Group Paper

Data Programming in R

Group O - Sam Anderson, David Sorensen, Will Cobian-Ruelas

Accuracy of Rankers for College Football Recruiting

**Introduction**

**Getting the Data**

In order for us to be as accurate as possible in judging the recruit ranking process, we had to get as much data over as many years as possible. We need quantifiable football statistics that we could use to calculate the performance of players over the course of their college career. Unfortunately, the majority of the user friendly data is sold at extremely high prices to presumably be used as a betting aid for sports gamblers. Luckily, we were able to find a complete set of statistics for years 2005-2013 that was archived from CFBStats.com before they made their data private and expensive. The data package was exhaustive set of spreadsheets that broke down each season into perfect detail. It used keys for players and games that allowed us to calculate per-game points earned as well as player assignment. Although it had the season broken down into incredible detail, we were hoping for statistics that covered more than just those years. Ultimately, we determined that we could reasonably perform our analysis using years provided as none of us were eager to pay over $500 for an updated set.

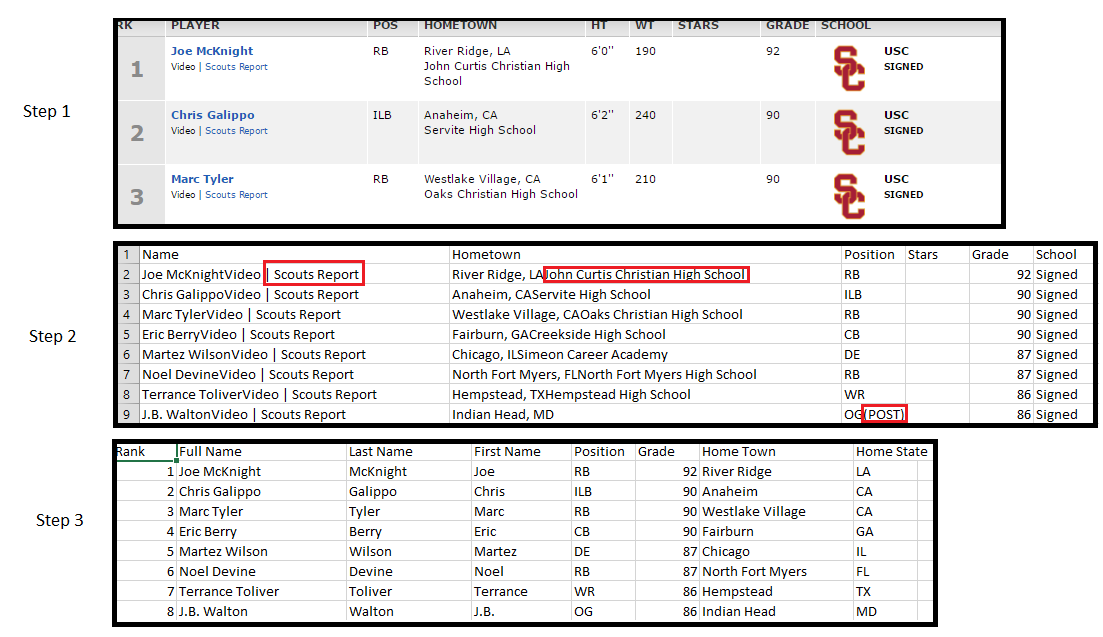
After obtaining the statistics data, we needed to retrieve the second piece of the puzzle: the recruits. We got nowhere nearly as lucky in acquiring this data in a set of spreadsheets, and decided that we would have to result in scraping websites for the data that we needed. ESPN has a Recruiting Database that goes back to 2007 in which every recruit, graded or not, is present. They display this data by showing 30 recruits per page, at which point you have to click “Next Page” to see the next 30. Considering some years had over 11,000 recruits, spread over 480 pages, we knew that this was going to be an intensive process.

We originally started scripting in Perl, and tried various approaches (lynx, HTTP:Mechanize), but the process was so granular and class oriented that that we knew we would end up looking at days of processing time going this route. As fate would have it, in the next R class, we talked about read\_html and parsing tables from html pages.

After investigating ESPN’s Database site, we learned that we could manipulate the URL at two points in order to get a fresh set of recruits: year and page number. By creating a loop of html\_read processes, we could simply increment the page count and keep appending the new table data to an existing dataframe. Since we had several years’ worth of data to gather, we decided into reading the tables in parallel by using the foreach package. This gave us an amount of threads equal to the number of CPU cores minus one. Each thread would independently retrieve an html page and parse out the table. The **.inorder = TRUE** and **.combine = “rbind”** arguments made sure the pages were returned in order, and that the result of the foreach would give us a dataframe. The parallelization improvement was so big that we ended up using it to read the CSV files during the program runtime as well.

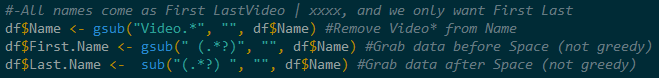
**Cleaning the Data**

Scraping the recruiting data from ESPN got us about 60% of the way to usable data.

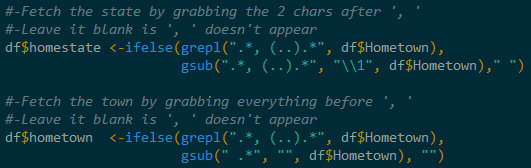


Cleaning the recruiting data was a little more difficult than anticipate because we had several fringe cases to work around. We decided to do all of the cleaning in R by using a combination of **grep**, **grepl**, **gsub**, and **regular expressions**.

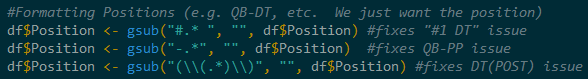
1. The first and easiest step was removing “Video | Scouts Report” from the name. Additionally, we wanted to add two new columns: First Name and Last Name.



1. The second step was fixing the Hometown column. Some rows had City, State School, some just City and State, and some just school. We wanted to end up with two columns: Home Town and Home State. We were able to grab the state by first determining its existence (check for 2 characters following “, “. If it didn’t exist, we left it blank. The same logic was used on the left side of the comma for Hometown.

****

1. The way the position column had been populated, there was extra information in some cases to state whether a Quarterback was a Pocket Passer or a Dual Threat. It also indicated different attributes in parenthesis about (POST) or transfer, or indicated the position number of a certain player. Since that wasn’t part of our analysis, we needed to strip that down base position.



The end result was a set of tidy yearly recruit data that we could easily use to match up with our statistics.

**Functions**

1. **getRecruits()**
   1. This is the function in RecruitingScraper.R that retrieves the recruiting ranking from the ESPN Recruiting Database and then builds and cleans yearly dataframes for use in the program.
2. **getCombinedRecruits()**
   1. This function reads the CSV files that were created by **getRecruits** and returns a single dataframe of each recruit. It also checks for duplicate players and retains only the most recent, as we discovered players can be ranked more than once if they attend a Junior College or other non-FBS school.
3. **getCombinedPlayers()**
   1. This is the first time we read our statistics from CFBStats.com, and gives us our critical component for our recruits: the Player Code. The Player Code is a unique number that is assigned to each player and is used throughout the spreadsheets. The spreadsheets that we’re reading simply a list of a players on a football roster. It does not mean they played any games; simply that they were on an FBS team. This is an important distinction to make, because as we look at the number of years played and average points per year, we don’t want to confuse the year a player was rostered as an active year by default.
4. **dfRecruitStats()**
   1. Now that we have the Player Code, we can read the game statistics files to get information regarding player performance. Since we’re only looking at offensive players, the game statistics that we are using to quantify performance are Rushing Yards, Rushing Touchdowns, Passing Yards, Passing Touchdowns, Intercepted Passes, Receiving Yards, Receiving Touchdowns, Kickoff Return Yards, Kickoff Return Touchdowns, Punt Return Yards, Punt Return Touchdowns, Miscellaneous Returning Yards, Miscellaneous Returning Touchdowns and Successful Two-Point Conversion.
   2. By using the Fantasy Football point assignment, we calculate each player’s points per game, and then return a dataframe that consists of the amount of points each of our recruits earned per year. These are considered active years, and are used in our calculations and Average Points per Year.
5. **createYearlyPlots()**
   1. This function creates and saves various plots related to overall yearly statistics as well as position specific information. This gives a good insight into how the ranking difference changes against time as well as position.
6. **createCareerPlots()**
   1. This function breaks the players out of their recruiting classes to give us a bigger picture of overall trends. We again break it up by overall ranking differences and position ranking differences, but we don’t separate it by year. These plots also use variance, which can be positive or negative, meaning we can see if one group is over/underrated in comparison to other positions.
7. **plotMeanDifference()**
   1. Bringing the program to a close, this creates plots to summarize our findings by averaging the differences for classes and positions. This allows us to see the average difference across all recruits by year.

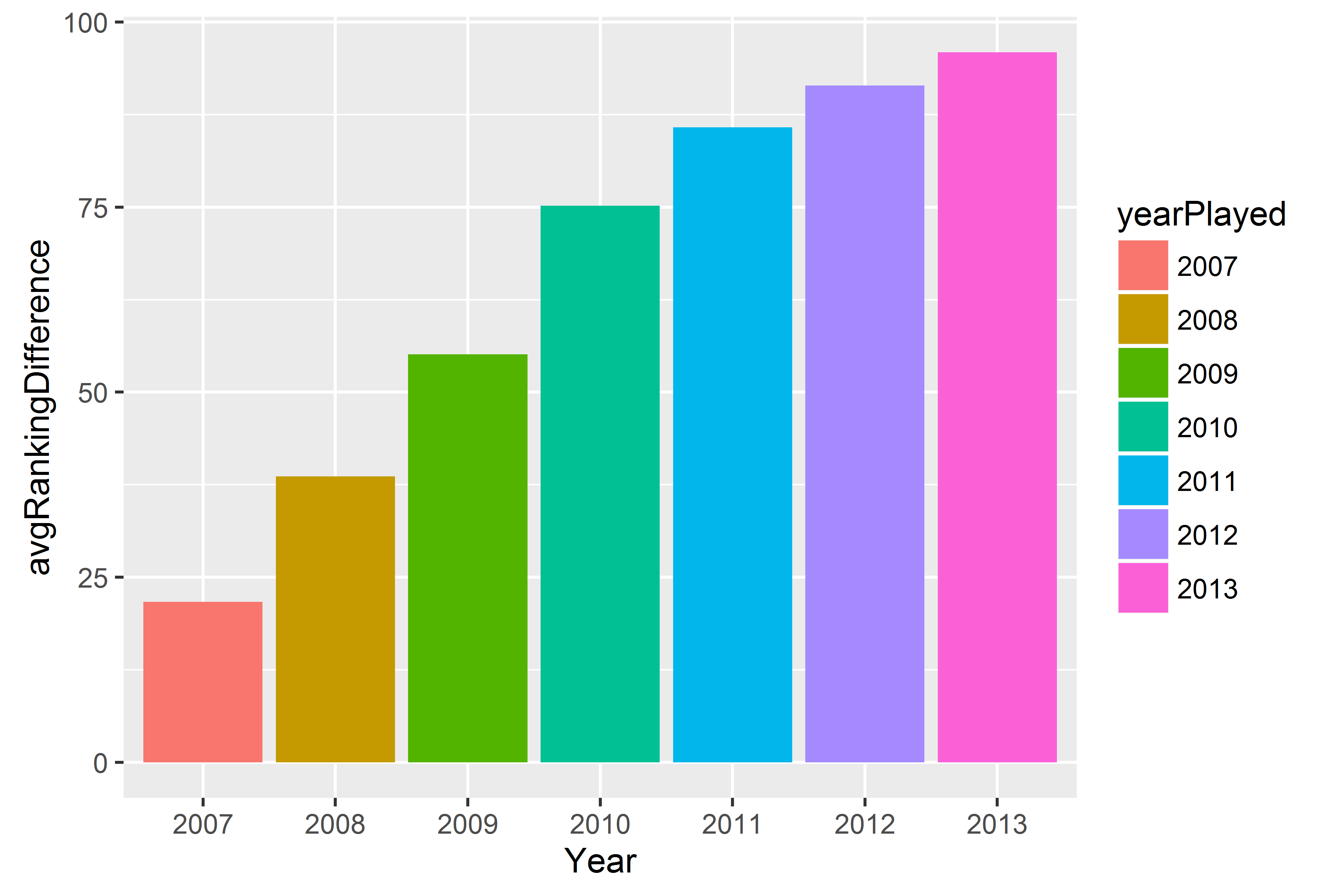
**Problems**

**Outcome**

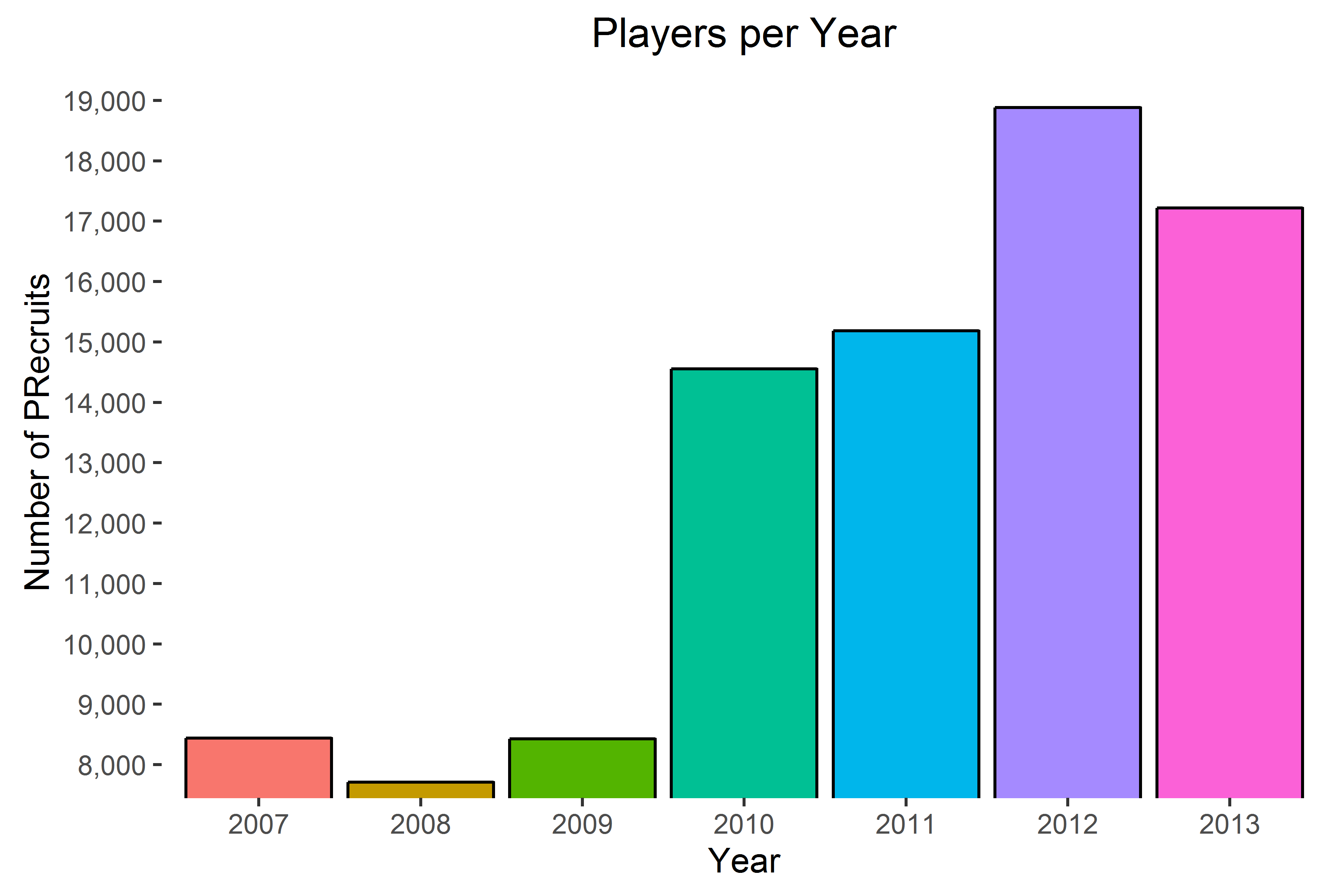
To determine the accuracy of rankers for college recruiting, we took the difference between a players ranking out of high school and compared it to their ranking after each college season that they played. We then took the average difference by position and year to determine how accurate the rankers turned out to be. Note here that we used difference instead of variance. To determine the average, we need to be working with a positive number, but variance was giving us how far off the rankers were in a positive and negative direction (i.e. whether they over-ranked or under-ranked each of the prospects). So we defined the difference to be the absolute value of this difference so the positive and negative numbers would not cancel each other out when we calculated the average.

The results of our study can be broken down into two different situations based on how we decided to deal with NA value in the data pulled from ESPN. Our two different handlings of the NA data results in two different interpretations of the data. I will go into both ways we handled NA below as well as what we concluded from the results.

The first time through, we replaced all of the NA grades with a 49. We settled on a grade of 49 because that was one point lower than the lowest graded prospect. In our minds, if the ranker decided not to grade a recruit, then there was no difference between them and anyone else with an NA grade. The reasoning behind using a 49 to replace NA was sound at the beginning of our analysis, but we saw flaws with it when we started to analyze our results. From 2007 to 2013, we saw the average difference in rankings increase greatly (see chart below).



Initially, we concluded that rankers were getting worse as the years passed. But then we looked at the data further and found what might actually be the cause of this increase. From 2007 to 2013, the number of recruits per year in our data also increase greatly (see chart below). Recruiters were still ranking approximately the same number of recruits, so the number of NA’s per year increase as a result. Some of these NA’s produced at a high level when they played in college, so it increased the average difference in rankings per a year. This result is really a reflection of the increase in the number of high school recruits trying to get ranked, and it is not a good measure of how accurate recruiters are in their rankings. It just shows us that there are too many players out there to be ranked every year. Because of this determination, we decided to handle the NA values differently and come to a real conclusion about the rankers.



Our second time around, we decided to drop any recruits with an NA rank from our data. This resulted in a much better measure of the accuracy of the rankers. We didn’t the see the steep decline in year-to-year accuracy that we saw in our first attempt. We also saw accuracy overall get much better than what we saw when we replaced the NA grades with a 49. Our new positional and yearly charts (see below) now had a difference of about 30 spots as a ceiling, down from the nearly 100 spot difference we were seeing before. This allowed us to conclude that when a player is ranked by the experts, they are actually pretty good about determining future output at the collegiate level. They were better at some positions than others, but overall they did a pretty good job. We would be comfortable using the expert’s rankings as a solid tool for determining whether or not a player will have success as a college football player.

