PlayByPlayMiner

Release 1.0

Jan L. Moffett

Jul 29, 2019

CONTENTS:

CHAPTER

ONE

CREATING AN INSTANCE OF BEAUTIFULSOUP

1.1 `SoupChef` – Functions to turn html files into ‘soup’

SoupChef.**open_soup**(*htm_string*)

This function requires the filename for an html file, a string ending in ‘.htm’ or ‘html’, as a parameter and returns an instance of the BeautifulSoup class.

SoupChef.**get_file_info**(*file_end_string*)

This function assumes that the input html files are named with the following format: ‘AWAY_HOME_0101201901.html’. This function takes such a filename as a parameter and returns a dict containing a Game ID string, the away team’s ID code, and the home team’s ID code.

CHAPTER

TWO

CREATING A GAME OBJECT

2.1 `Game` – Game object class and functions

class Game.**Game**(*BeautifulSoup_Object*)

This class requires an instance of the BeautifulSoup class as a parameter (see https://www.crummy.com/software/BeautifulSoup/). BeautifulSoup is imported and the instance created in the SoupChef module.

month
The two-digit month in which the game occured.

day
The two-digit day on which the game occurred.

year
The four-digit year in which the game occurred.

starttime
Start time of game, in *1:00 pm* format.

duration
Duration of the game, in minutes.

attendance
Attendance count for game.

city
City where the game occurred.

state
State where the game occurred.

park
Park where game occurred.

weather
Temperature and conditions at the start of the game.

umpires
Names of the umpires who officiated the game.

daynight
A string, either ‘day’ or ‘night’, indicating when game occurred.

The following list attributes iterate together:

awaypositions
A list of player names and positions for the away team.

homepositions
A list of player names and positions for the home team.

awaynames
A list of player names for the away team.

homenames
A list of player names for the home team.

awaystartsub
A list of strings, either 'starter' or 'sub', indicating player status for the away team.

homestartsub
A list of strings, either 'starter' or 'sub', indicating player status for the home team.

The following list attributes iterate together:

paunits
A list of play-by-play strings, scraped from the html file, each describing the events of a plate appearance, a baserunning event, or a substitution.

paside
A list of strings, either 'home' or 'away', corresponding to paunits.

paunitids
A list of unique alphanumeric identifiers for each element of paunits.

get_game_info()
Scrapes game data from html file and sets Game attributes.

get_month_day_year()
Sets month, day and year attributes based on date.

get_game_site()
Sets city, state, and park attributes based on site.

get_game_duration()
Converts game duration to minutes.

get_day_night()
Assigns 'day' or 'night' value to daynight attribute based on start time.

get_lineup_positions()
Scrapes strings containing player names and positions from html file and fills *awaypositions* and *homepositions* lists.

get_lineup_names()
Extracts player names from strings in *awaypositions* and *homepositions* and fills *awaynames* and *homenames* lists, as well as the *awaystartsub* and *homestartsub* lists.

get_action_units()
Scrapes and analyzes strings of play by play data from html file and fills the *paunits*, *paside*, and *paunitids* lists.

do_stuff()
Runs all class functions in correct order.

print_stuff()
Prints all class attributes for testing purposes.

CHAPTER

THREE

GATHERING AND ORGANIZING LINEUPS

3.1 `Lineup` – Lineup object class and functions

This module requires import of StdzNames module.

class `Lineup.`**`Lineup`**(*away_name_list*, *away_pos_list*, *away_startsub_list*, *home_name_list*, *home_pos_list*, *home_startsub_list*, *gameid*)

Initializing the Lineup class requires the following parameters:

- *away_name_list* - A list of names for away team output by Game class.
- *away_pos_list* - A list of positions corresponding to away_name_list.
- *away_startsub_list* - A list, ‘starter’ or ‘sub’, corresponding to away_name_list.
- *home_name_list* - A list of names for home team output by Game class.
- *home_pos_list* - A list of positions corresponding to home_name_list.
- *home_startsub_list* - A list, ‘starter’ of ‘sub’, corresponding to home_name_list.
- *gameid* - A unique alpha-numeric identifier for the game being analyzed.

`awaystarters`
A list of starters’ last names in proper case, in batting order, for the away team.

`homestarters`
A list of starters’ last names in proper case, in batting order, for the home team.

`awaystarterpos`
A list of positions (‘dh’,’p’,’c’,. . .) corresponding to *awaystarters*.

`homestarterpos`
A list of positions (‘dh’,’p’,’c’,. . .) corresponding to *homestarters*.

`awaysubs`
A list of subs’ last names in proper case, in order of appearance, for the away team.

`homesubs`
A list of subs’ last names in proper case, in order of appearance, for the home team.

`awaysubpos`
A list of positions (‘p’,’ph’,. . .) corresponding to *awaysubs*.

`homesubpos`
A list of positions (‘p’,’ph’,. . .) corresponding to *homesubs*.

`awayrelievers`
A list of relievers’ last names in proper case, in order of appearance, for the away team.

homerelievers
A list of relievers' last names in proper case, in order of appearance, for the home team.

awayoffsubs
A list of substitute position players' last names in proper case, in order of appearance, for the away team.

homeoffsubs
A list of substitute position players' last names in proper case, in order of appearance, for the home team.

awayorder
A list of hitters' last names in proper case, in batting order, for the away team.

homeorder
A list of hitters' last names in proper case, in batting order, for the home team.

get_starters_subs()
Separates starting players from substitutes and sets *awaystarters*, *awaystarterpos*, *awaysubs*, *awaysubpos*, *homestarters*, *homestarterpos*, *homesubs*, and *homesubpos*.

get_subs()
Separates relievers from offensive subs and sets *awayrelievers*, *awayoffsubs*, *homerelievers*, *homeoffsubs*.

get_batting_orders()
Sets *awayorder* and *homeorder*.

do_stuff()
Runs all SubUnit functions in the correct order.

print_stuff()
Prints all SubUnit attributes for testing purposes.

get_data_string()
Returns csv string of all variables generated by SubUnit class.

CHAPTER

FOUR

SORTING PLAY-BY-PLAY DESCRIPTIONS BY TYPE

4.1 `PlateAppearance` – PlateAppearance object class and functions

class `PlateAppearance.`**`PlateAppearance`**(*pa_unit_list*, *pa_side_list*, *pa_id_list*)

Initialization of the PlateAppearance class requires the following parameters:

- *pa_unit_list* - A list of 'plate appearance unit' strings output by the Game class.
- *pa_side_list* - A list of the values 'away' or 'home' corresponding to *pa_unit_list*.
- *pa_id_list* - A list of unique identifiers corresponding to *pa_unit_list*.

`actionunits`

A list of play-by-play strings, broken into player-specific units of action.

`autypes`

A list of the values "bat", "br", "batbr", "sub", or 'other' corresponding to *actionunits*.

`ausides`

A list of the values "away" or "home" corresponding to *actionunits*.

`aupaids`

A list of 'plate appearance unit id's' corresponding to *actionunits*. Up to four 'action units' can share the same 'plate appearance unit id'.

`auhalfinnings`

A list of integers denoting the half-inning in which the 'action unit' occurs, corresponding to *actionunits*.

`get_action_units``()`

Divides 'plate appearance unit' strings of play-by-play descriptions into player-specific units of action and sets *actionunits*, *aupaids*, and *ausides*.

`get_au_types``()`

Assigns a type value: 'bat', 'batbr', 'br', 'sub', or 'other' to each unit in *actionunits*.

`get_au_innings``()`

Sets *auhalfinnings*.

`do_stuff``()`

Runs all PlateAppearance class functions in the correct order.

`print_stuff``()`

Prints all PlateAppearance class attributes for testing purposes.

CHAPTER

FIVE

TURNING PLAY-BY-PLAY DESCRIPTIONS INTO DATA

5.1 `ActionUnit` – BatUnit, BRUnit, and SubUnit classes

Import of StdzNames and BaseOut modules required for this module.

5.1.1 Mining play-by-play batting descriptions

5.1.1.1 `BatUnit` – BatUnit class

class BatUnit(*bat_unit_str*, *team*, *batting_order*, *start_bsot*)

Initializing the BatUnit class requires the following parameters:

- *bat_unit_str* - A string of play-by-play batting information.
- *team* - A string, either 'away' or 'home'.
- *batting_order* - A list of names in the batting order of the specified team.
- *start_bsot* - A dict containing the base-out information for the start of the plate appearance.

hit_verbs
A list of verbs associated with base hits.

fo_verbs
A list of verbs associated with field outs.

fo_locations
A list of field out locations.

bip_verbs
A list of verbs associated with balls in play.

unit_string
Original string of play-by-play batting information.

team
A string, either 'away' or 'home'.

bo
A list containing names in batting order for the specified team.

bsotstart
A dict containing the base-out information for the start of the plate appearance.

bsotend
A dict containing the base-out information for the end of the plate appearance.

`batter_name`
A string, the last name of the batter in proper case.

`hole`
An integer representing the batter's place in batting order, the index of *batter_name* in *bo* + 1.

`count_balls`
A number as a string, the final ball count of the plate appearance.

`count_strikes`
A number as a string, the final strike count of the plate appearance.

`pitch_string`
A string of letters denoting balls, strikes, fouls, etc, such as 'BFKS'.

`balls`
An integer, the number of 'B' characters in pitch_string representing the number of balls thrown in plate appearance.

`fouls`
An integer, the number of 'F' characters in pitch_string representing the number of fouls hit in plate appearance.

`swingingstrikes`
An integer, the number of 'S' characters in pitch_string representing the number of swinging strikes in plate appearance.

`calledstrikes`
An integer, the number of 'K' characters in pitch_string representing the number of called strikes in plate appearance.

`BIPs`
An integer, the number of pitches that put a ball in play in plate appearance, either 0 or 1.

`pitches`
An integer, the total number of pitches based on pitch_string and BIP.

`BIP`
An integer, 1 if ball in play, 0 otherwise.

`H`
An integer, 1 if hit, 0 otherwise.

`bases`
An integer, the number of bases gained in hit (0, 1 for single, 2 for double,. . .).

`singles`
An integer, 1 if single, 0 otherwise.

`doubles`
An integer, 1 if double, 0 otherwise.

`triples`
An integer, 1 if triple, 0 otherwise.

`homers`
An integer, 1 if home run, 0 otherwise.

`hitloc`
A string description of hit location, if applicable. *("right center", "down the lf line",. . .)*

`hitqual`
A string description of hit quality, if applicable. *("soft", "medium", "hard")*

`hittype`
A string description of hit type, if applicable. *(“ground ball”, “fly ball”,. . .)*

`infh`
An integer, 1 if infield hit, 0 otherwise.

`fieldouts`
An integer, 1 if field out, 0 otherwise.

`foloc`
A string description of field out location, if applicable. *(“rf”, “1b”,. . .)*

`foqual`
A string description of field out quality, if applicable. *(“soft”, “medium”, “hard”)*

`fotype`
A string description of field out type, if applicable. *(“ground ball”, “fly ball”,. . .)*

`DP`
A integer, 1 if double play, 0 otherwise.

`doubleplaytype`
A string of numbers representing double play type, if applicable. *(“543”,”63”,. . .)*

`is_rch`
A boolean, True if ‘reached’ appears in play-by-play description, False otherwise.

`RBOE`
An integer, 1 if batter reached base on error, 0 otherwise.

`FC`
An integer, 1 if batter reached on fielder’s choice, 0 otherwise.

`SAC`
An integer, 1 if sacrifice bunt, 0 otherwise.

`SF`
An integer, 1 if sacrifice fly, 0 otherwise.

`BB`
An integer, 1 if walk, 0 otherwise.

`IBB`
An integer, 1 if intentional walk, 0 otherwise.

`UBB`
An integer, 1 if unintentional walk, 0 otherwise.

`SO`
An integer, 1 if strike out, 0 otherwise.

`kswing`
An integer, 1 if batter struck out swinging, 0 otherwise.

`klook`
An integer, 1 if batter struck out looking, 0 otherwise.

`UCTS`
An integer, 1 if uncaught third strike, 0 otherwise.

`SOreach`
An integer, 1 if batter reached on uncaught third strike, 0 otherwise.

trueoutcome
An integer, 1 if walk, strike out, or home run, 0 otherwise.

HBP
An integer, 1 if hit by pitch, 0 otherwise.

basesadded
An integer, the number of bases gained by batter during plate appearance. *(1 for single, walk, hit by pitch, etc, 2 for double,. . .)*

runsadded
An integer, the number of runs scored by the batter, not including rbi, either 0 or 1.

outsadded
An integer, the number of outs made by batter, not including other baserunners, either 0 or 1.

get_batter_name()
Finds batter name in *unit_string* and sets *batter_name*.

get_count()
Finds parentheses in unit string and sets *count_balls*, *count_strikes*, and *pitch_string*.

get_pitches()
Processes *pitch_string* and sets *balls*, *fouls*, *calledstrikes*,. . . , *pitches*.

is_bip()
Determines if there is a ball in play and sets *BIP*.

is_hit()
Determines if there is a hit and sets *H*.

get_hit()
Determines type of base hit and sets *bases*, *basesadded*, *singles*,. . .

get_hit_info()
Reads unit string and sets *infh* and *hitloc*, *hitqual*, and *hittype*, if applicable.

is_field_out()
Determines if there is a field out and sets *fieldouts*, *doubleplays*, and *outsadded*.

get_field_out()
Determines type of field out and sets *foqual*, *fotype*, and *foloc*, if applicable.

get_double_play()
Determines type of double play and sets *doubleplaytype*, if applicable.

is_reach()
Determines if ‘reached’ appears in unit string and sets *is_rch*.

get_reach()
Determines what type of play allowed the batter to reach base, if applicable, and sets *RBOE* and *FC*.

is_sac()
Determines if there is a sacrifice bunt and sets *SAC*.

is_sf()
Determines if there is a sacrifice fly and sets *SF*.

is_bunt()
Determines if there is a bunt and sets *bunt*.

is_walk()
Determines if there is a walk and sets *BB*, *bases*, and *basesadded*.

get_walk_type()
Determines if a walk is intentional or unintentional and sets *IBB* and *UBB*, if applicable.

is_strikeout()
Determines if there is a strike out and sets *SO*.

get_so_type()
Determines type of strike out and sets *kswing*, *klook*, *UCTS*, and *SOreach*.

is_true_outcome()
Determines if there is a true outcome (BB, SO, or HR) and sets *trueoutcome*.

is_hbp()
Determines if batter is hit by pitch and sets *hbp*.

update_bod()
Updates information in base-out dict.

do_stuff()
Runs all BatUnit functions in correct order.

print_stuff()
Prints all BatUnit attributes for testing purposes.

get_baseout_end()
Returns updated dict of base-out information.

get_data_string()
Returns csv string of all variables generated by BatUnit class.

5.1.2 Mining play-by-play baserunning descriptions

5.1.2.1 `BRUnit` – BRUnit class

class BRUnit(*unit_str*, *team*, *batting_order*, *start_bsot*, *end_bsot*)
Initializing the BatUnit class requires the following parameters:

- *unit_str* - A string of play-by-play baserunning information.
- *team* - A string, either ‘away’ or ‘home’.
- *batting_order* - A list of names in the batting order of the specified team.
- *start_bsot* - A dict containing the base-out information for the start of the plate appearance.
- *end_bsot* - A dict containing the base-out information for the end of the plate appearance.

unitstring
Original string of play-by-play baserunning information.

side
A string, either ‘home’ or ‘away’.

bo
A list containing names in batting order for the specified team.

bsotstart
A dict containing the base-out information for the start of the plate appearance.

bsotend
A dict containing the base-out information for the end of the plate appearance.

action
A string description of action performed by baserunner ("advanced","stole","scored",. . .).

runnername
A string, the last name of the baserunner in proper case.

hole
An integer representing the baserunner's place in batting order, index of *runnername* in *bo* + 1.

advanced
A boolean, True if runner advanced, False otherwise.

stole
A boolean, True if runner stole a base, False otherwise.

scored
A boolean, True if runner scored, False otherwise.

out
A boolean, True if runner was out, False otherwise.

fielder1
A string denoting the position of first fielder involved in runner out ("1b","ss",. . .).

fielder2
A string denoting the position of second fielder involved in runner out ("2b","3b",. . .).

startbase
An integer representing the base occupied by runner at start of plate appearance (0,1,. . . ,4).

endbase
An integer representing the base occupied by runner at end of plate appearance (0,1,. . . ,4).

basesadded
An integer representing the number of bases gained by runner, *endbase* - *startbase*.

outsadded
An integer, 1 if runner out, 0 otherwise.

runsadded
An integer, 1 if runner scored, 0 otherwise.

sba
An integer, 1 if stolen base attempt, 0 otherwise.

sb
An integer, 1 if stolen base, 0 otherwise.

poa
An integer, 1 if pickoff attempt, 0 otherwise.

fpo
An integer, 1 if failed pickoff attempt, 0 otherwise.

po
An integer, 1 if pickoff, 0 otherwise.

cs
An integer, 1 if caught stealing, 0 otherwise.

wp
An integer, 1 if wild pitch, 0 otherwise.

pb
: An integer, 1 if passed ball, 0 otherwise.

aoe
: An integer, 1 if runner advanced on error, 0 otherwise.

outatbase
: An integer representing the base where runner was called out, if applicable.

get_runner_name(self):
: Finds runner name in unit string and sets runnername and hole.

get_start_base()
: Determines base occupied by runner at start of plate appearance.

get_start_base_batbr()
: Determines base occupied by batter-runner after reaching initially.

get_action()
: Determines action performed by baserunner and sets action, advanced, stole, scored, and out.

is_pickoff()
: Determines if there is a pickoff and sets poa and po.

get_advanced()
: Determines where and why runner advanced, and sets *endbase*, *wp*, *pb*, and *aoe*, if applicable.

get_stole()
: Determines what base runner stole and sets *sba*, *sb*, and *endbase*, if applicable.

get_scored()
: Sets endbase and runsadded and sets wp, pb, and aoe if applicable.

def get_out()
: Determines where runner was called out and what fielders made the play.

update_bsot()
: Updates information in base-out dict.

get_bases_added()
: Determines bases added by baserunner during plate appearance.

do_stuff()
: Runs all BRUnit functions in correct order.

do_stuff_batbr()
: Runs all BRUnit functions for batter-runner case in correct order.

print_stuff()
: Prints all BRUnit information for testing purposes.

get_baseout_end()
: Returns updated dict of base-out information.

get_data_string()
: Returns csv string of all variables generated by BRUnit class.

5.1.3 Tracking substitutions

5.1.3.1 `SubUnit` – SubUnit class

class SubUnit(*sub_unit_str*, *team*, *away_batting_order*, *away_off_subs*, *away_relievers*, *home_batting_order*, *home_off_subs*, *home_relievers*)

Initializing the SubUnit class requires the following parameters:

- *sub_unit_str* - The original play-by-play string of substitution information.
- *team* - A string, either 'away' or 'home'.
- *away_batting_order* - A list of names in batting order for the away team.
- *away_off_subs* - A list offensive subs for the away team.
- *away_relievers* - A list of relief pitchers for away team.
- *home_batting_order* - A list of names in batting order for the home team.
- *home_off_subs* - A list of offensive subs for the home team.
- *home_relievers* - A list of relief pitchers for the home team.

off_pos
A list of positions in format used in play-by-play descriptions ("dh","p",...).

ogawaybattingorder
The original batting order for the away team, a list of last names in proper case.

oghomebattingorder
The original batting order for home team, a list of last names in proper case.

awaybattingorder
The updated batting order for the away team after substitution.

homebattingorder
The updated batting order for the home team after substitution.

awayoffsubs
Offensive subs for the away team, a list of last names in proper case.

homeoffsubs
Offensive subs for the home team, a list of last names in proper case.

awayrelievers
Relief pitchers for the away team, a list of last names in proper case.

homerelievers
Relief pitchers for the home team, a list of last names in proper case.

offense
A boolean, True if substitution affects a batting order for one of the teams, False otherwise.

playerin
The last name of the player subbing in, in proper case.

playerout
The last name of the player coming out, in proper case, or 'NA' if none.

subpos
The position being subbed ("rf","p",...).

subhole
An integer, the subbed player's place in their team's batting order (list index + 1).

get_players()
    Finds player names in unit string and sets playerout and playerin.

get_pos()
    Determines position that is being substituted.

get_offense()
    Determines if substitution requires change to a batting order.

get_hole()
    Determines subbed player's place in batting order.

mod_batting_order()
    Modifies batting order based on substitution.

do_stuff()
    Runs all SubUnit functions in the correct order.

get_updated_bo()
    Returns post-substitution batting order lists for both teams.

print_stuff()
    Prints all SubUnit information for testing purposes.

get_data_string()
    Returns csv string of all variables generated by SubUnit class.

ActionUnit.**get_bat_data_header**()
Returns a comma-separated header string with variable names for batting dataset.

ActionUnit.**get_br_data_header**()
Returns a comma-separated header string with variable names for baserunning dataset.

ActionUnit.**get_sub_data_header**()
Returns a comma-separated header string with variable names for substitution dataset.

CHAPTER

SIX

STANDARDIZING PLAYER NAME FORMATS

6.1 `StdzNames` – Name Format Standardization

`StdzNames.`**`last_first`**`(`*name_string*`)`

Processes raw name string and returns names *Last, First* in proper case.

`StdzNames.`**`last_only`**`(`*name_string*`)`

Processes raw name string and returns last name in proper case.

`StdzNames.`**`first_only`**`(`*name_string*`)`

Processes raw name string and returns first name in proper case.

`StdzNames.`**`lasts_firsts`**`(`*name_list*`)`

Takes list of raw name strings and returns *LastList, FirstList*, lists of last names and first names in proper case.

`StdzNames.`**`lasts_only`**`(`*name_list*`)`

Takes list of raw name strings and returns list of last names in proper case.

`StdzNames.`**`firsts_only`**`(`*name_list*`)`

Takes list of raw name strings and returns list of first names in proper case.

CHAPTER

SEVEN

ASSIGNING CODES TO BASE-OUT STATES

7.1 `BaseOut` – Base-out state coding functions

`BaseOut.`**`get_base_state`**(*on_first*, *on_second*, *on_third*)

Returns alpha code for base state in the range {'A','B','C',...,'H'}.

- *on_first, on_second, on_third* - String indicating runner name or NA if base is empty.

Code	Base Configuration
A	___
B	__1
C	_2_
D	3__
E	_21
F	3_1
G	32_
H	321

`BaseOut.`**`get_base_out_state`**(*outs*, *on_first*, *on_second*, *on_third*)

Returns alpha-numeric code for base-out state in the range {'A0',...,'A3',...,'H0',...,'H3'}.

- *outs* - An int or string indicating number of outs.
- *on_first, on_second, on_third* - String indicating runner name or NA if base is empty.

CHAPTER

EIGHT

STORING MINER'S OUTPUT IN CSV

8.1 `CsvStuff` – CSV string and file functions

`CsvStuff.`**`csv_print`**(*data_list*)
Prints a list of any type as a string of comma-separated values.

`CsvStuff.`**`make_csv_header`**(*var_list*)
Takes a list of variable names and returns a comma-separated header string.

`CsvStuff.`**`make_csv_row`**(*data_list*)
Takes a list of any type and returns a comma-separated string of values.

`CsvStuff.`**`make_csv_file`**(*file_name*, *header_string*)
Creates a new CSV file with a header of variable names.

`CsvStuff.`**`add_row_csv_file`**(*file_name*, *csv_row_string*)
Adds a row of data to an existing csv file.

CHAPTER NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX