Objective: A Run Expectancy Matrix for College Baseball

My main objective with this project, most simply put, was to create a run-expectancy matrix for college baseball. I knew from the research I did for two past projects that in order to create a run expectancy matrix, one needs a large dataset of plate appearances accompanied by some detailed information for each one, most importantly the customary tallies of outcomes (hits, strikes, walks, etc.), the starting and ending base-out states, and the observed value of the random variable **R** - the number of runs that will be scored between the start of an observation and the end of a half-inning, which is calculated by using another variable, **R'** - the number of runs scored during an individual plate appearance.

In order to fill the cells of the *run expectancy matrix*, we need to sort plate appearances into twenty-four groups according to starting base-out state and find the expected value of **R** for each. Once the values of the run expectancy matrix are known, starting and ending run expectancy values can then be assigned to each plate appearance in the database, their difference can be found and added to **R'** (the number of runs scored during the plate appearance) to assign a run expectancy amount to the plate appearance. Grouping plate appearances by outcome (e.g. single, strike out, sac fly, etc.), then averaging **R** yields an average run expectancy amount for that outcome. Further grouping plate appearances with a shared outcome according to starting base-out state, then averaging **R**, yields an entire run expectancy matrix for that outcome.

Furthermore, if observations are properly tagged with player identifications, we can also use this dataset to valuate college players in terms of expected runs, and possibly eventually formulate something resembling a college WAR.

So, I have identified the sort of dataset I need, a large number of plate appearances with the following variables attached: the identities of the batter and runners, the starting and ending base-out states, the outcome, and the number of runs scored. If I were performing this sort of operation on Major League data, I could quickly download a csv from Baseball Savant with the observations I need. The twist here is that I want this dataset for college baseball, so things are not as easy.

The Data Source

The Bad News

I haven't searched the internet exhaustively, but I doubt a dataset like the one described above readily exists and is available to the public. The only place I have found college baseball data on the plate appearance level is in the play-by-play descriptions on individual team websites. These descriptions are found on pages generally labeled 'Box Score,' and linked to on a season's schedule page. As far as I can tell, this data exists only in strings of text, and would require a person or a machine to quantify it and arrange it in spreadsheets before it could be analyzed and sifted for insight. Tables are provided for some inning-level totals, but these are not detailed enough for calculating run expectancy.

The Good News

Most college teams use the same company's services to enter and store their stats and build their websites, so the structure and content of these strings is fairly predictable, and it is possible to build an application to read the strings and arrange the data they contain in csv files.

Precautions

The data can be read and tabulated, but still not necessarily trusted. These play-by-play strings are created by an application that is operated by a college coach or student manager during the course of a game. I sat beside such an operator, my fellow student manager for the Eastern Kentucky University baseball team, Brady Salisbury, through several games as he used this application. As the events of the game unfold, the operator taps a combination of buttons on the screen to record them. Not every operator uses the application in the same way or with the same level of care. This introduces variability into the structure of the strings and the level of detail included in them. A certain number of inaccuracies due to human error are also to be expected.

The Project's Origin and Intended Audience

I first felt the urge to carry out this project more than two years ago. College baseball has long been the target of my sabermetric curiosities—I formed an attachment to the college game through my location (Kentucky), a liking for the informal, cool-weather atmosphere of fall practices, and a fascination with baseball scouts at work—and even before I earned a Statistics degree and learned to program, I noticed the scarcity of college data online and began to think about building a program that could glean more detailed stats from the information stored on college baseball websites.

I'm approaching this project as a member of the public, with no special access to NCAA or Trackman data, with only the html pages described above at my disposal. I'm assuming the reader is in a similar position.

The Domain

I chose the SEC, arguably the most dominant conference in college baseball, as the league I wanted to analyze, limiting the set to intra-conference games to reduce variability, and to the 2019 season to keep the size of the dataset manageable for a first attempt. The web design is mostly standard among SEC team pages, but there are some irregularities that present an obstacle in the cases of Arkansas, LSU, and Vanderbilt. Arkansas stores their data in pdf files, and LSU and Vanderbilt's pages are formatted differently from the rest. For this reason, three games between LSU and Arkansas and three games between Arkansas and Vanderbilt are missing from the dataset. This leaves a total of 206 html files, each one representing a game, that I saved in a directory.

Before I began to quantify the play-by-play strings, I needed to plan for storing and organizing the output. Early in the process of developing the application, it became apparent that the output would consist of multiple datasets, and I would need to link the observations common to these datasets with unique identifiers that could be used as keys in a database.

I began to create these identifiers by assigning unique four-letter codes to each SEC team. I used these to assign unique alpha-numeric identifiers to each of the 206 games which consisted of the away team's code, the home team's code, the 8-digit date, and two additional digits to distinguish between games in a double-header. I used these Game ID's as the file names, and thus also as a method of passing the Game ID and information it contained into the application.

I put these lists of ID's into tables and began to form a database. I also built a table of SEC players active in 2019, assigned each one a unique identifier, and gathered their names, heights, weights, and batting and throwing hands from the online team rosters, as well as draft information from Baseball Reference.

Note: I quickly realized in assembling this player dataset that many brothers, including several sets of twins, play baseball in the SEC, sometimes simultaneously, sometimes successively, both as teammates and opponents. One set of brothers' unique identifiers are distinguished only by their jersey number: Georgia's Cole and Connor Tate ('Co_Tate00_GEOR_15' and 'Co_Tate00_GEOR_23'). The Tates even appear in the some of the same games together, and in one game bat back to back in the order. This brother phenomenon, as well as that of players who transfer between schools and appear on multiple SEC teams, highlight the importance of carefully tracking the identities of players.

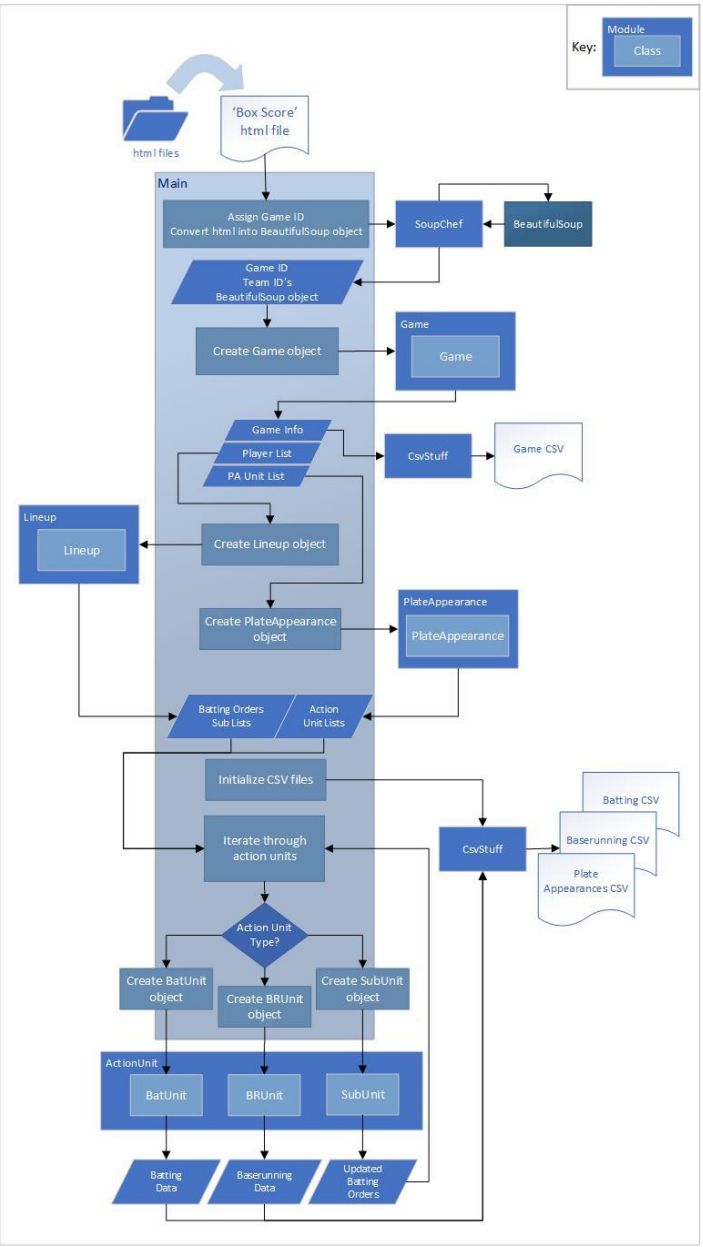
The Python Application, *PlayByPlayMiner*

The first phase of this project was building the application to do the dirty work of digging the data out of play-by-play text. I built a preliminary version of such an application in Java in 2018, but switched to Python for my second attempt because of its facility with strings, and inputting and outputting data.

Note: I also used R while building the Python application to analyze trends and outliers among the collected set of play-by-play strings, which informed the algorithm I developed to process them, and helped me either prevent or efficiently resolve many logic errors.

The python application, which I've named *PlayByPlayMiner*, essentially loops through a directory of html files, reads each one and outputs its data into csv files.

To parse the html game files, I used the existing web-scraping library *Beautiful Soup,* https://www.crummy.com/software/BeautifulSoup/bs4/doc/. This library includes functions to locate and extract strings of text from html code, and organize them into lists.



The PlayByPlayMiner application consists of a *Main* function and eight modules: *SoupChef, Game, Lineup, PlateAppearance, ActionUnit, StdzName, BaseOut, and CsvStuff*. The *Game, Lineup,* and *PlateAppearance* modules contain classes of the same name. The *ActionUnit* module contains three classes: *BatUnit, BRUnit, and SubUnit*. The *StdzName, BaseOut*, and *CsvStuff* modules contain functions to assist the class functions with specific recurring tasks like creating strings of comma-separated values and standardizing names that are input with unknown formatting.

Some of the lists created with the help of *Beautiful Soup* contain the lineups for each team, consisting of position players in batting order followed by pitchers in order of appearance, with subs indented beneath the starters they replaced, accompanied by their abbreviated position. These lists are passed to the Lineup class and organized there into standardized lists of last names in proper case (in future versions this will be the unique player ID). These lists have accompanying lists that hold the players' position and whether they started or were a substitute. The Lineup class also creates lists for each team's batting order, each team's relievers, and each team's offensive substitutes for use in the ActionUnit classes later.

The other lists contain the play-by-play descriptions of the events of a game. These are passed to the Play-By-Play class where they are assigned a unique identifier I called a 'Plate Appearance ID,' though not all strings with this id are technically a plate appearance. Some are substitutions, e.g. "Smith to p for Robinson," and some are running events that occur between and independent of hitting events, e.g. "Jordan stole second; Johnson out at third, caught stealing." Most, however, are batting events, followed by accompanying running events, if applicable, e.g. "Borman singled to left field (0-2 SSBF); Lovell advanced to second." The PlayByPlay class has a method that further breaks these compound batting and running event strings into what I call "action units," each of which begins with a player's name (with exceptions), and is either a batting event, a running event, or a substitution (or an 'other' string of some miscellaneous information, such as a review or a rain delay, that can be omitted from the data). I called these "action units", and made the ActionUnit module with the BatUnit, BRUnit, and SubUnit classes to handle them.

These classes gather and output csv files of data for each game. These csv files are then linked in a database by identifiers they contain.

Some Notable Logic Problems Encountered in the Process

Problem: Any type of name formatting that can be used, will be used.

One of the biggest challenges I ran up against was correctly isolating the part of the string that was a player's name. This proved to be surprisingly hard, as the data entry application apparently allows the operator to choose their own formatting, in up to 12 characters, when entering the names of the players on their team. This likely leads to at least two different formats being used for names within the same game, and across enough games, any and every way of formatting a name will appear at some point.

For example, the name Pete Conrad might appear in the following ways:

P. Conrad
Conrad, P.
Conrad, P
CONRAD, P
CONRAD, Pe
conrad
Conrad
Pete Conrad
pete conrad

Or, if he went by P.J. or had a father with the same name it could be:

Conrad, P.J.
Conrad, PJ
Conrad,Jr.,P
...

The liberty the data entry application bestowed on the operators in entering player names threw all sorts of wrenches into any attempt to detect the boundary between name and description. Both sides of the divide could contain commas, or not. Both could contain periods, or not. There might be one, two, or zero spaces in the name, and while it could occupy no more than twelve characters, it could occupy any number up to twelve. It seemed that no single character could be relied upon as a reliable landmark—only very specific combinations of characters could, after locating other important components of the string and ruling them out. This problem was so complex it required its own module to help resolve it, and more work is yet to be done to refine the process of identifying the players referenced in the strings and linking those identities to the output data. The current version uses the last name only, in proper case, as the standard player id, leaving the work of matching last names to unique player id's to be done in the database.

Problem: Knowing when to 'remember' and when to 'forget' who's on base.

I divided the batting and baserunning parts of plate appearances from each other in order to analyze them and gather their stats, but this complicated the problem of chaining the events together and 'remembering' and 'forgetting' who was on base and how many outs there were as needed, and tagging each observation with their beginning base-out state, updating the base-out state according to the

evens in the observation, then tagging the observation again with the ending base-out state and passing it to the next event in the list.

I did this by maintaining 'starting' and 'ending' dicts of base-out information in the main function, entering them as parameters when initializing a class from the ActionUnit module, including functions to update and return the ending dict in each class.

I managed this with a combination of coding in the Python application and performing some operations in R on the output dataset.

I used 'BaseOut' dicts in Main, built methods to update them, passed them in and out of classes. I assigned unique ID's to plate appearances early on and kept them in a list that iterated along with the strings. They contained numbers for the half-inning, and I set a watch on when the number changed as a signal to reset *Outs* to 0 and the *BaseState* to empty.

The Database

The second phase of this project was assembling the datasets output by the PlayByPlayMiner application in a database. The application outputs three datasets for each game it processes: a batting dataset, containing stats that pertain to the batting events that occurred in the game, a baserunning dataset that contains baserunning stats, and a quality dataset that aids auditing of base-out data and contains the variables needed for the run expectancy calculations.

After processing the 206 html files I saved in a directory, and storing the output csv files in a database, I had a set of 16,986 unique plate appearances*, 16,134 unique batting observations, and 6,892 unique baserunning observations.

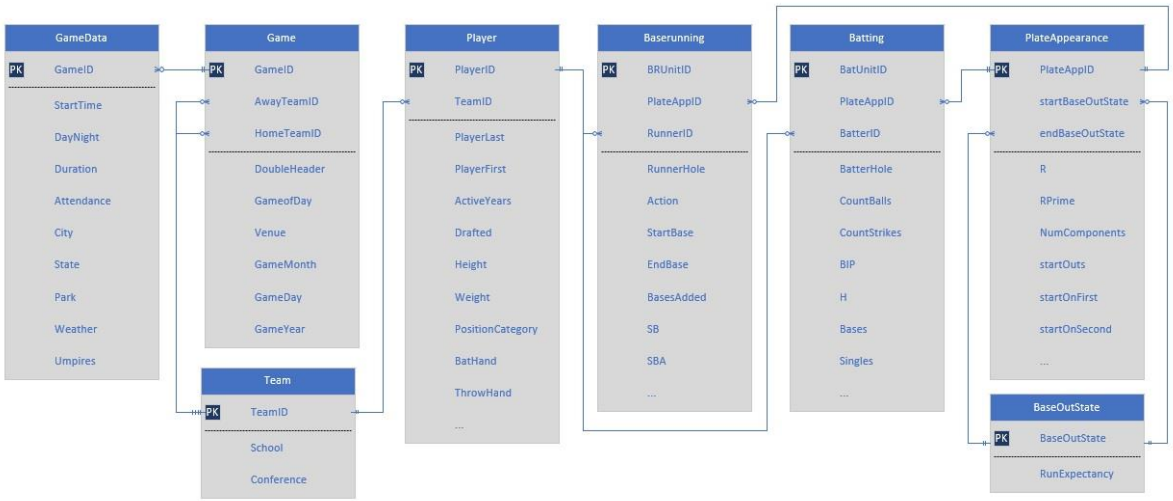
*These might include free-standing baserunning events that occur between batting events, such as pick offs or wild pitches.

The variability in the strings that necessitated the asterisk above presented another important challenge in the project. A 'plate appearance,' as anyone who watches baseball knows, can be a lot of things. For the purposes of this project, it consists of a *batting unit* and between zero and three *baserunning units*, unless it is a stand-alone baserunning event, consisting of between one and three *baserunning units*. No matter how many components a plate appearance has, however, it will still only have one starting base-out state and one ending base-out state, one total number of runs scored (**R'**), and, after calculation, one observed value for **R**.

Calculating R from R'

Some stats could only be calculated after the data was organized into tables. I tracked the accumulation of runs, outs, and the movements of baserunners with the events described in every batting and baserunning component, but for the dataset of unique plate appearances, I was only interested in the total runs and outs accrued, and the very beginning and very final base-out states for the plate appearance as a whole. I solved this problem by outputting a csv I named 'quality' (I also used it for quality control purposes). I then wrote a script in R which removes unneeded rows and columns, then collapses rows that share a plate appearance ID while totalling **R'** and keeping the correct starting and ending base-out states. The script then calculates **R** from **R'** and fills the column. These records are kept

in the ‘plate_appearances’ table in the database, and linked to the batting and baserunning tables through the ‘PlateAppID’. These tables are linked to the Player table through the ‘PlayerID.’



The Answers

Include the run expectancy matrix I created. Go over some general insights about the SEC in 2019.

BaseState	A	B	C	D	E	F	G	H
Outs	___	__1	_2_	3__	_21	3_1	32_	321
0	0.60122	1.03432	1.33993	1.82979	1.83051	1.55208	2.12346	2.61111
1	0.31734	0.66301	0.81326	0.95395	1.18679	1.30928	1.52212	1.74747
2	0.11785	0.26866	0.34153	0.37919	0.49834	0.57193	0.66525	0.84716

Do some studies of specific players.

The Future

I would like to expand the database to include more years for the SEC, as well as the ACC and PAC-12.