

d'Overbroeck's College

NBA Record Predictions



Computer Science Project

Ian Cheng

March-30-2019

Contents

1	Analysis.....	4
1.1	Overview.....	4
1.1.1	Useful Knowledge.....	4
1.2	Research.....	5
1.2.1	Available Systems	5
1.2.2	Methods to the problem.....	5
1.2.2.1	Linear Regression.....	5
1.2.2.2	Limitations of Linear Regression.....	6
1.3	Proceeding method	6
1.3.1	Retrieving statistical data.....	7
1.3.1.1	Pandas	8
1.3.2	Data Storage	8
1.3.2.1	CSV.....	9
1.3.2.2	MySQL.....	9
1.3.2.3	SQLAlchemy	9
1.3.3	Generating team score and record	10
1.3.3.1	Scoring efficiently	10
1.3.3.2	'Free points'	11
1.3.3.3	Offensive Rebounds	11
1.3.3.4	Protecting the ball.....	11
1.3.4	Optimization.....	12
1.3.4.1	Genetic Algorithms	12
1.4	Limitations.....	15
1.5	Objectives	16
2	Documented Design.....	18
2.1	Program Flowchart	18
2.2	Retrieving Statistical Data.....	18
2.2.1	Web Scrape algorithm.....	19
2.2.2	Manipulating data.....	19
2.2.3	External Libraries Used	21
2.3	Database connection and storage.....	21
2.3.1	Python-SQL connection.....	21
2.3.2	Database Storage	22
2.3.3	External Libraries Used	24
2.4	Pseudocode for data retrieval and storage.....	24
2.5	Querying	24
2.5.1	How to Query	24
2.5.2	Pseudocode for querying.....	26
2.5.3	External Libraries Used	27
2.6	Generating Team Score.....	27
2.6.1	Team score formula	27
2.6.2	Pseudocode for generating team score	27

2.7	Determining Record	28
2.7.1	Method	28
2.7.2	Getting the teams	28
2.7.3	Simulating games	28
2.7.4	Simulate Season.....	29
2.7.5	Pseudocode for determining record.....	29
2.7.6	External Libraries Used	31
2.8	Genetic Algorithm	31
2.8.1	Initialization	31
2.8.2	Fitness Function	32
2.8.3	Evolution – Selection.....	32
2.8.4	Evolution – Crossover.....	32
2.8.5	Evolution – Mutations.....	33
2.8.6	Pseudocode for genetic algorithm.....	33
2.8.7	External Libraries Used	34
2.9	Simulating next season.....	34
2.9.1	Simulating next season	34
3	Technical Solution.....	35
3.1	Web Scrape and Data Storage (.py)	35
3.2	Querying from Database (.py)	36
3.3	Generating Team Score (.py).....	38
3.4	Determining Record (.py).....	39
3.5	Genetic Algorithm (.py)	49
3.6	Usage Code (.py)	53
3.7	Teams Text File (.txt).....	55
3.7.1	Western.txt	55
3.7.2	Eastern.txt.....	56
3.8	Teams' Record (.json).....	56
4	Testing.....	61
4.1	Retrieving statistical data + storage.....	62
4.1.1	Retrieval results table	62
4.1.2	Test Evidence	63
4.2	Querying	67
4.2.1	Querying Results Table	67
4.2.2	Test Evidence	67
4.3	Generating Team Score.....	69
4.3.1	Generating Team Score Results Table	69
4.4	Determining Record	72
4.4.1	Determining Record Results Table	72
4.4.2	Test Evidence	72
4.5	Genetic Algorithm	75
4.5.1	Genetic Algorithm Results Table	75
4.5.2	Test Evidence	76
4.6	Prediction.....	78

4.6.1	Prediction Results Table.....	78
4.6.2	Test Evidence	79
5	Evaluation.....	81
5.1	Objectives Reflection.....	81
5.2	Functionality of entire program evaluation	83
5.2.1	Prediction	83
5.2.2	Program run time	83
5.2.3	Improvements	83
References		85

1 Analysis

1.1 Overview

In recent years, the usage of statistics in sports has grown in popularity. Statistical analysis has been used by sports teams to make major decisions such as which players to draft or trade for, how to play defense on specific teams and how much effect a player has on the team's performance. This was first popularized by Oakland Athletics' General Manager Billy Beane of Major League Baseball (MLB) where he prioritized the use of data analysis to make personnel decisions in the late 1990s. His success with the team, despite being at a heavy disadvantage in terms of the money they had to pay players, led to other teams in the MLB and more importantly franchises from other sports such as basketball to adopt the approach of using statistics.

There are a lot of research into statistical analysis in baseball but significantly less for basketball and with the growing popularity of basketball around the world (also being a big basketball fan myself), I decided to make a program that predicts the record (wins/losses) for future seasons of any team in the National Basketball Association (NBA).

The NBA is a professional men's basketball league based in North America with a total of 30 teams. It is regarded as the best professional basketball league with the best players in the world and in turn have many statistics that are recorded of players and teams. The goal for this program is to take these past statistics, process them and make a prediction on a team's future record.

1.1.1 Useful Knowledge

This section contains a list of knowledge / explanation of jargon that should be noted to better understand this report.

- There are two conferences in the NBA. The Western and the Eastern conference and each conference has 3 divisions which contain 5 teams each totaling to 30 teams in the NBA
 - Different divisions play differing number of games with different opponents
- But each team in the NBA plays 82 games total in a season
- Each game is 48 minutes split into four 12-minute quarters
- There are **no ties** in a game. If the scores are equal at the end of the regulation time, overtime (5 minutes) is played. If the scores are still equal, they play another overtime and keep on doing this until a winner is decided
- Record, presented like 'wins-loss' (49-33) are a way of showing how many games a team won and lost in a season. Adds up to 82

- Eight teams with the best record in each conference make the Playoffs

1.2 Research

1.2.1 Available Systems

I first searched on the internet to look if there are any systems that do something similar to this. I found that there are some systems that predict the result of individual NBA games and some systems which predict a team's record as a whole.

Many of these systems are developed by betting websites. For example, oddsshark.com (a betting website) has a page on upcoming games and the odds of a team winning coupled with the odds of a bet. However, only stats that are of interest to bettors are displayed and "irrelevant" ones aren't. Another example is FiveThirtyEight (538). 538 is a website that is focused on analysis of politics, economics and sports. They have a projected record displayed on their website predicted from their in-house system CARMELO (Career-Arc Regression Model Estimator with Local Optimization). CARMELO forecasts a player's future performance based on trajectories of similar players and uses this to generate a team rating to develop of record projection.

Other projection systems are simply expert's polls combined with non-expert voters. These are used to give teams a power ranking (A ranking of teams based on a derived rating) and determine a record this way. This is generally the opinion of a few individuals (in the case of NBA's official power ranking, one person's opinion). Though objective, still opinion based.

Upon my research, I also found some undergraduate and graduate [1], [2], [3] thesis papers which tackles this problem.

These are systems and people who also set out to tackle some variation of essentially the same problem. However, none of which is readily available for a user to use.

1.2.2 Methods to the problem

During my research into available systems, I found that there are a few methods that can be used to tackle this problem

1.2.2.1 Linear Regression

Many of the aforementioned thesis papers used linear regression as a model to predict some result. This uses some variables to find the combination of coefficients that give the lowest sum of the squared differences between the observed dataset and the actual

dataset.

$$\sum (O_i - E_i)^2$$

Where O is the predicted value and E is the true value.

Team, opponent and location can be used as variables to predict point differentials [3] or the teams can be used as a variable and scores as values or using a PER (a metric that measures a player's effectiveness) [1] as input to output a team's win ratio.

1.2.2.2 Limitations of Linear Regression

A major limitation of using a linear regression model of past data is that it doesn't account for players leaving or joining the team when predicting future records. Linear regression will find the relationship between variables and will base it upon the input data. i.e. the roster of the input team (if a player leaves the team, it will not affect the result).

A team's roster is a very big factor in basketball. Even one player (especially if they are All Star caliber) joining or leaving a team can have a very big impact on the team's performance. For instance, LeBron James and the Cleveland Cavaliers had a record of 61-21 in the 2009-2010 season, topping the eastern conference. However, next season when LeBron James left the team to join the Miami Heat, Cavaliers' record plummeted to just 19-63. The following three seasons were also disastrous with their best record being 33-49 until LeBron James came back in the 2014-2015 season. Instantly their record increased to 53-29 becoming the second in the eastern conference and went to the NBA Finals, compared to them not even making the Playoffs the previous four years without James. Furthermore, LeBron James left again before the 2018-2019 season and (as of 4th March 2019) Cavaliers have a record of 19-48. With 15 games left to play this season, even if they win all of their remaining games (which is very unlikely) they would still have a losing record of 34-48.

This shows how important accounting for the players on a team is when predicting a team's record.

1.3 Proceeding method

I decided to come up with a solution that takes into account the roster of a team every year. This method first obtains all the data for every player in the NBA and in a form that can be easily requested. Secondly, the data will be combined in a way that would give every team a power score based on the stats of the players on the team. This will account for players changing teams as their value is added to their new team. With a score for

every team, the program can compare the scores to each other. However, not every team will put the same emphasis on parts of the game. Some teams may aim to only take high percentage shots making their Field goal percentage higher than other teams or teams with very good Big men will try to get more offensive rebounds so the importance of each stat to each team is different and in turn the weighting of each stat is unique to the team.

So, this method will mainly rely on optimizing the weights put on each factor of the score. It will find the most representable weights for each team.

1.3.1 Retrieving statistical data

First thing I need is to get ahold of the past data. The ideal situation would be an NBA statistics API where requests on the needed statistics can be made and the server sends that data back. This would be good as I will be able to get real time stats in the easiest way possible. However, stats.nba.com (Official NBA stats website) limits the number of APIs that are available to the public and other available APIs are paid subscription based making it very hard to retrieve stats this way. There are also no public databases that can be queried from.

Another valid option is to utilize web scraping. Even though there aren't available databases, there are a lot of websites that display the required statistics. Basketball-reference.com is a particularly good website to web scrape from because they have every stat of every player per season displayed on one single web page. This means that everything in a season can be obtained with one single web scrape.

Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
1	Alex Abrines	SG	24	OKC	75	8	1134	115	291	.395	84	221	.380	31	70	.443	.540	39	46	.848	26	88	114	28	38	8	25	124	353
2	Quincy Acy	PF	27	BRK	70	8	1359	130	365	.356	102	292	.349	28	73	.384	.496	49	60	.817	40	217	257	57	33	29	60	149	411
3	Steven Adams	C	24	OKC	76	76	2487	448	712	.629	0	2	.000	448	710	.631	.629	160	286	.559	384	301	685	88	92	78	128	215	1056
4	Bam Adebayo	C	20	MIA	69	19	1368	174	340	.512	0	7	.000	174	333	.523	.512	129	179	.721	118	263	381	101	32	41	66	138	477
5	Arron Affalo	SG	32	ORL	53	3	682	65	162	.401	27	70	.386	38	92	.413	.485	22	26	.846	4	62	66	30	4	9	21	56	179
6	Cole Aldrich	C	29	MIN	21	0	49	5	15	.333	0	0	0	5	15	.333	.333	2	6	.333	3	12	15	3	2	1	1	11	12
7	LaMarcus Aldridge	C	32	SAS	75	75	2509	687	1347	.510	27	92	.293	660	1255	.526	.520	334	399	.837	246	389	635	152	43	90	111	161	1735
8	Jarrett Allen	C	19	BRK	72	31	1441	234	397	.589	5	15	.333	229	382	.599	.596	114	147	.776	144	244	388	49	28	88	82	147	587
9	Kadeem Allen	PG	25	BOS	18	1	107	6	22	.273	0	11	.000	6	11	.545	.273	7	9	.778	4	7	11	12	3	2	9	15	19
10	Tony Allen	SF	36	NOP	22	0	273	44	91	.484	4	12	.333	40	79	.506	.505	11	21	.524	20	27	47	9	11	3	19	49	103
11	Al-Farouq Aminu	PF	27	POR	69	67	2072	230	582	.395	125	339	.369	105	243	.432	.503	59	80	.738	97	428	525	84	79	40	79	136	644
12	Justin Anderson	SF	24	PHI	38	0	519	87	202	.431	34	103	.330	53	99	.535	.515	28	38	.737	25	68	93	25	15	7	16	54	236
13	Kyle Anderson	SF	24	SAS	74	67	1978	231	438	.527	19	57	.333	212	381	.556	.549	104	146	.712	84	312	396	202	115	60	94	114	585
14	Ryan Anderson	PF	29	HOU	66	50	1725	207	480	.431	131	339	.386	76	141	.539	.568	72	93	.774	94	237	331	60	24	21	42	126	617
15	Ike Anigbo	C	19	IND	11	0	30	4	9	.444	0	0	0	4	9	.444	.444	5	6	.833	5	4	9	0	1	3	2	1	13
16	Giannis Antetokounmpo	PF	23	MIL	75	75	2756	742	1402	.529	43	140	.307	699	1262	.554	.545	487	641	.760	156	597	753	361	109	106	223	231	2014
17	Carmelo Anthony	PF	33	OKC	78	78	2501	472	1168	.404	169	474	.357	303	694	.437	.476	148	193	.767	67	386	453	103	47	49	99	197	1261
18	OG Anunoby	SF	20	TOR	74	62	1481	163	346	.471	73	197	.371	90	149	.604	.577	39	62	.629	44	140	184	55	52	14	45	130	438
19	Ryan Arcidiacono	PG	23	CHI	24	0	304	17	41	.415	9	31	.290	8	10	.800	.524	5	6	.833	1	24	25	35	13	0	13	18	48
20	Trevor Ariza	SF	32	HOU	67	67	2269	268	651	.412	170	462	.368	98	189	.519	.542	76	89	.854	33	261	294	105	98	13	52	132	782
Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
21	Darrell Arthur	PF	29	DEN	19	1	141	22	47	.468	8	23	.348	14	24	.583	.553	2	3	.667	3	12	15	9	8	3	15	22	54
22	Jamel Artis	SG	25	ORL	15	1	279	31	79	.392	8	29	.276	23	50	.460	.443	7	12	.583	4	34	38	18	2	3	8	11	77
23	Omer Asik	C	31	TOT	18	0	182	9	22	.409	0	0	0	9	22	.409	.409	4	13	.308	9	38	47	3	2	4	9	20	22
23	Omer Asik	C	31	NOP	14	0	121	7	16	.438	0	0	0	7	16	.438	.438	4	12	.333	7	30	37	2	1	2	5	14	18
23	Omer Asik	C	31	CHI	4	0	61	2	6	.333	0	0	0	2	6	.333	.333	0	1	.000	2	8	10	1	1	2	4	6	4
24	D.J. Augustin	PG	30	ORL	75	36	1760	244	540	.452	114	272	.419	130	268	.485	.557	164	189	.868	30	130	160	287	54	0	123	94	766
25	Luke Babbitt	SF	28	TOT	50	14	715	91	215	.423	60	156	.385	31	59	.525	.563	17	22	.773	7	89	96	31	8	7	19	54	259

Figure 1: Top of table on basketball-reference

519	Derrick Williams	PF	26	LAL	2	0	9	1	4	.250	0	2	.000	1	2	.500	.250	0	0	0	1	1	0	0	0	0	0	2	
520	Lou Williams	SG	31	LAC	79	19	2589	582	1337	.435	186	518	.359	396	819	.484	.505	432	491	.880	40	158	198	417	85	19	234	106	1782
Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
521	Marvin Williams	PF	31	CHO	78	78	2006	258	563	.458	126	305	.413	132	258	.512	.570	102	123	.829	74	296	370	96	56	38	66	116	744
522	Matt Williams	SG	24	MIA	3	0	11	2	6	.333	1	5	.200	1	1	1.000	.417	0	0	0	0	1	1	0	0	0	1	1	5
523	Troy Williams	SF	23	TOT	21	1	307	51	109	.468	11	38	.289	40	71	.563	.518	20	30	.667	22	41	63	16	20	4	19	34	133
523	Troy Williams	SF	23	HOU	4	0	17	2	9	.222	0	5	.000	2	4	.500	.222	1	3	.333	1	3	4	1	1	0	1	2	5
523	Troy Williams	SF	23	NYK	17	1	290	49	100	.490	11	33	.333	38	67	.567	.545	19	27	.704	21	38	59	15	19	4	18	32	128
524	D.J. Wilson	PF	21	MIL	22	0	71	9	16	.563	2	5	.400	7	11	.636	.625	1	2	.500	2	8	10	3	3	1	4	7	21
525	Jamil Wilson	SF	27	LAC	15	10	274	38	81	.469	27	63	.429	11	18	.611	.636	2	4	.500	3	29	32	10	5	8	8	24	105
526	Justise Winslow	PF	21	MIA	68	25	1680	207	488	.424	49	129	.380	158	359	.440	.474	66	104	.635	64	306	370	148	54	33	77	140	529
527	Jeff Withey	C	27	DAL	9	0	39	6	16	.375	2	10	.200	4	6	.667	.438	1	2	.500	1	9	10	0	0	3	2	3	15
528	Nate Wolters	PG	26	UTA	5	0	19	1	6	.167	0	0	0	1	6	.167	.167	0	0	0	1	1	2	1	0	0	0	1	2
529	Brandon Wright	PF	30	TOT	28	1	381	59	102	.578	0	0	0	59	102	.578	.578	21	33	.636	33	62	95	13	13	26	8	32	139
529	Brandon Wright	PF	30	MEM	27	1	366	57	99	.576	0	0	0	57	99	.576	.576	21	33	.636	32	61	93	13	13	25	8	29	135
529	Brandon Wright	PF	30	HOU	1	0	15	2	3	.667	0	0	0	2	3	.667	.667	0	0	0	1	1	2	0	0	1	0	3	4
530	Delon Wright	PG	25	TOR	69	4	1433	201	432	.465	56	153	.366	145	279	.520	.530	97	117	.829	45	153	198	200	72	33	78	81	555
531	Guerschon Yabusele	PF	22	BOS	33	4	235	26	61	.426	12	37	.324	14	24	.583	.525	15	22	.682	17	35	52	16	4	5	12	23	79
532	James Young	SG	22	PHI	6	0	61	5	14	.357	3	10	.300	2	4	.500	.464	4	6	.667	0	2	2	2	0	0	1	4	17
533	Joe Young	PG	25	IND	53	1	558	80	186	.430	25	66	.379	55	120	.458	.497	22	29	.759	12	51	63	39	14	1	26	39	207
534	Nick Young	SG	32	GSW	80	8	1393	201	488	.412	123	326	.377	78	162	.481	.538	56	65	.862	20	105	125	36	38	7	39	100	581
535	Thaddeus Young	PF	29	IND	81	81	2607	421	864	.487	58	181	.320	363	683	.531	.521	55	92	.598	184	328	512	152	135	36	105	175	955
536	Cody Zeller	C	25	CHO	33	0	627	85	156	.545	2	3	.667	83	153	.542	.551	61	85	.718	67	110	177	31	14	21	33	81	233
537	Tyler Zeller	C	28	TOT	66	34	1109	187	334	.560	10	28	.357	177	306	.578	.575	57	79	.722	110	195	305	47	15	35	47	126	441
537	Tyler Zeller	C	28	BRK	42	33	703	125	229	.546	10	26	.385	115	203	.567	.568	40	60	.667	63	131	194	28	8	21	35	78	300
537	Tyler Zeller	C	28	MIL	24	1	406	62	105	.590	0	2	.000	62	103	.602	.590	17	19	.895	47	64	111	19	7	14	12	48	141
538	Paul Zipser	SF	23	CHI	54	12	824	81	234	.346	37	110	.336	44	124	.355	.425	19	25	.760	13	118	131	46	20	15	43	86	218
539	Ante Zizic	C	21	CLE	32	2	214	49	67	.731	0	0	0	49	67	.731	.731	21	29	.724	24	36	60	5	2	13	11	30	119
540	Juice Zubac	C	20	LAL	43	0	410	61	122	.500	0	1	.000	61	121	.504	.500	39	51	.765	45	78	123	25	8	15	26	47	161

Figure 2: Bottom of table on basketball-reference

Another advantage is that the data is displayed in the form of a HTML table which will make it convenient for us to web scrape. Additionally, the difference between URL from season to season is just the year number so it's easy to create a loop to web scrape from different seasons in the future.

1.3.1.1 Pandas

Pandas (Python data analysis library) is an external library for Python which is designed for data manipulation and analysis. There are a few inbuilt functions like `read_html()` and `to_sql` which are useful to what I want to do.

pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

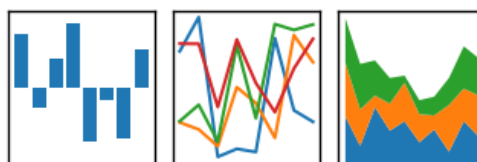


Figure 3: Pandas logo

1.3.2 Data Storage

Web scraped data in Pandas is returned to a Pandas DataFrame which is an array that holds data with labels. This is no good because this is essentially a variable array which disappears when the program closes. There should be only one initial web scrape and store that data somewhere so that we don't have to web scrape every time we run the program as this can take time to do. We also want to store it in a way that we can query players by team efficiently.

1.3.2.1 CSV

A way of storage can be putting the data into a comma separated values file. This allows us to only web scrape once and get the data from the CSV file when we need it. However, a disadvantage of storing in a CSV file is that even though we will not need to web scrape every time, we will still need to load the file into a DataFrame and store it at the start of the program then get the necessary stats. Considering the seasons, players and the number stats/ fields there are, this can take up more than necessary memory.

Considering these factors, I decided to store the stats in a database management system. Because of the nature of the data, it makes sense to store it in a relational database management system following the relational database model. Storing in a database solves all our problems. We only need one initial web scrape and the data does not need to be loaded into a variable at the start of the program. Instead we can query only the necessary data that we wish for. Avoiding unnecessary usage of memory to store the DataFrame.

1.3.2.2 MySQL

MySQL is a good database to use as it is open-source relational database management system. It uses tables as the main management system which works well with our database model.



Figure 4: MySQL Logo

1.3.2.3 SQLAlchemy

SQLAlchemy is a third-party toolkit that allows for database interactions with Python. It can create a connection between Python and the database whereby data can be queried or put in the database from a Python program.



Figure 5: SQLAlchemy Logo

1.3.3 Generating team score and record

A team score will be generated by combining every team's player statistics the same way. The method I will use is Dean Oliver's "Four Factors of Basketball Success" [4]. Dean Oliver is a statistician who contributed significantly to the use of APBRmetrics (usage of statistics in the analysis of basketball). These four factors are: to score efficiently, get as many 'free' points as possible, grab as many rebounds as possible and to protect the basketball. I am going to use these factors because they are closely related to the team's performance. They are most related to the team getting as many points as possible (which is ultimately the decider for any basketball game.)

1.3.3.1 Scoring efficiently

The most efficient team in terms of scoring will be to make shots on every possession but it's practically impossible that every possession will result in points added for the team. However, the more efficiently a team does this, undoubtedly, will be better off.

A general statistic to measure this is by Field Goal Percentage (FG%). This is essentially the number of shots a player makes divided by the number of attempted shots. However, this is not the best representation of scoring efficiency because on every possession, a player can either score a 2-point field goal or a 3-point field goal.

Consider Player A taking 10 shots and making 2 2-pointers and 2 3-pointers. His FG% would be 40% and he added 10 points to his team. Now consider Player B taking 10 shots but making 5 2-pointers and 0 3s. His FG% is 50% which is higher than Player A's but he has added the same number of points to his team. He's scoring 'more efficiently' even though he's added the same value.

A solution to this is to use a metric called **effective Field Goal Percentage (eFG%)**. eFG% accounts for the fact that the three point is worth 1 more than the two point.

$$eFG\% = \frac{FGM + 0.5 \times 3PM}{FGA}$$

Equation 1: eFG% Formula

Equation 1 shows the formula for calculating effective field goal percentage. Considering the scenario above, both of the players now have eFG% of 50%. This gives us the best relative measure for points per field goal attempted. In other words, how efficient the player is scoring.

1.3.3.2 'Free points'

Another way of scoring points is at the free throw line. Every free throw is worth 1 point and are at a set range where 1 player shoots without anyone contesting which makes it a guaranteed attempt at scoring as oppose to possessions where teams are not guaranteed to have an attempt at scoring (turnovers).

$$FTRate = \frac{FT}{FGA}$$

Equation 2: Free throw rate formula

The free throw rate formula measures how often a team gets to the free throw line and how often they make them. The number of free throws made can be divided by the number Field Goal Attempts instead of the number of possessions is because usually teams average under one FGA per possession.

1.3.3.3 Offensive Rebounds

If a team is unable to score every possession, the best option is to grab an offensive rebound so that the team has a second chance of scoring. Offensive rebounds extend a team's possession and gives them an extra attempt at a field goal.

$$ORB\% = \frac{\left(\frac{ORB}{82}\right)}{100}$$

Equation 3: Offensive rebound formula

Equation 3 shows our formula for calculating offensive rebounds for a team. We divide the total number of offensive rebounds by 82 (number of games) and then divide by 100.

1.3.3.4 Protecting the ball

This is measured with turnover percentage (TOV%). A turn over means that a team does not get an attempt at field goal so the lower the TOV% the better. TOV% is calculated using the following formula:

$$TOV\% = \frac{TOV}{(FGA + 0.44 \times FTA + TOV)}$$

Equation 4: Turnover percentage formula

Players who are on the court more, have the ball more and make more plays will more often than not will have more turnovers than other players. This formula takes that into account by including FTA and FGA.

By generating a power score for every team, a team's record can be determined by comparing the score to other teams' score. Every team plays each opponent different times but the general formula is that a team plays:

- 4 games against the 4 other teams in the **same division**
- 4 games against 6 teams that are in the **same conference** but **different division**
- 3 games against 4 remaining teams in the **same conference** but **different division**
*
- 2 games against the 15 teams in the **opposing conference**

* five-year rotation determines which out of division team is played 3 times and which are played 4 [5]

1.3.4 Optimization

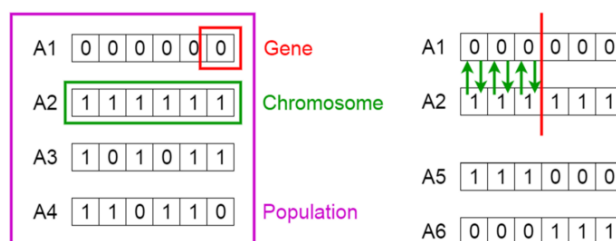
Dean Oliver assigned different weights to each of the four factor that we use to determine our power score. He says that scoring is worth 40%, turnovers are worth 25%, rebounding 20% and free throws are worth 15%. As I previously mentioned, different teams put different emphasis on parts of the game so it would be more representable to find the unique weights for each team.

Because of the nature of basketball games, we cannot 100% predict the outcome of any given match therefore we can model the problem using a metaheuristic optimization algorithm that finds a sufficiently good solution.

1.3.4.1 Genetic Algorithms

Genetic Algorithm is a metaheuristic inspired and based on Charles Darwin's Theory of Evolution and Natural Selection.

Source: [6]



Source: Adapted from [6]

Initialization [6]

Page 13 of 85

proposed solutions to the problem. A fitness function is also proposed. Fitness function is an objective that is to be reached with an individual, and optimal solution to the problem. Usually this is a single figure of merit.

Fitness assessment [6]

The fitness of all the individuals in the population is assessed using the fitness function. This is usually evaluated on the basis of how close it is to the final solution.

Evolution – Selection [6]

From selection onwards is where the process of evolution comes in. Selection from the existing population is made to breed new generations. Every individual in the population has a fitness score based on the fitness assessment. In the selection process, a portion of the top performers are selected to go to the next generation (Natural Selection). Similarly, lesser performers are also selected to promote genetic diversity. This is done to avoid getting stuck at a local solution and hence not being able to find the real optimal solution.

Evolution – Crossover [6]

In the crossover stage, ‘parents’ (selected individuals) are bred together to form the new population. The ‘children’ are bred until the population returns to its original size. Crossover algorithms used are problem dependent but the most common ones are single point crossover: a point on the parents’ chromosome is randomly chosen and any genes behind that point is swapped between the two parents to form new off springs.

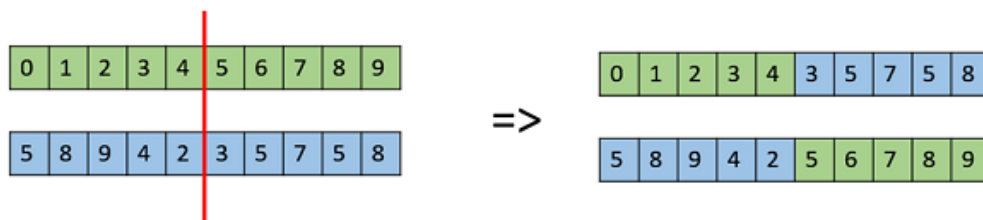


Figure 8: Single Point crossover

Source: [8]

Multi-point crossover: two points on the parents’ chromosomes are randomly chosen and the genes in the segments are swapped

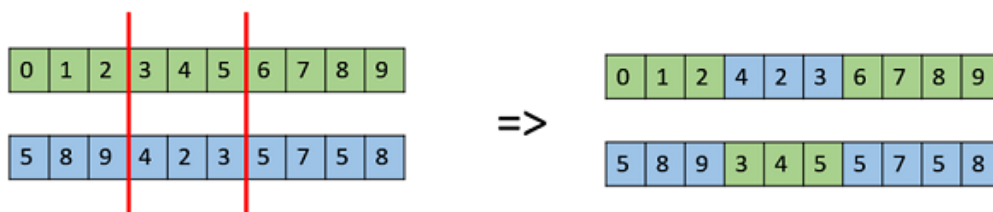


Figure 9: Multi-point crossover

Source: [8]

Uniform crossover: Chromosomes are not divided into segments, rather each gene has a 50/50 change of getting swapped.

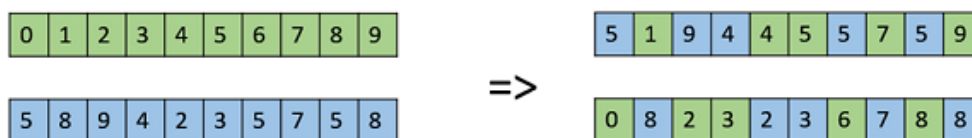


Figure 10: Uniform crossover

Source: [8]

Evolution – Mutation [6]

Mutation is a process where there is random chance that a random gene is randomly modified. This is also a process to promote genetic diversity.

Stopping Criteria [6]

After every generation, the stopping criteria is assessed based on the fitness function to see if an individual has reached the optimal solution yet. If not, the algorithm repeats from fitness assessment and every stage is carried out again until the stopping criteria is reached.

1.4 Limitations

Although the program doesn't need to web scrape every time it runs, it does however, need to web scrape when statistics for the most recent season comes out. Basketball-reference updates the website every day but if we want to predict the record for next season, we have to wait until the current season is over to have all the data to process. This is also a second limitation as we don't know a team's roster for the season before hand, we can only make predictions when they come out which is usually close to the start of the season. For this reason, we are limited to predicting one season at a time. During the season, players can also be traded from team to team. Sometimes these can be very impactful players so a prediction at the start of the season is not always representable for the entire season, however, a trade deadline in the middle of the season (usually the start of February) means that there are rarely any big moves after this. Predictions after this date can be representable.

There are also uncontrollable factors that affect a player's performance. This includes the frequency of games played in a given period of time and how much they need to travel. This can have an effect on a team's performance due to players' fatigue which will

become more apparent later into the season. The difference in schedule between the seasons will mean that using past data as part of the model will not be as accurate.

1.5 Objectives

Setup

1. Set up a database – This should be an empty database ready for data to be stored
2. Acquire all relevant data from basketball-reference – This should be web scraped from the web pages its self and into a DataFrame in Pandas
3. Manipulate the data so that it is the correct data types – Initially all acquired data are in text form. Manipulate it so that all data which are strings should stay as text and any data with numbers in it should be changed to double or int/BigInt
4. Manipulate the data so that it is compatible with SQL – Manipulate data so that it doesn't contain anything that violate SQL heading rules. E.g. Special characters
5. Create a connection between MySQL database and Python – Connection should be able to allow Python to query and input new data to MySQL
6. Store all the data – This should be done so that data can be queried from database instead of the program web scraping every time it runs

Program

1. Query current season players for the team – This should obtain relevant data from the database and store return it in a list to pass it forwards
2. Query previous season relevant stats for the players – Should take in the list of names as a parameter which then queries stats from database based on these names
3. Produce a team score – Stats queried for the team should be combined into a representable score that reflects how good a team is
4. Simulate season - Every game of the season should be simulated many times using randomly generated coefficients (weights on stats) to determine a record. Taking into account home court advantage and randomness of each game.
5. Genetic Algorithm for optimization - Using a Genetic Algorithm, evolve the coefficients by using the actual record as fitness function
6. Find correct coefficients – This should find the coefficients that determine the correct record for that team

7. Simulate next season – Using the correctly found coefficients, simulate every game of next season
8. Repeat simulation – Simulate next season 5 times and average the records to decrease the randomness (uncertainty)
9. Display the average predicted record for next season – Display wins and losses

2 Documented Design

2.1 Program Flowchart

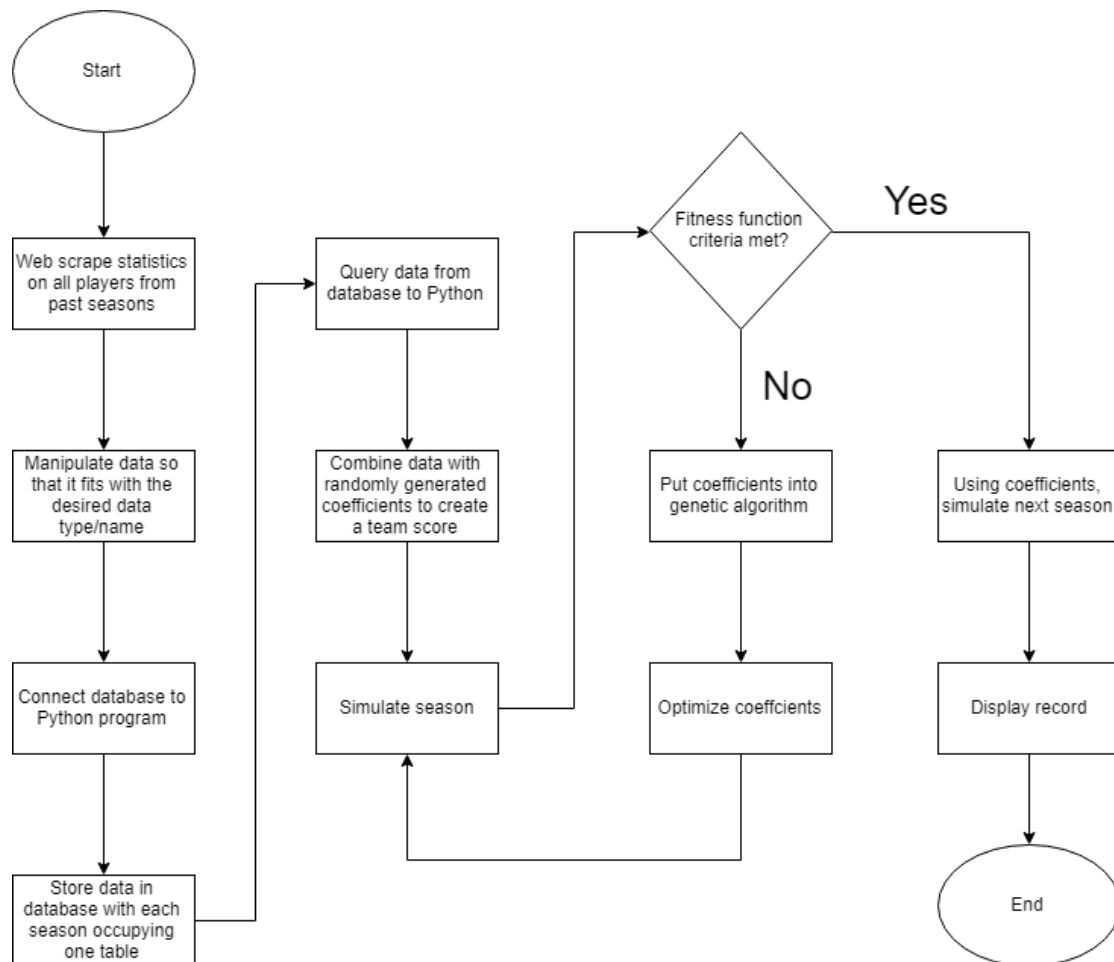


Figure 11: Program functions flowchart

2.2 Retrieving Statistical Data

The tools for web scraping I will use is the Pandas library for Python along with lxml, BeautifulSoup4 and html5lib libraries to parse HTML tables.

Pandas is used because they have an inbuilt function 'read_html()' which makes it very easy to obtain data by automating the HTML parsing, resolving some issues that may occur. One of the issues is that the lxml library may fail to parse the table due to it having strict Markup Validations for HTML tables. So, if this happens the function switches to html5lib and BeautifulSoup4 automatically to guarantee a valid return. Using lxml as the primary library instead of html5lib and BeautifulSoup4 despite the issue is because lxml

is much faster.

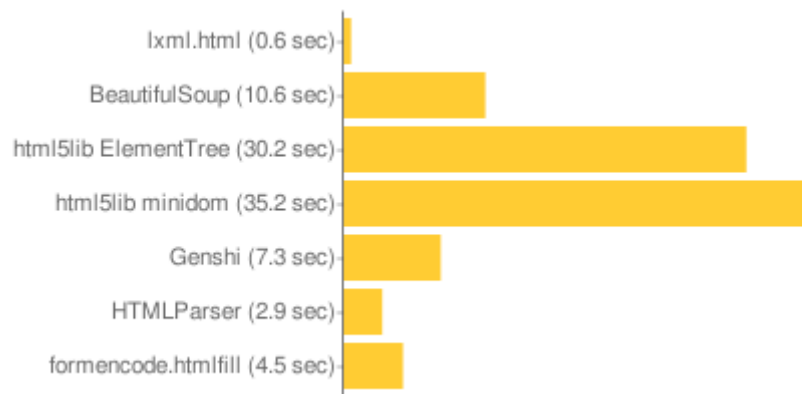


Figure 12: HTML parsing library time comparisons

Source: [9]

2.2.1 Web Scrape algorithm

https://www.basketball-reference.com/leagues/NBA_2018_totals.html

Basketball-reference URL to total stats for season 17-18

https://www.basketball-reference.com/leagues/NBA_2017_totals.html

Basketball-reference URL to total stats for season 16-17

Because the URL for pages on basketball-reference vary only by year number, it is possible to use a for loop to web scrape all the data.

2.2.2 Manipulating data

Due to basketball-reference's data layout and data type representation, some modifications are needed after web scraping. The modifications will be done on every web scraped DataFrame to return a desired DataFrame which will then be stored.

- 1) On **Figure 1** and **Figure 2** (tables on basketball-reference) it can be seen that there are rows of headings in the middle of the table (occurs every 20 rows). This is for easy viewing on the website so that the viewer knows which numbers correspond to which stats when looking at the middle or bottom of the table. As this is no use to us, we need to delete all of these rows except for the top headings as this will be used as the heading in the database. Pandas has a drop() function which allows us to drop rows or columns. However, this only drops one row or a column at a time and because there are many of these redundant rows, we will need to use a different method.

```
1 stats = stats[stats['Player'] != 'Player']
```

```

2 # This get rids of all the rows which are headings in the middle of the table
3 # From 'column_name' select all the rows that value != values (player)

```

***from DataFrame 'stats', get all the rows in column 'Player' (header) which doesn't have the value 'Player' and store it in new DataFrame called stats.

- 2) The headings on basketball-reference uses special characters like '%' and '/'. These are invalid characters to use for headings in MySQL which means that if we keep the headings, we will not be able to query anything from the database as it will return an error. The heading will need to be changed to something that is accepted by MySQL. We do this by using the rename() function.

```

1 stats.rename(columns={'FG%':'FGp', '3P%':'3Pp', '2P%':'2Pp', 'eFG%':'eFGp',
2 'FT%':'FTp'}, inplace = True )
3 #Headings violate mysql rules with special characters. Changes headings to fit those
4 rules.

```

- 3) The retrieved data are all in the text data type as shown in **Figure 13**.

```
mysql> DESCRIBE 2k18;
```

Field	Type	Null	Key	Default	Extra
Player	text	YES		NULL	
Pos	text	YES		NULL	
Age	text	YES		NULL	
Tm	text	YES		NULL	
G	text	YES		NULL	
GS	text	YES		NULL	
MP	text	YES		NULL	
FG	text	YES		NULL	
FGA	text	YES		NULL	
FG%	text	YES		NULL	
3P	text	YES		NULL	
3PA	text	YES		NULL	
3P%	text	YES		NULL	
2P	text	YES		NULL	
2PA	text	YES		NULL	
2P%	text	YES		NULL	
eFG%	text	YES		NULL	
FT	text	YES		NULL	
FTA	text	YES		NULL	
FT%	text	YES		NULL	
ORB	text	YES		NULL	
DRB	text	YES		NULL	
TRB	text	YES		NULL	
AST	text	YES		NULL	
STL	text	YES		NULL	
BLK	text	YES		NULL	
TOV	text	YES		NULL	
PF	text	YES		NULL	
PTS	text	YES		NULL	

```

29 rows in set (0.01 sec)

mysql>

```

Figure 13: Data types before modification

As most of everything stored is a number and that we are working with numbers, we need to change all the variables to the correct data type i.e. changing all the numbers to data type double and keeping all the strings as text. The function to_numeric()

changes all domains which contain a numeric number in other data types to doubles.

```
1 stats = stats.apply(pd.to_numeric, errors = 'ignore')
2 #Changes all datatypes from string to double instead of numbers in text
3 #errors = ignore, keeps all the columns with just text as string
```

- 4) Some players' name has an apostrophe in them. E.g. "D'Angelo Russell". The apostrophe can cause problems when querying. We can remove apostrophe in the names. This won't cause querying differences as long as remove it from all the names. This can be done by replacing every instance of apostrophe with nothing. This will join the name together. E.g. "D'Angelo Russell" becomes "DAngelo Russell".
- 5) In 2014, there was a team name change. The Charlotte Bobcats changed names to Charlotte Hornets. With this, the team's identifier also changed from "CHA" to "CHO". To ensure that identifier matching still works with older seasons, we should change all instances of the old identifier, "CHA" to "CHO".

2.2.3 External Libraries Used

- 1) **Pandas** – Web Scrape function
- 2) **lxml** – Main HTML table parsing
- 3) **BeautifulSoup4 / html5lib** – Backup HTML table parsing

2.3 Database connection and storage

2.3.1 Python-SQL connection

To create the connection between Python and MySQL, we use the external library SQLAlchemy and the connector PyMySQL. SQLAlchemy function `create_engine()`, creates an Engine which connects to the DBAPI. The DBAPI (Python Database API Specification) is a commonly used specification for Python to define patterns for different connection packages. It is used so that the Python application can talk to the database. The Engine connects to the DBAPI through a connection Pool and a Dialect. These are instructions on how to talk to specific kinds of DBAPI.

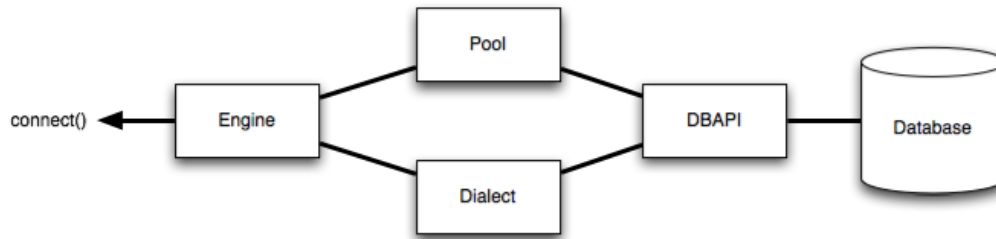


Figure 14: SQLAlchemy Engine Structure

Source: [10]

Figure 14 shows a diagram of the Engine structure for connection. The Engine references both the Pool and the Dialect interpret the behavior of that specific database.

General form to create an Engine:

```
sqlalchemy.create_engine('dialect+driver://username:password@host:port/database')
```

Dialect is the identifying name for the dialect. In the case of SQLAlchemy it is the name of the database all in lowercase. E.g. 'sqlite', 'oracle', 'mysql', 'posgresql' etc. For our Engine, it would be 'mysql'. The driver is the name of the DBAPI used to connect to the database (also in lowercase). The driver I will be using is PyMySQL as this is a driver for MySQL. The rest of the parameter are for details of the database like the username and password, the host and the port and the name of the database within the system.

Our Engine:

```

1 def sqlconnection():
2     #This function creates the connection engine between python and the database in mysql
3     Engine =
4     sqlalchemy.create_engine('mysql+pymysql://root:qazwsxedcrfv@localhost:3306/stats')
5     #dialect+driver://username:password@host:port/database
6     return engine
  
```

2.3.2 Database Storage

The data will be stored in the database with each season occupying 1 relation. The attributes will be each player's name and the stats names (Field goal, point per game, games played etc.). The tuple will the all the players' numbers.

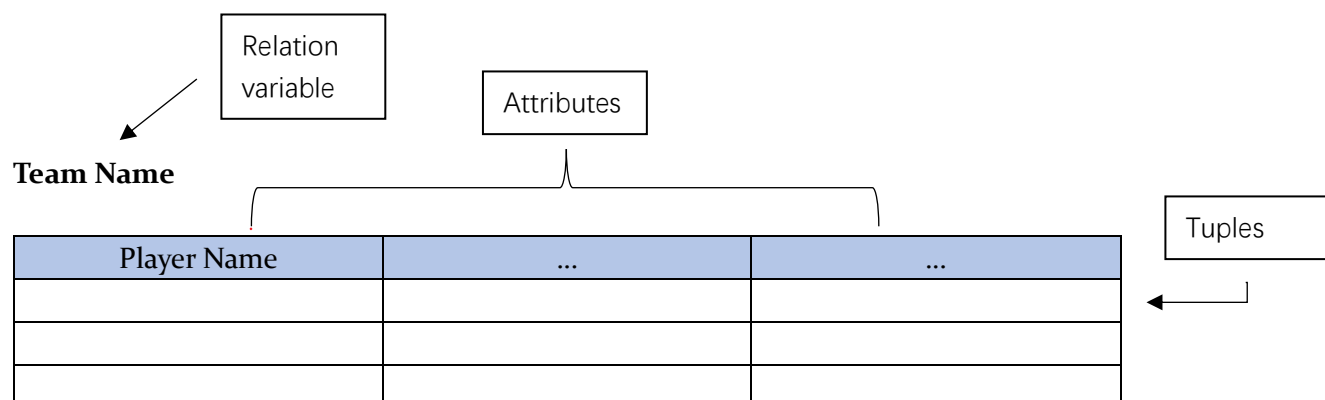


Figure 15: Database model

Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FGp	3P	3PA	3Pp	2P	2PA	2Pp	eFGp	FT	FTA	FTp	ORB	DRB	TRB	AST	STL	BLK	TOV
Alex Abrines	SG	24	OKC	75	8	1134	115	291	0.395	84	221	0.38	31	70	0.443	0.54	39	46	0.848	26	88	114	28	38	8	25
Quincy Acy	PF	27	BRK	70	8	1359	130	365	0.356	102	292	0.349	28	73	0.384	0.496	49	60	0.817	40	217	257	57	33	29	60
Steven Adams	C	24	OKC	76	76	2487	448	712	0.629	0	2	0	448	710	0.631	0.629	160	286	0.559	384	301	685	88	92	78	128
Ben Adebayo	C	20	MIA	69	19	1368	174	340	0.512	0	7	0	174	333	0.523	0.512	129	179	0.721	118	263	381	101	32	41	66
Arron Affalo	SG	32	ORL	53	3	682	65	162	0.401	27	70	0.386	38	92	0.413	0.485	22	26	0.846	4	62	66	30	4	9	21
Cole Aldrich	C	29	MIN	21	0	49	5	15	0.333	0	0	0.333	5	15	0.333	0.333	2	6	0.333	3	12	15	3	2	1	1
LaMarcus Aldridge	C	32	SAS	75	75	2509	687	1347	0.51	27	92	0.293	660	1255	0.526	0.52	334	399	0.837	246	389	635	152	43	90	111
Jarrett Allen	C	19	BRK	72	31	1441	234	397	0.589	5	15	0.333	229	382	0.599	0.596	114	147	0.776	144	244	388	49	28	88	82
Kadeem Allen	PG	25	BOS	18	1	107	6	22	0.273	0	11	0	6	11	0.545	0.273	7	9	0.778	4	7	11	12	3	2	9
Tony Allen	SF	36	NOP	22	0	273	44	91	0.484	4	12	0.333	40	79	0.506	0.505	11	21	0.524	20	27	47	9	11	3	19
Al-Farouq Aminu	PF	27	POR	69	67	2072	230	582	0.395	125	339	0.369	105	243	0.432	0.503	59	80	0.738	97	428	525	84	79	40	79
Justin Anderson	SF	24	PHI	38	0	519	87	202	0.431	34	103	0.33	53	99	0.535	0.515	28	38	0.737	25	68	93	25	15	7	16
Kyle Anderson	SF	24	SAS	74	67	1978	231	438	0.527	19	57	0.333	212	381	0.556	0.549	104	146	0.712	84	312	396	202	115	60	94
Ryan Anderson	PF	29	HOU	66	50	1725	207	480	0.431	131	339	0.386	76	141	0.539	0.568	72	93	0.774	94	237	331	60	24	21	42
Ike Anigbogu	C	19	IND	11	0	30	4	9	0.444	0	0	0.444	4	9	0.444	0.444	5	6	0.833	5	4	9	0	1	3	2
Giannis Antetokounmpo	PF	23	MIL	75	75	2756	742	1402	0.529	43	140	0.307	699	1262	0.554	0.545	487	641	0.76	156	597	753	361	109	106	223
Carmelo Anthony	PF	33	OKC	78	78	2501	472	1168	0.404	169	474	0.357	303	694	0.437	0.476	148	193	0.767	67	386	453	103	47	49	99
OG Anunoby	SF	20	TOR	74	62	1481	163	346	0.471	73	197	0.371	90	149	0.604	0.577	39	62	0.629	44	140	184	55	52	14	45
Ryan Arcidiacono	PG	23	CHI	24	0	304	17	41	0.415	9	31	0.29	8	10	0.8	0.524	5	6	0.833	1	24	25	35	13	0	13
Trevor Ariza	SF	32	HOU	67	67	2269	268	651	0.412	170	462	0.368	98	189	0.519	0.542	76	89	0.854	33	261	294	105	98	13	52
Darrell Arthur	PF	29	DEN	19	1	141	22	47	0.468	8	23	0.348	14	24	0.583	0.553	2	3	0.667	3	12	15	9	8	3	15
Jamel Artis	SG	25	ORL	15	1	279	31	79	0.392	8	29	0.276	23	50	0.46	0.443	7	12	0.583	4	34	38	18	2	3	8
Omer Asik	C	31	TOT	18	0	182	9	22	0.409	0	0	0.409	9	22	0.409	0.409	4	13	0.308	9	38	47	3	2	4	9
Omer Asik	C	31	NOP	14	0	121	7	16	0.438	0	0	0.438	7	16	0.438	0.438	4	12	0.333	7	30	37	2	1	2	5
Omer Asik	C	31	CHI	4	0	61	2	6	0.333	0	0	0.333	2	6	0.333	0.333	0	1	0	2	8	10	1	1	2	4
D.J. Augustin	PG	30	ORL	75	36	1760	244	540	0.452	114	272	0.419	130	268	0.485	0.557	164	189	0.868	30	130	160	287	54	0	123
Luke Babbitt	SF	28	TOT	50	14	715	91	215	0.423	60	156	0.385	31	59	0.525	0.563	17	22	0.773	7	89	96	31	8	7	19
Luke Babbitt	SF	28	ATL	37	9	570	80	168	0.476	49	111	0.441	31	57	0.544	0.622	17	22	0.773	5	76	81	26	7	5	15
Luke Babbitt	SF	28	MIA	13	5	145	11	47	0.234	11	45	0.244	0	2	0	0.351	0	0	0.833	2	13	15	5	1	2	4
Dwight Bacon	SG	22	CHO	53	6	713	72	192	0.375	11	43	0.256	61	149	0.409	0.404	20	25	0.8	4	120	124	38	16	2	23
Ron Baker	SG	24	NYK	29	1	385	20	59	0.339	11	33	0.333	9	26	0.346	0.432	20	26	0.769	5	25	30	47	26	6	18
Wade Baldwin	PG	21	POR	7	0	80	14	21	0.667	4	5	0.8	10	16	0.625	0.762	6	10	0.6	1	7	8	5	2	1	4

Figure 16: MySQL 2018 Table

Figure 16 shows how a table looks like after all the stats have been stored. The tables in the database are kept like this and are not normalized. Normalizing the database serves to reduce data redundancy so that problems like data loss and insertion/ deletion anomalies doesn't happen when the database is updated.

However, the reason why we don't need to normalize it is because we will not need to update our database. The stats stored in the database will never change because it is what they got in that year. They can't go back to 2015 and score 2 more points for example. For this reason, the aforementioned problems will not happen and the efficiency of querying will be improved as it will not need to call to different tables.

The stats are stored in the database after the modifications in the loop for web scrape.

```

1 def savetosql(df, num):
2     #This function puts the called dataframe into the database using the connection
3     engine = sqlconnection()

```



```

4 df.to_sql(name = '2k' + num , con = engine, index = False, if_exists = 'replace')
5 # name = table name, con = connection if_exists (if the table already exists, replace
6     everything in it)

```

This function is called which calls the connection function to create a connection in the variable engine. Pandas function `to_sql()` takes in the parameters including the connection and puts the DataFrame into MySQL. This is done for each season.

2.3.3 External Libraries Used

- 1) **SQLAlchemy** – To create a connection between Python program and MySQL
- 2) **PyMySQL** – Driver for SQLAlchemy connection
- 3) **Pandas** – Save DataFrame to MySQL

2.4 Pseudocode for data retrieval and storage

```

Import libraries

def sqlconnection():
    engine ← create engine using SQLAlchemy
    return engine

def savetosql(dataframe):
    engine ← sqlconnection()
    dataframe to sql using engine

y ← user input year
y2 ← y - 5

for range years y2 to y:
    table ← read_html( front of URL + year number + end of URL)
    Manipulation of the DataFrame
    Savetosql(table)

```

2.5 Querying

2.5.1 How to Query

To query from Python code, SQLAlchemy is again used. SQLAlchemy can execute SQL commands written in the Python program in MySQL. The `connection.execute(query)`

function takes in a parameter which is the query statement in SQL and executes this in the database linked to connection. The return result can be stored in a variable. The return result is an object as a result of the query. To get the actual data asked for, `object.fetchall()` function is used on the returned result.

When querying, we want to get the roster of the year we are predicting but the previous year stats for those players. This is because when we make the predictions, the season has not played yet so the players won't have any stats.

The first step is to query for a return of all the players on the team for this year. Store all the players in a names list and use the names list to query all the stats associated with the player from the year before.

```

1 def roster(year, team):
2     query = 'SELECT Player FROM 2k'+ str(year) + ' WHERE Tm = ' + '\'' + team + '\''
3     resultProxy = connection.execute(query)
4     ''' This executes the query statement, 'query', from dB in connection and
5     stores it in resultProxy '''
6     #resultProxy is the object returned by .execute() method
7     resultSet = resultProxy.fetchall()
8     #Actual data requested when using fetch method on resultProxy
9
10    x = []
11    for row in resultSet:
12        x.append (row[0])
13        #fetchall() return all rows and all fields
14        #Need to iterate over the rows to access the fields and get the data
15    return x

```

This function gets the current roster. A for loop is used because the return we get is everything albeit these fields would be NULL values, we still only want the names. So, we only append the zeroth element of every row (this is the index where the names are) to the names list.

```

1 def stats(year, team):
2     players = roster(year, team)
3     #players is a list with players from predict year
4     playerStats = []
5     for i in players:
6         query = 'SELECT * FROM 2k'+ str(year - 1) + ' WHERE Player = ' + '\'' + i + '\'' + ' and Tm =
7             \''TOT\' '
8         ''' Players who played for multiple teams will have multiple records in the table
9         This is to Query for just the reccord of their totals as oppose to a specific team '''
10        resultProxy = connection.execute(query)

```

```

11     resultSet = resultProxy.fetchall()
12
13     if not resultSet:
14         #If list is empty
15         query = 'SELECT * FROM 2k'+ str(year - 1) + ' WHERE Player = ' + '\''+ i + '\''
16         resultProxy = connection.execute(query)
17         resultSet = resultProxy.fetchall()
18
19     if resultSet:
20         for row in resultSet:
21             playerStats.append( {'eFG%': row[16], 'FGA': row[8], 'FTA': row[18], 'TOV' :
22                                 row[26], 'FT': row[17], 'ORB': row[20]})
23
24     return playerStats

```

This is the function for getting the stats. PlayerStats is a list of dictionaries. Each dictionary is the stats for each player. For example, playerStats = [{Stephen Curry's stats}, {Kevin Durant's stats}, {Klay Thompson's Stats} ...]. Doing this, the list is the stats for a single team however, the dictionaries will not have a player's name tied to it because from this point onwards, it doesn't matter which stats belongs to who as long as it's a player on the same team.

2.5.2 Pseudocode for querying

Import libraries

Engine \leftarrow create engine using SQLAlchemy

Connection \leftarrow engine.connect()

def roster(year, team):

SQL query statement \leftarrow select players from 'year' where team == team

resultProxy \leftarrow connection.execute()

resultSet \leftarrow resultProxy.fetchall()

namesList \leftarrow list

for row in resultSet:

append index of row in resultSet to names

return namesList

def stats(year, team):

names = roster(year, team)

for I in names:

SQL query statement \leftarrow select stats from 'year -i' where player == names (i)

resultProxy \leftarrow connection.execute()

```

resultSet ← resultProxy.fetchall()
playerStats ← append necessary stats to this list of dictionaries

return playerStats

```

2.5.3 External Libraries Used

- 1) **SQLAlchemy** – To create a connection between Python and MySQL and to execute query statements in MySQL from Python
- 2) **PyMySQL** – Driver for SQLAlchemy connection

2.6 Generating Team Score

2.6.1 Team score formula

The Four Factors are going to be used to determine a team score. They are going to be combined with positive stat adding to the score and negative stat taking away from the score. Each factor is going to have a weight on it.

$$Score = x \cdot eFG\% + y \cdot ORB\% + z \cdot FTRate - w \cdot TOV\%$$

Equation 5: General team score formula

Equation 5 shows the formula for calculating the team score. *eFG%*, *ORB%* and *FTRate* are all positive multipliers for a team. They should all add to the power score for the team whereas turnover percentage measures a team's carelessness. The higher the *TOV%*, the worst off the team will be therefore this should be taken off of the power score. In the above formula *x*, *y*, *z* and *w* represent the weights for each team that is going to be optimized.

The factors in **Equation 5** are **team** factors. It is the sum of individual players' factors

2.6.2 Pseudocode for generating team score

```

Import functions from querying

def teamScore():
    call function from querying to get list of dictionaries
    loop through the items in list (dictionaries)
        add each players' factor score to the team's factor score
    apply formula for each factor
    calculate team score

```

2.7 Determining Record

2.7.1 Method

For every team, we call the team score function to generate a score for every team.

To determine the record, we simulate every game 1000 times. Taking into account randomness and home court advantage. We simulate a total of 82,000 games per season for one team and count the number of wins. We don't need to count the number of losses because the number of games per season is always 82. We just need to subtract the number of wins from that to determine the losses.

2.7.2 Getting the teams

Every team is stored in a text file. One for teams in the East and one for teams in the West. We read the files into Python and create two 2d arrays. Each array inside the two 2d arrays is an array containing the five teams in the same division.

Example:

```
West      =      [[DEN,OKC,POR,UTA,MIN],      [GSW,LAC,SAC,LAL,PHO],
[HOU,SAS,DAL,NOP,MEM]]
```

```
East = [[TOR,PHI,BOS,BRK,NYK], [MIL,IND,DET,CHI,CLE], [CHO,ORL,MIA,WAS,ATL]]
```

The reason why we do this is because a team plays a different amount of games against opponents from different divisions and different conference. So, a 2d array allows for a call to the same division to play the same number of games and different division and conference for different number of games.

2.7.3 Simulating games

When simulating a game, we randomly add a score from 0 to 0.25 for the home team and 0 to 0.15 for the away team. This gives a slight advantage to the home team as there is a chance of a higher number added to their score. The added-on score is random so that a team doesn't have a 100% chance of winning from their calculated score previously as this is not the case in NBA games. Majority of NBA games have a score difference at the end of the game within 10 points and many games are decided on a last second shot so no team has a definitive chance of winning when playing against a specific opponent.

There will be a function to simulate a game when the team we are simulating is at home

and when the team is away. The coefficients will only modify the main team we are simulating so the coefficients are passed as a parameter to modify either the home or the away team in each of the functions.

After the 1000 simulations, we calculate which team won more of the 1000 games and that team is determined to be the winner of that one game. If they win the same number of games, then it is counted as a win for the away team.

2.7.4 Simulate Season

The function to simulate the season calls upon the simulate games function to simulate every game of the season to determine a record. Four games will be simulated against the four other teams in the same division. Two will be at home and two will be away. For the 15 teams in the other conference, two games will be simulated for each team with one at home and one away. As with the remaining 10 teams in the same conference but different division, 4 games will be played against 6 of these teams and the remaining 4 teams will be played 3 times. Which teams gets played three times or four times is determined by a five-year rotation in the NBA.

As I do not know the NBA's five-year rotation and even if I did, it will be different for every team, I decided to random which 6 of the 10 teams gets played four times and which 4 gets played three times. For the teams that gets played four times, two will be at home and two will be away as with the teams that get played three times, once at home and once away. The last game will be random as to who will be home or away.

2.7.5 Pseudocode for determining record

```

Import libraries
Import functions

def westTeams():
    read text file for west teams
    for line in file
        append teams for first division
        append teams for second division
        append teams third division
    close file

def eastTeams():
    read text file for east teams
    for line in file
        append teams for first division
        append teams for second division

```

```
        append teams third division
    close file

def simulateGameHome():
    call function to get team scores
    For 1000 times:
        home team score add number between 0 - 0.25
        away team score add number between 0 - 0.15
        if homeScore > awayScore:
            homeWin + 1
        else
            awayWin + 1

    If homeWin > awayWin:
        return won
    else
        return lost

def simulateGameAway():
    call function to get team scores
    For 1000 times:
        home team score add number between 0 - 0.25
        away team score add number between 0 - 0.15
        if homeScore > awayScore:
            homeWin + 1
        else
            awayWin + 1

    If homeWin > awayWin:
        return lost
    else
        return won

def SimulateWest():
    same division – four teams
    Simulate 2 games at home
    Simulate 2 games away

    Other conference
    Simulate 1 game at home
    Simulate 1 game away

    Same conference different division
    Random 6 teams to play 4 times
```

```

    Simulate 2 at home
    Simulate 2 away
    Random 4 teams to play 3 times
    Simulate 1 at home
    Simulate 1 away
    Random either simulate at home or away

def Simulateeast():
    same division – four teams
    Simulate 2 games at home
    Simulate 2 games away

    Other conference
    Simulate 1 game at home
    Simulate 1 game away

    Same conference different division
    Random 6 teams to play 4 times
    Simulate 2 at home
    Simulate 2 away
    Random 4 teams to play 3 times
    Simulate 1 at home
    Simulate 1 away
    Random either simulate at home or away

```

2.7.6 External Libraries Used

random – generate random values to be added on to the teams score

2.8 Genetic Algorithm

2.8.1 Initialization

In this stage, a function is created to generate four random values to act as the coefficients. The range in which we generated our coefficients is seeded, meaning we choose it from a particular area where the solution is most likely instead of just any number. This range is from 0.75 to 1.25 inclusive. This range is chosen because the lowest score and the highest score these weights produce results in the team either losing or winning all of their games in a season. For example, the lowest score a team can have is when the first three factors have the weight of 0.75 and the TOV% factor have the weight of 1.25. This results in the team winning zero games the entire season. On the other hand, the highest score a team can have is when 1.25 is the weight of the first three factors and 0.75 as the weight of TOV%.

This will result in any team winning all 82 games of their entire season. With these two extremes, we can conclude that the optimal result for the weights to predict how many games a team wins will lie in this range making it suitable to use this range. Seeding the range can decrease the time of convergence to a result.

The population size will be 12. This is quite small but it is a tradeoff between program run time and how accurate each population is. However, because our initialization is seeded to a range of 0.5 a small population wouldn't cause a big issue in accuracy so the advantages gained in decreased run time outweighs the disadvantage to accuracy issues.

2.8.2 Fitness Function

The fitness assessment criteria used is the record for the previous year. The fitness function will call the function to determine a record and compare it to the actual record. The function returns an absolute value of the observed number of wins and the expected number of wins. This essentially is a measure of how far away from the correct solution the individuals are.

2.8.3 Evolution – Selection

The selection process is simple. We take the top performing one third of the individuals and randomly select one third of the worse performing ones. This is to promote genetic diversity. Two thirds of the population are selected to be parents meaning there will be a low crossover rate as only one third (four) individuals are bred, this is because a low crossover rate will keep more of the strong performing individual's gene.

2.8.4 Evolution – Crossover

With two thirds of the population selected previously, we now have to breed the last one third of the population so that it returns to the original size. Different crossover methods have benefits and downsides to them. A single point crossover means that the off-springs will not be that genetically diverse from their parents, keeping much of the traits of the parents. A uniform crossover will create off-springs that are very different from their parents.

Considering this, the breeding algorithm I chose to use is the multipoint crossover method. This is a mix of single-point and uniform crossover. I do not want the off-springs to be too different from the parents (as the seed range is small so there is accuracy already) but I also want it to diversify enough (as population size is small) so that it can actually converge even if initializing population is poor. A multipoint crossover gets the best of both worlds so it's a suitable crossover technique to use.

2.8.5 Evolution – Mutations

The chance of mutations is chosen to be 5%. A random individual in the population is chosen and, in the individual, a random gene is chosen to be mutated. In mutation, we replace the gene with a randomly generated number in our seed range.

2.8.6 Pseudocode for genetic algorithm

```

Import libraries
Import functions

def individual(n,range):
    list ← n numbers in range
    return x

def population (count,n,range):
    loop in range count
        list ← individual (n,range)

    return population

def fitnessFunction():
    fitnessScore ← absolute value of difference between expected and observed
    return fitnessScore

def evolution(target, pop):
    x ← fitness of passed in population
    if x:
        return pop
    while true
        append the best scoring individuals to a new list

    random select left over individuals

    while len(newPop) != len(oldPop):
        multipoint crossover breeding
        random father
        random mother
        random first point
        random second point
        everything in mother between first and second point is now in father
        everything in father between first and second point is now in mother

```

append to newPop as off-spings

mutations

5% chance

Random individual

Random gene

Random new value

Gene replaced with new value

2.8.7 External Libraries Used

Random – generate random values for: individual initialization,

- Generate random parameters for selecting lesser performers, mutations, parents, crossover positions

2.9 Simulating next season

2.9.1 Simulating next season

Once the genetic algorithm has ran and determined the coefficients for different years, it is averaged out to create one set of four coefficients. These coefficients will be used to simulate the games for next season by calling the determine record function. Instead of coefficients from the genetic algorithm, the predicted coefficients are passed in. A predicted record is generated off of this.

3 Technical Solution

3.1 Web Scrape and Data Storage (.py)

```

1  import pandas as pd
2  # 'as pd' so can use pd.command instead of pandas.command
3  import sqlalchemy
4
5  def sqlconnection():
6      # This function creates the connection engine between python and the database in mysql
7      engine =
8      sqlalchemy.create_engine('mysql+pymysql://root:qazwsxedcrfv@localhost:3306/stats')
9      # dialect+driver://username:password@host:port/database
10     return engine
11
12 def savetosql(df, num):
13     # This function puts the called dataframe into the database using the connection
14     engine = sqlconnection()
15     df.to_sql(name = '2k' + num , con = engine, index = False, if_exists = 'replace')
16     # name = table name, con = connection if_exists (if the table already exists, replace
17     # everything in it)
18
19 def editSQL(num):
20     # Removes all the apostrophes in every name in every table
21     engine = sqlconnection()
22     connection = engine.connect()
23     query = "UPDATE 2k" + str(num) + " SET Player = REPLACE (Player, '\\\\' , '')"
24     resultProxy = connection.execute(query)
25
26     if num == 13 or num == 14:
27         query = "UPDATE 2k" + str(num) + " SET Tm = REPLACE (Tm, 'CHA', 'CHO')"
28         resultProxy = connection.execute(query)
29         # There was a team name change
30         # Changes it to match the new name
31
32 y = int(input('Enter the last two digits of the year '))
33 y2 = y - 5
34
35 for i in range(y2,y+1):
36     yearNum = str(i)
37     stats, = pd.read_html ('https://www.basketball-
38     reference.com/leagues/NBA_20'+yearNum+'_totals.html', header = None)

```

```

39     # The ',' is used to unpack the tuple of values
40     # Right Hand Side returns a tuple of values that can be unpacked into the left hand side
41     using ','
42     # Read_html is a built in function in the Pandas library to scrape tabular data from html
43     pages
44
45     stats.drop('Rk', axis = 1, inplace = True)
46     # This line removes the column 'Rk'
47     # axis = 1 is a way to tell pandas that it is a column
48     # by default drop() function only displays the changed dataframe and not save it. inplace
49     = True replaces the dataframe
50
51     stats = stats[stats['Player'] != 'Player']
52     # This gets rid of all the rows which are headings in the middle of the table
53     # From 'column_name' select all the rows that value != values (player)
54
55     stats.rename(columns={'FG%':'FGp', '3P%':'3Pp', '2P%':'2Pp', 'eFG%':'eFGp', 'FT%':'FTp'},
56                 inplace = True )
57     # Headings violate mysql rules with special characters. Changes headings to fit those
58     rules.
59
60     stats = stats.apply(pd.to_numeric, errors = 'ignore')
61     # Changes all datatypes from string to double instead of numbers in text
62     # errors = ignore, keeps all the columns with just text as string
63
64     savetosql(stats, yearNum)
65
66     editSQL(i)
67
68     print('DONE')

```

3.2 Querying from Database (.py)

```

1     import sqlalchemy
2
3     engine = sqlalchemy.create_engine('mysql+pymysql://root:qazwsxedcrfv@localhost:3306/stats')
4     # Engine for connection
5     connection = engine.connect()
6
7     def roster(year, team):
8         query = 'SELECT Player FROM 2k'+ str(year) + ' WHERE Tm = '+ '\'' + team + '\''
9         resultProxy = connection.execute(query)
10         ''' This executes the query statement, 'query', from dB in connection and stores it in
11             resultProxy '''

```

```

12     # resultProxy is the object returned by .execute() method
13     resultSet = resultProxy.fetchall()
14     # Actual data requested when using fetch method on resultProxy
15
16     x = []
17     for row in resultSet:
18         x.append (row[0])
19         ''' fetchall() return all rows and all fields. Need to iterate over the rows to access
20             the fields and get the data '''
21     return x
22
23 def stats(year, team):
24     players = roster(year, team)
25     # players is a list with players from predict year
26     playerStats = []
27     for i in players:
28         query = 'SELECT * FROM 2k'+ str(year - 1) + ' WHERE Player = ' + '\''+ i + '\'' and Tm =
29             '\TOT\ '
30         ''' Players who played for multiple teams will have multiple records in the table. This
31             is to Query for just the reccord of their totals as oppose to a specific team '''
32         resultProxy = connection.execute(query)
33         resultSet = resultProxy.fetchall()
34
35         if not resultSet:
36             query = 'SELECT * FROM 2k'+ str(year - 1) + ' WHERE Player = ' + '\''+ i + '\''
37             resultProxy = connection.execute(query)
38             resultSet = resultProxy.fetchall()
39
40         if resultSet:
41             for row in resultSet:
42                 playerStats.append( {'eFG%': row [16], 'FGA': row[8], 'FTA': row[18], 'TOV' :
43 row[26], 'FT': row[17], 'ORB': row[20]})
44
45     return playerStats
46
47 def get(year, team):
48     x = stats(year, team)
49
50     return x

```

3.3 Generating Team Score (.py)

```

1  from get_data import *
2
3  def teamScore(year, team, coefficients):
4      co = coefficients
5      eFGp = 0
6      TOV = 0
7      FGA = 0
8      FTA = 0
9      ORB = 0
10     FT = 0
11     count = 0
12     x = get(year, team)
13
14     for i in x:
15         if x[count]['eFG%'] != None:
16             eFGp = eFGp + x[count]['eFG%']
17         if x[count]['TOV'] != None:
18             TOV = TOV + x[count]['TOV']
19         if x[count]['FGA'] != None:
20             FGA = FGA + x[count]['FGA']
21         if x[count]['FTA'] != None:
22             FTA = FTA + x[count]['FTA']
23         if x[count]['ORB'] != None:
24             ORB = ORB + x[count]['ORB']
25         if x[count]['FT'] != None:
26             FT = FT + x[count]['FT']
27         # Gets the sum of all players for all these stats
28         count = count + 1
29
30     #Four factors
31     eFGp = eFGp / count
32     TOVp = TOV / (FGA + 0.44 * FTA + TOV)
33     ORB = ORB / 82 / 100
34     FT = FT / FGA
35
36     score = co[0] * eFGp + co[1] * ORB + co[2] * FT - co[3] * TOVp
37     return score

```

3.4 Determining Record (.py)

```

1  from combine_data import *
2  import random
3
4  def teams(conference):
5      if conference == 'west':
6          file = open('western.txt','r')
7      elif conference == 'east':
8          file = open('eastern.txt','r')
9
10     count = 1
11     x=[]
12     y=[]
13     z=[]
14     team = []
15     for line in file:
16         line = line.rstrip()
17         if count == 1 or count == 2 or count == 3 or count == 4 or count == 5:
18             x.append(line)
19             count += 1
20             # These teams are in one division
21
22         elif count == 6 or count == 7 or count == 8 or count == 9 or count == 10:
23             y.append(line)
24             count += 1
25             # These teams are in another division
26         else:
27             z.append(line)
28             count += 1
29             # These teams are in the final division for this conference
30
31     team.append(x)
32     team.append(y)
33     team.append(z)
34     file.close()
35     return team
36     #Returns a 2d list, each list in this list are the 5 teams that are in the same division
37
38 def simulateMainAtHome(main,t1, t2, year):
39     '''This function simulates the chance that team 1 wins against team 2 or vice versa in ONE
40     Head to head game. It accounts for some randomness and home court advantage. This one

```



```

41     game is simulated many times and the team that wins majority of the time will be
42     considered to be the winner of this game '''
43     home = t1
44     away = t2
45     x = [1,1,1,1]
46     homeInit = main
47
48     awayInit = teamScore(year, away, x)
49     #Get home and away team score first
50
51     homeWin = 0
52     awayWin = 0
53
54     for i in range(0,1000):
55
56         randomHome = random.randint(0,25) / 100
57         # Home team has advantage
58         randomAway = random.randint(0,15) / 100
59
60         homeScore = homeInit + randomHome
61         awayScore = awayInit + randomAway
62
63
64         if homeScore >= awayScore:
65             homeWin = homeWin + 1
66         else:
67             awayWin = awayWin + 1
68
69     if homeWin > awayWin:
70         print(home, 'won', homeWin / 10 , '% of the time')
71         return 'won'
72     else:
73         print(away, 'won' , awayWin / 10 , '% of the time')
74         return 'lost'
75
76     ''' One function for home and one for away because our coefficients will only modify the main
77         team we are testing. The main team will be sometimes home and sometimes away so we need
78         the coefficients parameter to be passed to the homeInit or the awayInit when calling
79         teamScore() '''
80
81 def simulateMainAtAway(t1, t2, main, year):
82     home = t1
83     away = t2
84     x = [1,1,1,1]

```

```

85     homeInit = teamScore(year, home, x)
86
87     awayInit = main
88     #Get home and away team score first
89
90     homeWin = 0
91     awayWin = 0
92
93
94     for i in range(0,1000):
95
96         randomHome = random.randint(0,25) / 100
97         # Home team has advantage
98         randomAway = random.randint(0,15) / 100
99
100        homeScore = homeInit + randomHome
101        awayScore = awayInit + randomAway
102
103
104        if homeScore >= awayScore:
105            homeWin = homeWin + 1
106        else:
107            awayWin = awayWin + 1
108
109    if homewin > awayWin:
110        print(home, 'won', homewin / 10 ,'% of the time')
111        return 'won'
112    else:
113        print(away, 'won' , awayWin / 10 ,'% of the time')
114        return 'lost'
115
116    def simulateWest(team, division, index, year, coefficients):
117        ''' division and index is the position of the team in the list of teams. Passed into the
118            function to use as conditions so the team doesn't play themselves'''
119        mainScore = teamScore(year, team, coefficients)
120
121        west = teams('west')
122        east = teams('east')
123        wins = 0
124
125        if division == 0:
126            d1 = 1
127            d2 = 2
128        elif division == 1:

```

```

129     d1 = 0
130     d2 = 2
131     elif division == 2:
132         d1 = 0
133         d2 = 1
134     # used to know the other two divisions when randoming the teams later
135
136     for i in range(2):
137         for j in range(0,5):
138             if j != index:
139                 x = simulateMainAtHome(mainScore, team, west[division][j],year)
140                 if x == 'won':
141                     wins += 1
142     # Plays every team in the same divion twice at HOME
143
144     for i in range(2):
145         for j in range(0,5):
146             if j != index:
147                 x = simulateMainAtAway(west[division][j],team, mainScore, year)
148                 if x == 'lost':
149                     wins += 1
150     # Plays every team in the same divion twice at AWAY
151
152
153     for i in range(0,3):
154         for j in range(0,5):
155             x = simulateMainAtHome(mainScore, team, east[i][j],year)
156             if x == 'won':
157                 wins += 1
158     # Plays every team in the oppsing conference once at HOME
159
160     for i in range(0,3):
161         for j in range(0,5):
162             x = simulateMainAtAway(east[i][j],team, mainScore, year)
163             if x == 'lost':
164                 wins += 1
165     # Plays every team in the oppsing conference once at AWAY
166
167     # Random 6 teams in same conference but differet division to play 2 home 2 away
168     rd1 = [0,1,2,3,4]
169     rd2 = [0,1,2,3,4]
170
171     for i in range(6):
172         if len(rd1) != 0 and len(rd2) != 0:

```

```

173     n = random.choice([d1,d2])
174     # If they are both not empty, randomly choose one of the conferences
175     if n == d1:
176         y = random.choice(rd1)
177         # Randomly choose a team in this conference
178         for i in range(2):
179             x = simulateMainAtHome(mainScore, team, west[d1][y],year)
180             if x == 'won':
181                 wins += 1
182                 # Play 2 games at HOME
183
184         for i in range(2):
185             x = simulateMainAtAway(west[d1][y],team, mainScore, year)
186             if x == 'lost':
187                 wins +=1
188                 # Play 2 games AWAY
189         rd1.remove(y)
190
191     elif n == d2:
192         y = random.choice(rd2)
193         for i in range(2):
194             x = simulateMainAtHome(mainScore, team, west[d2][y],year)
195             if x == 'won':
196                 wins += 1
197
198         for i in range(2):
199             x = simulateMainAtAway(west[d2][y], team, mainScore, year)
200             if x == 'lost':
201                 wins +=1
202         rd2.remove(y)
203
204
205     elif len(rd1) == 0:
206         # If one of the list is empty (because it was picked) i.e. teams have all been
207         # played
208         y = random.choice(rd2)
209         # Choose from the other list
210         for i in range(2):
211             x = simulateMainAtHome(mainScore, team, west[d2][y],year)
212             if x == 'won':
213                 wins += 1 # 2 at HOME
214
215         for i in range(2):
216             x = simulateMainAtAway(west[d2][y], team, mainScore, year)

```

```

217         if x == 'lost':
218             wins +=1 # 2 AWAY
219         rd2.remove(y)
220
221     elif len(rd2) == 0:
222         y = random.choice(rd1)
223         for i in range(2):
224             x = simulateMainAtHome(mainScore, team, west[d1][y],year)
225             if x == 'won':
226                 wins += 1
227
228
229         for i in range(2):
230             x = simulateMainAtAway(west[d1][y], team, mainScore, year)
231             if x == 'lost':
232                 wins +=1
233         rd1.remove(y)
234
235
236     # next have to random 2 teams to play 2 home games and 1 away game
237     for i in range(2):
238         if len(rd1) == 0:
239             # if conference 1 is empty, random 2 teams from conference 2
240             g = d2
241             f = random.randint(0, len(rd2)-1)
242         elif len(rd2) == 0:
243             g = d1
244             f = random.randint(0, len(rd1)-1)
245         else:
246             g = random.choice([d1,d2])
247             # both not empty, random conference then random team
248             if g == d1:
249                 f= random.randint(0, len(rd1) - 1)
250             elif g == d2:
251                 f = random.randint(0, len(rd2) - 1)
252
253         for i in range(2):
254             x = simulateMainAtHome(mainScore, team, west[g][f], year)
255             if x == 'won':
256                 wins += 1 # 2 at HOME
257
258         x = simulateMainAtAway(west[g][f], team, mainScore, year)
259         if x == 'lost':
260             wins += 1 # 1 AWAY

```

```

261
262     if g == d1:
263         rd1.remove(rd1[f])
264     else:
265         rd2.remove(rd2[f])
266
267
268     # Last 2 teams, 1 away 1 home
269     while len(rd1) != 0:
270         # Take all teams from this conference. At this point, there is either 2 left, or 1 or
271         none
272         for i in range(len(rd1)):
273             x = simulateMainAtHome(mainScore, team, west[d1][i],year)
274             if x == 'won':
275                 wins += 1 # 1 at HOME
276
277             for i in range(2):
278                 x = simulateMainAtAway(west[d1][i], team, mainScore,year)
279                 if x == 'lost':
280                     wins +=1 # 1 AWAY
281         rd1 = [] # Remove all from list
282
283     while len(rd2) != 0:
284         # Same as above
285         for i in range(len(rd2)):
286             x = simulateMainAtHome(mainScore, team, west[d2][i], year)
287             if x == 'won':
288                 wins += 1
289
290             for i in range(2):
291                 x = simulateMainAtAway(west[d2][i], team, mainScore, year)
292                 if x == 'lost':
293                     wins +=1
294         rd2 = []
295
296     print(wins, 'wins')
297     return(wins)
298
299
300 def simulateEast(team, division, index, year, coefficients):
301     ''' This and simulateWest are essentially the same function but one is for a team in the
302     western conference and the other in the eastern conference. Two functions are needed
303     because of the difference in the teams and the number of games they play per team due
304     to them being different conferences '''

```

```
305
306     mainScore = teamScore(year, team, coefficients)
307
308     west = teams('west')
309     east = teams('east')
310     wins = 0
311
312     if division == 0:
313         d1 = 1
314         d2 = 2
315     elif division == 1:
316         d1 = 0
317         d2 = 2
318     elif division == 2:
319         d1 = 0
320         d2 = 1
321
322     for i in range(2):
323         for j in range(0,5):
324             if j != index:
325                 x = simulateMainAtHome(mainScore, team, east[division][j],year)
326                 if x == 'won':
327                     wins += 1
328
329     for i in range(2):
330         for j in range(0,5):
331             if j != index:
332                 x = simulateMainAtAway(east[division][j],team, mainScore, year)
333                 if x == 'lost':
334                     wins += 1
335
336
337     for i in range(0,3):
338         for j in range(0,5):
339             x = simulateMainAtHome(mainScore, team, west[i][j],year)
340             if x == 'won':
341                 wins += 1
342
343     for i in range(0,3):
344         for j in range(0,5):
345             x = simulateMainAtAway(west[i][j],team, mainScore, year)
346             if x == 'lost':
347                 wins += 1
348
```

```

349     rd1 = [0,1,2,3,4]
350     rd2 = [0,1,2,3,4]
351
352     for i in range(6):
353         if len(rd1) != 0 and len(rd2) != 0:
354
355             n = random.choice([d1,d2])
356             if n == d1:
357                 y = random.choice(rd1)
358                 for i in range(2):
359                     x = simulateMainAtHome(mainScore, team, east[d1][y],year)
360                     if x == 'won':
361                         wins += 1
362
363                 for i in range(2):
364                     x = simulateMainAtAway(east[d1][y],team, mainScore, year)
365                     if x == 'lost':
366                         wins +=1
367                 rd1.remove(y)
368
369             elif n == d2:
370                 y = random.choice(rd2)
371                 for i in range(2):
372                     x = simulateMainAtHome(mainScore, team, east[d2][y],year)
373                     if x == 'won':
374                         wins += 1
375
376                 for i in range(2):
377                     x = simulateMainAtAway(east[d2][y], team, mainScore, year)
378                     if x == 'lost':
379                         wins +=1
380                 rd2.remove(y)
381
382         elif len(rd1) == 0:
383             y = random.choice(rd2)
384             for i in range(2):
385                 x = simulateMainAtHome(mainScore, team, east[d2][y],year)
386                 if x == 'won':
387                     wins += 1
388
389         for i in range(2):
390             x = simulateMainAtAway(east[d2][y], team, mainScore, year)
391             if x == 'lost':
392                 wins +=1

```



```

393         rd2.remove(y)
394
395     elif len(rd2) == 0:
396         y = random.choice(rd1)
397         for i in range(2):
398             x = simulateMainAtHome(mainScore, team, east[d1][y],year)
399             if x == 'won':
400                 wins += 1
401
402         for i in range(2):
403             x = simulateMainAtAway(east[d1][y], team, mainScore, year)
404             if x == 'lost':
405                 wins +=1
406         rd1.remove(y)
407
408
409     for i in range(2):
410         if len(rd1) == 0:
411             g = d2
412             f = random.randint(0, len(rd2)-1)
413         elif len(rd2) == 0:
414             g = d1
415             f = random.randint(0,len(rd1)-1)
416         else:
417             g = random.choice([d1,d2])
418             if g == d1:
419                 f= random.randint(0, len(rd1) - 1)
420             elif g == d2:
421                 f = random.randint(0, len(rd2) - 1)
422
423         for i in range(2):
424             x = simulateMainAtHome(mainScore, team, east[g][f], year)
425             if x == 'won':
426                 wins += 1
427
428         x = simulateMainAtAway(east[g][f], team, mainScore,year)
429         if x == 'lost':
430             wins += 1
431
432         if g == d1:
433             rd1.remove(rd1[f])
434         else:
435             rd2.remove(rd2[f])
436

```

```

437
438     while len(rd1) != 0:
439         for i in range(len(rd1)):
440             x = simulateMainAtHome(mainScore, team, east[d1][i],year)
441             if x == 'won':
442                 wins += 1
443
444         for i in range(2):
445             x = simulateMainAtAway(east[d1][i], team, mainScore,year)
446             if x == 'lost':
447                 wins +=1
448         rd1 = []
449
450     while len(rd2) != 0:
451         for i in range(len(rd2)):
452             x = simulateMainAtHome(mainScore, team, east[d2][i], year)
453             if x == 'won':
454                 wins += 1
455
456         for i in range(2):
457             x = simulateMainAtAway(east[d2][i], team, mainScore, year)
458             if x == 'lost':
459                 wins +=1
460         rd2 = []
461
462     print(wins, 'wins')
463     return(wins)

```

3.5 Genetic Algorithm (.py)

```

1  import random
2  from determine_record import *
3
4  west = teams('west')
5  east = teams('east')
6
7  def individual(N,min,max):
8      x = []
9      for i in range(N):
10         x.append( random.randint(min,max) / 100 )
11     return x
12
13     '''randomly generates a number between min, max and appends them into a list. does it N
14     times '''

```

```

15     # Creates an individual in a population
16
17 def population(count,N,min,max):
18     x = []
19     for i in range(count):
20         x.append(individual(N,min,max))
21     return x
22
23     # Creates the population using the individual function
24
25 def fitnessFunction(conference, numbers, target, cNum, tNum, year):
26     # numbers is the list of coefficients
27     print(numbers)
28     if conference == 'east':
29         observed = simulateEast(east[cNum][tNum], cNum, tNum, year, numbers)
30     elif conference == 'west':
31         observed = simulateWest(west[cNum][tNum], cNum , tNum, year, numbers)
32
33     score = abs(target - observed)
34     print(score, 'fitness score')
35     return score
36
37     # Takes in a LIST and a target
38     # Generates a score based on the difference between the sum of the list and the target
39
40
41 def evolution(prevPop, target, conf, cNum, tNum, year):
42     newPop = []
43     targetLength = len(prevPop)
44     noHP = (1/3)
45     noLP = (1/3)
46
47     while len(newPop) != int(targetLength * noHP):
48         min = 100
49         for i in range(len(prevPop)):
50             x = fitnessFunction(conf, prevPop[i], target, cNum, tNum, year)
51             if x <= 0:
52                 return prevPop[i]
53             ''' If we find the solution within the predefined population we return it
54                 straight away there is no need for evolution '''
55         else:
56             if x <= min:
57                 k = i
58                 min = x

```

```

59
60     newPop.append(prevPop[k])
61     prevPop.pop(k)
62     ''' loop through everything if the individual is in our desired range of the target, we
63         return it straight away otherwise , take the SMALLEST fitnessfunction score
64         individuals (which would be 1D list) (smaller fitnessFunction scores are BETTER
65         performers) (one of the arrays in the 2D array) put in new pop pop it off prevPop '''
66
67     # Randomly select lesser performers
68     for i in range(int(targetLength * noLP)):
69         ran = random.randint(0, len(prevPop)-1)
70         newPop.append(prevPop[ran])
71         prevPop.pop(ran)
72
73     # newPop are parents of next generation
74     # 2/3 is selected to be parents
75     # 1/3 is breeded
76
77     # Breeding
78     parentsLength = len(newPop)
79
80     children = []
81     child1 = []
82     child2 =[]
83     childrenTargetLength = targetLength - parentsLength
84
85     while len(children) != childrenTargetLength:
86         fatherNo = random.randint(0, parentsLength - 1)
87         motherNo = random.randint(0, parentsLength - 1)
88         ''' Randomly chooses mother and father from the group of parents With some high
89             performers and some lower performers '''
90
91         if fatherNo != motherNo:
92             # Ensures father and mother are not the same
93             father = newPop[fatherNo]
94             mother = newPop[motherNo]
95
96             firstPoint = random.randint(0, len(father)-1)
97             secondPoint = random.randint(0, len(father) - 1)
98             # Multipoint crossover: Chooses two points in an individual as crossover points
99
100             if firstPoint != secondPoint and secondPoint > firstPoint:
101                 # First point != second point (otherwise no crossover)

```

```

102         # Second point > first point (prevents weird stuff from happening because of
103         # how child1 and child2 are defined)
104         child1 = father[:firstPoint] + mother[firstPoint: secondPoint] +
105             father[secondPoint:]
106         child2 = mother[:firstPoint] + father[firstPoint: secondPoint] +
107             mother[secondPoint:]
108
109         children.append(child1)
110         children.append(child2)
111
112     # Mutations
113     chanceToMutate = random.randint(1,20)
114     # 5% chance
115     if chanceToMutate == 1:
116         individualToMutate = random.randint(0,len(children)-1)
117         geneToMutate = random.randint(0, len(children[individualToMutate])-1)
118         # Randomly chooses which individual and which gene in that individual to mutate
119
120         children[individualToMutate][geneToMutate] = random.randint(75, 125) / 100
121         # randint is any number for generating individual
122
123         print(individualToMutate, geneToMutate, 'mutated')
124
125     newPop.extend(children)
126
127     print(newPop)
128     return newPop
129
130
131 def runGA(target, conference, cNum, tNum, year ):
132     size = 12
133     # not 4
134     test = population(size ,4,75,125)
135     repeat = True
136     count = 0
137
138     while repeat == True:
139         count = count + 1
140         test = evolution(test, target, conference, cNum, tNum, year)
141         solution = test
142         print(count, 'evolution(s)')
143         if len(solution) != size:
144             return solution
145         repeat = False

```

```

146         break
147         ''' If the returning value is a solution, the len of the 1 d list will be four this
148             is not 'size' so we know that what was returned was the solution
149
150             If returning value is the new population for next evolution, the length will be 6
151             and the same as size so we know this is the pop for next evolution so these
152             functions are not executed '''

```

3.6 Usage Code (.py)

```

1  from determine_record import *
2  from genetic_algorithm import runGA
3  import json
4  import time
5  start_time = time.time()
6
7  west = teams('west')
8  east = teams('east')
9
10 def teamIndex(conf, team):
11     # gets the index of the team in the list. Function for determining record takes position
12     as parameter
13     for i in range(len(conf)):
14         try:
15             c = i
16             index = conf[i].index(team)
17             return (c, index)
18             break
19         except ValueError:
20             pass
21
22 def allSs(record, co, num1, num2, year):
23     x = []
24     sol = []
25     sol2 = []
26     one = 0
27     two = 0
28     three = 0
29     four = 0
30
31     sol = runGA(record,co,num1, num2, year)
32     x.append(sol)
33
34     for i in range(1,3):

```

```

35     year = year - 1
36     sol = runGA(record,co,num1, num2, year)
37     x.append(sol)
38
39     for i in range(0,len(x)):
40         one += x[i][0]
41         two += x[i][1]
42         three += x[i][2]
43         four += x[i][3]
44
45     sol2.append( round(one / len(x), 2) )
46     sol2.append( round(two / len(x), 2) )
47     sol2.append( round(three / len(x), 2) )
48     sol2.append( round(four / len(x), 2) )
49     print(sol2)
50
51     return sol2
52
53 def getJson():
54     with open('records.json','r') as file:
55         x = json.load(file)
56
57     return x
58 # Teams' record stored in JSON file
59
60 allRecords = getJson()
61
62 print('')
63 co = input('The team\'s conference ')
64 t = input('Input the team initials ')
65 year = int(input('Input the last two numbers of the year you want to find the record for. e.g.
66 input 18 for 2018 '))
67
68 year = year - 1
69
70 if co == 'west':
71     c = west
72     nums = teamIndex(c, t)
73 elif co == 'east':
74     c = east
75     nums = teamIndex(c,t)
76
77 if co == 'west':
78     n = 0

```

```

79 elif co == 'east':
80     n = 1
81
82     record = allRecords["20" + str(year)][n][t]
83
84     #sol = allSs(record, co, nums[0], nums[1], year)
85     sol = runGA(record, co, nums[0], nums[1], year)
86
87     print('')
88     print(sol)
89     print('These are the solution coefficients. Using this to predict next season\'s record.')
90
91 if co == 'west':
92     w = 0
93     for i in range(5):
94         w = w + simulateWest(west[nums[0]][nums[1]], nums[0], nums[1], year + 1, sol )
95     w = w // 5
96     # because of the randomness, we can take an average
97 elif co == 'east':
98     w = 0
99     for i in range(5):
100         w = w + simulateEast(east[nums[0]][nums[1]], nums[0], nums[1], year + 1, sol )
101     w = w // 5
102
103 print(t, 'is predicted to win', w, 'games in 20' + str(year + 1), 'Their record:', w , '-' ,
104      82 - w)
105
106 print ("My program took", time.time() - start_time, "to run")

```

3.7 Teams Text File (.txt)

3.7.1 Western.txt

```

1 DEN
2 OKC
3 POR
4 UTA
5 MIN
6 GSW
7 LAC
8 SAC
9 LAL
10 PHO

```



```
11 HOU
12 SAS
13 DAL
14 NOP
15 MEM
```

3.7.2 Eastern.txt

```
1 TOR
2 PHI
3 BOS
4 BRK
5 NYK
6 MIL
7 IND
8 DET
9 CHI
10 CLE
11 CHO
12 ORL
13 MIA
14 WAS
15 ATL
```

3.8 Teams' Record (.json)

```
1 {
2   "2018" : [{
3     "HOU": 65,
4     "GSW": 58,
5     "POR": 49,
6     "OKC": 48,
7     "UTA": 48,
8     "NOP": 48,
9     "SAS": 47,
10    "MIN": 47,
11    "DEN": 46,
12    "LAC": 42,
13    "LAL": 35,
14    "SAC": 27,
15    "DAL": 24,
16    "MEM": 22,
17    "PHO": 21
```

```
18 }, {
19 "TOR": 59,
20 "BOS": 55,
21 "PHI": 52,
22 "CLE": 50,
23 "IND": 48,
24 "MIA": 44,
25 "MIL": 44,
26 "WAS": 43,
27 "DET": 39,
28 "CHO": 36,
29 "NYK": 29,
30 "BRK": 28,
31 "CHI": 27,
32 "ORL": 25,
33 "ATL": 24
34 }],
35 "2017" : [{
36 "HOU": 55,
37 "GSW": 67,
38 "POR": 41,
39 "OKC": 47,
40 "UTA": 51,
41 "NOP": 38,
42 "SAS": 61,
43 "MIN": 31,
44 "DEN": 40,
45 "LAC": 51,
46 "LAL": 26,
47 "SAC": 32,
48 "DAL": 33,
49 "MEM": 43,
50 "PHO": 24
51 }, {
52 "TOR": 51,
53 "BOS": 53,
54 "PHI": 28,
55 "CLE": 51,
56 "IND": 42,
57 "MIA": 41,
58 "MIL": 42,
59 "WAS": 49,
60 "DET": 37,
61 "CHO": 36,
```

```
62 "NYK": 31,
63 "BRK": 20,
64 "CHI": 41,
65 "ORL": 29,
66 "ATL": 43
67 }],
68 "2016" : [{
69 "HOU": 41,
70 "GSW": 73,
71 "POR": 44,
72 "OKC": 55,
73 "UTA": 40,
74 "NOP": 30,
75 "SAS": 67,
76 "MIN": 29,
77 "DEN": 33,
78 "LAC": 53,
79 "LAL": 17,
80 "SAC": 33,
81 "DAL": 42,
82 "MEM": 42,
83 "PHO": 23
84 }, {
85 "TOR": 56,
86 "BOS": 48,
87 "PHI": 10,
88 "CLE": 57,
89 "IND": 45,
90 "MIA": 48,
91 "MIL": 33,
92 "WAS": 41,
93 "DET": 44,
94 "CHO": 48,
95 "NYK": 32,
96 "BRK": 21,
97 "CHI": 42,
98 "ORL": 35,
99 "ATL": 48
100 }],
101 "2015" : [{
102 "HOU": 56,
103 "GSW": 67,
104 "POR": 51,
105 "OKC": 45,
```

```
106 "UTA": 38,
107 "NOP": 45,
108 "SAS": 55,
109 "MIN": 16,
110 "DEN": 30,
111 "LAC": 56,
112 "LAL": 21,
113 "SAC": 29,
114 "DAL": 50,
115 "MEM": 55,
116 "PHO": 39
117 }, {
118 "TOR": 49,
119 "BOS": 40,
120 "PHI": 18,
121 "CLE": 53,
122 "IND": 38,
123 "MIA": 37,
124 "MIL": 41,
125 "WAS": 46,
126 "DET": 32,
127 "CHO": 33,
128 "NYK": 17,
129 "BRK": 38,
130 "CHI": 50,
131 "ORL": 25,
132 "ATL": 60
133 }],
134 "2014": [{
135 "HOU": 54,
136 "GSW": 51,
137 "POR": 54,
138 "OKC": 59,
139 "UTA": 25,
140 "NOP": 34,
141 "SAS": 62,
142 "MIN": 40,
143 "DEN": 36,
144 "LAC": 57,
145 "LAL": 27,
146 "SAC": 28,
147 "DAL": 49,
148 "MEM": 50,
149 "PHO": 48
```

```
150 }, {  
151 "TOR": 48,  
152 "BOS": 25,  
153 "PHI": 19,  
154 "CLE": 33,  
155 "IND": 56,  
156 "MIA": 54,  
157 "MIL": 15,  
158 "WAS": 44,  
159 "DET": 29,  
160 "CHO": 43,  
161 "NYK": 37,  
162 "BRK": 44,  
163 "CHI": 48,  
164 "ORL": 23,  
165 "ATL": 38  
166 }]]}
```

4 Testing

In this testing section, I will be testing the functionality of each individual module of this program. I will mainly be using Golden State Warriors (GSW) as the reference team to be tested on. This is so that I don't need to carry out tests for all 30 teams as the principles of these tests are the same for every team.

Initialization test:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sakila |
| stats |
| sys |
| world |
+-----+
7 rows in set (0.01 sec)

mysql> USE stats;
Database changed
mysql>
mysql> SHOW TABLES;
Empty set (0.00 sec)

mysql>
```

Figure 17: Empty Database

Figure 17 shows the database 'stats' before the Python script is run. As shown, the database is empty.

After the script is run, what should be seen is:

- Store five tables of increasing year up until (not including) the predicting year.
- Remove all instances of apostrophe in players' names
- All datatypes should be changed to be the correct ones
- All heading names adjusted to fit SQL rules
- Adjust the team name
- Not useful rows not stored

See 2.2.2

4.1 Retrieving statistical data + storage

4.1.1 Retrieval results table

Test	Stage	Description	Type	Expected Result	Result	Reference Screenshot
1	Store season stats	Store season stats using year 2018 as input	Normal data	Stats from 2013 – 2018 (inclusive) should be stored	✓	Test 1 - Figure 18 Result 1 - Figure 19
2	Store season stats	Store season stats using 2019 as input	Boundary Data	Stats from 2014 – 2019 (inclusive) should be stored	✓	Test 2 - Figure 20 Result 2 - Figure 21
3	Store season stats	Store season stats using 2020 as input	Erroneous data	Error message	✓	Test 3 - Figure 22 Result 3 - Figure 23
4	Remove apostrophe in names	Automatically remove apostrophe in names	Normal data	Apostrophes are removed from names	✓	Test/Result 4 - Figure 24
5	Adjust Datatypes	Change the data types of stored values to the correct ones	Normal data	All headings with numbers are to be changed to double or integers and strings stay as text	✓	Test/Result 5 - Figure 25
6	Adjust heading names	Remove special characters from headings. So that it works with SQL	Normal data	Special characters replaced with letters	✓	Test/Result 6 - Figure 25

7	Adjust team name	There was a team name change. Change it so it fits will all other data	Normal data	Change “CHA” to “CHO” for all years before 2015	✓	Test/Result 7 - Figure 26
8	Not useful rows not stored	‘Rk’ rows are not useful. Don’t store it.	Normal data	None of these rows are stored	✓	Test/Result 8 - Figure 27

4.1.2 Test Evidence

Test 1:

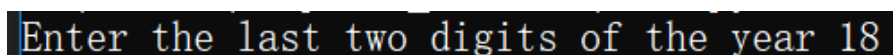


Figure 18: Stat Retrieve - Test 1

Result 1:

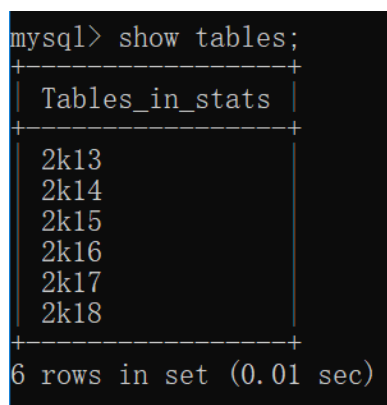


Figure 19: Stat Retrieve - Result 1

Figure 18 shows the test with the year 2018 as input. What should happen is 6 tables should be stored in the database from (year – 5) to year.

Figure 19 shows just this. Years 2013 – 2018 is stored. From this, we can also conclude that our SQLAlchemy connection is working as everything from Python is stored into MySQL.

Test 2:

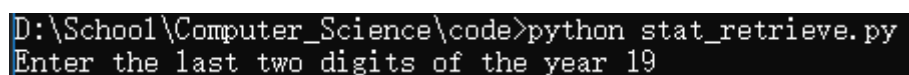


Figure 20: Stat Retrieve - Test 2

Result 2:

```
mysql> show tables;
+-----+
| Tables_in_stats |
+-----+
| 2k14             |
| 2k15             |
| 2k16             |
| 2k17             |
| 2k18             |
| 2k19             |
+-----+
6 rows in set (0.01 sec)
```

Figure 21: Stat Retrieve - Result 2

Figure 20 shows the test with the year 2019 as input. **Figure 21** shows Years 2015 – 2019 is stored.

Test 3:

```
Enter the last two digits of the year 20
```

Figure 22: Stat Retrieve - Test 3**Result3:**

```
D:\School\Computer_Science\code>python stat_retrieve.py
Enter the last two digits of the year 20
C:\Users\Ian Cheng\AppData\Local\Programs\Python\Python37\
s\pymysql\cursors.py:170: Warning: (1366, "Incorrect string
A\\xD7\\xBC\\xCA\\xB1..." for column 'VARIABLE_VALUE' at
result = self._query(query)
Traceback (most recent call last):
  File "stat_retrieve.py", line 36, in <module>
    stats, = pd.read_html('https://www.basketball-refere
20'+yearNum+'_totals.html', header = None)
  File "C:\Users\Ian Cheng\AppData\Local\Programs\Python\
-packages\pandas\io\html.py", line 987, in read_html
    displayed_only=displayed_only)
  File "C:\Users\Ian Cheng\AppData\Local\Programs\Python\
-packages\pandas\io\html.py", line 815, in _parse
    raise_with_traceback(retained)
  File "C:\Users\Ian Cheng\AppData\Local\Programs\Python\
-packages\pandas\compat\__init__.py", line 404, in raise_
    raise exc.with_traceback(traceback)
urllib.error.HTTPError: HTTP Error 404: Not Found
```

Figure 23: Stat Retrieve - Result 3

When using year 2020 as input (**Figure 22**), tables should be year 2015 – 2020. **Figure 23** shows there's an error. This is because the season 2020 has not happened yet. So, there is not data to be web scraped and consequently none to be stored. This is as expected.

Test / Result 4:

```
DAngelo Russell
```

Figure 24: Stat Retrieve - Test/Result 4

Figure 24 shows D'Angelo Russell's name as a value in MySQL. It can be seen that the apostrophe in the name has been removed.

Test/ Result 5 and 6:

```
mysql> DESCRIBE 2k18;
```

Field	Type	Null	Key	Default	Extra
Player	text	YES		NULL	
Pos	text	YES		NULL	
Age	bigint(20)	YES		NULL	
Tm	text	YES		NULL	
G	bigint(20)	YES		NULL	
GS	bigint(20)	YES		NULL	
MP	bigint(20)	YES		NULL	
FG	bigint(20)	YES		NULL	
FGA	bigint(20)	YES		NULL	
FGp	double	YES		NULL	
3P	bigint(20)	YES		NULL	
3PA	bigint(20)	YES		NULL	
3Pp	double	YES		NULL	
2P	bigint(20)	YES		NULL	
2PA	bigint(20)	YES		NULL	
2Pp	double	YES		NULL	
eFGp	double	YES		NULL	
FT	bigint(20)	YES		NULL	
FTA	bigint(20)	YES		NULL	
FTp	double	YES		NULL	
ORB	bigint(20)	YES		NULL	
DRB	bigint(20)	YES		NULL	
TRB	bigint(20)	YES		NULL	
AST	bigint(20)	YES		NULL	
STL	bigint(20)	YES		NULL	
BLK	bigint(20)	YES		NULL	
TOV	bigint(20)	YES		NULL	
PF	bigint(20)	YES		NULL	
PTS	bigint(20)	YES		NULL	

```
29 rows in set (0.00 sec)

mysql>
```

Figure 25: Stat Retrieve - Test/Result 5 and 6

Figure 25 shows the return when we ask for description of table in SQL.

Test 5: In the field column, all the headings do not contain any special characters.

Test 6: In the type column everything is in the correct data type as opposed to the result before changing (See **Figure 13: Data types before modification**).

Test/ Result 7:

```
mysql> SELECT DISTINCT Tm FROM 2k14
-> ORDER BY Tm ASC;
+-----+
| Tm     |
+-----+
| ATL    |
| BOS    |
| BRK    |
| CHI    |
| CHO    |
| CLE    |
| DAL    |
| DEN    |
| DET    |
| GSW    |
| HOU    |
| IND    |
| LAC    |
| LAL    |
| MEM    |
| MIA    |
| MIL    |
| MIN    |
| NOP    |
| NYK    |
| OKC    |
| ORL    |
| PHI    |
| PHO    |
| POR    |
| SAC    |
| SAS    |
| TOR    |
| TOT    |
| UTA    |
| WAS    |
+-----+
31 rows in set (0.01 sec)

mysql>
```

Figure 26: Stat Retrieve - Test/Result 7

Figure 26 shows the result when querying for the team names in the year 2014. Because the team name change happened in 2014, it didn't take effect until 2015. However, from our query from year 2014, the team name stored is correct meaning that we changed it from "CHA" to "CHO".

Test / Result 8:

To test if we have removed all the useless rows in the middle of the table, we can query for the value 'Rk' in the column headed 'Player'. This is the value of rows we do not want. If the query returns some rows, then it means that we were unsuccessful in removing these rows. However, if the query returns nothing then we were successful.

```
mysql> SELECT Player FROM 2k18 WHERE Player = 'Rk' ;
Empty set (0.00 sec)

mysql>
```

Figure 27: Stat Retrieve - Test/Result 8

From **Figure 27**, we can see that we have removed all of the rows we don't want as an empty set is returned when we try to query for them.

4.2 Querying

4.2.1 Querying Results Table

Test	Stage	Description	Type	Expected Result	Actual Result	Reference Screenshot
9	Query Roster	Query roster from year 2017 for GSW	Normal data	A list of names of players on GSW from 2017	✓	Test 9 Result 9 - Figure 28
10	Query Roster	Query roster from year 2018 for GSW	Boundary Data	A list of names of players on GSW from 2018	✓	Test 10 Result 10 - Figure 29
11	Query Roster	Query roster from year 2020 for GSW	Erroneous data	Error message	✓	Test 11 Result 11 - Figure 30
12	Query Stats	Query stats for year 2018 for GSW	Boundary data	List of dictionaries of stats from 2018	✓	Test 12 Result 12 - Figure 31
13	Query Stats	Query stats for year 2018 for 'ABC' (not a team name)	Erroneous data	Error	✓ Empty List	Test 13 Result 13 - Figure 32

4.2.2 Test Evidence

Test 9:

```
1 print( roster(17, 'GSW') )
```

Result 9:

```
['Matt Barnes', 'Ian Clark', 'Stephen Curry', 'Kevin Durant', 'Draymond Green', 'Andre Iguodala', 'Damian Jones', 'Shaun Livingston', 'Kevon Looney', 'James Michael McAdoo', 'Patrick McCaw', 'JaVale McGee', 'Zaza Pachulia', 'Klay Thompson', 'Anderson Varejao', 'Briante Weber', 'David West']
```

Figure 28: Querying - Result 9

Test 10:

```
1 print( roster(18,'GSW') )
```

Result 10:

```
[Jordan Bell, Chris Boucher, Omri Casspi, Quinn Cook, Stephen Curry, Kevin Durant, Draymond Green, Andre Iguodala, Damian Jones, Shaun Livingston, Kevon Looney, Patrick McCaw, JaVale McGee, Zaza Pachulia, Klay Thompson, David West, Nick Young]
```

D:\School\Computer_Science\code>

Figure 29: Querying - Result 10**Test 11:**

```
1 print( roster(20,'GSW') )
```

Result 11:

```
sqlalchemy.exc.ProgrammingError: (pymysql.err.ProgrammingError) (1146, "Table 'stats.2k20' doesn't exist") [SQL: "SELECT Player FROM 2k20 WHERE Tm = 'GSW'"] (Background on this error at: http://sqlalche.me/e/f405)
```

Figure 30: Querying - Result 11

Querying for roster of 2020 returns an error. This is because only years 2014 – 2019 is stored. There is no 2020 table so nothing can be queried.

Test 12:

```
1 print( stats(18,'GSW') )
```

Result 12:

```
[{'eFG%': 0.515, 'FGA': 164, 'FTA': 29, 'TOV': 29, 'FT': 17, 'ORB': 27}, {'eFG%': 0.583, 'FGA': 66, 'FTA': 5, 'TOV': 13, 'FT': 2, 'ORB': 1}, {'eFG%': 0.58, 'FGA': 1443, 'FTA': 362, 'TOV': 239, 'FT': 325, 'ORB': 61}, {'eFG%': 0.594, 'FGA': 1026, 'FTA': 384, 'TOV': 138, 'FT': 336, 'ORB': 39}, {'eFG%': 0.481, 'FGA': 650, 'FTA': 213, 'TOV': 184, 'FT': 151, 'ORB': 98}, {'eFG%': 0.605, 'FGA': 415, 'FTA': 102, 'TOV': 58, 'FT': 72, 'ORB': 51}, {'eFG%': 0.5, 'FGA': 16, 'FTA': 10, 'TOV': 6, 'FT': 3, 'ORB': 9}, {'eFG%': 0.549, 'FGA': 316, 'FTA': 60, 'TOV': 62, 'FT': 42, 'ORB': 28}, {'eFG%': 0.533, 'FGA': 107, 'FTA': 34, 'TOV': 17, 'FT': 21, 'ORB': 44}, {'eFG%': 0.516, 'FGA': 245, 'FTA': 37, 'TOV': 36, 'FT': 29, 'ORB': 21}, {'eFG%': 0.652, 'FGA': 319, 'FTA': 111, 'TOV': 40, 'FT': 56, 'ORB': 100}, {'eFG%': 0.534, 'FGA': 307, 'FTA': 126, 'TOV': 87, 'FT': 98, 'ORB': 140}, {'eFG%': 0.565, 'FGA': 1376, 'FTA': 218, 'TOV': 128, 'FT': 186, 'ORB': 49}, {'eFG%': 0.542, 'FGA': 252, 'FTA': 56, 'TOV': 78, 'FT': 43, 'ORB': 47}, {'eFG%': 0.564, 'FGA': 633, 'FTA': 90, 'TOV': 36, 'FT': 77, 'ORB': 25}]
```

D:\School\Computer_Science\code>

Figure 31: Querying - Result 12**Test 13:**

```
1 print( stats(18,'ABC') )
```

Result 13:

```
[ ]
```

D:\School\Computer_Science\code>

Figure 32: Querying - Result 13

The reason that **Test 13** returns an empty list and not an error message is because it queries for team 'ABC' in the table 2018. Even though 'ABC' is not a team, the table 2018 is still a correct table and no queries from this table match with 'ABC' so nothing is returned therefore an empty list is printed. As opposed to **Test 11**, where 2019 is not a table so an error message is returned.

4.3 Generating Team Score

4.3.1 Generating Team Score Results Table

Test	Stage	Description	Type	Expected Result	Actual Result	Reference Screenshot
14	Generate Neutral Score	Generate score with weights 1.	Normal data	Same score every time.	✓	Test 14 Result 14 - Figure 33
15	Generate Neutral Score	Generate score with weights 1 on team 'ABC'	Erroneous data	Error	✓	Test 15 Result 15 - Figure 34
16	Generate Weighted score	Generate lowest possible score	Boundary data	Lowest score A Score	✓	Test 16 Result 16 - Figure 35
17	Generate Weighted score	Generate highest possible score	Boundary data	Highest score	✓	Test 17 Result 17 - Figure 36
18	Generate Weighted score	Generate score using any numbers from 0.75 to 1.25	Normal data	A Score	✓	Test 18 Result 18 - Figure 37

Test 14:

```
1 print( teamScore(18, 'GSW', [1,1,1,1]) )
```

Result 14:

```

D:\School\Computer_Science\code>python combine_data.py
C:\Users\Ian Cheng\AppData\Local\Programs\Python\Python37-
Warning: (1366, "Incorrect string value: '\xB1\xEA\xD7
UE' at row 518")
  result = self._query(query)
0.7193772996962753

D:\School\Computer_Science\code>python combine_data.py
C:\Users\Ian Cheng\AppData\Local\Programs\Python\Python37-
Warning: (1366, "Incorrect string value: '\xB1\xEA\xD7
UE' at row 518")
  result = self._query(query)
0.7193772996962753

D:\School\Computer_Science\code>

```

Figure 33: Generating Team Score - Result 14

Figure 33 shows the result when generating Golden State Warriors' neutral score from 2018. As can be seen, the score is the same every time it is run.

Test 15:

```
1 print(teamScore(18, 'ABC', [1,1,1,1]))
```

Result 15:

```

Traceback (most recent call last):
  File "combine_data.py", line 40, in <module>
    print(teamScore(18, 'ABC', [1,1,1,1]))
  File "combine_data.py", line 31, in teamScore
    eFGp = eFGp / count
ZeroDivisionError: division by zero

D:\School\Computer_Science\code>

```

Figure 34: Generating Team Score - Result 15

Generating a score using an invalid team generates a zero-division error because one part of the calculation requires a number to be divided by the number of items in the list from the function stats (tested in **test 12, 13**). From **Test 13** And **Figure 32**, we can see that an invalid team returns a list with zero items in it. Hence, the zero- division error.

Test 16:

```
1 print( teamScore(18, 'GSW', [0.75,0.75,0.75,1.25]) )
```

Result 16:

```

0.4776131703333474

D:\School\Computer_Science\code>

```

Figure 35: Generating Team Score - Result 16

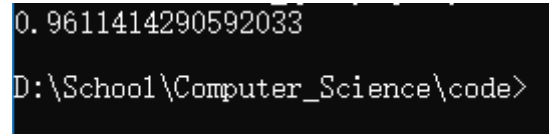
Figure 35 shows the resulting score when the first three weights is 0.75 and the last weight

is 1.25. These are the lowest and the highs number the weights can take on. Meaning this is the lowest team score.

Test 17:

```
1 print( teamScore(18, 'GSW', [1.25,1.25,1.25,0.75]) )
```

Result 17:



```
0.9611414290592033
D:\School\Computer_Science\code>
```

Figure 36: Generating Team Score - Result 17

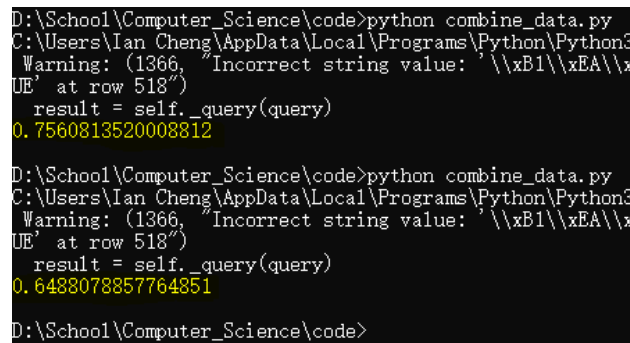
Figure 36 shows the other end of extreme weights. This is the highest team score.

Test 18:

```
1 print( teamScore(18, 'GSW', [1.02,0.83,1.15,0.91]) )
```

```
1 print( teamScore(18, 'GSW', [0.98,1.21,0.73,1.20]) )
```

Result 18:



```
D:\School\Computer_Science\code>python combine_data.py
C:\Users\Ian Cheng\AppData\Local\Programs\Python\Python3
Warning: (1366, "Incorrect string value: '\\xB1\\xEA\\x
UE' at row 518")
result = self._query(query)
0.7560813520008812
D:\School\Computer_Science\code>python combine_data.py
C:\Users\Ian Cheng\AppData\Local\Programs\Python\Python3
Warning: (1366, "Incorrect string value: '\\xB1\\xEA\\x
UE' at row 518")
result = self._query(query)
0.6488078857764851
D:\School\Computer_Science\code>
```

Figure 37: Generating Team Score - Result 18

Figure 37 shows the team score with two sets of four random weights. The return value lies in between the extreme results. Which makes sense.

Weighted Score Results table for GSW

Weights	Nature of weights	Score
[0.75,0.75,0.75,1.25]	Extreme	0.4776131703333474
[0.98,1.21,0.73,1.20]	Normal	0.6488078857764851
[1.00,1.00,1.00,1.00]	Neutral	0.7193772996962753
[1.02,0.83,1.15,0.91]	Normal	0.7560813520008812
[1.25,1.25,1.25,0.75]	Extreme	0.9611414290592033

4.4 Determining Record

4.4.1 Determining Record Results Table

Test	Stage	Description	Type	Expected Result	Actual Result	Reference Screenshot
19	Simulate Game	Simulate game between GSW and ORL	Normal data	Return the team that won and percentage simulations won	✓	Test 19 Result 19 - Figure 38
20	Simulate Game	Simulate game between GSW and CLE	Normal Data	Return the team that won and percentage simulations won	✓	Test 20 Result 20 - Figure 39
21	Simulate Game	Simulate game between GSW and 'ABC'	Erroneous data	Zero-Division Error	✓	Test 21 Result 21 - Figure 40
22	Simulate Season	Simulate GSW season with neutral coefficients	Normal data	Results of every game displayed and a final number of wins in the season	✓	Test 22 Result 22 - Figure 41
23	Simulate Season	Simulate season coefficients that generate lowest score	Boundary data	Simulation with no wins	✓	Test 23 Result - Figure 42

4.4.2 Test Evidence

Test 19:

```

1 for i in range(10):
2     print( simulateMainAtHome(score, 'GSW', 'ORL', 18) )

```

Result 19

```

GSW won 91.1 % of the time
won
GSW won 92.1 % of the time
won
GSW won 90.7 % of the time
won
GSW won 92.5 % of the time
won
GSW won 91.1 % of the time
won
GSW won 91.0 % of the time
won
GSW won 89.6 % of the time
won
GSW won 92.6 % of the time
won
GSW won 91.7 % of the time
won
GSW won 91.5 % of the time
won
D:\School\Computer_Science\code>

```

Figure 38: Determining Record - Result 19

When simulating games, a random score is added on to both teams. The size of that score depends on if they are home or away. As every game is simulated 1000 times, this means that a team shouldn't win the same number of those games every simulation.

Figure 38 shows the result when we run the small test above on the function. The test runs the 1000 games simulation 10 times. As shows in the result, even though GSW won all 10 times against ORL (Orlando Magic), the number of times they won each game differed. This result is reliable because it is consistent with real life. GSW is one of the best teams in the NBA whereas ORL is one of the worst so it makes sense that Golden State has a very high chance of winning a game however that chance is not 100%.

Test 20:

```

1 for i in range(10):
2     print( simulateMainAtHome(score, 'GSW', 'CLE', 18) )

```

Result 20:

```

GSW won 51.2 % of the time
won
GSW won 50.9 % of the time
won
CLE won 52.4 % of the time
lost
GSW won 51.2 % of the time
won
GSW won 50.6 % of the time
won
CLE won 50.7 % of the time
lost
GSW won 51.9 % of the time
won
GSW won 50.1 % of the time
won
GSW won 53.7 % of the time
won
GSW won 50.6 % of the time
won
D:\School\Computer_Science\code>

```

Figure 39: Determining Record - Result 20

As can be seen in **Figure 39**, this time GSW has won much less of the time and even losing on two games. This test further shows the reliability of the model because the Cavaliers was the team that the Warriors played against in the playoff finals that year. So, it's natural that they have less of a chance to win and even lose to a much better team than previous.

Test 21:

```

1 for i in range(10):
2     print( simulateMainAtHome(score, 'GSW', 'ABC', 18) )

```

Result 21:

```

Traceback (most recent call last):
  File "determine_record.py", line 1, in <module>
    from combine_data import *
  File "D:\School\Computer_Science\code\combine_data.py", line 40, in <module>
    print(teamScore(18,'ABC', [1,1,1,1]))
  File "D:\School\Computer_Science\code\combine_data.py", line 31, in teamScore
    eFGp = eFGp / count
ZeroDivisionError: division by zero

```

Figure 40: Determining Record - Result 21

The same error happened as **Test 15** because simulating games calls the function tested in **Test 15**. As 'ABC' is not a real team, the error occurs with the same reason as why **Test 15** resulted in an error.

Test 22:

```

1 print( simulateWest(west[1][0], 1, 0, 18, [1,1,1,1] ) )

```

Result 22:

```

GSW won 83.7 % of the time
UTA won 53.1 % of the time
GSW won 56.4 % of the time
GSW won 62.3 % of the time
DEN won 78.0 % of the time
GSW won 58.2 % of the time
DEN won 78.7 % of the time
OKC won 75.5 % of the time
GSW won 65.2 % of the time
DEN won 77.4 % of the time
OKC won 75.9 % of the time
51 wins

```

Figure 41: Determining Record - Result 22

This is a successful test as the function simulates all the games and determine a record. This record might seem low now but that is because this was tested using neutral coefficients as I am only testing if it's doing what it should.

Test 23:

```
1 print( simulateWest(west[1][0], 1, 0, 18, [0.75,0.75,0.75,1.25] ) )
```

Result 23:

```

SAS won 96.0 % of the time
HOU won 100.0 % of the time
SAS won 100.0 % of the time
0 wins
0

```

Figure 42: Determining Record - Result 23

The coefficients used generate the lowest possible score for GSW. This leads to them losing all the games.

4.5 Genetic Algorithm

4.5.1 Genetic Algorithm Results Table

Test	Stage	Description	Type	Expected Result	Actual Result	Reference Screenshot
24	Generic Genetic Algorithm	A non-problem specific genetic algorithm that finds 4 numbers that sums to 150	Normal data	A list of four numbers that sums to 150	✓	Test 24 Result 24 - Figure 43 & Figure 44

25	Specific Genetic Algorithm	A problem specific genetic algorithm. Find coefficients for GSW in year 2017	Normal Data	A list of four coefficients.	✓	Test 25 Result 25 - Figure 45
----	----------------------------	---	-------------	------------------------------	---	--

4.5.2 Test Evidence

Test 24:

```

1 target = 150
2 repeat = True
3 testPop = population(20,4,0,50)
4 print(testPop)
5 count = -1
6 while repeat == True:
7     for i in testPop:
8         if fitnessFunction(i,target) == 0:
9             count = count + 1
10            print('Solution found after', count,'evolution(s)')
11            print(i)
12            repeat = False
13            break
14        else:
15            count = count + 1
16            print(count,'evolution(s)')
17            print(testPop)
18            testPop = evolution(testPop,target)
19            break

```

This code is run on the Genetic Algorithm function. It finds the solution to the problem in varying number of evolutions.

Result 24

```

, 44, 34], [23, 44, 26, 19], [48, 43, 28, 12]] sad
0 evolution(s)
[[11, 25, 36, 1], [42, 50, 46, 30], [29, 22, 48, 17], [9, 50, 28, 26], [30, 19, 24, 21], [18, 15, 39,
12], [49, 47, 36, 13], [17, 36, 44, 4], [27, 41, 35, 38], [38, 13, 42, 7], [26, 17, 4, 39], [32, 49, 3
6, 12], [24, 6, 19, 48], [37, 46, 9, 7], [32, 38, 31, 24], [50, 19, 22, 21], [18, 33, 28, 49], [28, 32
, 44, 34], [23, 44, 26, 19], [48, 43, 28, 12]]
1 evolution(s)
[[49, 47, 36, 13], [27, 41, 35, 38], [28, 32, 44, 34], [42, 50, 46, 30], [48, 43, 28, 12], [32, 49, 36
, 12], [18, 33, 28, 49], [29, 22, 48, 17], [48, 43, 36, 12], [49, 47, 28, 13], [32, 49, 28, 12], [18,
33, 36, 49], [42, 33, 28, 30], [18, 50, 46, 49], [42, 32, 44, 30], [28, 50, 46, 34], [18, 33, 46, 49],
[42, 50, 28, 30], [49, 50, 46, 13], [42, 47, 36, 30]]
Solution found after 2 evolution(s)
[42, 50, 28, 30]

```

Figure 43: Genetic Algorithm - Result 24.1

The above result shows the Genetic Algorithm finding the correct solution in 2 evolutions. The sum of 42, 50, 28 and 30 is indeed 150. There are varying number of solutions found after a different number of evolutions.

```
[[29, 23, 3, 31], [33, 4, 1, 36], [14, 43, 14, 40], [24, 42, 7, 10], [19, 22, 4, 44], [17, 12, 39, 25],
 [48, 31, 15, 12], [30, 43, 20, 7], [31, 31, 42, 3], [6, 38, 19, 44], [13, 5, 2, 27], [21, 34, 8, 39],
 [14, 24, 42, 11], [46, 41, 18, 18], [33, 34, 28, 7], [20, 42, 22, 40], [0, 6, 43, 47], [48, 46, 8, 2
8], [32, 46, 32, 10], [33, 26, 39, 36]] sad
0 evolution(s)
[[29, 23, 3, 31], [33, 4, 1, 36], [14, 43, 14, 40], [24, 42, 7, 10], [19, 22, 4, 44], [17, 12, 39, 25],
 [48, 31, 15, 12], [30, 43, 20, 7], [31, 31, 42, 3], [6, 38, 19, 44], [13, 5, 2, 27], [21, 34, 8, 39],
 [14, 24, 42, 11], [46, 41, 18, 18], [33, 34, 28, 7], [20, 42, 22, 40], [0, 6, 43, 47], [48, 46, 8, 2
8], [32, 46, 32, 10], [33, 26, 39, 36]]
1 evolution(s)
[[33, 26, 39, 36], [48, 46, 8, 28], [20, 42, 22, 40], [46, 41, 18, 18], [32, 46, 32, 10], [14, 43, 14,
40], [6, 38, 19, 44], [33, 4, 1, 36], [32, 43, 14, 40], [14, 46, 32, 10], [46, 41, 19, 18], [6, 38, 1
8, 44], [33, 26, 39, 40], [20, 42, 22, 36], [33, 4, 39, 36], [33, 26, 1, 36], [33, 46, 1, 36], [32, 4,
32, 10], [33, 46, 32, 36], [32, 26, 39, 10]]
2 evolution(s)
[[33, 46, 32, 36], [33, 26, 39, 40], [33, 26, 39, 36], [48, 46, 8, 28], [32, 43, 14, 40], [46, 41, 19,
18], [20, 42, 22, 36], [33, 4, 39, 36], [46, 41, 19, 40], [33, 26, 39, 18], [33, 41, 19, 36], [46, 26
, 39, 18], [48, 43, 14, 28], [32, 46, 8, 40], [32, 43, 14, 36], [33, 26, 39, 40], [33, 43, 14, 36], [3
2, 26, 39, 40], [33, 42, 22, 36], [20, 46, 32, 36]]
3 evolution(s)
[[33, 46, 32, 36], [46, 41, 19, 40], [33, 26, 39, 40], [33, 26, 39, 40], [32, 26, 39, 40], [20, 46, 32
, 36], [33, 26, 39, 36], [48, 43, 14, 28], [33, 46, 39, 36], [33, 26, 32, 36], [33, 26, 39, 40], [32,
46, 32, 36], [20, 46, 32, 36], [33, 46, 32, 36], [20, 41, 32, 36], [46, 46, 19, 40], [33, 26, 39, 40],
[32, 26, 39, 40], [46, 41, 19, 40], [32, 26, 39, 40]]
4 evolution(s)
[[46, 46, 19, 40], [33, 46, 32, 36], [33, 46, 32, 36], [46, 41, 19, 40], [32, 46, 32, 36], [33, 46, 39
, 36], [33, 26, 39, 40], [33, 26, 32, 36], [46, 46, 32, 40], [33, 26, 19, 36], [33, 46, 19, 40], [46,
41, 32, 36], [46, 46, 19, 36], [32, 46, 32, 40], [46, 46, 19, 40], [32, 46, 32, 36], [33, 46, 39, 36],
[46, 46, 19, 40], [32, 46, 19, 40], [46, 46, 32, 36]]
Solution found after 5 evolution(s)
[32, 46, 32, 40]
D:\School\Computer_Science>
```

Figure 44: Genetic Algorithm - Result 24.2

Test 25:

```
1 def runGA(target, conference, cNum, tNum, year ):
2     size = 12
3     # not 4
4     test = population(size ,4,75,125)
5     repeat = True
6     count = 0
7
8     while repeat == True:
9         count = count + 1
10        test = evolution(test, target, conference, cNum, tNum, year)
11        solution = test
12        print(count, 'evolution(s)')
13        if len(solution) != size:
14            return solution
15            repeat = False
16            break
17        ''' If the returning value is a solution, the len of the 1 d list will be Four
18        this is not 'size' so we know that what was returned was the solution
19
20        if returning value is the new population for next evolution, the length
21        will be 6 and the same as size so we know this is the pop for next evolution
```

```

22         so these functions are not executed '''
23 print( runGA(67, 'west', 1, 0, 17) )

```

Result 25:

```

67 wins
0 fitness score
3 evolution(s)

[0.98, 1.2, 1.21, 1.2]
These are the solution coefficients. Using this to predict next season's record.

```

Figure 45: Genetic Algorithm - Result 25

Figure 45 shows the result when the function is run to predict the record in 2018 using the record from 2017 as fitness function. It successfully converges to a solution and finds the coefficients after three evolutions.

4.6 Prediction

4.6.1 Prediction Results Table

In this section, I will be trying to predict the record in 2018 even if the season has already happened. This is so that I can see how my program performs by having a result to compare to. In 2018, the Warriors had a record of 58 – 24 so I will be comparing the success of my program to this.

Test	Stage	Description	Type	Expected Result	Actual Result	Reference Screenshot
26	Predict season records	Prediction using all previous season stats	Normal data	A record prediction of year 2018	✓	Test/Result 26 - Figure 46, Figure 47
27	Predict season records	Prediction with only the one previous season stat	Normal Data	A record prediction of year 2018	✓	Test/Result - Figure 49, Figure 48, Figure 50, Figure 51, Figure 52, Figure 54, Figure 53, Figure 55, Figure 56,

4.6.2 Test Evidence

Test / Result 26:

```

GWS is predicted to win 75 games in 2018 Their record: 75 - 7
My program took 2202.1908707618713 to run

```

Figure 46: Prediction - Test/Result 26.1

```

GWS is predicted to win 69 games in 2018 Their record: 69 - 13
My program took 2726.2916004657745 to run

```

Figure 47: Prediction - Test/Result 26.2

As can be seen in **Figure 46**, GSW is predicted to win 75 games. This is a pretty bad prediction seeing as the NBA record of wins in a season is 73 games and that we know they actually won 58 games. Running it a second time, **Figure 47** also show a big difference between predicted and actual. The program also took 2202 seconds (≈ 37 minutes) and 2726 (≈ 45 minutes) to run, this is quite long.

Test / Result 27:

```

GWS is predicted to win 61 games in 2018 Their record: 61 - 21
My program took 1217.1231956481934 to run

```

Figure 49: Prediction - Test/Result 27.1

```

GWS is predicted to win 58 games in 2018 Their record: 58 - 24
My program took 1425.09405875206 to run

```

Figure 48: Prediction - Test/Result 27.2

```

GWS is predicted to win 55 games in 2018 Their record: 55 - 27
My program took 2085.4885334968567 to run

```

Figure 50: Prediction - Test/Result 27.3

```

GWS is predicted to win 58 games in 2018 Their record: 58 - 24

```

Figure 51: Prediction - Test/Result 27.4

```

GWS is predicted to win 63 games in 2018 Their record: 63 - 19
My program took 72.59346008300781 to run

```

Figure 52: Prediction - Test/Result 27.5


```
GSW is predicted to win 62 games in 2018 Their record: 62 - 20  
My program took 1901.391215801239 to run
```

Figure 54: Prediction - Test/Result 27.6

```
GSW is predicted to win 60 games in 2018 Their record: 60 - 22  
My program took 381.9618318080902 to run
```

Figure 53: Prediction - Test/Result 27.9

```
GSW is predicted to win 61 games in 2018 Their record: 61 - 21  
My program took 98.7295606136322 to run
```

Figure 55: Prediction - Test/Result 27.7

```
GSW is predicted to win 64 games in 2018 Their record: 64 - 18  
My program took 650.1366879940033 to run
```

Figure 56: Prediction - Test/Result 27.8

Results generated were much better. Two out of the nine times tested, the program actually predicted the record correctly in **Figure 48** and **Figure 51**. Other results were also in the ball park with the highest difference between expected and observed games won being 6 in **Figure 56** where 64 wins was predicted. The run time however, is not consistent with a range of 2013 seconds (\approx 34 minutes). The shortest run time was 72 seconds and the longest time being nearly as long as it took to run multiple years worth of stats.

5 Evaluation

5.1 Objectives Reflection

Objective number	Objective	Met?	Reference Test	Comment
Setup – 1	Set up a database	✓	Initialization test	Set up an empty database ready for stats to be stored
Setup – 2	Acquire all relevant data from basketball-reference	✓	Test 2	Successfully web scraped 5 seasons worth of data into Pandas DataFrame
Setup – 3	Manipulate the data so that it is the correct data types	✓	Test 5	Successfully changed types from all Text to correct ones: double, BigInt, text
Setup – 4	Manipulate the data so that it is compatible with SQL	✓	Test 5 Test 6	Successfully changed heading names so that it doesn't contain any special characters
Setup – 5	Create a connection between MySQL database and Python	✓	Test 1	Successfully completed this objective by using SQLAlchemy to create a connection so requests can be made
Setup – 6	Store all the data	✓	Test 1	Successfully automated the process of storing data from DataFrame MySQL using

				SQLAlchemy and Pandas
Program – 1	Query current season players for the team	✓	Test 10	Successfully generated a list of names queried from database, used to pass into function to get stats
Program – 2	Query previous season relevant stats for the players	✓	Test 12	Successfully used the list of names as the parameter to query stats from the database
Program – 3	Produce a team score	✓	Test 18	Successfully produced a unique score for every team based on the players' stats
Program – 4	Simulate season	✓	Test 22	Successfully simulated every game of the season taking into account randomness and advantage. Results of each game displayed
Program – 5	Genetic Algorithm for optimization	✓	Test 25	Successfully evolved the coefficients using a genetic algorithm and real record as fitness function
Program – 6	Find correct coefficients	✓	Test 25	Successfully found correct coefficients that determine the correct record for the team
Program – 7	Simulate next season	✓	Test 26 Test 27	Successfully used the correct coefficients to

				simulate the next season
Program – 8	Repeat simulation	✓	Test 26 Test 27	Repeated the simulation of the next season 5 times.
Program – 9	Display the average predicted record for next season	✓	Test 26 Test 27	Display the average predicted record

5.2 Functionality of entire program evaluation

5.2.1 Prediction

Task	Description	Result
Predict record correctly	Predict within 5 games of actual record	✓, ✗

Although all the objectives have been met, the accuracy of prediction is a success and a fail. The reason why this was a success and a fail is because using the initial model (All previous stats), the record generated was pretty inaccurate, as can be seen in the Testing section. However, changing the model by using only the one previous year gave some very good predictions.

5.2.2 Program run time

The duration the program runs for varies significantly from a minute to somewhere like 30 – 45 minutes. Some runs take longer than this and still don't converge to a solution. This big difference is largely due to the population size of our initial coefficients. Using a small population size somewhat relies on the initialization of individuals to be close to the solution.

5.2.3 Improvements

There are a few things that may lead to this program's prediction to be unreliable. One thing is that it doesn't take into account any new (rookie) players coming into the league. Because they have never played in the league, there are no statistics available for them on basketball-reference. However, sometimes rookie players can have a very big impact on a team especially for the worst teams in the league as they get high draft picks for the best players out of college or from overseas.

In light of the test results, it prompted me to think just how good a model using the past 5 years data is. Although using only 5 years may seem way too little, my test results suggests that it might actually be too far back and even data from just 5 years ago is not applicable to the present. This may not be totally incorrect as basketball is a constantly evolving game and a player can have a break out year any time and can improve significantly just in one off-season.

Furthermore, I am currently putting the records in a JSON file as a key-value pair that is loaded into Python and obtained that way. This is not the best method as it means I will have to input the new records for future seasons. I am doing this because there is no way for me to web scrape that data as every single record is listed on a different web page and the record is part of a larger table on that web page.

If I was to revisit this problem again, I would try to include the value of high drafted rookies into the strength of the team. This will improve the reliability of the prediction as power score of the team is more representable. I will also be using the past two years or just the past year data. Also, I would find a better way to include the teams' record for use in the fitness function.

References

- [1] Y. (. Yang, "Predicting Regular Season Results of NBA Teams," Berkeley, California, 2015.
- [2] J. Uudmae, "CS229 Final Project: Predicting NBA Game," Stanford, California.
- [3] Yale Undergraduate Sports Analytics Group, "NBA Model Math," 2018. [Online]. Available: <https://sports.sites.yale.edu/nba-model-math>. [Accessed 24 February 2019].
- [4] J. Jacobs, "Introduction to Oliver's Four Factors," 5 September 2017. [Online]. Available: <https://squared2020.com/2017/09/05/introduction-to-olivers-four-factors/>. [Accessed 28 February 2019].
- [5] NBAstuffer, "How the NBA Schedule is Made," [Online]. Available: <https://www.nbastuffer.com/analytics101/how-the-nba-schedule-is-made/>. [Accessed 22 February 2019].
- [6] V. Mallawaarachchi, "Introduction to Genetic Algorithms—Including Example Code," 8 July 2017. [Online]. Available: <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>. [Accessed 5 January 2019].
- [7] t. f. e. Wikipedia, "Genetic algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Genetic_algorithm. [Accessed 5 February 2019].
- [8] "Genetic Algorithms - Crossover," [Online]. Available: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm. [Accessed 5 February 2019].
- [9] I. Bicking, "Python HTML Parser Performance," 30 March 2008. [Online]. Available: <http://www.ianbicking.org/blog/2008/03/30/python-html-parser-performance/index.html>. [Accessed 12 March 2019].
- [10] SQLAlchemy, "SQLAlchemy 1.3 Documentation," [Online]. Available: <https://docs.sqlalchemy.org/en/latest/core/engines.html>. [Accessed 16 March 2019].