**Final Project: An Analysis of MLB Data (2010-2016)**

**CS544 Foundations of Analytics**

**Fall 2017**

**Anthony Valencia**

**Data Preparation:**

[Retrosheet](http://www.retrosheet.org/) is an online source of baseball data. Additionally, the provide a few tools to convert this data into alternative formats. One of these tools, bevent.exe, is used in this project to create readable play-by-play files. All of the Retrosheet utilities are Windows-compatible only. I do not own a Windows machine, so in order to convert this data, I launched an AWS EC2 instance running Windows Server 2016. From this machine, I downloaded bevent.exe and every Retrosheet event file since 1940. I organized the files as follows:

saber**/** # all sabermetric data files  
 eve**/** # all event files  
 BEVENT.exe # play-by-play file utility  
 convert.py # runs bevent.exe creating play-by-play CSVs  
 2000seve**/** # 2000 event decade file  
 2010seve**/** # 2010 event decade file  
 2010**/** # 2010 event year file  
 2011**/** # 2011 event year file  
 2012**/** # 2012 event year file  
 2013**/** # 2013 event year file  
 2014**/** # 2014 event year file  
 2015**/** # 2015 event year file  
 2016**/** # 2016 event year file  
 TEAM2016 # 2016 Retrosheet team file  
 **\***.EVA # event files of American League teams  
 **\***.EVN # event files of National League teams  
 **\***.ROS # roster files  
 csv**/** # csv directory  
 **\***.EVA.csv # bevent.exe generated CSVs for AL  
 **\***.EVN.csv # bevent.exe generated CSVs for NL

Next, I run convert.py (attached in python directory):

**import** os  
**import** subprocess  
  
decades **= [**i **for** i **in** range**(**4, 10**)] + [**0**]  
  
for** decade **in** decades**:** decade\_prefix **= '19' +** decade.\_\_str\_\_**()** d **=** decade\_prefix **+ '0'** years **= [**decade\_prefix **+** str**(**i**) for** i **in** range**(**10**)]  
  
 for** year **in** years**:** cmd **= ['bevent.exe'**, **'-y'**, year**]** eve\_dir **=** os.path.join**(**os.path.dirname**(**os.path.realpath**(**\_\_file\_\_**))**, d **+ 'seve'**, year**)** files **= [**f **for** f **in** os.listdir**(**eve\_dir**) if (**os.path.isfile**(**os.path.join**(**eve\_dir, f**)) and '.EV' in** f**)]  
  
 for** f\_name **in** files**:** print**(**f\_name**)  
 with** open**(**os.path.join**(**eve\_dir, **'csv'**, f\_name **+ '.csv')**, **'w') as** out\_file**:** full\_cmd **=** cmd **+ [**f\_name**]** print**(**full\_cmd, eve\_dir**)** subprocess.call**(**full\_cmd, stdout**=**out\_file, cwd**=**eve\_dir**)**

to create play-by-play CSVs from all event files. Note that this script creates all of these in their respective CSVs. The script is then modified appropriately to produce play-by-play event files for 2000-2016.

All CSVs are copied to S3. Then an Ubuntu 16.04 t2.micro EC2 instance is launched and the CSVs are downloaded and organized by decade. Using a simple bash script, each file for a decade is appended to create a single YYYY-present-event.csv. These files are subsequently scp’d locally.

All decade CSVs from 1970 to present are converted to RData files using generate\_rdata\_files.R (attached in R directory). As it turns out, all files other than that for the decade 2010 are very large and take too long to run operations for this project. Therefore, only 2010-present-event.RData is used for this project. It would be interesting to run the 1970-present-event.RData file using RSpark, or the CSV equivalent using PySpark.

Furthermore, additional files for which RData files are created from generate\_rdata\_files.R include:

* player\_manager\_coach.RData (full names, and time active)
* parkcode.RData (historical ball park data)
* current\_names.RData (current team names)
* event\_types.RData (play event types including codes)
* positions.RData (field positions)
* park\_details.RData (additional ballpark detail including distance to center field)

All of these files are attached in the rdata directory.

**Analyzing the Data:**

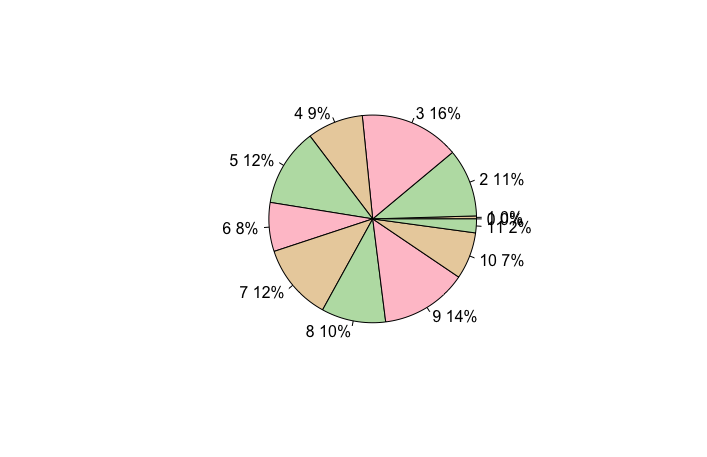
**Categorical:**

First load the 2010-present events and merge with event types and positions. Next I want to see how home runs, strike outs and batter hits were distributed according to the category of batter position, where positions have the following indices:

|  |  |
| --- | --- |
| **position\_index** | **position** |
| 1 | pitcher |
| 2 | catcher |
| 3 | 1b |
| 4 | 2b |
| 5 | 3b |
| 6 | ss |
| 7 | lf |
| 8 | cf |
| 9 | rf |
| 10 | designated hitter |
| 11 | pinch hitter |

**HRs by position (%):**

P: 0.4602; C: 10.5961; 1B: 15.4576; 2B: 8.706; 3B: 12.141; SS: 7.6139; LF: 11.9169; CF: 10.0134; RF: 13.5306; DH: 7.3061; PH: 2.1635



**SOs by position (%):**

P: 5.579; C: 10.5649; 1B: 11.3785; 2B: 9.3261; 3B: 10.2901; SS: 9.1646; LF: 11.2577; CF: 11.4731; RF: 11.434; DH: 5.7713; PH: 3.7529

**Hit by pitcher by position (%):**

P: 0.9422; C: 11.3789; 1B: 11.1161; 2B: 12.1308; 3B: 10.9803; SS: 9.2317; LF 12.312; CF: 11.4332; RF: 11.4242; DH: 5.6894; PH: 3.352

**Numerical:**

Merge events file with player, manager, coach dataset and find players with more than 125 RBIs in one year.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **res\_batter** | **year** | **RBI/Y** | **LAST** | **FIRST** | **DEBUT** |
| cabrm001 | 2012 | 139 | Cabrera | Miguel | 6/20/03 |
| davic003 | 2013 | 138 | Davis | Chris | 6/26/08 |
| cabrm001 | 2013 | 137 | Cabrera | Miguel | 6/20/03 |
| arenn001 | 2016 | 133 | Arenado | Nolan | 4/28/13 |
| arenn001 | 2015 | 130 | Arenado | Nolan | 4/28/13 |
| hamij003 | 2012 | 128 | Hamilton | Josh | 4/2/07 |
| encae001 | 2016 | 127 | Encarnacion | Edwin | 6/24/05 |
| ortid001 | 2016 | 127 | Ortiz | David | 9/2/97 |
| kempm001 | 2011 | 126 | Kemp | Matt | 5/28/06 |
| cabrm001 | 2010 | 126 | Cabrera | Miguel | 6/20/03 |

Next, I dropped all players who played less than 10 games to focus on those playing regularly.

Setting the remaining players’ RBIs/year to the variable x, I measured the following statistics:

**Mean**: 24.85745

**Variance**: 822.0773

**Standard deviation**: 28.67189

**Quantiles**: 0%: 0; 25%: 2; 50%: 13; 75%: 41; 100%: 139

With the stem plot producing:

0 | 00000000000000000000000000000000000000000000000000000000000000000000+2469

1 | 00000000000000000000000000000000000000000000000000000000000000000000+587

2 | 00000000000000000000000000000000000000000000001111111111111111111111+420

3 | 00000000000000000000000000000000000000111111111111111111111111111111+305

4 | 00000000000000000000000000000000000000000011111111111111111111111111+276

5 | 00000000000000000000000000000000001111111111111111111111111111111112+239

6 | 00000000000000000011111111111111111111122222222222222222222222222223+138

7 | 00000000000000000000001111111111111111112222222222222222222233333333+105

8 | 00000000000000001111111111111122222222222222222222222222333333333333+88

9 | 00000000000111111111111222222222233333333444444444555555555566666666+14

10 | 00000000000111112222222223333333333333344444444555555555555666677788+2

11 | 0001111112233455667778899

12 | 0345566778

13 | 03789

So, dropping those with 10 games or less didn’t seem to have much effect on the balance toward zero.

**Central Limit Theorem:**

Here, I set out to quantify the home field advantage. Each play record notes the score before the play occurs. In order to measure the score after the play, we evaluate if the base runners or batter reached home plate. Additionally, we check if each base runner or batter reached home due to one of two types of errors. Finally, we add these possibilities to the runs prior to the game (depending on who is batting, home team or visiting). Next, we have a calculation to filter out all records that are not the final play of the game. This leaves us with a data frame containing the final scores. The calculation for final scores assumes the game ends in the 9th inning or after. This causes us to lose 44 games, all of which ended early. We find the following statistics associated with the final score:

**Median**: 1; **Mean**: 0.1293993; **SD**: 4.195234

**Sample size**: 5; **Median**: 0; **Mean**: 0.058; **SD**: 1.77828

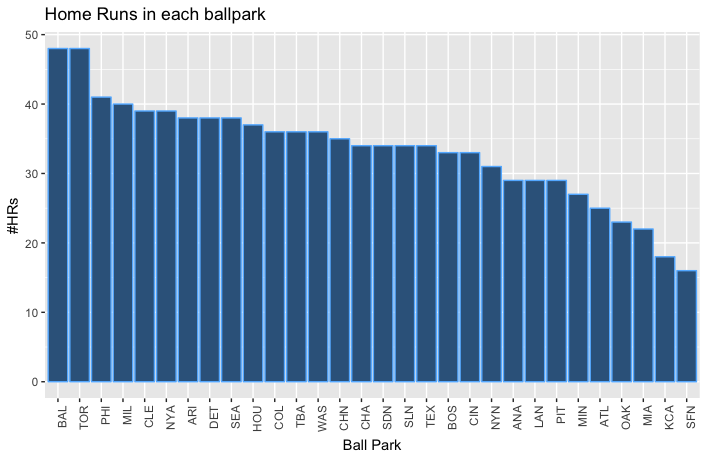
**Sample size**: 20; **Median**: 0.15; **Mean**: 0.16335; **SD**: 0.9516275

**Sample size**: 100; **Median**: 0.15; **Mean**: 0.14457; **SD**: 0.422206

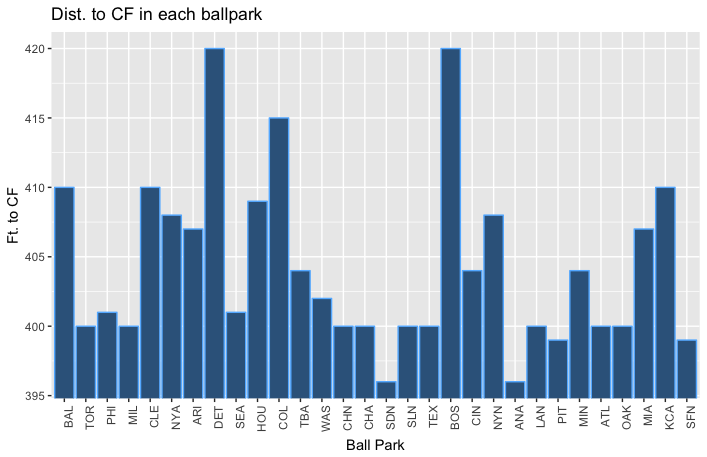
So, we see that there is a slight home field advantage, as we subtracted home field final score from visiting. As noted by the CLT, we observe the mean converging as the standard deviation starts to decrease.

**Sampling Methods:**

We merge our home run events variable (obtained earlier) with our park details dataset to get ballpark distances. Next we select 1000 random samples without replacement and plot a histogram of home runs by ballpark in sorted order.



Next, we plot the ballparks by distance to center field in the same order as above.



It seems that there is no advantage. One thing to note is that ballparks come in various shapes and there may be some region of the field that can be exploited by the batter which is not center field. This would not be reflected in the analysis. More research is required here before any conclusions can be drawn. The same evaluation was performed using systematic sampling with similar results.

**Confidence Levels:**

In gauging the confidence that the value of the home field advantage falls within 80 and 90 percent around the mean, we find the following:

80% Conf Level (alpha = 0.20), z: -1.28, 1.28

90% Conf Level (alpha = 0.10), z: -1.64, 1.64

Final score SD: 4.195234

Sample size=10, Mean: -1.9

80% Conf Level (alpha = 0.20), CI = (-3.60, -0.20), Precision = 3.40

90% Conf Level (alpha = 0.10), CI = (-4.08, 0.28), Precision = 4.36

Sample size=100, Mean: 0.13

80% Conf Level (alpha = 0.20), CI = (-0.41, 0.67), Precision = 1.08

90% Conf Level (alpha = 0.10), CI = (-0.56, 0.82), Precision = 1.38

Sample size=1000, Mean: 0.278

80% Conf Level (alpha = 0.20), CI = (0.11, 0.45), Precision = 0.34

90% Conf Level (alpha = 0.10), CI = (0.06, 0.50), Precision = 0.44

Confirming our earlier assertion that there is a slight home field advantage. Furthermore, we note confidence interval tightening and precision decreasing as expected from our confidence interval measurements.

**Additional Features:**

Some additional analytic methods explored here are the concepts of On Base Percentage (OBP) and Slugging Average (SLG). Furthermore, we are going to analyze the data with Python’s pandas, numpy and plotly to demonstrate some new techniques.

OBP = (H + BB + HBP)/(AB + BB + HBP + SF)

H: hits

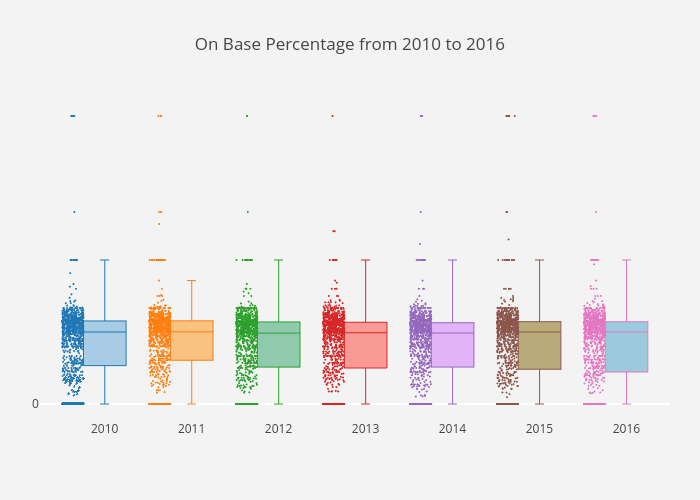
BB: base on balls

HBP: hit by pitch

AB: at bats

SF: sacrifice fly

Since every play in the event records dataframe constitutes an at bat, this one is easy to measure. All others use the event type, hit value and SH flag (for SF). Next, we group by player and year and calculate all OBPs for each player for each year. Next, we use plotly to create box plots for each year.



Seems that not much has changed since 2010 in batter performance when measured by OBP.

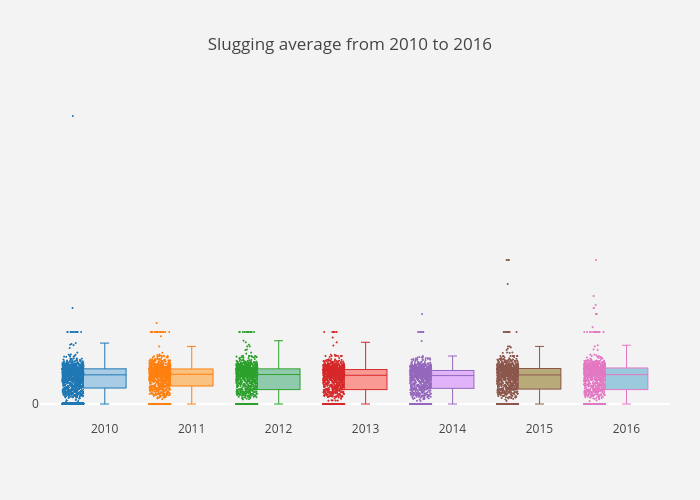
Finally, we calculate SLG by first calculating:

TB = 1B + 2\*2B + 3\*3B + 4\*HR

SLG = TB/AB

TB: total bases

Since batter destination value 5 indicates scored and unearned and 6 indicates team unearned, we don’t know how far the batter earned his bases. We’ll have to assume 4 here.

[](https://plot.ly/~antvalencia/4)

Again, not much has changed since 2010 in batter performance when measured by SLG.

**References:**

* Retrosheet.org