec 12,~ L      Virginia (6)             79        91
```

At this point, we have scraped the results of all 1407 Duke games in the Coach K era!

## Basic modeling

Let's tackle a simple, yet interesting question through some basic modeling techniques. The simple question is: What if we wanted to predict next seasons win percentage from this years data? Let's determine whether the current season win percentage or current season point differential is a better predictor of next season win percentage.

First, we need to group by season and find each seasons win percentage and point differential. Then, we need to compare these values to prior seasons. To do this, we will use `dplyr::lag()`.

```
seasons_summarized <- game_logs_unnest %>%
  # group by season
  dplyr::group_by(season) %>%
  # summarise relevant stats
  dplyr::summarise(
    games = n(),
    wins = sum(team_score > opp_score),
    win_pct = wins / games,
    point_diff = sum(team_score - opp_score)
  ) %>%
  dplyr::ungroup() %>%
  dplyr::select(season, point_diff, win_pct) %>%
  # lag to find previous metrics
  dplyr::mutate(
    prev_win_pct = dplyr::lag(win_pct, n = 1),
    prev_point_diff = dplyr::lag(point_diff, n = 1)
  )

head(seasons_summarized)
```

```
## # A tibble: 6 x 5
##   season  point_diff win_pct prev_win_pct prev_point_diff
##   <chr>        <dbl>   <dbl>        <dbl>           <dbl>
## 1 1980-81         92   0.567       NA                 NA
## 2 1981-82       -161   0.370        0.567             92
## 3 1982-83        -98   0.393        0.370           -161
## 4 1983-84        168   0.706        0.393            -98
## 5 1984-85        341   0.742        0.706            168
## 6 1985-86        504   0.925        0.742            341
```

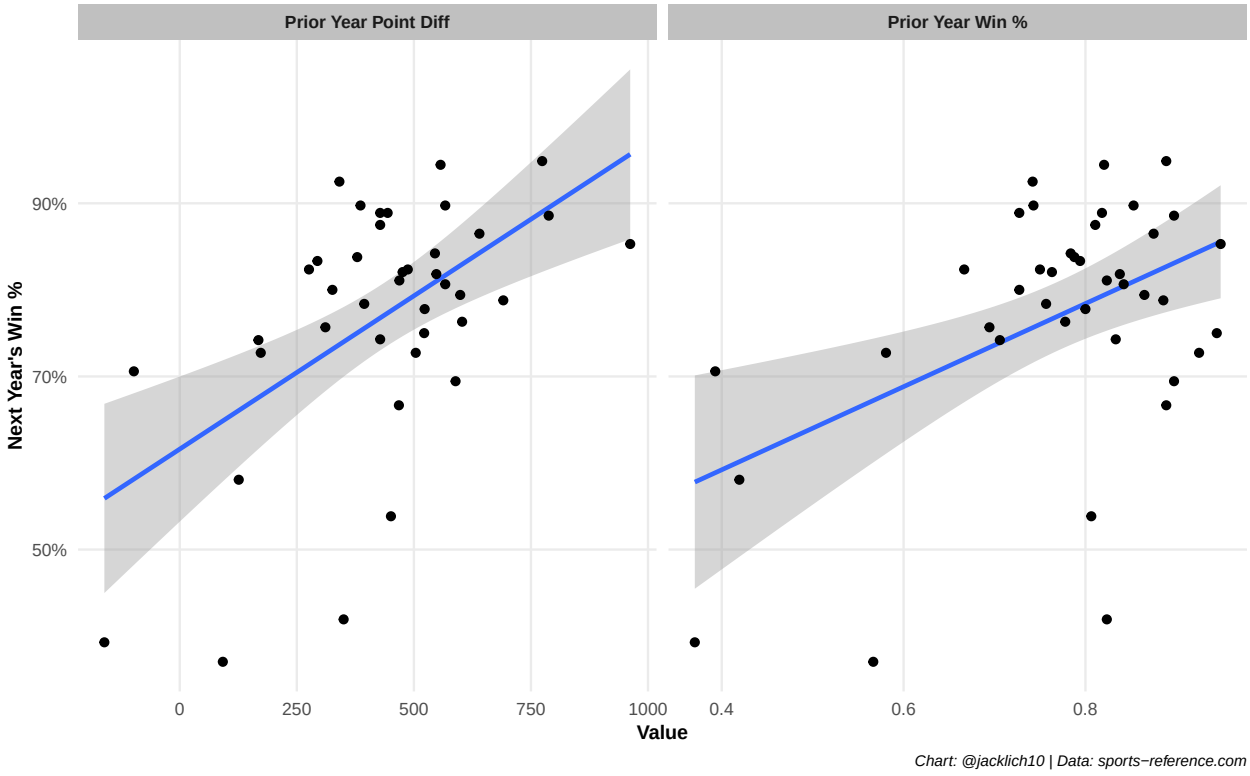
Next, we need to "pivot" the dataframe into a "long" format so that we can compare both metrics to next year's win percentage. To do so, we will use `tidyr::pivot_longer()`.

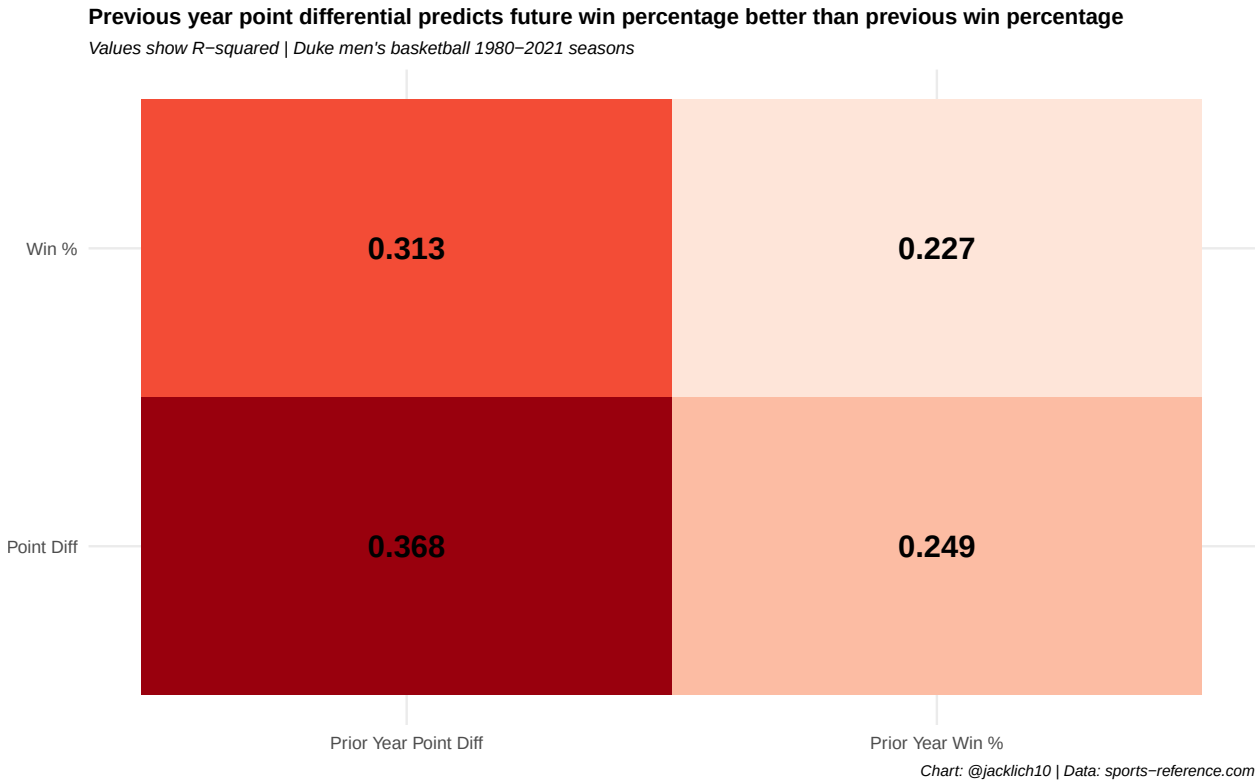
```
long <- seasons_summarized %>%
  # pivot both metrics to long format
  tidyr::pivot_longer(
    cols = c(prev_win_pct, prev_point_diff),
    names_to = "predictor",
    names_prefix = "prev_"
  ) %>%
  # make prettier labels
  dplyr::mutate(predictor = ifelse(predictor == "win_pct",
    "Prior Year Win %", "Prior Year Point Diff"
```

```
  ))

head(long)
```

```
## # A tibble: 6 x 5
##   season  point_diff win_pct predictor                value
##   <chr>        <dbl>   <dbl> <chr>                    <dbl>
## 1 1980-81         92   0.567 Prior Year Win %        NA
## 2 1980-81         92   0.567 Prior Year Point Diff   NA
## 3 1981-82       -161   0.370 Prior Year Win %         0.567
## 4 1981-82       -161   0.370 Prior Year Point Diff   92
## 5 1982-83        -98   0.393 Prior Year Win %         0.370
## 6 1982-83        -98   0.393 Prior Year Point Diff -161
```

We have essentially "doubled" the number of observations in the dataframe by pivoting from "wide" to "long". This makes for easier comparisons between prior year win percentage/point differential and next year's win percentage. Let's plot these relationships:



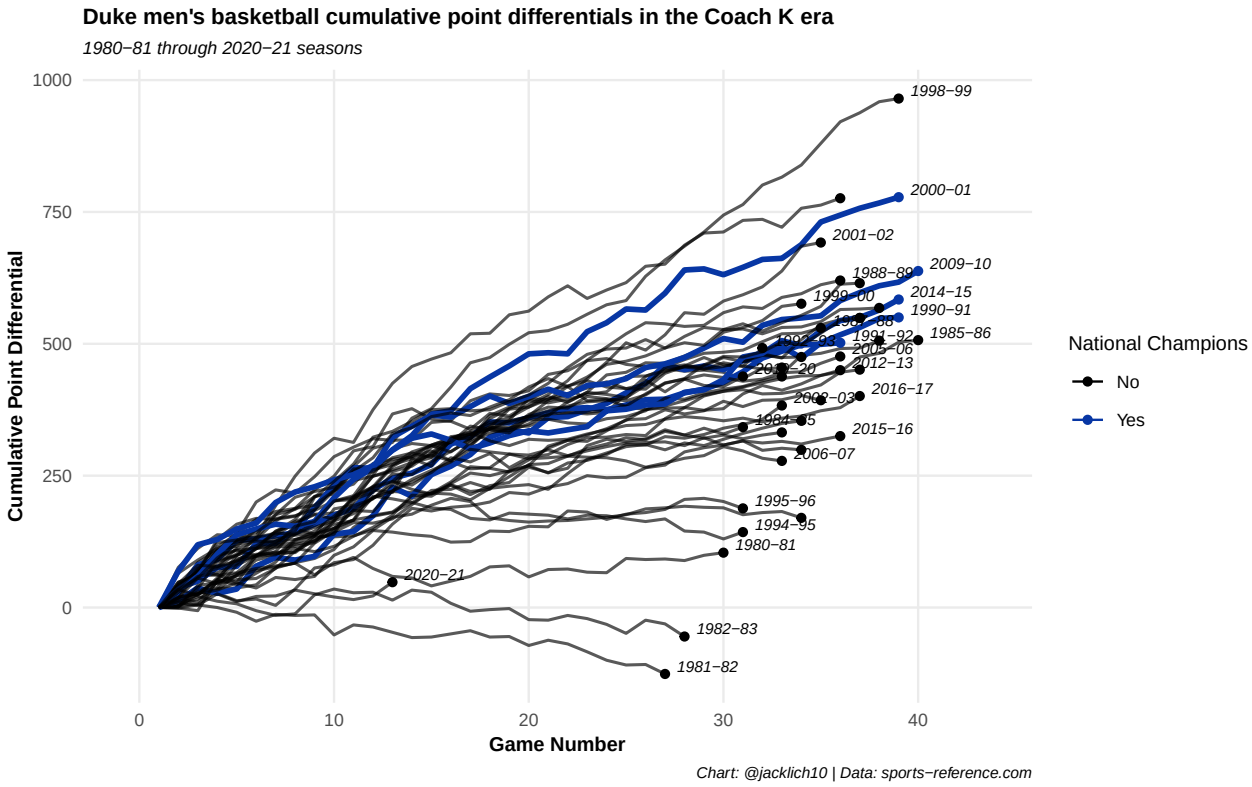



Awesome! So we have found some evidence that prior year's point differential is a better predictor of next year's win percentage than prior year's win percentage! This is a replication of a commonly found idea across many sports.

## Data visualization

Let's make a visualization! One idea I have is to visualize the strength of a Duke team through their cumulative point differential. To find cumulative point differential by season, we need to `dplyr::group_by()` season and use `cumsum()`.

```
game_logs_unnest <- game_logs_unnest %>%
  # add championship years
  dplyr::mutate(
    point_diff = team_score - opp_score,
    champions = ifelse(season %in% c("1990-91", "1991-92", "2000-01", "2009-10", "2014-15"),
      "Yes", "No"
    )
  ) %>%
  dplyr::filter(!is.na(point_diff)) %>%
  # group by season and find cumulative point differential by season
  dplyr::group_by(season) %>%
  dplyr::arrange(game_number) %>%
  dplyr::mutate(cumulative_pt_diff = dplyr::lag(cumsum(point_diff), default = 0)) %>%
  dplyr::ungroup()
```




Duke basketball in the Coach K era has been nothing short of dominant. It's also interesting that you can make an argument that Duke's best team (1998-99) didn't even win the National Championship.

## Explore the four factors

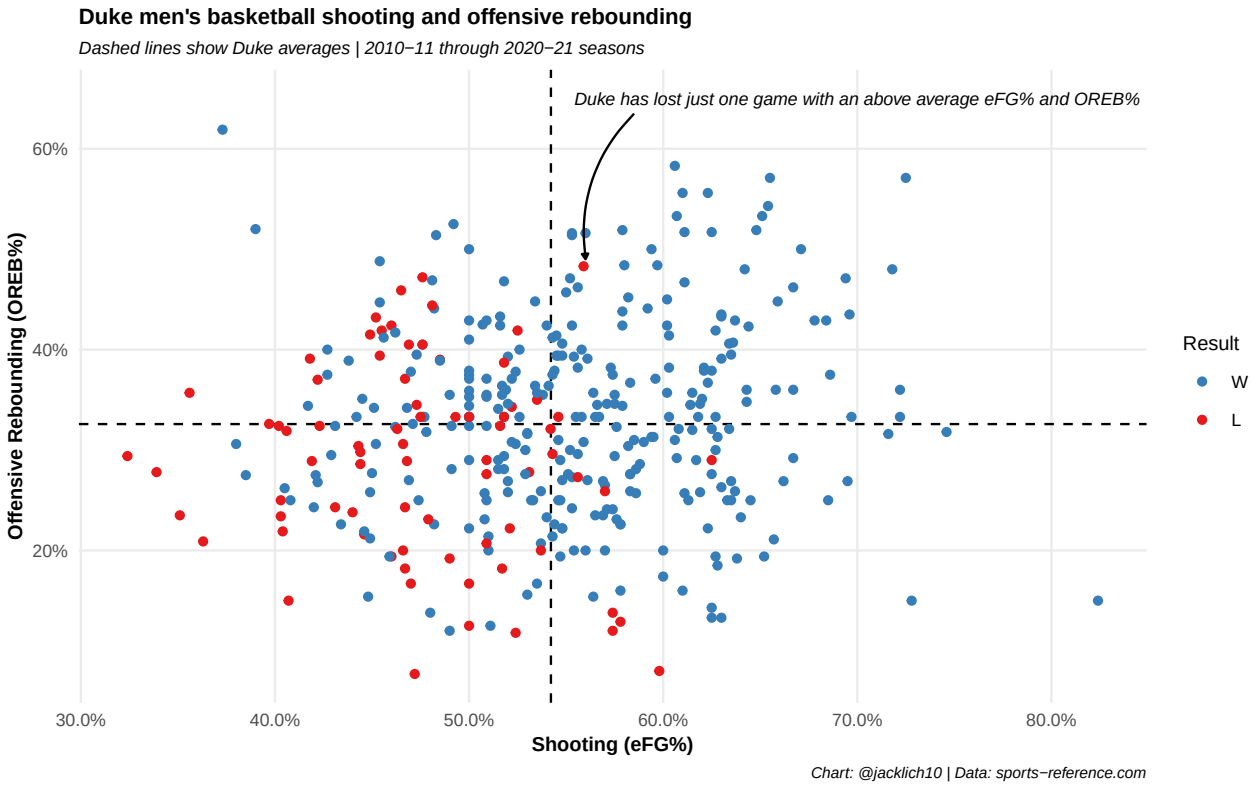
Lastly, let's explore the "four factors" of basketball, coined by Dean Oliver. We will construct a similar function to scrape these statistics from sports-reference.com. The function is below:

```
# we can make this a function to grab from multiple seasons!
scrape_four_factors <- function(url) {
  xml2::read_html(url) %>%
    # grab the tables
    rvest::html_table(fill = T) %>%
    # pluck the first table
    purrr::pluck(1) %>%
    # make the first row the column names
    janitor::row_to_names(row_number = 1) %>%
    # clean column names
    janitor::clean_names() %>%
    # convert to tibble
    dplyr::as_tibble() %>%
    # keep only rows where there is a score
    dplyr::filter(tm != "", opp_2 != "", opp != "Opp") %>%
    # grab only what we need (four factors)
    dplyr::transmute(
      date = as.Date(date),
      opponent = opp,
```

```
      team_score = tm,
      opp_score = opp_2,
      # true_shooting = ts_percent,
      eff_fg_pct = e_fg_percent,
      tov_pct = tov_percent,
      oreb_pct = orb_percent,
      # ft_rate = ft_fga,
      ft_rate = f_tr
    ) %>%
    # change values to numeric and on 0-1 scale
    dplyr::mutate(
      dplyr::across(c(everything(), -date, -opponent), suppressWarnings(as.numeric)),
      dplyr::across(tov_pct:oreb_pct, ~ . / 100)
    )
}

# sports-reference.com has data dating back to 2010-11 season
four_factors <- dplyr::tibble(
  season_url = 2011:2021,
  season = paste0(season_url - 1, "-", stringr::str_sub(season_url, start = 3)),
  url = paste0("https://www.sports-reference.com/cbb/schools/duke/", season_url, "-gamelogs-advanced.ht
) %>%
  dplyr::mutate(data = purrr::map(url, purrr::possibly(scrape_four_factors, otherwise = NA)))

four_factors_unnest <- four_factors %>%
  # don't need the url anymore
  dplyr::select(-url, -season_url) %>%
  # unnest the data
  tidyr::unnest(data) %>%
  dplyr::mutate(
    result = ifelse(team_score > opp_score, "W", "L"),
    result = forcats::fct_rev(result)
  )

head(four_factors_unnest)
```

```
## # A tibble: 6 x 10
##   season date       opponent team_score opp_score eff_fg_pct tov_pct oreb_pct
##   <chr>  <date>     <chr>         <dbl>     <dbl>      <dbl>   <dbl>    <dbl>
## 1 2010-~ 2010-11-14 Princet~         97        60      0.662   0.165    0.269
## 2 2010-~ 2010-11-16 Miami (~         79        45      0.5     0.11     0.333
## 3 2010-~ 2010-11-19 Colgate         110        58      0.556   0.117    0.462
## 4 2010-~ 2010-11-22 Marquet~         82        77      0.594   0.214    0.313
## 5 2010-~ 2010-11-23 Kansas ~         82        68      0.537   0.169    0.207
## 6 2010-~ 2010-11-27 Oregon           98        71      0.637   0.203    0.259
## # ... with 2 more variables: ft_rate <dbl>, result <fct>
```

Great, we now have Duke's offensive four factors since 2010-11 (the first season sports-reference.com) has data. Let's do some basic plotting:




Duke has lost only one game in which they had an above average (relative to Duke) effective field goal percentage and above average offensive rebounding rate! The four factors are amazing *descriptive* measures, but can also be leveraged for *prediction*.

Here is all of the four factor data in tabular form. Click on a column name to sort! Code for the table can be found in the source code.

| Season | Date ↓ | | Opponent | Result | Score | eFG% | TO% | OREB% | FTA/FGA |
|---|---|---|---|---|---|---|---|---|---|
| 2020-21 | February 1, 2021 | | Miami (FL) | L | 75 - 77 | 0.5 | 0.158 | 0.333 | 0.206 |
| 2020-21 | January 30, 2021 | | Clemson | W | 79 - 53 | 0.541 | 0.115 | 0.364 | 0.279 |
| 2020-21 | January 26, 2021 | | Georgia Tech | W | 75 - 68 | 0.491 | 0.16 | 0.281 | 0.379 |
| 2020-21 | January 23, 2021 | | Louisville | L | 65 - 70 | 0.537 | 0.205 | 0.2 | 0.167 |
| 2020-21 | January 19, 2021 | | Pitt | L | 73 - 79 | 0.443 | 0.122 | 0.304 | 0.271 |
| 2020-21 | January 12, 2021 | | Virginia Tech | L | 67 - 74 | 0.467 | 0.141 | 0.182 | 0.25 |
| 2020-21 | January 9, 2021 | | Wake Forest | W | 79 - 68 | 0.582 | 0.176 | 0.452 | 0.164 |
| 2020-21 | January 6, 2021 | | Boston College | W | 83 - 82 | 0.507 | 0.141 | 0.425 | 0.373 |
| 2020-21 | December 16, 2020 | | Notre Dame | W | 75 - 65 | 0.583 | 0.114 | 0.259 | 0.083 |
| 2020-21 | December 8, 2020 | | Illinois | L | 68 - 83 | 0.44 | 0.139 | 0.238 | 0.239 |
| | | | | | Average | 54.2% | 13.9% | 32.6% | 38.6% |

1–10 of 373 rows Previous 1 2 3 4 5 ... 38 Next

## Wrap-up

Hopefully, this piece served as a helpful introduction into leveraging publicly available college basketball data from sports-reference.com into interesting visualizations and insights. Don't forget that while I chose to only scrape Duke basketball statistics, the functions we created can be easily modified to scrape other team's statistics (or many team's statistics at once). Happy R programming!