Liam Fruzyna
MATH 4630
Final Project

Introduction

On a crazy vacation to Las Vegas you wake up in an unfamiliar location. You make you way outside and are blinded by neon. You ask a passerby what day it is, he responds, “April 9th, 1968.” “1968!”, you exclaim, “It was 2019 just yesterday!” “Well, sounds like you should bet on some sports”, the stranger cracks. “I may as well take advantage of this opportunity”, you think. You look up and see a giant neon sign reading, “Golden Nugget Gambling Hall”. You go inside to place bets on the upcoming MLB season, which coincidentally starts tomorrow. But, what team should you bet on?

The 1968 MLB consisted of 20 teams, 10 in the National League, 10 in the American League. Inter-league play and the designated hitter had not been introduced yet, but the 162 game season had been in the rotation for 7 seasons. This resulted in each team playing every other team in its respective league 18 times each season, 9 times home, 9 times away. Once the regular season was complete, a winner of each league was crowed and the two champions played for the world championship in the World Series. In order to simulate this season 1620 regular season games must be simulated, plus 7 World Series games. But, games are more complicated than a winner and a loser. Each game will have at least 9 innings, consisting of 3 outs on each side. Each match up between the batter and the pitcher will form an interaction resulting in a hit, walk, or out. Eventually, resulting in runs being scored deciding the winner of the game. Given enough time, one team should balance out to be the most winning team. The question is, which team is most likely to win the 1968 World Series?

Model

To keep the model relatively simple, some assumptions will be made. First, each team will consist of 9 players, all of whom will bat and one of whom will pitch. The 8 batters will be chosen by their number of plate appearances and the pitcher by his number of batters faced. The batter will either hit a single, double, triple, or home run, be walked, or be struck out. If none of these cases occur the batter will be assumed out on contact (fly ball or ground out). The defense will not be given opportunities to collect double or triple plays.

In the data used each player will have a number of plate appearances, which is the total of all interactions at the plate when used to compute probabilities. Then the number of singles, doubles, triples, home runs, walks, and strike outs will be used against the number of plate appearances to calculate the probability of each happening again. These values will be averaged with the corresponding values for the pitcher who has a number of batters faced. A randomly drawn value will be used to decide which event happens for every plate appearance.

$$1.0=\frac{P(1B_B)+P(1B_P)}{2}+\frac{P(2B_B)+P(2B_P)}{2}+\frac{P(3B_B)+P(3B_P)}{2}+\frac{P(HR_B)+P(HR_P)}{2}+\frac{P(BB_B)+P(BB_P)}{2}+\frac{P(K_B)+P(K_P)}{2}+P(O)$$

These plate appearances add up to innings, which add up to games, which add up to seasons. Each season will have a winner, the amount of times each team wins the World Series will determine

the likeliness that they are to win the real 1968 World Series. We are looking for the team that is most likely to win the World Series, so the team that wins the most will be our choice.

Analysis

Due to the complexity and repetitiveness of the model, it was developed in Python in a very modular way. Once the source data is read in, cleaned, and organized. A custom “Bases” object is used to keep track of who is on the bases. This object is reset on every side of the inning.

Each at bat is run individually in the “runAtBat” function which requires a batter and a pitcher. The odds for each event that could happen is calculated by summing the odds of each event, averaging that of the batter and the pitcher, and generating a random value from 0 to 1 to choose which event happens. The at bats are run over each batter in the line up in the “runInning” function until 3 outs are achieved. The score is increased each time a batter makes it to home.

After the third out, the next side of the inning is run, once that side is complete the “runGame” function runs the next inning until at least 9 innings are completed and one team has more runs than the other. Each game is run as part of a series in the “runSeries” function. The series consists of a home and away team and a number of games. For each game a row is create for a log which contains which teams played, the score, and the winner.

Each series is part of a league and run in the “runLeague” function. A league consists of 10 teams, each playing every other team 9 times at home. Once each league is run, standings are calculated for each. The winners of each league the play each other in a 7 game series and the winner of that is declared the winner of the season. 1000 seasons are simulated for a total 1,627,000 games.

Results

After running the model 1000 times, the model proved fairly accurate. The Detroit Tigers won the American League and defeated the National League’s St. Louis Cardinals in the 1968 World Series. But according to the model if the season was played many times, St. Louis would have won the world series the most, 42% of the time. Detroit was second, winning 17% of the time. Most of the other teams follow in order. Of the 14 teams that won the World Series at least once in the simulation, all 10 of the top 10 teams of 1968 were represented and only 2 were not from the top 14. Of the 14 winners there were 2 teams that stood out. The Minnesota Twins who were 13th in 1968, but won 13% of the time in the simulation and the New York Mets who were 18th in 1968 but won 3% of simulations.

Simulation Ranking	Team	Winning %	1968 Ranking
1	SLN	41.6	2
2	DET	17.3	1
3	BAL	13.2	3
4	MIN	12.7	13
5	SFN	04.4	4
6	NYN	03.1	18
7	CLE	02.7	5

8	CHN	01.7	6
9	PIT	01.0	12
10	BOS	00.9	7
11	ATL	00.5	10
12	NYA	00.5	9
13	CIN	00.2	8
14	HOU	00.2	19

Figure 1: World Series Simulation Winning Percentage of Teams

Discussion and Summary

In 1968, the underdog Detroit Tigers overcame a 3 - 1 deficit to beat the defending world champion, the St. Louis Cardinals. In game 5 of the series Detroit overcame an early 3 - 0 deficit to extend the series to 5th game, had they not the St. Louis would have won the series. According to the model, St. Louis should have ended the series there. As mentioned St. Louis won almost 3 times more than Detroit. This shows the comeback win of 1968 was likely a fluke and would be hard to repeat.

In 1968 the Minnesota Twins finished 13th in the league. But in the simulation, they were the fourth most common winner, winning 13% of the time. In 1968 the Twins had 3 All-Stars, including first baseman Harmon Killebrew. Unfortunately for the Twins in that All-Star Game, Harmon ruptured his left medial hamstring and missed the second half of the season. His batting stats weren't amazing that season, but his first half of the season resulted in 17 home runs in only 295 at bats meaning he hit a home run 5.8% of the time, more than anyone else on the team. Because the model is based on number of plate appearances, Harmon's injury is invisible to the model. Meaning his 5.8 home run percentage was carried through the entire season, likely boosting the Twins' performance.

The New York Mets placed 18th in 1968, but in the simulation they were the sixth most common winner, winning 3% of the time. Despite their low ranking, 1968 was the Mets' best season since their inception in 1962. The next season, 1969, would be the Mets' first National League Pennant and World Series title. Perhaps the model was able to capture something that didn't produce results until the next year. The 1968 Mets also played 15 extra-inning games that year, but lost 13 of those games. Their inability to perform in extras may be a factor not represented in the model. In their next season the Mets would win 10 of their 16 extra-inning games, a drastic improvement.

In general, the model proved to be fairly accurate, considering that the model does not account for changes within teams during the season. To answer the initial condition, if one were to bet on the winner of the World Series, the St. Louis Cardinals would be the team to choose.

Appendices

In [1]:
```
# Liam Fruzyna
# MATH 4630
# Final Project

# Data from Sean Lahman's Baseball Database
# http://www.seanlahman.com/baseball-archive/statistics/
```

In [2]:
```
# imports
import pandas as pd
import random as rd
from statistics import mean
```

In [3]:
```
# function for improved logging
printLogs = False
def log(*objs, force=False):
    if force or printLogs:
        strs = []
        for obj in objs:
            strs.append(str(obj))
        print(' '.join(strs))
```

In [4]:
```
# read in and format batting data
b = pd.read_csv('Batting.csv')
b = b[b['yearID'] == 1968]
b['1B'] = b['H'] - b['2B'] - b['3B'] - b['HR']
b['O'] = b['AB'] - (b['H'] + b['SO'])
b['PA'] = b['AB'] + b['BB']
b = b[['playerID', 'teamID', 'lgID', 'PA', 'H', '1B', '2B', '3B', 'HR', 'BB', '
SO', 'O']]
b.head()
```

Out[4]:

	playerID	teamID	lgID	PA	H	1B	2B	3B	HR	BB	SO	O
46520	aaronha01	ATL	NL	670.0	174.0	108.0	33.0	4.0	29.0	64.0	62.0	370.0
46521	aaronto01	ATL	NL	304.0	69.0	55.0	10.0	3.0	1.0	21.0	37.0	177.0
46522	abernte02	CIN	NL	20.0	0.0	0.0	0.0	0.0	0.0	3.0	12.0	5.0
46523	adairje01	BOS	AL	217.0	45.0	42.0	1.0	0.0	2.0	9.0	28.0	135.0
46524	adamsmi01	BAL	AL	3.0	1.0	1.0	0.0	0.0	0.0	0.0	2.0	0.0

In [5]:
```
# convert batting data to percentages
bp = b.copy()
bp['H'] = b['H'] / b['PA']
bp['BB'] = b['BB'] / b['PA']
bp['SO'] = b['SO'] / b['PA']
bp['O'] = b['O'] / b['PA']
bp['1B'] = b['1B'] / b['PA']
bp['2B'] = b['2B'] / b['PA']
bp['3B'] = b['3B'] / b['PA']
bp['HR'] = b['HR'] / b['PA']
bp.head()
```

Out[5]:

	playerID	teamID	lgID	PA	H	1B	2B	3B	HR	BB
46520	aaronha01	ATL	NL	670.0	0.259701	0.161194	0.049254	0.005970	0.043284	0.095522
46521	aaronto01	ATL	NL	304.0	0.226974	0.180921	0.032895	0.009868	0.003289	0.069079
46522	abernte02	CIN	NL	20.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.150000
46523	adairje01	BOS	AL	217.0	0.207373	0.193548	0.004608	0.000000	0.009217	0.041475
46524	adamsmi01	BAL	AL	3.0	0.333333	0.333333	0.000000	0.000000	0.000000	0.000000

In [6]:
```
# read in and format pitching data
p = pd.read_csv('Pitching.csv')
p = p[p['yearID'] == 1968]
p['O'] = p['BFP'] - (p['H'] + p['BB'] + p['SO'])
p = p[['playerID', 'teamID', 'lgID', 'BFP', 'H', 'HR', 'BB', 'SO', 'O']]
p.head()
```

Out[6]:

	playerID	teamID	lgID	BFP	H	HR	BB	SO	O
18474	abernte02	CIN	NL	562.0	111	9	55	64	332.0
18475	adamsmi01	BAL	AL	36.0	9	2	4	4	19.0
18476	aguirha01	LAN	NL	167.0	32	0	13	25	97.0
18477	akerja01	OAK	AL	330.0	72	6	33	44	181.0
18478	arrigge01	CIN	NL	853.0	181	13	77	140	455.0

In [7]:
```
# convert pitching data to percentages
pp = p.copy()
pp['H'] = p['H'] / p['BFP']
pp['BB'] = p['BB'] / p['BFP']
pp['SO'] = p['SO'] / p['BFP']
pp['O'] = p['O'] / p['BFP']
pp['HR'] = p['HR'] / p['BFP']
pp.head()
```

Out[7]:

	playerID	teamID	lgID	BFP	H	HR	BB	SO	O
18474	abernte02	CIN	NL	562.0	0.197509	0.016014	0.097865	0.113879	0.590747
18475	adamsmi01	BAL	AL	36.0	0.250000	0.055556	0.111111	0.111111	0.527778
18476	aguirha01	LAN	NL	167.0	0.191617	0.000000	0.077844	0.149701	0.580838
18477	akerja01	OAK	AL	330.0	0.218182	0.018182	0.100000	0.133333	0.548485
18478	arrigge01	CIN	NL	853.0	0.212192	0.015240	0.090270	0.164127	0.533411

In [8]:
```
# get a list of teams and franchise info
teamIds = b['teamID'].unique()
teamIds
```

Out[8]:
```
array(['ATL', 'CIN', 'BOS', 'BAL', 'HOU', 'NYN', 'LAN', 'OAK', 'WS2',
       'PHI', 'PIT', 'MIN', 'CHA', 'SFN', 'CLE', 'NYA', 'CHN', 'CAL',
       'SLN', 'DET'], dtype=object)
```

In [9]:
```
# build a roster of 8 batters and a pitcher for each team
teams = {}
for team in teamIds:
    # find single most used pitched (batters faced)
    pitchers = pp[pp['teamID'] == team]
    pitcher = pitchers.nlargest(1, columns=['BFP']).iloc[0]
    teams[team + '-pitcher'] = pitcher
    # find top 8 most used batters (at bats), plus pitcher
    batters = bp[bp['teamID'] == team]
    pitcherBat = batters[batters['playerID'] == pitcher['playerID']]
    teams[team + '-batters'] = batters.nlargest(8, columns=['PA']).append(pitch
erBat)
```

In [10]:
```
# example of batter data for the Cubs
teams['CHN-batters']
```

Out[10]:

	playerID	teamID	lgID	PA	H	1B	2B	3B	HR	BB	
46860	kessido01	CHN	NL	693.0	0.226551	0.194805	0.020202	0.010101	0.001443	0.054834	0
47213	willibi01	CHN	NL	690.0	0.268116	0.169565	0.043478	0.011594	0.043478	0.069565	0
46560	beckegl01	CHN	NL	674.0	0.280415	0.227003	0.041543	0.005935	0.005935	0.045994	0
47094	santoro01	CHN	NL	673.0	0.210996	0.142645	0.025260	0.004458	0.038633	0.142645	0
46815	hundlra01	CHN	NL	592.0	0.211149	0.162162	0.030405	0.006757	0.011824	0.065878	0
46555	bankser01	CHN	NL	579.0	0.234888	0.132988	0.046632	0.000000	0.055268	0.046632	0
47027	phillad01	CHN	NL	486.0	0.218107	0.139918	0.041152	0.010288	0.026749	0.096708	0
46839	johnslo01	CHN	NL	211.0	0.236967	0.151659	0.066351	0.014218	0.004739	0.028436	0
46828	jenkife01	CHN	NL	106.0	0.150943	0.103774	0.037736	0.000000	0.009434	0.056604	0

In [11]:
```
# example of pitcher data for the Cubs
teams['CHN-pitcher']
```

Out[11]:
```
playerID    jenkife01
teamID            CHN
lgID               NL
BFP              1231
H              0.207149
HR             0.021121
BB            0.0528026
SO              0.21121
O              0.528838
Name: 18598, dtype: object
```

In [12]:
```
# function to build a list of odds into a list of brackets
def sumOdds(odds):
    for i in range(1, len(odds)):
        odds[i] += odds[i-1]
    return odds
```

In [13]:
```
# object to track and manage which bases are occupied
class Bases:
    def __init__(self):
        self.bases = [False, False, False, False]
        self.runs = 0

    def __repr__(self):
        bases = ''
        for b in range(1, 4):
            if self.bases[b]:
                bases += ' ' + str(b)
        return str(self.runs) + ' scored with men on' + bases

    def play(self, earned):
        log('Bases:', earned)
        if earned > 0:
            for b in range(len(self.bases)-1, 0, -1):
                if self.bases[b]:
                    reached = b + earned
                    self.bases[b] = False
                    if reached >= 4:
                        self.runs += 1
                    else:
                        self.bases[reached] = True
            if earned == 4:
                self.runs += 1
            else:
                self.bases[earned] = True
```

In [14]:
```
# process a single at bat of a pitcher vs a batter
def runAtBat(batter, pitcher):
    log('Batting:', batter['playerID'])
    odds = sumOdds([mean([batter['1B'], pitcher['H']/4]), mean([batter['2B'], p
itcher['H']/4]),
                    mean([batter['3B'], pitcher['H']/4]), mean([batter['HR'], p
itcher['HR']]),
                    mean([batter['BB'], pitcher['BB']]), mean([batter['SO'], pi
tcher['SO']])])
    log(odds)
    play = rd.random()
    if play <= odds[0]:
        log('Single')
        return 1, 0
    elif play <= odds[1]:
        log('Double')
        return 2, 0
    elif play <= odds[2]:
        log('Triple')
        return 3, 0
    elif play <= odds[3]:
        log('Home Run')
        return 4, 0
    elif play <= odds[4]:
        log('Base on Balls')
        return 1, 0
    elif play <= odds[5]:
        log('Strike Out')
        return 0, 1
    log('Out')
    return 0, 1
```

In [15]:
```
# run a single side of an inning (3 outs)
def runInning(offTeam, defTeam, leadOff):
    lineup = teams[offTeam + '-batters']
    pitcher = teams[defTeam + '-pitcher']
    bnum = leadOff
    outs = 0
    bases = Bases()
    while outs < 3:
        batter = lineup.iloc[bnum]
        b, o = runAtBat(batter, pitcher)
        bases.play(b)
        outs += o
        bnum += 1
        log(bases)
        if bnum >= 9:
            bnum = 0
    return bases.runs, bnum
```

In [16]:
```
# run a single game, a home team against a visitor
def runGame(homeTeam, awayTeam):
    home = 0
    away = 0
    inning = 1
    homeNext = 0
    awayNext = 0
    while inning <= 9 or home == away:
        log('---')
        log('Top', inning)
        r, n = runInning(awayTeam, homeTeam, awayNext)
        away += r
        awayNext = n
        log('---')
        log('Bottom', inning)
        r, n = runInning(homeTeam, awayTeam, homeNext)
        home += r
        homeNext = n
        inning += 1
    winner = 'HOME'
    if home < away:
        winner = 'AWAY'
    return home, away, winner
```

In [17]:
```
# play a series of games between 2 teams
def runSeries(home, away, games, seriesLen):
    for i in range(seriesLen):
        log(away, '@', home, '#' + str(i+1))
        games['homeTeam'].append(home)
        games['awayTeam'].append(away)
        h, a, w = runGame(home, away)
        games['homeScore'].append(h)
        games['awayScore'].append(a)
        games['winner'].append(w)
        log('Final:', h, a)
        log('')
```

In [18]:
```
# process a while league of teams, assuming a certain amount of home games agai
nst each team in the league
def runLeague(teams, seriesLen):
    games = {'homeTeam': [], 'awayTeam': [], 'homeScore': [], 'awayScore': [],
'winner': []}
    for home in teams:
        for away in teams:
            if home != away:
                runSeries(home, away, games, seriesLen)
    return pd.DataFrame(data=games)
```

In [19]:
```
# process a while league of teams, assuming a certain amount of home games agai
nst each team in the league
def runPlayoff(teamA, teamB):
    games = {'homeTeam': [], 'awayTeam': [], 'homeScore': [], 'awayScore': [],
'winner': []}
    teams = pd.concat([teamA, teamB], axis=1).T.sort_values(by=['wins', 'homeWi
ns', 'awayWins', 'team'], ascending=False)
    teamA = teams['team'].iloc[0]
    teamB = teams['team'].iloc[1]
    runSeries(teamA, teamB, games, 4)
    runSeries(teamB, teamA, games, 3)
    return pd.DataFrame(data=games)
```

In [20]:
```
# generate a leaderboard for a given set of teams
def standings(league):
    board = {'team': [], 'wins': [], 'losses': [], 'homeWins': [], 'homeLosses'
: [], 'awayWins': [], 'awayLosses': []}
    for team in league['homeTeam'].unique():
        homeGames = league[league['homeTeam'] == team]
        homeWins = len(homeGames[homeGames['winner'] == 'HOME'].index)
        awayGames = league[league['awayTeam'] == team]
        awayWins = len(awayGames[awayGames['winner'] == 'AWAY'].index)
        totalGames = len(homeGames.index) + len(awayGames.index)
        board['team'].append(team)
        board['wins'].append(homeWins + awayWins)
        board['losses'].append(totalGames - (homeWins + awayWins))
        board['homeWins'].append(homeWins)
        board['homeLosses'].append(len(homeGames.index) - homeWins)
        board['awayWins'].append(awayWins)
        board['awayLosses'].append(len(awayGames.index) - awayWins)
    return pd.DataFrame(data=board).sort_values(by=['wins', 'homeWins', 'awayWi
ns', 'team'], ascending=False)
```

In [21]:
```
# returns the best team in a standings
def getWinner(standings):
    return standings.iloc[0]
```

In [22]:
```
# process the national and american leagues independently
national = runLeague(b.loc[b['lgID'] == 'NL']['teamID'].unique(), 9)
american = runLeague(b.loc[b['lgID'] == 'AL']['teamID'].unique(), 9)
```

In [23]:
```
# national league standings
nlstand = standings(national)
nlstand
```

Out[23]:

	team	wins	losses	homeWins	homeLosses	awayWins	awayLosses
9	SLN	108	54	50	31	58	23
3	NYN	88	74	43	38	45	36
7	SFN	87	75	46	35	41	40
8	CHN	81	81	49	32	32	49
1	CIN	81	81	42	39	39	42
2	HOU	81	81	39	42	42	39
6	PIT	79	83	35	46	44	37
0	ATL	71	91	37	44	34	47
4	LAN	69	93	31	50	38	43
5	PHI	65	97	33	48	32	49

In [24]:
```
# american league standings
alstand = standings(american)
alstand
```

Out[24]:

	team	wins	losses	homeWins	homeLosses	awayWins	awayLosses
9	DET	103	59	54	27	49	32
1	BAL	93	69	42	39	51	30
4	MIN	90	72	47	34	43	38
6	CLE	88	74	40	41	48	33
0	BOS	79	83	36	45	43	38
8	CAL	77	85	34	47	43	38
7	NYA	75	87	39	42	36	45
2	OAK	75	87	33	48	42	39
3	WS2	66	96	36	45	30	51
5	CHA	64	98	38	43	26	55

In [25]:
```
# simulate a world series
ws = runPlayoff(getWinner(nlstand), getWinner(alstand))
worldstand = standings(ws)
worldstand
```

Out[25]:

	team	wins	losses	homeWins	homeLosses	awayWins	awayLosses
1	DET	5	2	2	1	3	1
0	SLN	2	5	1	3	1	2

In [26]:
```
# display world champion
getWinner(worldstand)['team']
```

Out[26]: 'DET'