Data Analytics Web App for NBA Roster Construction

Kailen Shinmoto

December 2021

Abstract

This paper covers the process of creating a web-app focused on using data analytics to improve an NBA team's roster construction. This is done through the use of the Four Factors model constructed by Dean Oliver to identify an NBA team's strength and weaknesses, and then suggesting players from around the league that could potentially impact the designated team's weakness in a positive way. The web-app is developed with respect to previous examples of roster construction analytics in sports, as well as understanding the importance of what translates to winning more basketball games in the modern NBA. Once finishing the web-app, the project is then reflected upon with an evaluation of the finished product as well as an open conversation of how the project could be further improved or additional features that can be added.

1 Introduction

With each passing year it seems that data analysis deepens its grasp on everyday life and better maximizes the efficiency of all things we do as humans. It is said that on average, every human created at least 1.7 MB of data per second in 2020. With all this data being created, the possibilities of how to use this information is endless, including using the data we gather from the major entertainment industry that is the sporting world here in the United States. Ever since the practices of "Moneyball" seeped into the sports world, the way statistics and data analysis has been used in sports has changed forever.

Many people have heard the story of the Oakland Athletics and the movie that was later made based on the 2002 team along with their general manager at the time, Billy Beane. This story of a small budget franchise team using statistics and data analysis to compete with big name organizations and win more professional baseball games is one of the first major stories of data analysis in sports of this century, and was a sign of the era of statistics that we are in and that is still continuing to develop today. However, this project won't be focusing on baseball, but will be focusing on a data analysis project in basketball instead. So what are the ways data analytics is used on roster construction in basketball?

Well, the truth is that a lot of the information used in the decision-making process of crafting an NBA roster is mostly unavailable to the public. Whatever analytics process is used for the numerous NBA teams in adding or trading for players is mostly kept behind closed doors. However, this project aims to contribute a resource to the decision-making process of constructing an NBA roster. Roster construction was a big focus in the story of "Moneyball" and how they focused on the stat of on-base percentage to help them win games. So what if there was a program out there that could analyze a team's roster and suggest current NBA players for that team to consider signing or trading for? That is what this project is about: creating a program that will identify NBA teams' strengths and weaknesses, and suggest possible roster changes that could be made to address the teams' weaknesses and create a more efficient roster construction.

2 Background/Lit Review:

First off, before we continue with the paper, here is a list of basketball terms and stats that you will need to know going forward and their meanings:

- Games/GP = Games Played in a Season
- GS = Games Started
- FGM = Field Goals Made (Field goals are shots made excluding free throws)
- FGA = Field Goals Attempted
- FG% = Field Goal Percentage (Percentage of shots made)
- 3pt/3ptM = Three Pointers Made
- 3ptA = Three Pointers Attempted
- 3pt% = Three Pointers Percentage
- 2pt/2ptM = Two Pointers Made
- 2ptA = Two Pointers Attempted
- 2pt% = Two Pointers Percentage
- 13.EFG% = Effective Field Goal Percentage, (FGM + (3ptM * 1.5)) / FGA
- FTM = Free Throws Made
- FTA = Free Throws Attempted
- FT% = Free Throws Percentage
- FT Rate = Ratio of Free Throw Makes per 100 FGA
- Rebounds = Grabbing a missed shot
- ORb = Offensive Rebound (Rebound gained on offense)
- ORb% = One of the Four Factors, percentage of possessions out of 100 possessions where the team gets an offensive rebound
- DRb = Defensive Rebound (Rebound gained on defense)
- Def ORb% = One of the Four Factors, percentage of possessions out of 100 possessions where the opposing team gets an offensive rebound
- Assists = Passing to a teammate who scores after
- Steals = Taking the ball from the other team
- Blocks = Hitting the ball on its upward motion during a shot by the opposing team
- Turnovers = Losing the ball or causing a foul that gives the ball to the opposing team
- PF = Personal Fouls
- PPG = Points Per Game
- PPS = Points Per Shot, (Total Points off of 2ptA and 3ptA)/(Total FGA)
- Off Pts/Poss = Offensive Points per 100 Possessions

- Def Pts/Poss = Defensive Points per 100 Possessions
- D_FGM = Defensive field goals made, how many FGM an opposing player averages when defended by this player
- D_FGA = Defensive field goals attempted, how many FGA an opposing player averages when defended by this player
- D_FG% = Defensive field goal percentage, the FG% an opposing player shoots when defended by this player
- PCT_PM = Percentage plus minus, the change in FG% between an opposing players normal FG% and their FG% when defended by this player
- Upcoming FA = Upcoming Free Agent, will be a potential free agent after this season
- Future FA = Future Free Agent, will be a potential free agent after next season
- Multiyear Player = Is on contract for 3+ years, players you will probably have to trade for

To restate our project's goal, we want to create a web-app that uses data analytics to identify NBA teams' strengths and weaknesses, and in return suggests NBA players from across the league that could positively impact the teams' areas of improvement. This is a project that we found to be interesting in the fact that there are not many resources available to the public on what happens behind the scenes of free agency in the the NBA, along with the analytics involved that goes into a team's decision-making process when deciding whether or not to sign a player. To add to this, each year players are being picked up in the free agency process or traded for, and they do not always fit into the team dynamic or address that team's weaknesses. The impact this project can make as well as why it is important is that it offers a new resource for teams to view their possible areas of improvement, along with the player suggestions from our program so that they are able to potentially make better informed, stats-based decisions when signing or trading for a player.

The first step to this process of determining our player suggestions is dependent on each team's specific weaknesses. However, there are so many stats involved in basketball besides just how many wins and losses a team has, and whether or not they have a "good" offense or "bad" defense. We're sure that there are numerous ways for our project to designate a team's strength or weakness, but the final choice made for how we would determine these aspects of a team was through the Four Factors model. The Four Factors model is a term coined by Dean Oliver in his book "Basketball on Paper" written in 2004 [11]. The model focuses on four main principles that Oliver believed to highly correlate with winning more basketball games. These four principles were: to score as often as possible, grab as many rebounds as possible, limit the amount of turnovers your team produces, and to get to the free throw line as much as possible. The way the four principles are measured in statistics were through Effective Field Goal Percentage (EFG%), Offensive Rebounding Percentage (ORb%), Turnover Percentage (TOV%), and Free Throw Rate (FT Rate). For a refresher on what these all mean, the definitions of the Four Factors and how they are measured is shown in the list of terms for the reader to know at the beginning of this section. When viewing a team's Four Factors, we can also look at the factors from both the offensive and defensive side of the ball, thus turning our Four Factors into eight factors instead (as we can measure the same four factors for a team's opponent). It is through these Four Factors (on both their offensive and defensive outputs) that we will identify a team's strengths and weaknesses.

So how did we end up on the Four Factors as our measurement for how we will will be identifying NBA teams' areas of improvement? There are multiple reasons why we chose the Four Factors as the way we would observing the NBA teams. These reasons are its accuracy

in predicting a team's win projection as well, the stats for each team are well documented, and it keeps a simplistic approach for us to be suggesting players off of. For the first point, there have been articles done on the accuracy of the Four Factors model. An article by Konstantinos Kotzias found that the Four Factors model led to a more accurate win prediction rate when compared to using each team's offensive and defensive efficiencies [3]. The book "Mathletics" also touches upon the Four Factors model in its basketball section where it contributes a 1% change in any of the Four Factors is equivalent to a team winning 3.5 more games in a season [14].

Another benefit to the Four Factors model is that it is well documented. Since the Four Factors have been around since 2004. The site that we will be using to supply our Four Factors stats is the site Cleaning the Glass created by Ben Falk. This site has an offensive and defensive rating for each of the Four Factors for every NBA team. And the last benefit of using the Four Factors model in our project is the simplicity of it. Since the information of league rankings for each of the factors are supplied by the Cleaning the Glass website, we can easily determine how a team ranks in each of the Four Factors when compared to the rest of the league. This makes it simple for our project to determine a team's strengths and weaknesses in the Four Factors.

However, the Four Factors model is not the only way to observe an NBA team, as we can use numerous other statistics to determine areas they could improve. It is important to be aware of the other ways that this project could be reproduce without the Four Factors model. One other way that we could observe NBA teams are using machine learning to determine which stats correlate the most with winning in the NBA today. An article of a project done by David Peterson showed the change in stats correlating to winning NBA games of the years 1990-2020 [4]. The article touches upon numerous team stats that they found to relate to winning, and in the most recent season at the time, two-point field goal percentage was the stat most correlated to winning in the 2019-20 season. We could have used an approach like this in our project but settled on the Four Factors model since it would not fluctuate from season to season like the other stats correlated to winning will, as well as the Four Factors model having a clearer sense of organization.

Lastly, it is important to touch upon previous examples in sports where roster construction analytics have been used. The most prevalent example of this is the story of the Oakland Athletics that we have mentioned previously [10]. The Oakland Athletics were heavily outmatched in payroll when compared to other financially muscular franchises in the MLB (luckily the NBA has salary caps), but were able to use data analytics to help themselves win games and compete at the same level as the franchises who would attempt to "buy" their wins. The Athletics were able to use data analytics to find that on-base percentage was a stat that lead to more wins (since the more your players get on base, the higher your chances of scoring). In a sense, we are doing the same thing except with the Four Factors model, being that the Four Factors are our indicators of what translates to winning more games in the NBA. With all these factors in mind of previous literature related to our topic, we can now move on to our methods of actually producing our web-app.

3 Methods:

3.1 Project Approach:

We know our goal for the project: to make a web-app that will identify a team's strengths and weaknesses, and based on those weaknesses suggest players from around the league that could impact those weaknesses positively and potentially fit into their team dynamic better than some of their current players. To keep the approach to the project simple, the tasks that we will need our web-app to do are: 1) determine team weaknesses, 2) identify players that are possible contributors to the team's weaknesses, 3) suggest players that would address those weaknesses based on the players identified as contributors.

3.2 Web-Scraping:

The first step to accomplishing our tasks we want our web-app to do is that we will need stats. The whole project is based around data so we will need to collect the stats we will be using to create our web-app. The first task we want our web-app to do is determine team weaknesses. We will be doing this through the Four Factors model that is mentioned previously where each team is given a league ranking in the four factors of EFG%, ORb%, TOV%, and FT Rate for both their offense and defense. This information is supplied through the Four Factors page on the website Cleaning the Glass. A screenshot of this page is shown below.

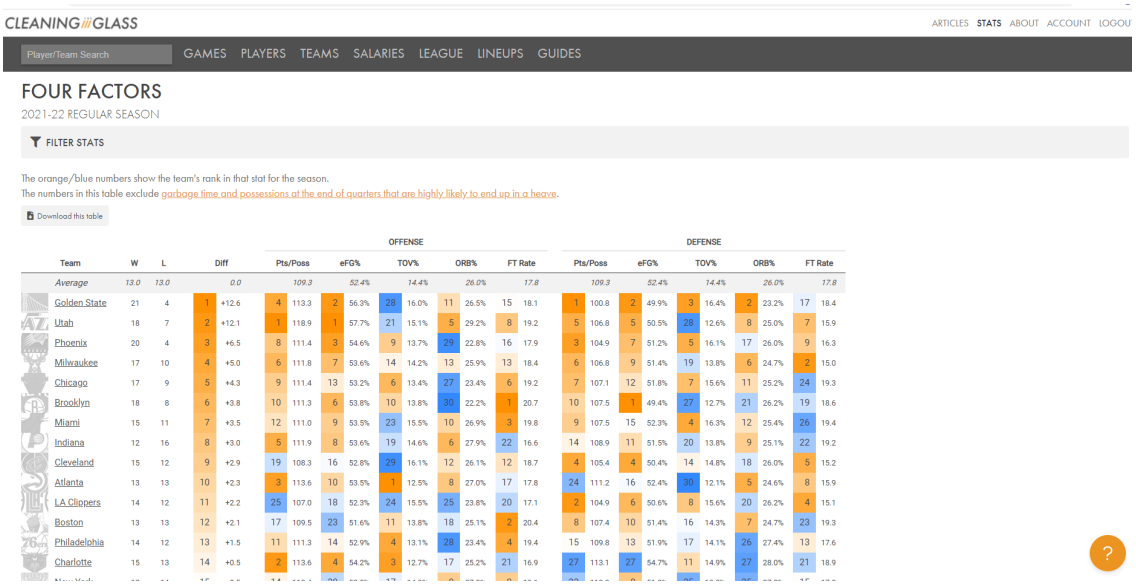


Figure 1: Screenshot of Four Factors from Cleaning the Glass

After scraping the information on the Four Factors and each team's league ranking for each of the factors, we can then move on to the web-scraping of the player data. For our player data we scraped information from both of these sites: basketball-reference.com and nba.com/stats/. These two sites have a wide range of information available on every team and player in the NBA as well as information that is updated after every game being played.

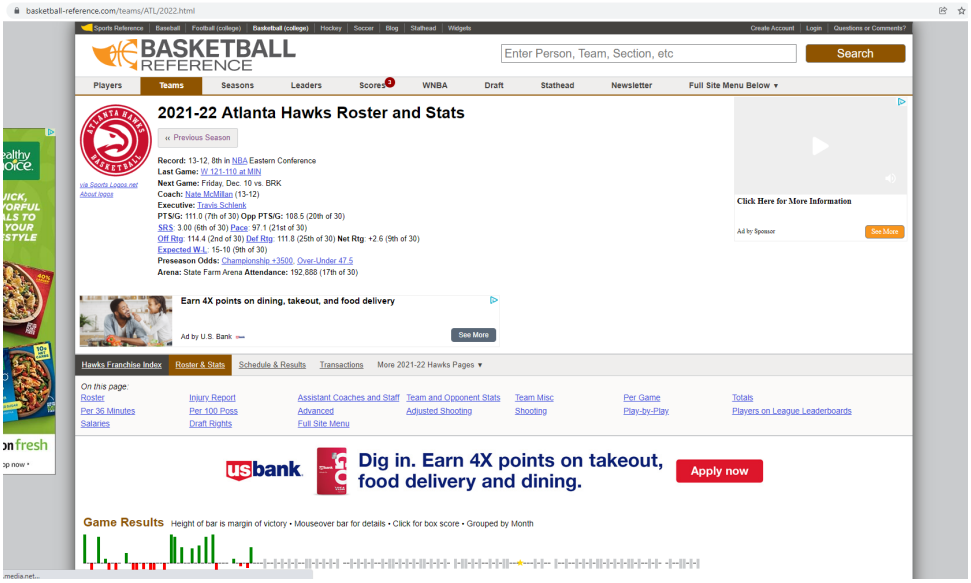


Figure 2: Screenshot of basketball-reference.com (Atlanta Hawks page

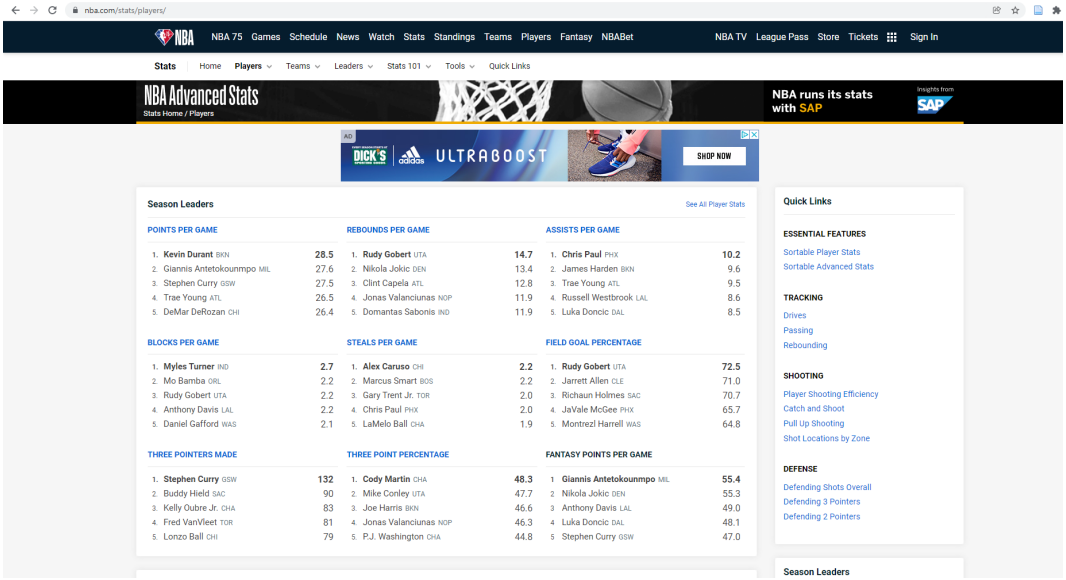


Figure 3: Screenshot of nba.com/stats/

First off, we will start with the process of scraping information from the basketball-reference site. In our process of web-scraping, we used Python, the pandas package, the BeautifulSoup package, and the requests package to scrape stats from basketball-reference.

```
import pandas as pd
import requests
from bs4 import BeautifulSoup
```

After importing these packages, you can move on to inspecting the html code of the basketball-reference pages so we know exactly how to get the stats that we are looking for. Since we are looking to get the basic player stats here from basketball-reference, the table that we want to scrape information from is the Totals table pictured below.

Totals Share & Export ▾ Glossary

Regular Season

Rk		Age	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	Trae Young	23	25	25	855	228	500	.456	69	176	.392	159	324	.491	.525	132	149	.886	18	81	99	236	23	3	99	43	657
2	John Collins	24	25	25	807	164	292	.562	31	75	.413	133	217	.613	.615	65	83	.783	43	153	196	54	18	34	27	72	424
3	Clint Capela	27	25	25	747	127	223	.570	0	1	.000	127	222	.572	.570	31	61	.508	101	216	317	33	22	36	16	61	285
4	Kevin Huerter	23	24	14	647	102	214	.477	42	107	.393	60	107	.561	.575	11	13	.846	7	74	81	58	8	6	30	44	257
5	Bogdan Bogdanović	29	20	20	564	89	199	.447	43	110	.391	46	89	.517	.555	11	14	.786	7	64	71	49	13	4	18	38	232
6	Danilo Gallinari	33	22	0	472	67	161	.416	32	82	.390	35	79	.443	.516	40	43	.930	12	76	88	31	11	5	10	25	206
7	Cam Reddish	22	21	0	472	83	201	.413	34	89	.382	49	112	.438	.498	41	46	.891	18	41	59	19	23	4	28	30	241
8	Delon Wright	29	23	0	322	24	62	.387	8	21	.381	16	41	.390	.452	13	17	.765	18	47	65	49	16	3	10	12	69
9	De'Andre Hunter	24	11	11	306	50	111	.450	15	38	.395	35	73	.479	.518	4	10	.400	4	26	30	7	7	3	14	32	119
10	Lou Williams	35	20	0	270	46	121	.380	13	36	.361	33	85	.388	.434	22	26	.846	6	23	29	26	11	2	18	19	127
11	Gorgui Dieng	32	21	0	177	19	45	.422	7	21	.333	12	24	.500	.500	9	14	.643	21	51	72	15	5	6	11	26	54
12	Timothé Luwawu-Cabarrot	26	18	4	173	23	57	.404	16	40	.400	7	17	.412	.544	5	6	.833	1	19	20	9	3	3	3	20	67
13	Solomon Hill	30	13	1	139	3	20	.150	2	13	.154	1	7	.143	.200	0	0		6	17	23	12	4	2	1	16	8
14	Jalen Johnson	20	7	0	19	5	10	.500	2	4	.500	3	6	.500	.600	2	2	1.000	0	5	5	0	0	0	0	0	14
15	Skylar Mays	24	7	0	19	4	5	.800	1	2	.500	3	3	1.000	.900	5	5	1.000	0	2	2	1	0	0	2	2	14
16	Sharife Cooper	20	5	0	11	1	4	.250	0	2	.000	1	2	.500	.250	0	0		0	0	0	4	0	0	1	0	2
			25		6000	1035	2225	.465	315	817	.386	720	1408	.511	.536	391	489	.800	262	895	1157	603	164	111	305	440	2776

The totals page shown is the stats for player's totals of the 2021-22 season that is currently happening. We decided to scrape information from both last year's stats as well as this year's stats since the current NBA season had not begun yet when we first started the project, as well as it currently only being around twenty-six games into the season at this point. We also scraped player's totals going back to the 2017-18 season since the original plan for the project was to allow users to alternate between different spans of seasons and see how the results would change, however our project was not able to reach that point.

In order to be able to scrape the information we want from the Totals table on basketball-reference, we will need multiple details: the site url, NBA team abbreviations, the season years we are scraping stats from, the table name we are scraping from, and the stat names we want from the table. To do the web-scraping we used a function called get_totals() that would take the inputs of team, year (season), and the table name. This function is pictured below.

```python
def get_totals(team, year, table):

    # assign site url to variable
    team_url = (f'https://www.basketball-reference.com/teams/{team}/{year}.html')

    # make request using site url
    team_res = requests.get(team_url)

    #BeautifulSoup Library parses the content of an HTML document, in this case clips_res
    team_soup = BeautifulSoup(team_res.content, 'lxml')

    # identify table we want to scrape data from (team totals)
    team_totals = team_soup.find(name = 'table', attrs = {'id' : table})

    # create list of dictionaries to hold each player's stats
    team_stats = []

    # print team to help keep track of any bugs and know where you are
    print(team + f' totals {year}')

    # loop through totals table for each player and get each stat
    for row in team_totals('tr')[1:]:
        # this loop will go through each row in the totals table (row represents a player in the table)
        # goes through each row and scrapes all stats in the row

        # create dict for the row(represents each player)
        player = {}

        # use .find to get info we want for every stat
        player['Name'] = row.find('td', {'data-stat' : 'player'}).text
        print(row.find('td', {'data-stat' : 'player'}).text) # print out names of player to help with bu

        # continue using .find to get rest of stats in table
        player['Age'] = row.find('td', {'data-stat' : 'age'}).text
        # use int() to turn info we scrape into a numeric value
        player['Games'] = int(row.find('td', {'data-stat' : 'g'}).text)

        # use try and except for cases where players don't have a value for the stat we are scraping
        try:
            player['GS'] = int(row.find('td', {'data-stat' : 'gs'}).text)
        except:
            player['GS'] = 0 # if a player doesn't have a value, assign stat value as 0

        # repeat int() and .find to scrape more stats
        player['Min'] = int(row.find('td', {'data-stat' : 'mp'}).text)
        player['FGM'] = int(row.find('td', {'data-stat' : 'fg'}).text)
        player['FGA'] = int(row.find('td', {'data-stat' : 'fga'}).text)

        # use try and except for cases where players don't have a value for the stat we are scraping
        try:
            # using float() instead of int() here since value has decimals
            player['Field Goal %'] = float(row.find('td', {'data-stat' : 'fg_pct'}).text)
```

Figure 4: Screenshot of $\text{get}_{t}otals()function, continues on next page$

```
        # use try and except for cases where players don't have a value for the stat we are scraping
        try:
            # using float() instead of int() here since value has decimals
            player['Field Goal %'] = float(row.find('td', {'data-stat' : 'fg_pct'}).text)
        except:
            player['Field Goal %'] = 0

        # continue scraping rest of stats in the row for the current player and turning them into numeric values
        player['3pt'] = int(row.find('td', {'data-stat' : 'fg3'}).text)
        player['3ptA'] = int(row.find('td', {'data-stat' : 'fg3a'}).text)
        try:
            player['3pt %'] = float(row.find('td', {'data-stat' : 'fg3_pct'}).text)
        except:
            player['3pt %'] = 0
        player['2pt'] = int(row.find('td', {'data-stat' : 'fg2'}).text)
        player['2ptA'] = int(row.find('td', {'data-stat' : 'fg2a'}).text)
        try:
            player['2pt %'] = float(row.find('td', {'data-stat' : 'fg2_pct'}).text)
        except:
            player['2pt %'] = 0
        try:
            player['EFG %'] = float(row.find('td', {'data-stat' : 'efg_pct'}).text)
        except:
            player['EFG %'] = 0
        player['FTM'] = int(row.find('td', {'data-stat' : 'ft'}).text)
        player['FTA'] = int(row.find('td', {'data-stat' : 'fta'}).text)
        try:
            player['FT %'] = float(row.find('td', {'data-stat' : 'ft_pct'}).text)
        except:
            player['FT %'] = 0
        player['ORb'] = int(row.find('td', {'data-stat' : 'orb'}).text)
        player['DRb'] = int(row.find('td', {'data-stat' : 'drb'}).text)
        player['Rebounds'] = int(row.find('td', {'data-stat' : 'trb'}).text)
        player['Assists'] = int(row.find('td', {'data-stat' : 'ast'}).text)
        player['Steals'] = int(row.find('td', {'data-stat' : 'stl'}).text)
        player['Blocks'] = int(row.find('td', {'data-stat' : 'blk'}).text)
        player['Turnovers'] = int(row.find('td', {'data-stat' : 'tov'}).text)
        player['PF'] = int(row.find('td', {'data-stat' : 'pf'}).text)
        player['Points'] = int(row.find('td', {'data-stat' : 'pts'}).text)

        # this makes sure to append only players since one row in the table will be the team totals with a min va
        if player['Min'] < 10000:
            team_stats.append(player)

    # turn list of stats into df
    team_roster_df = pd.DataFrame(team_stats)

    # return the final df of all players and their stat totals for the designated team
    return team_roster_df
```

What is happening in this function is that it is taking in the team, season, and table information to know which team and season stats it should be scraping from basketball-reference, as well as the table input since I scraped stats from both the regular season totals and playoff totals tables. When the code determines which table it will be scraping from and then assigns that table to the variable team_totals, it then loops through each of the rows in the table, taking down each of the players names and statistics about their totals from that table. Once it is done scraping all of the information from the table, it then appends the information to a list which is then turned into a dataframe using pandas. The dataframe is then returned at the end of the function. To summarize it all, the function scrapes the totals for all players of a designated team and returns that information in a dataframe. This is the process we used for the majority of our web-scraping done through basketball-reference.

After we had scraped all information from basketball reference, since we only grabbed the totals of player stats we needed to go through the stats to add them up and divide them by games played to get each players per game stats. The process for that can be seen through our function of get_player_stats where it would take in the inputs of a player and the range of seasons you want to find their per game stats for. If the season input was 1, then it would return the per game averages of our scraped stats for the player of the past season and current season. If the season input was 4, then it would return the per game averages of our scraped

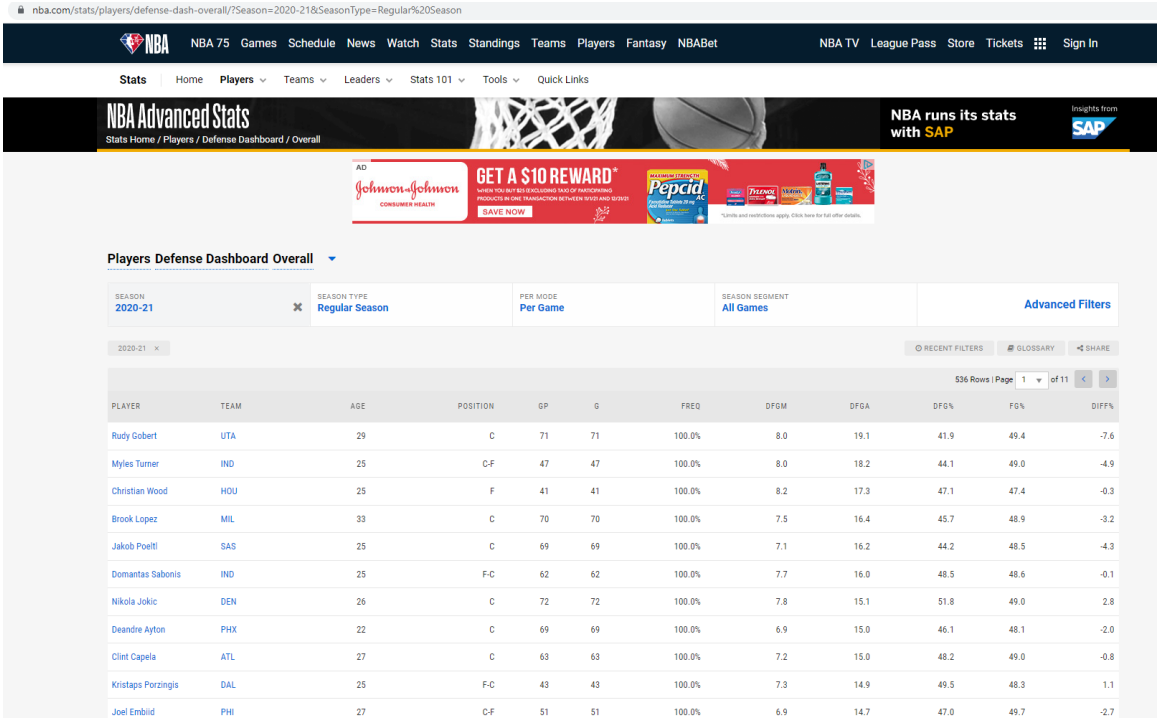
stats for the player of the past four seasons and the current season. Once it gets the totals of the determined span of season, it will then use simple mathematical formulas to get the per game and percentage stats for our players using the sums of the totals from each season. The coding behind how the per game and percentage stats are calculated is featured below.

```
#turn list into df
player_totals_df = pd.DataFrame(player_stats)

if len(player_totals_df) >= 1:

    #create row in df that adds up all stat totals of the seasons
    player_totals_df = player_totals_df.append(player_totals_df.sum(axis = 0, skipna = True), ignore_index= True)

    final_stats = {}

    # now go through sums of all stats and get per game averages for each
    final_stats['Name'] = player
    final_stats['Age'] = age
    final_stats['Games'] = player_totals_df.loc[len(player_totals_df)-1, 'Games']

    # create gp to hold total number of games player has played in allotted time
    gp = player_totals_df.loc[len(player_totals_df)-1, 'Games']
    final_stats['GS'] = player_totals_df.loc[len(player_totals_df)-1, 'GS']
    # round stats to second decimal mark, use gp to get per game avg for stats
    final_stats['Min'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'Min'] / gp),2)
    final_stats['FGM'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'FGM'] / gp),2)
    final_stats['FGA'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'FGA'] / gp),2)
    # create fga to get percentage stat
    fga = (player_totals_df.loc[len(player_totals_df)-1, 'FGA'])
    final_stats['FG %'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'FGM'] / fga),2) # fgm/fga
    final_stats['3pt'] = round(float(player_totals_df.loc[len(player_totals_df)-1, '3pt'] / gp),2)
    final_stats['3ptA'] = round(float(player_totals_df.loc[len(player_totals_df)-1, '3ptA'] / gp),2)
    # create threeptA to get percentage stat
    threeptA = (player_totals_df.loc[len(player_totals_df)-1, '3ptA'])
    final_stats['3pt %'] = round(float(player_totals_df.loc[len(player_totals_df)-1, '3pt'] / threeptA),2)
    final_stats['2pt'] = round(float(player_totals_df.loc[len(player_totals_df)-1, '2pt'] / gp),2)
    final_stats['2ptA'] = round(float(player_totals_df.loc[len(player_totals_df)-1, '2ptA'] / gp),2)
    # create twoptA to get percentage stat
    twoptA = (player_totals_df.loc[len(player_totals_df)-1, '2ptA'])
    final_stats['2pt %'] = round(float(player_totals_df.loc[len(player_totals_df)-1, '2pt'] / twoptA),2)
    # create twoptm for use in efg%
    twoptm = (player_totals_df.loc[len(player_totals_df)-1, '2pt'])
    # efg_threes represents value of 3ptm in efg%
    efg_threes = 1.5 * (player_totals_df.loc[len(player_totals_df)-1, '3pt'])
    # efg% = (1.5(3ptm) + twoptm) / fga
    final_stats['EFG %'] = round(float((twoptm + efg_threes) / fga),2)
    final_stats['FTM'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'FTM'] / gp),2)
    final_stats['FTA'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'FTA'] / gp),2)
    # create fta to get percentage stat
    fta = (player_totals_df.loc[len(player_totals_df)-1, 'FTA'])
    final_stats['FT %'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'FTM'] / fta),2)
    final_stats['ORb'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'ORb'] / gp),2)
    final_stats['DRb'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'DRb'] / gp),2)
    final_stats['Rebounds'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'Rebounds'] / gp),2)
    final_stats['Assists'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'Assists'] / gp),2)
    final_stats['Steals'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'Steals'] / gp),2)
    final_stats['Blocks'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'Blocks'] / gp),2)
    final_stats['Turnovers'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'Turnovers'] / gp),2)
    final_stats['PF'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'PF'] / gp),2)
    final_stats['PPG'] = round(float(player_totals_df.loc[len(player_totals_df)-1, 'Points'] / gp),2)
```

We repeated this process of finding each players per game stat averages and percentages by going through each team's current rosters (which was also scraped off of basketball-reference) and turning the final outputs into one large concatenated dataframe. Besides the game stats for players, we also scraped the information of current team rosters and player and team salaries through the same process of scraping the player stat totals, which details I will leave out due to the repetition of the process. This concludes the last of the web-scraping process we did through basketball-reference. Now to move on to the nba.com/stats site.

To web-scrape from the NBA stats site, the process is a little different and can be a little hard to understand without a quality tutorial. The first thing you need to do is determine which page of the NBA stats site you would like to scrape your data from. For our project,

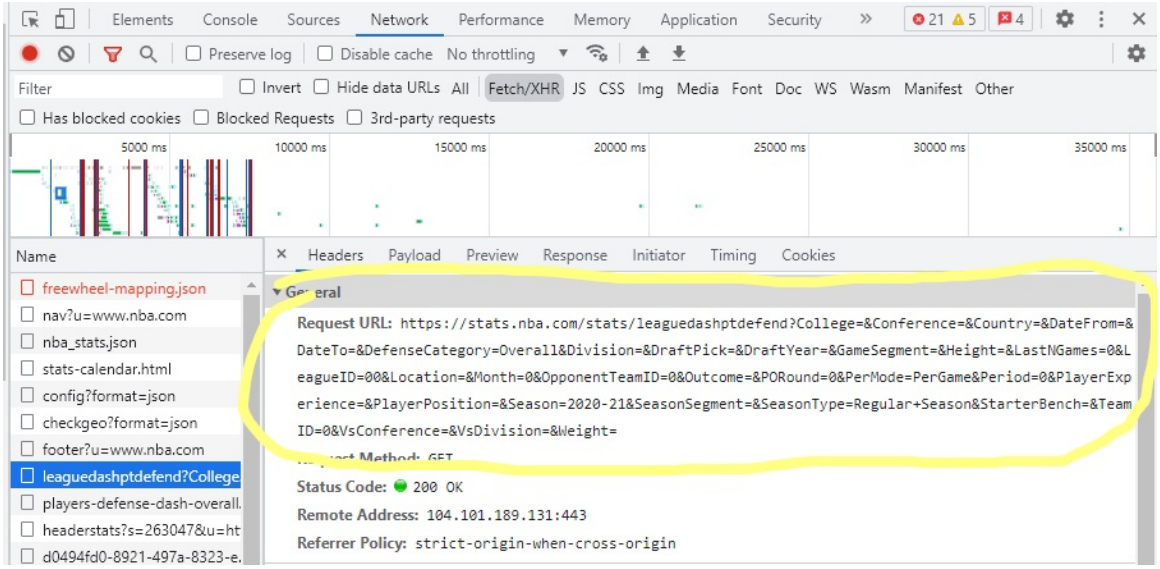
we needed defensive stats to round out our player profiles as most of the stats scraped from basketball-reference were offensive stats. The page we used was the defense dashboard overall page.



After you have found the page you want, you can then transition to the coding side of the web-scraping process. The function we used for web-scraping on the nba.com/stats/ pages was a little simpler compared to the function used to scrape basketball-reference. The function we used to scrape the NBA stats pages is called get_nba_stats, and has a similar logic to the function we used to scrape basketball-reference, however it has a few important differences. The most important information you need to scrape the NBA site is the headers information.

```
# headers used for web-scraping, very important!
headers = {'Connection': 'keep-alive',
    'Accept': 'application/json, text/plain, */*',
    'x-nba-stats-token': 'true',
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.
    'x-nba-stats-origin': 'stats',
    'Sec-Fetch-Site': 'same-origin',
    'Sec-Fetch-Mode': 'cors',
    'Referer': 'https://stats.nba.com/',
    'Accept-Encoding': 'gzip, deflate, br',
    'Accept-Language': 'en-US,en;q=0.9'}
```

The headers initialized in our web-scraping code is important because it is what allows us access to the stats we want to scrape from the NBA site, as it asks for this information in the web-scraping process. After we have our headers initialized, we will then need to inspect the NBA stats page we want to scrape and find the request url.



After finding the page's request url, we are now set to scrape the page for the stats we want. We will do this using the get_nba_stats function we mentioned earlier which can be seen below.

```
def get_nba_stats(url):

    # use url variable and request info from nba site, turn to json file
    json_file = requests.get(url, headers=headers).json()

    # from json file, assign variables that will represent the rows of the info and the headers/columns
    data = json_file['resultSets'][0]['rowSet']
    columns = json_file['resultSets'][0]['headers']

    # turn records of stats into a dataframe
    nba_stats = pd.DataFrame.from_records(data, columns=columns)

    return(nba_stats)
```

We can use this function which will take in our request url we found through inspecting the NBA stats page, and it will then return a pandas dataframe of the stats we want.

```
# get defensive stats using stats.nba.com url (2020-21 season)
defense_stats = get_nba_stats('https://stats.nba.com/stats/leaguedashptdefend?College=&Conference=&Country=&DateFrom=&
```

Once we run the function with our request url for the defensive dashboard overall page, we will have our dataframe with the relevant information we wanted to scrape. This process can be repeated for any of the stats pages found on the nba.com/stats/ site and will be useful for any future projects where stats from the NBA site will need to be scraped. This covers the process of how we scraped our stats from the multiple sites of Cleaning the Glass, Basketball-Reference, and the NBA stats site. Now we can continue on with how we were able to utilize these stats in progressing our project and reaching our end goal of our web-app.

3.3 Categorizing Players:

Now that we have our Four Factors information along with the league rankings in the Four Factors for all teams, we can move on to figuring out which players on each team are possibly contributing to their team's area(s) of weakness. In order to do this, we will need to compare each player and their stats to a certain benchmark. But how do we determine the benchmark, or what they will be compared to? Originally, the plan was to compare players to the league average in stats, but that would be unfair to the players with less playing time, as well as favoring the elite players. We then pivoted to comparing by position, but even then there are players that transcend multiple positions as basketball in the NBA continues to trend towards positionless basketball. Our final outcome in comparing players was to use machine learning and compare players and their stats to each other based on the role they played for their teams.

There are multiple examples of these being done in the past as far as using machine learning to observe player stats and designate roles or clusters for players based on the stats given. Recent examples of this are this article written by Todd Whitehead where he groups players by their offensive roles [6]. Unfortunately, Whitehead's version of grouping players relies on play-type information that was given by Synergy Sports, which would cost $40 monthly for us to access. Because of the lack of funds for the project, we would go a different route. An example of a similar way to group players that would not need to access to Synergy Sports information was the GitHub project done by Ahmeed Cheema where he uses 189 different features scraped from the NBA Stats site as the information used to create his player clusters [7]. If you are interested in the 189 features used for the machine learning process, there will be a txt file added to the project's GitHub repository including the features used.

The first step in this process of separating players into their clusters and assigning them roles is that we need stats to base these clusters off of. This means more web-scraping of course. All the web-scraping done for this machine learning process is done through the nba.com/stats site with the same process mentioned above of how we were able to retrieve our player's defensive stats. It was through this web-scraping process that we were able to retrieve the 189 features needed to create our player clusters, and all of the features scraped came from the most recent complete NBA season (2020-21 season). This is so that we have a full season to base each player's role off of and it will be the most recent information available, being that the current season does not have enough games played yet to be reliable information.

Once all players' stats were collected, the data needed to be cleaned to make sure all stats were treated as numeric values as well as nan values being replaced with zero values. After cleaning the data and placing all the information into one dataframe, we were then ready to move on to the next step of fitting the data for the clustering.

The first thing we will need to do before we move on to the machine learning side of our clustering, is we need to import the packages needed. The packages we will be using in this process are StandardScaler, PCA (Principal Component Analysis), KMeans, plt from matplotlib, pandas, and silhouette_score. The correct way to import these packages into a python code is shown below.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import silhouette_score
```

Once we have these packages imported, we can move on to the first step of our clustering process which is to use Principal Component Analysis (PCA) to get our data ready for the clustering. The way that PCA works is it gathers all of the 189 stats we scraped previously for every player, and turns those stats into two x and y values for each of the players. These x and y values will be used for plotting our players and helps with the KMeans process of clustering the players into groups. However, there are a few things we need to be sure of before using PCA. The first thing we want to do is make sure we get rid of players that didn't play enough games or minutes, as these outliers could sway the data and cause inaccuracies. The requirements we had in these two fields were that we got rid of any players with less than 8 minutes played per game, and less than 15 games played.

```
testdf = df[(df.MPG > 8) & (df.GP > 15)].reset_index(drop=True) # minimum requirements of minutes and games played
```

The other step we need to do before we use PCA is to fit the data and standardize them to each other, since the computer won't be able to understand the difference between 1 steal compared to 20 points, it only sees the numbers. This means we need to scale all the stats to each other before running our PCA.

```
# PRINCIPAL COMPONENT ANALYSIS

features = [x for x in df.columns if (x != 'PLAYER_NAME') &  (x != 'POSITION')]

x = testdf.loc[:, features].values
y = testdf.loc[:,['PLAYER_NAME']].values

x = StandardScaler().fit_transform(x) # standardize all values
```

Now that our stats are standardized to each other, we can run our PCA and get the two values for our axes for each player.

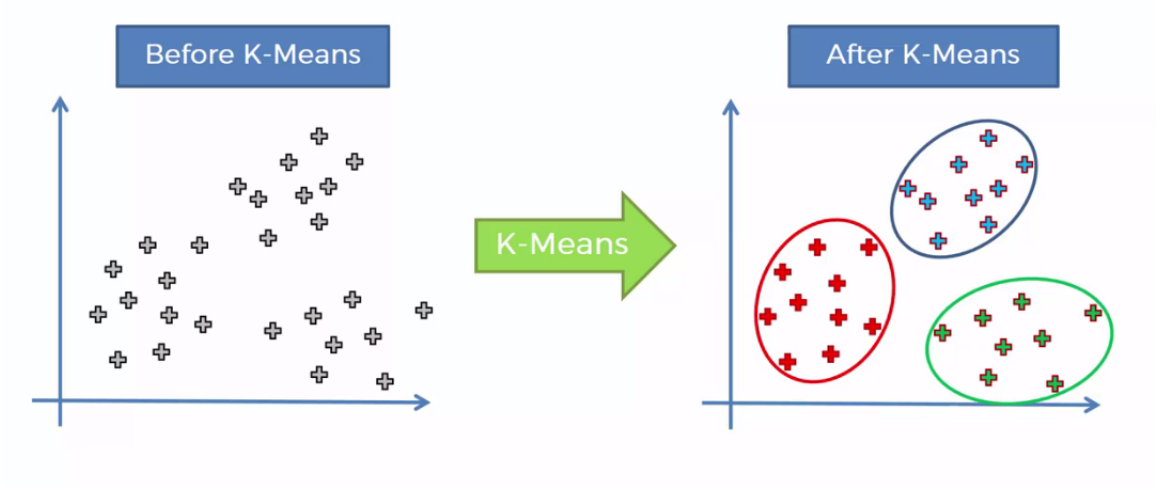
```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['pc1', 'pc2'])

final = pd.concat([principalDf, testdf[['PLAYER_NAME']]], axis=1)

final = final[['PLAYER_NAME','pc1','pc2']]
final.columns = ['player','pc1','pc2']
```

Index	player	pc1	pc2
0	Aaron Gordon	1.89385	0.90254
1	Aaron Holiday	-1.85504	-6.0703

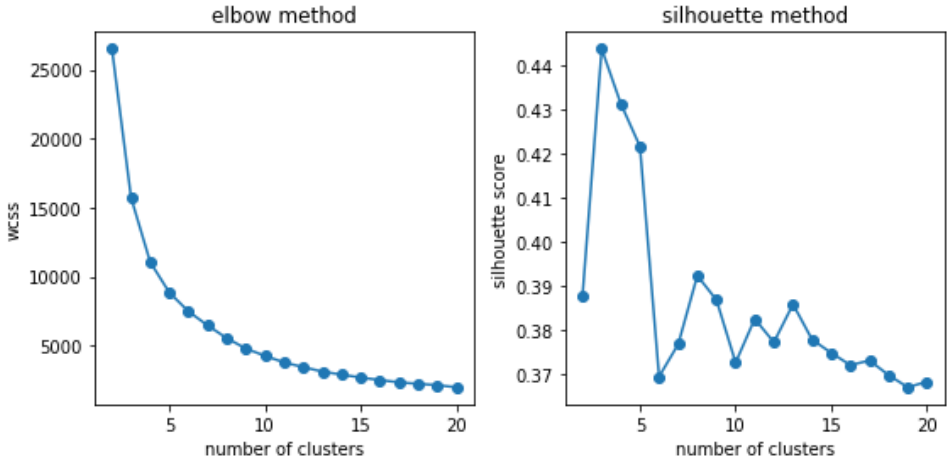
Figure 5: PCA Results Examples

As you can see from our code, we designated that we wanted two axes from our PCA by setting the n_components to 2, and then we ran our principalComponents using the 189 features we standardized previously. After that, we put the results into a dataframe where we could see every player as well as their two values from the PCA.

Now that we have our results from the PCA, we can move on to the process of our Kmeans clustering. First, we will explain briefly how KMeans clustering works. To put it simply, what Kmeans clustering does is finds centroids to base its clustering around. The user will choose how many clusters they want, and the Kmeans clustering will find that many centroids that are in the most optimized positions. To do this, it starts with randomly selected centroids, then repeats the process of going through each point of data until finding the most optimal centroids and the centroids have stabilized.



But how do users determine how many clusters is the right amount for their set of data? The answer to how many clusters a person should use when using KMeans is shown through two types of methods: the Elbow method and the Silhouette method.



These are examples of graphs for both the Elbow and Silhouette method based on our data for our NBA players. To read the elbow method graph, the lower a dot is on the graph, the more accurate that number of clusters will be. This is true for the silhouette method graph as well, where the lower a dot is on the graph, the more accurate that number of clusters will be. However, as the user, we can choose to use this information more as a guideline since we won't be wanting to separate our NBA players into twenty different clusters. When looking at both the graphs, it is clear that the area around ten clusters looks to be the most accurate and reasonable for our project, but we felt that ten clusters was still a lot to be separating our players into. Thus, we ended on the amount of eight clusters as that was the area where the

bend in the elbow graph starts to flatline more. Of course, the amount of clusters chosen in our project will heavily impact the end results so in the future we could incorporate a way for users of our web-app to adjust the amount of clusters or categories for the NBA players and see how that affects the outcomes of the player suggestions.

Since we now know our number of clusters that we want, we can move on to using KMeans clustering to actually create our groupings of players.

```
# KMEANS MODEL
kmeans = KMeans(n_jobs = -1, n_clusters = 8, init='k-means++')
kmeans.fit(principalComponents)
pred0 = kmeans.predict(principalComponents)

frame = pd.DataFrame(principalComponents)
frame['cluster'] = pred0
final['cluster'] = pred0
frame['cluster'].value_counts()
principalDf['pred'] = pred0
```

The code behind using the KMeans clustering to make our groups is relatively simple as it is just a few lines. The main lines are the first three lines where we use KMeans and designate how many clusters we want. Then we fit our KMeans model to our PrincipalComponents that have our two axes values for every player. Then we turn the outcome into a dataframe and add a column with the cluster results.

Since the outcomes of our KMeans can vary slightly everytime we run it, I wanted to make sure that players were ending up in their correct groupings, or the groupings that they were more often placed in. I did this by repeating the KMeans process three times and placing players in the group that they ended up in the most. Here is a final scatter plot of our player groupings.

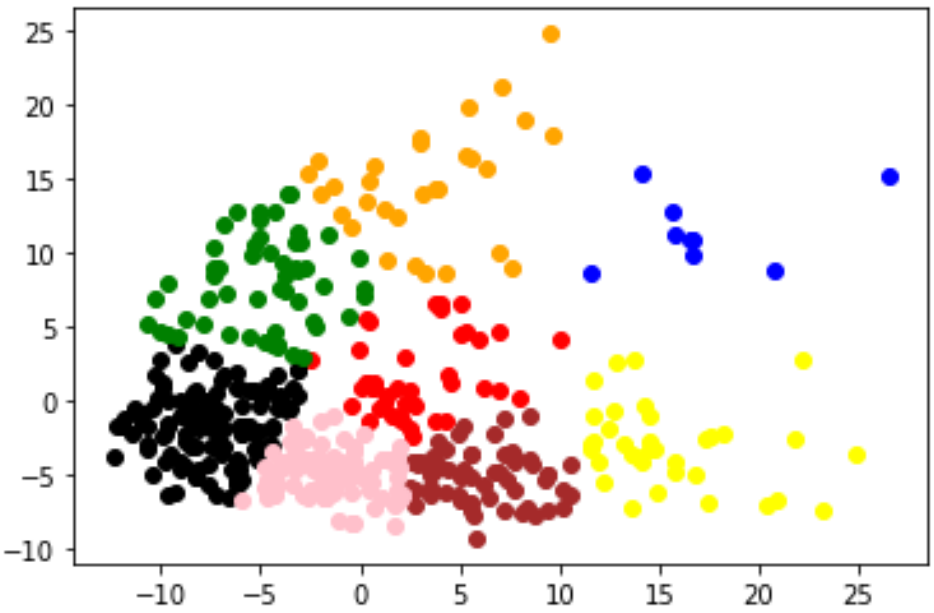


Figure 6: Scatter Plot of Player Groupings

After getting our player groupings, we then looked at all the players that got grouped together so we could designate labels for each of the final groupings of players. Here were the final groupings of players: Blue - High Usage Bigs, Yellow - Ball Dominant Playmakers, Orange - Traditional Bigs, Brown - Secondary Playmakers, Red - Versatile Forwards, Green - Roleplaying Bigs, Pink - Roleplaying Guards, Black - Off Bench Players.

As a quick rundown of what each role means: High Usage Bigs are All-Star worthy big men who are a main point of attack for their offense (Nikola Jokic, Joel Embiid, Domantas Sabonis), Ball Dominant Playmakers are usually a team's top perimeter player who is a main point of attack on offense (LeBron James, Chris Paul, Luka Doncic), Traditional Bigs are big men who play big roles for their teams but usually in the traditional form for big men that focuses more around screens, rebounds, and scoring around the basket, Secondary Playmakers are players who are quality perimeter players but are usually not the first option when it comes to scoring for their team (Lonzo Ball, CJ McCollum, Reggie Jackson), Versatile Forwards are usually wing or bigs who help their teams in multiple ways on offense and defense (Draymond Green, Harrison Barnes, Ben Simmons), Roleplaying Bigs are players that fill big men roles as a non-emphasis on the offensive end (Serge Ibaka, Mo Bamba, Javale McGee), Roleplaying Guards are players that fill guard roles as a non-emphasis on the offensive end (Alex Caruso, Danny Green, Luke Kennard), Off Bench Players were players that didn't get many minutes in game and usually have a limited usage on their teams.

Now that we have our groupings of players the next steps become quite clear. We will use the information we've scraped previously from both the Basketball-Reference and NBA stat sites and attach our player roles to each of the players and their stats that we've scraped. This means that we should now have a dataframe where we have players' game averages and along with their roles attached to them.

FTM	FTA	FT %	ORb	DRb	Rebounds	Assists	Steals	Blocks	Turnovers	PF	PPG	Team_x	Role
.04	7.97	0.88	0.61	3.13	3.74	9.44	0.92	0.14	4.08	1.82	26.08	ATL	Ball Dominant
.5	3.05	0.82	1.86	5.88	7.74	1.4	0.57	1.02	1.26	3.06	16.84	ATL	Traditional Big
.68	3.08	0.54	4.33	9.08	13.41	0.93	0.74	1.74	1.01	2.38	13.42	ATL	Traditional Big
.66	0.85	0.78	0.5	2.93	3.43	3.15	0.94	0.36	1.15	2.32	11.51	ATL	Secondary Playmaker
.33	1.66	0.8	0.97	2.92	3.88	3.8	1.36	0.38	1.08	1.01	8.3	ATL	Secondary Playmaker
.75	2.96	0.93	0.41	3.65	4.05	1.33	0.52	0.2	0.75	1.93	12.23	ATL	Versatile Forward
.2	2.61	0.84	0.8	2.67	3.47	1.2	1.22	0.29	1.31	2.14	11.45	ATL	Off Bench

Figure 7: Excerpt of dataframe of players and their stats along with their roles

Before moving on, it is important to highlight the weaknesses of KMeans and the potential downfalls it can cause when clustering through it. Since KMeans randomly picks centroids and has centroids as the main focus of how it groups clusters, it isn't able to find patterns among the groupings and just finds as many points as possible around the centroids it finds that are within a standard distance. An example of this can be seen in our figure below. The side on the right shows how the scatter plot was grouped when using KMeans clustering.

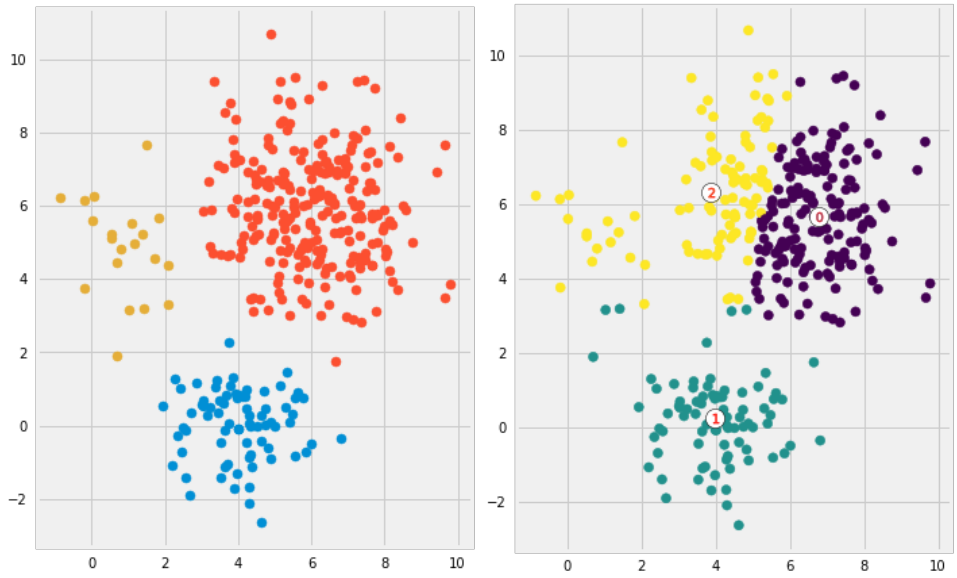


Figure 8: Excerpt of dataframe of players and their stats along with their roles

Here you can clearly see the difference between the grouping between the left and the right side. KMeans clustering focuses on finding the most optimized centroids for its clusters which leads to patterns like we see on the right, where the KMeans is not able to distinguish patterns that we as humans are able to recognize like the grouping on the left. Luckily, our scatter plot of NBA players doesn't have clearly laid out groupings or patterns like the side on the left in the figure above, however this doesn't mean that there aren't potentially better ways for our project to group players. This information is good to keep in mind if we ever want to redo the project or if others would like to improve upon our project while reproducing it.

3.4 Adding Player Percentiles:

At this point, we now have all the stats we need for our project collected, as well as the player roles we will be using to compare players to each other. This means we can move on to our next step of doing the comparisons and seeing how the NBA players and their stats stack up to their colleagues in the designated roles. This step is important because these comparisons will determine how a player ranks among their peers in each stat, and we will be using those rankings in our program to help us suggest players for teams depending on how high or low a player ranks in stats relating to their team's weaknesses. The way that we will be determining player ranks in each of the stats that we have scraped for every player is through percentile rankings. That is what our next step in our project is, to add a player percentile ranking for every stat of each player that will determine their percentile placement of the designated stat when compared to other players of the same role.

The first thing we need to do for this step is to create our function that will be adding percentile rankings for players as well as understanding how to determine a players percentile. The function we used in our project to do our percentile ranking is player_percentiles which took in an input for the player role we wanted to create our percentiles for. The function starts with looking through our dataframe of players and creating a separate dataframe of the players with a role matching the role input given to our function.

```
### repeat for defensive concat dataframe
def player_percentiles(role):
    role_list = []

    # loop through concat list and get role list
    for index, row in concat_4_defense.iterrows():
        if row['Role'] == role: # role
            role_list.append(row)

    # turn list into dataframe
    role_df = pd.DataFrame(role_list)
```

Once we have our dataframe of players from only the designated role we want, we then create a list of the player percentiles. We do this by creating a list of integers from 1 through the length of our role dataframe, and then divide each integer in the list by the length of the role dataframe. This gives us our list of player percentiles in a descending order.

```
    # create function that makes list of numbers 1 to len(ordered_list)
    def createList(r1, r2):
        return list(range(r1, r2+1))

    rank_list = createList(1, len(role_list))
    rank_list = sorted(rank_list, reverse=True)

    percentile_list = []

    for num in rank_list:
        percentile_list.append(round(num/len(role_list), 2))
```

Now that we have our list of player percentiles, the next steps are simple. Using our player percentile list, we will sort our role dataframe by each stat in a descending order (since our percentile list is also in descending order) that we want to create a ranking column for. And for every stat that we sort by, we then input our percentile list next to the stat it represents. We repeat this process for every numeric stat of the players that represents some form of production they have in game (we don't rank the players' minutes played or their salaries). Once we have done this, we have finished our function. Now we can move on to using the function on our collection of players and their stats based on their designated roles.

```
    # insert player percentiles rank for each stat
    role_df = role_df.sort_values(by=['FGM'], ascending=False)
    role_df.insert(5, 'FGM Rank', percentile_list)

    # repeat for rest of stats
    role_df = role_df.sort_values(by=['FGA'], ascending=False)
    role_df.insert(7, 'FGA Rank', percentile_list)

    role_df = role_df.sort_values(by=['FG %'], ascending=False)
    role_df.insert(9, 'FG% Rank', percentile_list)

    role_df = role_df.sort_values(by=['3pt'], ascending=False)
    role_df.insert(11, '3ptM Rank', percentile_list)
```

To get the percentile rankings for our players, we use our function of player_percentiles for each of our eight roles, and then after assigning each player their stat percentiles, we concatenate the information all into one dataframe. A final example segment of our dataframe with the player percentiles is shown below.

FGM Rank	FGM	FGA Rank	FGA	FG% Rank	FG %	3ptM Rank	3pt	3ptA Rank	3ptA	3pt% Rank	3pt %	2ptM Rank	2pt
0.53	8.32	0.71	18.95	0.12	0.44	0.62	2.4	0.65	6.89	0.35	0.35	0.47	5.91
0.93	6.54	0.97	11.75	0.37	0.56	0.97	1.26	0.93	3.2	0.97	0.4	0.77	5.27
0.87	5.87	0.9	9.94	0.47	0.59	0.33	0	0.13	0.01	0.13	0	0.9	5.87
0.32	4.46	0.33	10.14	0.58	0.44	0.42	1.94	0.44	5.3	0.47	0.37	0.26	2.52
0.04	3.07	0.02	6.74	0.81	0.46	0.05	0.84	0.05	2.24	0.42	0.37	0.23	2.23
0.43	3.81	0.45	8.9	0.18	0.43	0.82	1.86	0.7	4.62	0.91	0.4	0.14	1.96

3.5 Suggesting Potential Players for Teams:

With the percentile rankings for our players now in place, our program is finally ready for its most important step. This step is to determine player suggestions for each of the NBA teams based on their Four Factors weaknesses. The first step to doing this is to look at the NBA teams and figure out what each of their weaknesses are based on the Four Factors model. To do this we will look at each team individually, and look at that team's league ranking in each of their Four Factors. The way that we determined a team's weaknesses through the Four Factors was if the team was ranked below average in a factor when compared to the rest of the league. Being that there are thirty teams in the NBA, any team ranked above 15th would mean that they are below the median of the league and are in the bottom half of the NBA in that factor. To determine if a team is below the median of the league we simply go through each of their league rankings for all of the factors in the Four Factors model, and if their league ranking is above 15, then we save that ranking as a weakness as well as the difference between their ranking and the median.

```
def suggest_players(team, abbr):
    # create temp values for the four factors
    temp_values = {
        'temp_off_pts_poss': 0,
        'temp_off_efg': 0,
        'temp_off_tov': 0,
        'temp_off_orb': 0,
        'temp_off_ft': 0,
        'temp_def_pts_poss': 0,
        'temp_def_efg': 0,
        'temp_def_tov': 0,
        'temp_def_orb': 0,
        'temp_def_ft': 0,
    }

    # check to see which of the four factors are below average for designated team
    for index, row in four_factors.iterrows():
        if row['Team'] == team:
            if row['Off Pts/Poss Rank'] > 15: # rankings above 15 mean team is below league average
                temp_values['temp_off_pts_poss'] = row['Off Pts/Poss Rank'] - 15
            if row['EFG% Rank'] > 15:
                temp_values['temp_off_efg'] = row['EFG% Rank'] - 15
            if row['TOV% Rank'] > 15:
                temp_values['temp_off_tov'] = row['TOV% Rank'] - 15
            if row['ORb% Rank'] > 15:
                temp_values['temp_off_orb'] = row['ORb% Rank'] - 15
            if row['FT Rate Rank'] > 15:
                temp_values['temp_off_ft'] = row['FT Rate Rank'] - 15
            if row['Def Pts/Poss Rank'] > 15:
                temp_values['temp_def_pts_poss'] = row['Def Pts/Poss Rank'] - 15
            if row['Def EFG% Ramk'] > 15:
                temp_values['temp_def_efg'] = row['Def EFG% Ramk'] - 15
            if row['Def TOV% Rank'] > 15:
                temp_values['temp_def_tov'] = row['Def TOV% Rank'] - 15
            if row['Def ORb% Rank'] > 15:
                temp_values['temp_def_orb'] = row['Def ORb% Rank'] - 15
            if row['Def FT Rate Rank'] > 15:
                temp_values['temp_def_ft'] = row['Def FT Rate Rank'] - 15
```

Once we complete this process of determining areas of weakness for a team based on their

league rankings in the Four Factors model, we can then sort their weaknesses in descending order based on the difference between the factor's league ranking and the league median. This lets us know the order of a team's biggest weakness to their smallest weakness, as we would want to be suggesting players that will be impacting a team's biggest weaknesses first.

```
### sort value dict in descending order
sorted_values = sorted(temp_values.items(), key=lambda x: x[1], reverse=True)
```

Now that we are able to designate a team's weaknesses and the descending order of those weaknesses, we will need to determine which of our players' stats we have collected relate or impact the Four Factors and areas of weakness for a team. You can see the Four Factors and the stats that are related to each of the factors in the list created here.

```
off_pts_poss = ['EFG% Rank', 'PPG Rank', 'FGM Rank', 'FG% Rank']
off_efg = ['EFG% Rank', 'FG% Rank', '3pt% Rank', '2pt% Rank']
off_tov = ['TO Rank']
off_orb = ['ORb Rank', 'TRb Rank']
off_ft = ['FTM Rank', 'FTA Rank', 'FT% Rank']
def_pts_poss = ['PCT_PM Rank', 'D_FG% Rank', 'BLK Rank', 'STL Rank']
def_efg = ['PCT_PM Rank', 'D_FG% Rank', 'BLK Rank']
def_tov = ['STL Rank']
def_orb = ['DRb Rank', 'TRb Rank']
def_ft = ['PF Rank', 'STL Rank']
```

For the next step in our process of suggesting our players, now that we have the stats that relate to our Four Factors, we then need to determine how those stats would be weighted when we are comparing players to each other and doing our player suggestions. In our case, we decided to make the designated team's biggest weakness the most important factor to improve, thus it was given a weight of 3. The second biggest weakness for a team was then weighed at 2. And any weaknesses after that were weighted at 1. This means that if Offensive EFG% was a team's biggest weakness, the stats related to Offensive EFG% (EFG%, FG%, 3pt%, and 2pt%) would be given a weight of 3. These weights will be later used to determine which players on a team could be impacting the team's Four Factors that they are deficient in. How we save the stats and their weights is by going through a team's determined weaknesses, and then finding the related stats (related stats are shown in figure above) and saving the related stats for each weakness along with their weights.

There are many different ways to do weighted rankings, as this process is similar to many debates about medal weights given at the Olympics. There is an article that dives into the different ways Olympic medals could be weighted that in a sense could be related to how we choose to weigh our team weaknesses [5]. The reason we weighed the first and second weaknesses more than the other weaknesses that followed was because of the fact that we felt it would be in a team's best interest to improve the areas they are weakest in the most, as those are the areas they could see a drastic improvement in with the right decisions, and hopefully the large change in improving would be reflected in their records. Of course, if there were more time for the project, perhaps there would be a way for the user to adjust the weights of the weaknesses and see how the results of players suggested would change.

Now that we have determined how we are going to weight our stats, we can use the stat weights to determine which players on the designated team are possibly contributing to the team's factors that they are below the league median in. We will do this by going through our list of stats related to the team's Four Factors they are deficient in, then sorting players by each of those stats, and assigning the relevant weights of the stats to the five players with the

lowest percentiles of that stat. The reason we chose to only assign weights to the bottom five players of each stat is because the active NBA Roster is fifteen players, so five players are the bottom third of a team and any fraction larger than that would seem too large of a portion to be assigning weights to. However, if others would like to reproduce this project and change parameters of this process we would not object to it.

```
### create list to hold roster tallies
roster_counts = []

### gather tallies for worst five players of each important stat, append to roster_counts
for pair in tuple_list:
    i = 0
    team_roster = team_roster.sort_values(by=pair[0])
    for index, player in team_roster.iterrows():
        if i < 5:
            i += 1
            player_count = (player['Name'], pair[1])
            roster_counts.append(player_count)
```

Once we have gone through our list of stats relevant to the Four Factors and assigned weights of those stats to all players, we will then need to tally up the weights to get player totals. The players with a larger total are players who fell in the bottom five of the most stat columns related to the team's Four Factor deficiencies.

```
### create list of all roster players for us to add up tallies in
roster_sums = []
for index, row in team_roster.iterrows():
    player = {}
    player['Name'] = row['Name']
    player['Count'] = 0
    player['Role'] = row['Role']
    player['Status'] = row['Status']
    roster_sums.append(player)

### add up tallies
for tally in roster_counts:
    for player_sum in roster_sums:
        if tally[0] == player_sum['Name']:
            player_sum['Count'] = player_sum['Count'] + tally[1]
```

Once we know which players of the designated team have the largest weights, we can look at their player roles. By recording their player roles, we can then find other players around the league that play similar roles but excel in the stats related to the designated team's deficiencies.

```
### sort list of players by their counts
roster_sums = sorted(roster_sums, key=lambda i: i['Count'], reverse=True)

### get the roles for the top 3 players with highest counts
list_weak_roles = []
list_weak_players = []

for i in [0, 1, 2]:
    list_weak_roles.append(roster_sums[i]['Role'])
    list_weak_players.append(roster_sums[i]['Name'])

### get rid of any repeat roles
for role in list_weak_roles:
    count = 0
    for role2 in list_weak_roles:
        if role == role2:
            count += 1
            if count > 1:
                list_weak_roles.remove(role)
```

Once recording the players and their roles that are designated as the team's three potential contributors to their Four Factor deficiencies, we can use this information to narrow down the player pool that the designated team will be taking player suggestions from. This is because we want to make sure that any players we suggest would be playing similar roles as the players labeled as potential negative impacts on the team's Four Factors. For the coding side of this, we can make a copy of our dataframe of players and get rid of any players whose role does not match any of the roles from our identified players.

```
### make a temp copy of concat_4_defense
temp_concat = concat_4_defense

# get rid of all players with irrelevant roles
for index, row in temp_concat.iterrows():
    if len(list_weak_roles) == 3:
        if row['Role'] != list_weak_roles[0] and row['Role'] != list_weak_roles[1] and row['Role'] != \
                list_weak_roles[2]:
            temp_concat = temp_concat.drop(index)
    elif len(list_weak_roles) == 2:
        if row['Role'] != list_weak_roles[0] and row['Role'] != list_weak_roles[1]:
            temp_concat = temp_concat.drop(index)
    else:
        if row['Role'] != list_weak_roles[0]:
            temp_concat = temp_concat.drop(index)
```

Once we have our player pool that we want to pull player suggestions from, we can do the same process of assigning weights to players except we will now be giving weights based on the best players of the relevant stats (the opposite of how we determined the bottom players from the designated team's roster). We do this by sorting each player by the relevant stats, and assigning weights to the top ten players of the stat based on their percentile ranking. The number of top ten players is a decision we made just so that the players suggested were limited and that not every player was being suggested for the given team, however, when reproducing the project, the process of assigning weights to players can be adjusted to the project's liking.

```
### create list to hold roster tallies
fa_counts = []

### gather tallies for best ten players of each important stat, append to fa_count
for pair in tuple_list:
    i = 0
    temp_concat = temp_concat.sort_values(by=pair[0], ascending=False)
    for index, player in temp_concat.iterrows():
        if i < 10:
            i += 1
            player_count = (player['Name'], pair[1])
            fa_counts.append(player_count)
```

Once we assign the player weights, we can then tally the totals up like we did for the weights of the players on the team roster. After adding up all the weight totals, we can then get rid of any players with a weight total of zero since this means that they did not end up in the top ten players of any of the related stats to the factors we are trying to improve. We decided to keep any players that ended up with a weight tally above zero, as it shows they could help the team in some way no matter how small. But the players were still sorted in the order of their weights. Now that we have our final list of players that we want to suggest to the designated team, we then separated the players into their contract groupings. The three contract groupings were: Upcoming Free Agents/Two-Way players (players who will be potentially available after this season or are on G-League contracts), Future Free Agents (players who will be potentially available after next season), and Multiyear Players (players who are currently on contract for the next 3+ years).

Once we have made our three separate lists of player suggestions, we can then turn those lists of players into dataframes with their stats and percentiles. To end the function we then return the three dataframes of player suggestions, the players that are labeled as potential contributors to their team's deficiencies in the Four Factors, and the stat column names that are related to the team's weaknesses.

```
return upcoming_fa_df, future_fa_df, multiyear_df, list_weak_players, th_list
```

3.6 DB Browser/SQL Database:

Being that our goal was to make a web-app, this meant that we would most likely be working with a database for our project as well. The plan was to have a database that would hold all the stats for the players relevant to our project and its design. We later figured out that our web-app could have been designed without the need of a database, however we had already begun the use of a SQL database for the web-app and the project. The way that we were able to create our database was through the program DB Browser.

The reason that we chose to use DB Browser was because it made creating a database through csv files extremely easy. For all the stats and dataframes that we have mentioned in the methods previously, we created csv files for each of the important and finalized information, which we could then easily input into a SQL database through DB Browser. This is the process we used to create our final database for the web-app.

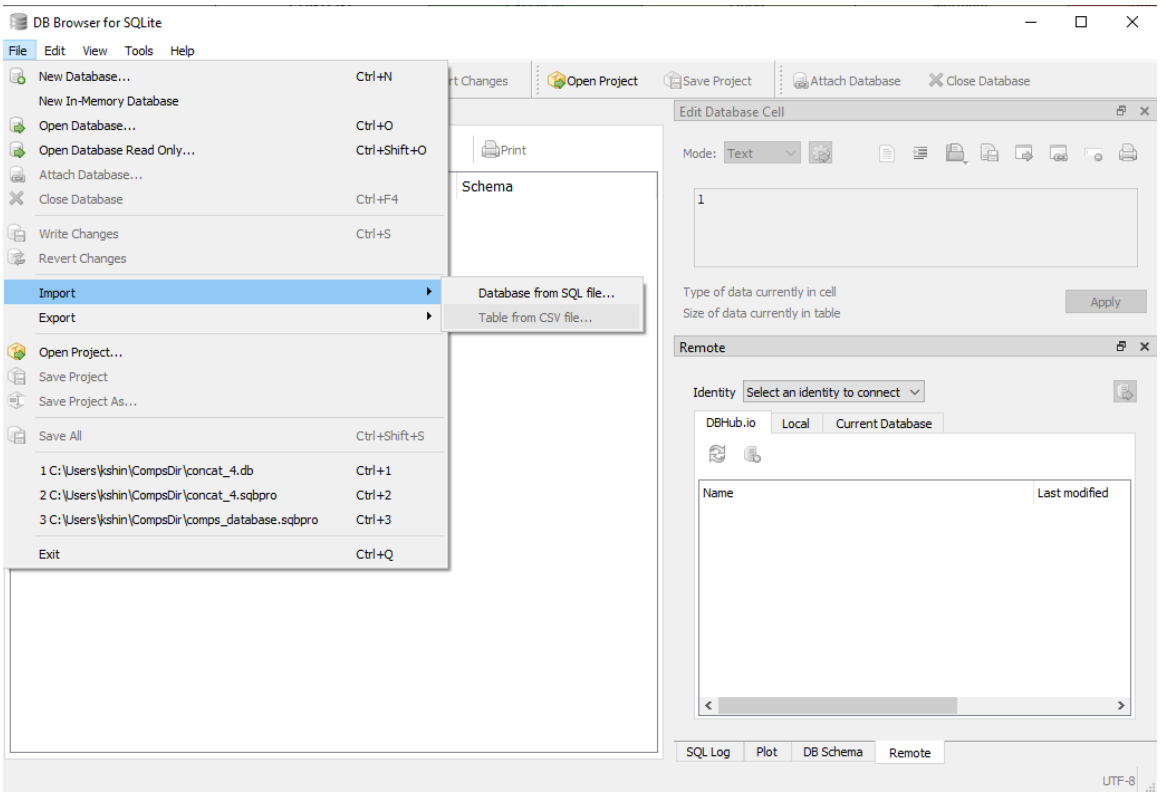


Figure 9: Input table through csv file

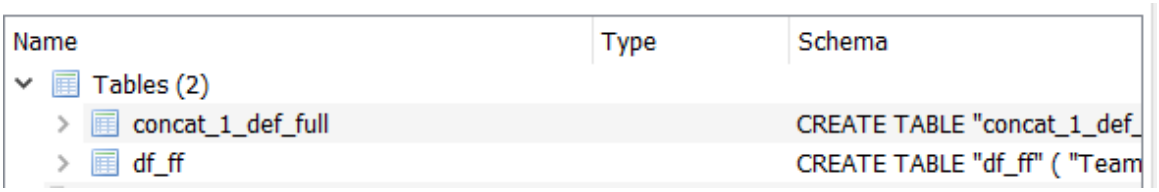


Figure 10: Database Tables

Name	Age_x	Games	GS	Min	FGMRank	FGM	FGARank	FGA	FG%Rank	FG%	3ptMRank	3pt	3ptARank	3ptA	3pt%Rank	3pt%	2ptMRank	2pt	2ptARank	2ptA	2pt%Rank	2pt%	EFG%Ran
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
Trae Young	23.0	102.0	102.0	34.42	0.53	8.32	0.71	18.93	0.12	0.44	0.59	2.37	0.62	6.84	0.29	0.35	0.47	5.95	0.53	12.09	0.15	0.49	0.18
John Collins	24.0	104.0	104.0	30.43	0.93	6.5	0.97	11.62	0.37	0.56	0.97	1.24	0.93	3.16	0.97	0.39	0.8	5.26	0.8	8.46	0.57	0.62	0.63
Clint Capela	27.0	104.0	104.0	30.31	0.87	5.91	0.9	10.0	0.47	0.59	0.17	0.0	0.13	0.01	0.13	0.0	0.9	5.91	0.93	9.99	0.27	0.59	0.37
Kevin Huerter	23.0	109.0	71.0	29.9	0.32	4.42	0.33	10.07	0.65	0.44	0.4	1.89	0.42	5.24	0.35	0.36	0.26	2.53	0.25	4.83	0.82	0.52	0.61
Cam Reddish	22.0	51.0	21.0	25.76	0.98	3.88	0.99	9.8	0.32	0.4	0.91	1.49	0.94	4.49	0.41	0.33	0.98	2.39	0.99	5.31	0.21	0.45	0.18
Delon Wright	29.0	84.0	39.0	24.26	0.04	3.12	0.02	6.87	0.67	0.45	0.05	0.83	0.05	2.26	0.53	0.37	0.23	2.29	0.21	4.61	0.68	0.5	0.39
Bogdan Bogdanović	29.0	82.0	65.0	30.09	0.58	5.52	0.58	12.35	0.68	0.45	0.84	2.88	0.81	7.15	0.81	0.4	0.33	2.65	0.3	5.21	0.74	0.51	0.81
Danilo Gallinari	33.0	89.0	4.0	23.43	0.43	3.74	0.45	8.85	0.09	0.42	0.8	1.84	0.7	4.61	0.91	0.4	0.14	1.9	0.25	4.25	0.07	0.45	0.45
Gorgui Dieng	32.0	58.0	1.0	12.34	0.16	1.74	0.24	3.52	0.31	0.5	0.8	0.59	0.73	1.5	0.87	0.39	0.07	1.16	0.07	2.02	0.44	0.57	0.6

Figure 11: Example of Table contents

3.7 Web-App/Flask:

With our back-end side of our web-app finished, we can now move on to the front-end of our web-app and create a finished product that shows our player suggestions for every team. To create our web-app, we used Flask and Python along with the SQL database mentioned before that we made through DB Browser.

The first step to making our web-app is to initialize it through Flask which includes installing all of the necessary packages. The packages we used to make our web-app are listed in the following figure.

```
from flask import Flask, render_template
import sqlite3
```

We can then initialize our Flask web-app using the code below. Once we have done this, we can move on to making the pages for our web-app.

```
app = Flask(__name__)
```

However, before we move on to the details of how the pages of the web-app were made, we must be able to view our web-app in a browser so we are able to see the changes we make to our web-app in real time. There are a few easy steps to running a Flask web-app on a local computer. The first step is making sure that we have Flask installed on our computer. We can do this in any Command Prompt or the Terminal on our computer. Once we have Flask installed, we must then go to the directory on our computer holding the Python file of the web-app.

```
(base) C:\Users\kshin\CompsDir>
```

Now that we are in the directory holding the web-app's Python file, we must set our Flask app to the Python file holding the code for our web-app.

```
(base) C:\Users\kshin\CompsDir>SET FLASK_APP=webapp.py
```

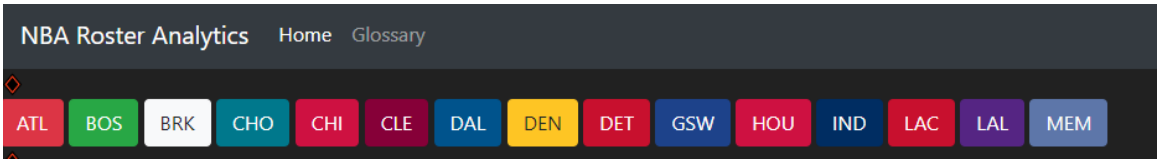
After setting the Flask app to our correct Python file, if we want to see real time changes to our web-app as we code them, we can set our Flask environment to development.

```
(base) C:\Users\kshin\CompsDir>SET FLASK_ENV=development
```

Once we have done these steps, our web-app should be ready to run. Use flask run to run the web-app, and we can view our web-app in our web browser at the address listed.

```
(base) C:\Users\kshin\CompsDir>flask run
 * Serving Flask app "webapp.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 228-114-250
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

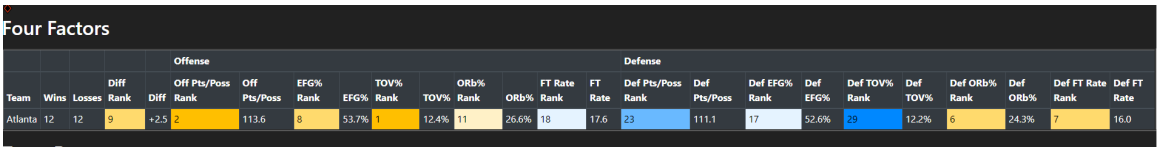
To make the pages of our web-app, we will need to use html files for each of the pages we want to make. A good practice to use when making a web-app is to have a single base html file that holds the general layout of the web-app as well as any formatting information. Our base file keeps track of numerous crucial details important to every page of our web-app. The information that the base file holds are the functions used when creating the tables in our web-app as well as formatting information like the stylesheet. It is also in charge of creating the navigation bar for each page and the buttons that navigate to the different pages for each NBA team. Each of our pages on our web-app use the base html file and extends from it.



For the purpose of simplifying the paper, we will view each section of a team's page and go over how each section was created. The code behind how each page is created through Flask is shown below. We will come back to the code by going through each section of our web-page and reference the web page code in explaining how each section was made.

```
@app.route('/atl')
def atl_page():
    conn = sqlite3.connect('concat_4.db')
    conn.row_factory = sqlite3.Row

    cur = conn.cursor()
    cur.execute("select * from concat_1_def_full where Team_x = 'ATL'")
    bos = cur.fetchall()

    cur.execute("select * from df_ff where Team = 'Atlanta'")
    df = cur.fetchall()

    upcoming_fa, future_fa, multiyear_fa, list_names, list_th = copy_comps_algo.suggest_players('Atlanta', 'ATL')

    return render_template("atl.html", BOS_=_bos, DF_=_df, weak_names_=_list_names, th_list_=_list_th,
                           up_row_data_=_list(upcoming_fa.values.tolist()),
                           fut_row_data_=_list(future_fa.values.tolist()),
                           my_row_data_=_list(multiyear_fa.values.tolist()))
```

The first section of a team's web page is their Four Factors table pictured below.



This table was created using our SQL database created earlier through DB Browser. The code for our web page connects to our created database, and then looks for the row of stats in the Four Factors table that matches the team we are making a page for (in this case the team is the Atlanta Hawks).

```
cur.execute("select * from df_ff where Team = 'Atlanta'")
df = cur.fetchall()
```

We then have an html file for the Atlanta Hawks page that extends from the base html file. It is in this html file for our Atlanta Hawks page that we create our Four Factors table. However, the table only holds the cell and header information. It is in our base file where we have written functions that change the background color of our cells holding the league rankings of the team's Four Factors. For the purposes of the paper, we will not go into great detail on these functions, but it essentially uses javascript to create if statements that will change the background of the cells depending on the cell's value.

We repeat a similar process to this in our roster table that is pictured below.

Team Roster

Name	Age	Games	GS	MIN	FGM	FGM Rank	FGA	FGA Rank	FG%	FG% Rank	3ptM	3ptM Rank	3ptA	3ptA Rank	3pt%	3pt% Rank	2ptM	2ptM Rank	2ptA	2ptA Rank	2pt%	2pt% Rank	EFG%	EFG% Rank	FTM	FTM Rank	FTA	FTA Rank	FT%	FT% Rank	ORb	ORb Rank	DRb	DRb Rank	TRb	TRb Rank	AST	AST Rank	STL	STL Rank	BLK	BLK Rank	TO	TO Rank
Trae Young	23.0	102.0	102.0	34.42	0.53	8.32	0.71	18.93	0.12	0.44	0.59	2.37	0.62	6.84	0.29	0.35	0.47	5.95	0.53	12.09	0.15	0.49	0.18	0.5	1.0	7.04	0.97	7.96	0.82	0.88	0.18	0.59	0.09	3.12	0.09	3.71	0.94	9.37	0.29	0.91	0.09	0.15	0.12	4.0
John Collins	24.0	104.0	104.0	30.43	0.93	6.5	0.97	11.62	0.37	0.56	0.97	1.24	0.93	3.16	0.97	0.39	0.8	5.26	0.8	8.46	0.57	0.62	0.63	0.61	0.87	2.5	0.77	3.06	0.97	0.82	0.17	1.85	0.63	5.87	0.57	7.71	0.6	1.38	0.47	0.58	0.5	1.03	0.5	1.2
Clint Capela	27.0	104.0	104.0	30.31	0.87	5.91	0.9	10.0	0.47	0.59	0.17	0.0	0.13	0.01	0.13	0.0	0.9	5.91	0.93	9.99	0.27	0.59	0.37	0.59	0.53	1.68	0.8	3.09	0.17	0.55	1.0	4.32	0.97	9.06	0.97	13.38	0.23	0.91	0.77	0.74	0.87	1.73	0.73	1.6
Kevin Huerter	23.0	109.0	71.0	29.9	0.32	4.42	0.33	10.07	0.65	0.44	0.4	1.89	0.42	5.24	0.35	0.36	0.26	2.53	0.25	4.83	0.82	0.52	0.61	0.53	0.05	0.64	0.05	0.83	0.14	0.77	0.46	0.51	0.39	2.92	0.39	3.43	0.37	3.15	0.54	0.95	0.58	0.36	0.84	1.1
Cam Reddish	22.0	51.0	21.0	25.76	0.98	3.88	0.99	9.8	0.32	0.4	0.91	1.49	0.94	4.49	0.41	0.33	0.98	2.39	0.99	5.31	0.21	0.45	0.18	0.47	0.99	2.2	0.99	2.61	0.87	0.84	0.73	0.8	0.77	2.67	0.77	3.47	0.77	1.2	0.99	1.22	0.61	0.29	0.01	1.3
Delon Wright	29.0	84.0	39.0	24.26	0.04	3.12	0.02	6.87	0.67	0.45	0.05	0.83	0.05	2.26	0.53	0.37	0.23	2.29	0.21	4.61	0.68	0.5	0.39	0.51	0.28	1.36	0.28	1.7	0.35	0.8	0.96	0.99	0.4	2.93	0.61	3.92	0.53	3.82	0.88	1.38	0.65	0.38	0.88	1.1
Bogdan Bogdanović	29.0	82.0	65.0	30.09	0.58	5.52	0.58	12.35	0.68	0.45	0.84	2.88	0.81	7.15	0.81	0.4	0.33	2.65	0.3	5.21	0.74	0.51	0.81	0.56	0.11	0.77	0.09	0.91	0.58	0.84	0.33	0.46	0.6	3.27	0.6	3.73	0.3	3.02	0.72	1.11	0.42	0.27	0.79	1.2
Danilo Gallinari	33.0	89.0	4.0	23.43	0.43	3.74	0.45	8.85	0.09	0.42	0.8	1.84	0.7	4.61	0.91	0.4	0.14	1.9	0.25	4.25	0.07	0.45	0.45	0.53	0.89	2.76	0.84	2.98	1.0	0.93	0.02	0.42	0.36	3.63	0.23	4.04	0.2	1.3	0.18	0.52	0.14	0.19	0.98	0.7
Gorgui Dieng	32.0	58.0	1.0	12.34	0.16	1.74	0.24	3.52	0.31	0.5	0.8	0.59	0.73	1.5	0.87	0.39	0.07	1.16	0.07	2.02	0.44	0.57	0.6	0.58	0.6	1.16	0.38	1.4	0.93	0.83	0.18	1.03	0.22	2.53	0.18	3.57	0.64	1.07	0.67	0.53	0.18	0.36	0.6	0.7
Lou Williams	35.0	102.0	6.0	19.0	0.11	3.51	0.11	8.5	0.26	0.41	0.07	0.92	0.07	2.3	0.77	0.4	0.28	2.59	0.53	6.2	0.09	0.42	0.09	0.47	0.47	1.93	0.44	2.2	0.84	0.88	0.04	0.27	0.02	1.6	0.02	1.87	0.28	2.83	0.21	0.67	0.07	0.09	0.7	1.4
Timothé Luwawu-Cabarrot	26.0	81.0	9.0	14.46	0.48	1.77	0.65	4.88	0.1	0.36	0.76	1.04	0.8	3.28	0.38	0.32	0.28	0.73	0.35	1.59	0.24	0.46	0.21	0.47	0.44	0.49	0.39	0.6	0.79	0.82	0.26	0.35	0.3	1.46	0.29	1.8	0.57	0.94	0.5	0.43	0.11	0.09	0.45	0.5
Solomon Hill	30.0	97.0	20.0	18.48	0.01	1.2	0.01	3.53	0.01	0.34	0.16	0.75	0.21	2.51	0.06	0.3	0.01	0.44	0.02	1.02	0.15	0.43	0.1	0.45	0.07	0.38	0.05	0.52	0.17	0.74	0.68	0.53	0.43	2.1	0.48	2.63	0.05	0.92	0.26	0.58	0.21	0.14	1.0	0.4
De'Andre Hunter	24.0	39.0	35.0	29.13	1.0	4.82	1.0	10.36	0.83	0.47	0.88	1.33	0.89	3.82	0.6	0.35	1.0	3.49	1.0	6.54	0.54	0.53	0.61	0.53	1.0	2.28	1.0	2.85	0.77	0.8	0.57	0.59	0.94	3.51	0.91	4.1	0.84	1.38	0.82	0.69	0.88	0.46	0.04	1.2

For our team roster we also use our database created previously to read in the stat information for the players on the team. The same goes for how the Team Roster table is created in the html file along with using a function in the base html file to color the backgrounds of the player percentile rankings based on the cell's value. The main difference between our Team Roster and the Four Factors table is that we also have highlighted the columns of stats that relate to the team's Four Factor weaknesses, as well as highlighting the players that are designated as the three players who rank lowest in their percentiles for the highlighted columns of stats. We are able to do this by running our python program that suggests players and saving the information of the columns used as well as the players highlighted from the Team Roster.

To be able to do this we must first make sure that we import the python file that runs the program suggesting players, as well as making sure that the file is located in the same directory as our file containing our web-app.

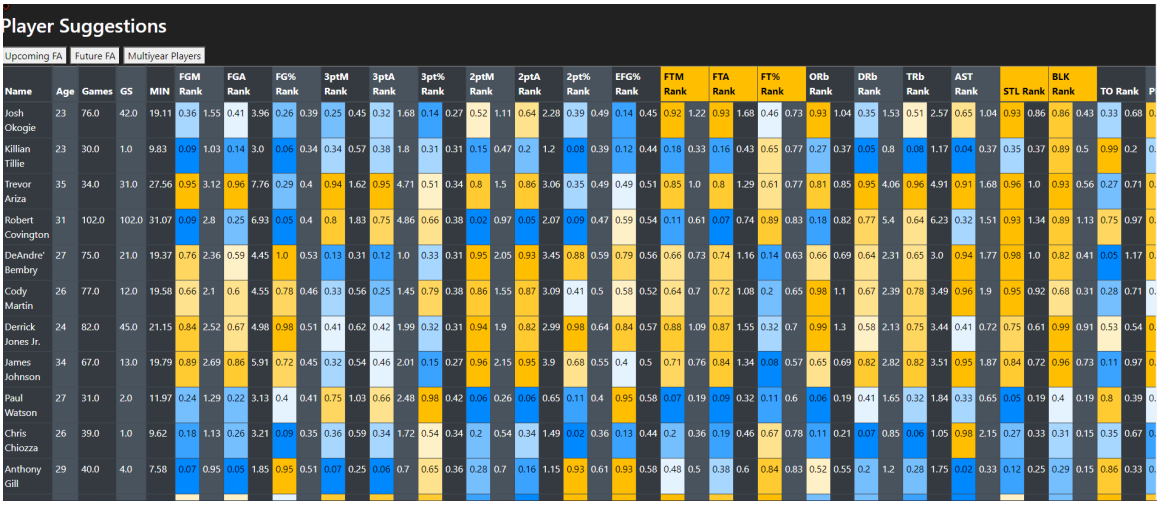
```
import copy_comps_algo
```

Once we have the correct file imported, we can then use the functions of that file (in this case, the entire Python file is wrapped in a function titled suggest_players) to get the values of the columns used and highlighted players from the team roster.

```
upcoming_fa, future_fa, multiyear_fa, list_names, list_th = copy_comps_algo.suggest_players('Atlanta', 'ATL')
```

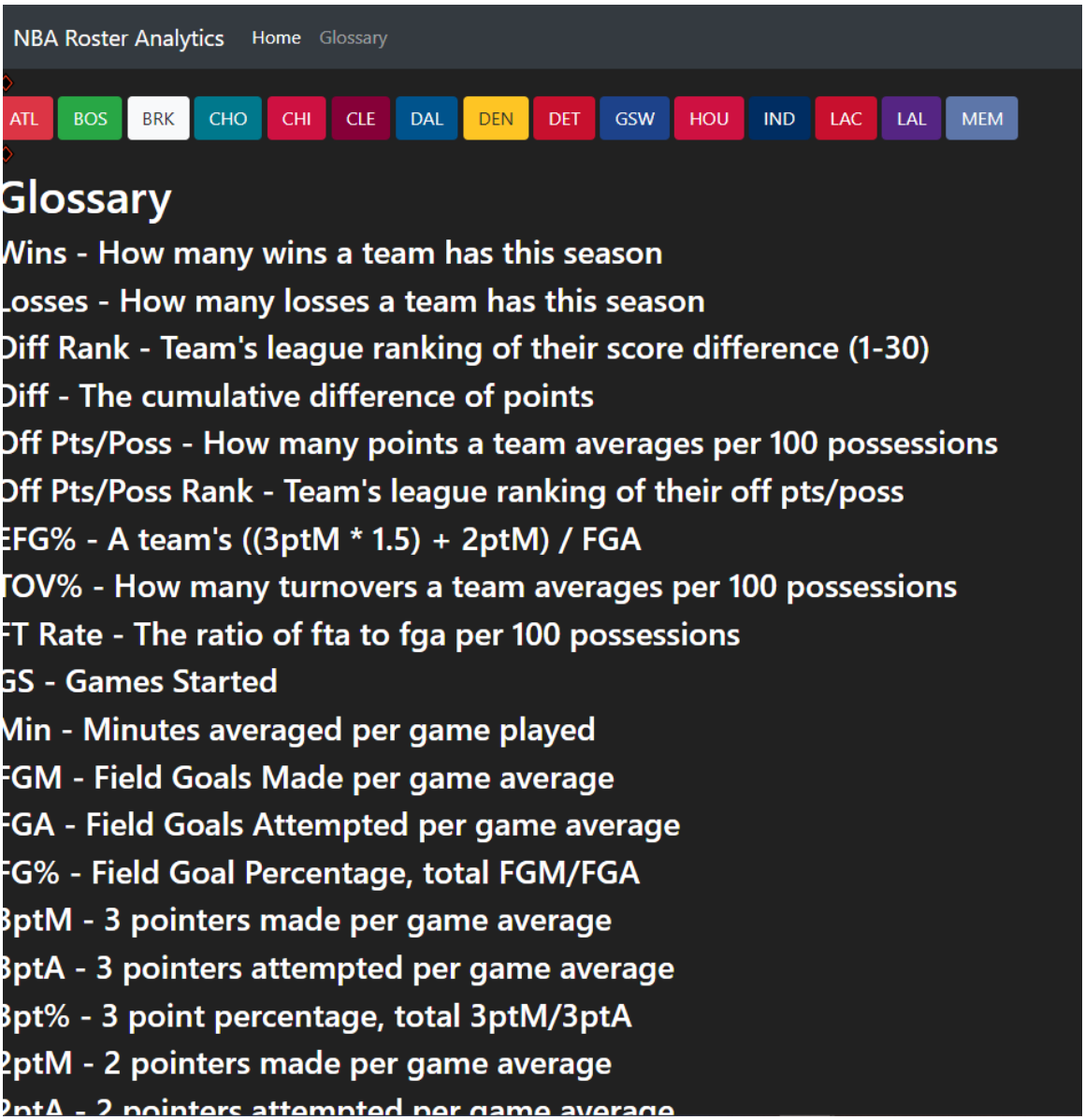
We then use this information when creating our Team Roster table so that when the value of a cell matches either the column name or a player name which was highlighted in our suggesting players file, that cell is then highlighted on our web-app.

Lastly, we created our three tables for the player suggestions of the designated team. The process of creating these tables is similar to how we created our Team Roster table, being that these tables also had the same highlighted stat columns. The stats for each of the player suggestion tables come from our values saved from running the player suggestion python file that we also received the highlighted column values from. These values can be seen in the figure above. The process of creating these tables are then repeated for each of the NBA teams.



Name	Age	Games	GS	MIN	FGM Rank		FGA Rank		FG% Rank		3ptM Rank		3ptA Rank		3pt% Rank		2ptM Rank		2ptA Rank		2pt% Rank		EFG% Rank		FTM Rank		FTA Rank		FT% Rank		ORb Rank		DRb Rank		TRb Rank		AST Rank		STL Rank		BLK Rank		TO Rank	
Josh Okogie	23	76.0	42.0	19.11	0.36	1.55	0.41	3.96	0.26	0.39	0.25	0.45	0.32	1.68	0.14	0.27	0.52	1.11	0.64	2.28	0.39	0.49	0.14	0.45	0.92	1.22	0.93	1.68	0.46	0.73	0.93	1.04	0.35	1.53	0.51	2.57	0.65	1.04	0.93	0.86	0.86	0.43	0.33	0.68	0.
Killian Tillie	23	30.0	1.0	9.83	0.09	1.03	0.14	3.0	0.06	0.34	0.34	0.57	0.38	1.8	0.31	0.31	0.15	0.47	0.2	1.2	0.08	0.39	0.12	0.44	0.18	0.33	0.16	0.43	0.65	0.77	0.27	0.37	0.05	0.8	0.08	1.17	0.04	0.37	0.35	0.37	0.89	0.5	0.99	0.2	0.
Trevor Ariza	35	34.0	31.0	27.56	0.95	3.12	0.96	7.76	0.29	0.4	0.94	1.62	0.95	4.71	0.51	0.34	0.8	1.5	0.86	3.06	0.35	0.49	0.49	0.51	0.85	1.0	0.8	1.29	0.61	0.77	0.81	0.85	0.95	4.06	0.96	4.91	0.91	1.68	0.96	1.0	0.93	0.56	0.27	0.71	0.
Robert Covington	31	102.0	102.0	31.07	0.09	2.8	0.25	6.93	0.05	0.4	0.8	1.83	0.75	4.86	0.66	0.38	0.02	0.97	0.05	2.07	0.09	0.47	0.59	0.54	0.11	0.61	0.07	0.74	0.89	0.83	0.18	0.82	0.77	5.4	0.64	6.23	0.32	1.51	0.93	1.34	0.89	1.13	0.75	0.97	0.
DeAndre' Bembry	27	75.0	21.0	19.37	0.76	2.36	0.59	4.45	1.0	0.53	0.13	0.31	0.12	1.0	0.33	0.31	0.95	2.05	0.93	3.45	0.88	0.59	0.79	0.56	0.66	0.73	0.74	1.16	0.14	0.63	0.66	0.69	0.64	2.31	0.65	3.0	0.94	1.77	0.98	1.0	0.82	0.41	0.05	1.17	0.
Cody Martin	26	77.0	12.0	19.58	0.66	2.1	0.6	4.55	0.78	0.46	0.33	0.56	0.25	1.45	0.79	0.38	0.86	1.55	0.87	3.09	0.41	0.5	0.58	0.52	0.64	0.7	0.72	1.08	0.2	0.65	0.98	1.1	0.67	2.39	0.78	3.49	0.96	1.9	0.95	0.92	0.68	0.31	0.28	0.71	0.
Derrick Jones Jr.	24	82.0	45.0	21.15	0.84	2.52	0.67	4.98	0.98	0.51	0.41	0.62	0.42	1.99	0.32	0.31	0.94	1.9	0.82	2.99	0.98	0.64	0.84	0.57	0.88	1.09	0.87	1.55	0.32	0.7	0.99	1.3	0.58	2.13	0.75	3.44	0.41	0.72	0.75	0.61	0.99	0.91	0.53	0.54	0.
James Johnson	34	67.0	13.0	19.79	0.89	2.69	0.86	5.91	0.72	0.45	0.32	0.54	0.46	2.01	0.15	0.27	0.96	2.15	0.95	3.9	0.68	0.55	0.4	0.5	0.71	0.76	0.84	1.34	0.09	0.57	0.65	0.69	0.82	2.82	0.82	3.51	0.95	1.87	0.84	0.72	0.96	0.73	0.11	0.97	0.
Paul Watson	27	31.0	2.0	11.97	0.24	1.29	0.22	3.13	0.4	0.41	0.75	1.03	0.66	2.48	0.98	0.42	0.06	0.26	0.06	0.65	0.11	0.4	0.95	0.58	0.07	0.19	0.09	0.32	0.11	0.6	0.06	0.19	0.41	1.65	0.32	1.84	0.33	0.65	0.05	0.19	0.4	0.19	0.8	0.39	0.
Chris Chiozza	26	39.0	1.0	9.62	0.18	1.13	0.26	3.21	0.09	0.35	0.36	0.59	0.34	1.72	0.54	0.34	0.2	0.54	0.34	1.49	0.02	0.36	0.13	0.44	0.2	0.36	0.19	0.46	0.67	0.78	0.11	0.21	0.07	0.85	0.06	1.05	0.98	2.15	0.27	0.33	0.31	0.15	0.35	0.67	0.
Anthony Gill	29	40.0	4.0	7.58	0.07	0.95	0.05	1.85	0.95	0.51	0.07	0.25	0.06	0.7	0.65	0.36	0.28	0.7	0.16	1.15	0.93	0.61	0.93	0.58	0.48	0.5	0.38	0.6	0.84	0.83	0.52	0.55	0.2	1.2	0.28	1.75	0.02	0.33	0.12	0.25	0.29	0.15	0.86	0.33	0.

The last page made for our web-app that is not an NBA team page is our glossary page where we list the stat terms used in our web-app along with the meaning of them. This is so that anyone who is unfamiliar with any of the terms on our web-page can view the glossary page and understand its meaning.



3.8 Additional Comments:

To conclude the methods section, it is important to touch upon the flaws in our process of making the web-app and if we were to do the project over again or had more time, certain aspects would be changed or included. To start, one important factor that is not taken into consideration when we are making the player suggestions are team salary caps as well as the contract information of the suggested players. This is crucial information that is missing from our process as even if a player is able to help a team, it will not matter unless that team is able to afford them. Other information that is missing from our process of suggesting players is that we could have collected more stats that are relevant to the Four Factors. For example, the main defensive stat that we used for Defensive TOV% were the players' steals counts. However, there are multiple ways to cause a turnover that were not taken into account. Stats that relate to causing turnovers are drawing charges or offensive fouls, or deflections from passes as more deflections usually leads to more steals. There are also shot clock violations that result in a team turnover when the offense does not shoot the ball before the shot clock expires. It is because of this that if the project were to be done again, many more stats would be taken into

account for the player suggestions and it would lead to more accurate results.

Another improvement we would make to the project with more time is to make the web-app self updating. Currently all the information is turned into csv files that we input into a database, which is a hassle because of the fact that this needs to be done manually. If we were to redo the project, we would figure out a way to just display the data without the need of a database, or a database that self updates at the end of each day to stay up to date with the current team standings and stats.

In addition to the previous improvements we have mentioned, with more time would have also liked to add an in-depth description of each of the features of our web-app on its home page. Currently our home page is blank, which leaves a lot of space for us to use in creating a road map through the layout of our NBA team pages and what each part of the web-app represents.

Lastly, with more time our project would have incorporated ways for the web-app to be more interactive for the user, such as being able to change the filter of clusters or weights like we have mentioned earlier in this methods section. Hopefully with more time, one day our project will be able to incorporate these changes that we have mentioned to end our methods section.

4 Evaluation:

4.1 Evaluation Metrics:

When evaluating our project, the most important things to keep in mind are our project's goal, and the target audience of our project. We want to make sure that we accomplish our project's goal but also be sure that our end result of our web-app is useful for the audience we are targeting. Of course the first benchmark we want our project to meet is our goal. Did we accomplish what we set out to do, which is create a web-app that uses data analytics to identify NBA teams' strengths and weaknesses, and suggest possible roster changes to improve the roster constructions of those teams? Once we determine if we've accomplished our goal, it is important to also make sure that we are receiving feedback from our targeted audience on the end results of our project as well.

In order to do this we must first establish the audience that we are targeting. Being that our project is a web-app made for analyzing NBA teams and their rosters, the audience that makes the most sense for the project would be NBA executives and general managers who have the authority over roster changing decisions. However, due to the limitations of our network, NBA executives are not the easiest audience to reach in terms of getting evaluation feedback. Thus, the next best audience we can reach out to for feedback on our project are coaches who have experience in crafting rosters as well as making roster changing decisions. The reason being for this is that coaches are a lot easier to receive feedback from when compared to NBA executives, as well as they still have had to make decisions for their team's roster construction which could result in relative feedback for our project. The coaches that we reached out to had backgrounds ranging from High School Varsity Basketball experience, to Division III Collegiate Basketball Coaching experience. When meeting with these coaches they were asked to fill out a Google Form for feedback on the web-app, as well as us taking notes on the conversion during each meeting.

The questions asked on the Google Form for each meeting were:

- Do you think this project could help a team/general manager make better roster decisions?

- Do you feel that the players recommended to teams are reasonable players to suggest?
- What are your thoughts on the process behind how players are suggested?
- What are your main concerns for the project?
- Do you have any additional comments?

These questions were asked in order to receive definitive answers resulting in a positive or negative statement towards the web-app, as well as constructive feedback on how the web-app could be improved or to reveal any flaws the evaluators discovered. By doing this we could get an accurate picture on the standing of our project from the viewpoint of coaches with roster making decision experience, and make any future adjustments to the project accordingly.

There are other ways that this project could be evaluated that were not explored, but could generate equally interesting results. One other form of evaluation would be a change in audience, as there are Data Analysts with work experience in the sports field that this project to be presented to for feedback as they would have real-world experience in what equates to a productive data analysis project. Another form of evaluation that would be beneficial for the project that we hope to explore in the future is the use of historical data. We could apply the same analysis we used for our project on previous season's data and see which players were suggested for teams in the past. By doing this we could then observe the differences between the suggested players our program chooses compared to the actual free agency decisions made by NBA teams in the past. After doing this there could be a possible way of measuring the differences between picking the players from our programs suggestions and the actual players chosen by a team in free agency to determine the usefulness of our web-app.

4.2 Evaluation Results:

When looking at our project and its results, the first question we want to answer is if we were able to accomplish our project's goal. In a sense, the answer to this is yes being that we had a functioning web-app that would suggest players from around the NBA that could possibly help an NBA team improve the factors in the Four Factors model that they were deficient in. However, like we have previously stated, it is important to receive feedback from our target audience as they will be the best indicators on our projects success and the viability of our web-app.

To get evaluation results we met with numerous basketball coaches about the project and the web-app that we developed, and asked them questions about their initial reactions and opinions after viewing the web-app. First off, the two beginning questions asked of our coaches are similar to the goal of our project: Do you think this project could help a team/general manager make better roster decisions and do you feel that the players recommended to teams are reasonable players to suggest? All the coaches in some shape or form answered yes to both of these questions, although they had their concerns for the project as well. Multiple coaches felt that the project had potential to help an NBA team, but it depended on the team's situation. As for how they felt about the selection process for suggesting players, all coaches felt there was a logical route taken to suggest the players based off of the designated team's Four Factors, however some coaches brought up concerns on the impact suggested players would make depending on their role, or if their replacing a player would impact the team negatively in the factors they are excelling at. Screenshots of coaches replies to the feedback form can be viewed below.

Yes I think it could, especially teams with players that are drastically bringing down certain areas of their four factors.

It has the potential to depending on the team.

Yes, I think in the right circumstances this could help out a team a lot in being more informed for their free agency decisions.

Figure 12: Coaches responses to question: Do you think this project could help a team/general manager?

Yes, they seem to be reasonable players depending on the specific situations. Of course, there are also situations where the players being suggested or the players being replaced aren't always playing a large role for the team. So the changes might only have a minimal impact.

Figure 13: Coach response to question: Do you feel that the players recommended to teams are reasonable players to suggest?

I think there are cases where certain players who are weak in four factors will be labeled as weaknesses but when they are replaced, could take away from how good a team is in their strengths for other factors of the four factors model.

Figure 14: Coach response to question: What are you main concerns for this project?

I'm impressed with the project and I am glad that it help synthesize a lot of information that could be helpful for NBA team general managers and puts the info into one page.

Figure 15: A coach's overall impression of the project

Based off of the feedback from the numerous coaches we had evaluate our project, it is safe to say that it had a positive overall first impression, however there are many improvements to still be made (as any first draft of a website will have). However, we are still glad to have gotten the opportunity to meet with many basketball minds on the topic of the web-app we created and the feedback received through these evaluations will greatly improve the quality and accuracy of the web-app when future revisions are made.

Lastly, we touched upon the topic of using historical data to evaluate our project in the previous section of Evaluation Metrics, but unfortunately did not have the time to do so given the time frame of the project. We hope to use this tactic to possibly evaluate a future version of the web-app where we take into consideration the numerous suggestions for improvement in the web-app and develop a refined version.

5 Ethical Concerns:

5.1 Effect on Players:

One of the main ethical considerations for this project comes from the main objective of what our project's goal is: identifying weaknesses or deficiencies in NBA teams and how to improve them. By doing this, our program will be clearly pointing out players that will be identified as possible weaknesses to a team or below average at their job compared to other players in that same role around the league. This is an ethical situation that we need to consider because our program could have potential effects on NBA players and how they are viewed, along with how these viewpoints could affect their mental health and their job security.

The mental health of the athletes that our program is analyzing must be highly considered, especially with the drastic effects that could potentially come along with a player feeling like they are under performing or failing at their job. Studies have shown that there is a prevalence of failure-based depression among elite athletes that the NBA players we are analyzing could also be susceptible to [9]. Being that our program targets areas and players on each team that the organizations can possibly improve upon, we must be cognizant of how the message is relayed. Just because a player is a potential cause of why a team is below average in a certain factor, this does not mean that they are not good in other areas, as well as potentially being beneficial for a different NBA team.

Another important factor to consider with how this program could affect NBA players is in the power aspect. A big ethical concern when it comes to data analytics and its usage is the fact that most analytics are aimed at helping the large corporations (in this case NBA Teams) while usually at the cost of their workers (in this case, NBA players). This is the same for our program as the purpose of our web-app is to improve NBA rosters, thus making NBA teams the main people gaining power, and players who are deemed as "under performing in their role" or causes for why a team could be below average in a certain factor will end up being the potential losers of this program. This could end up with players being cut from their teams, or being traded, or in general jeopardizing their jobs. It is because of these concerns that we need to make sure that our program and the way that it is implemented does not have any errors, as well as restating the fact that data analytics are meant to be purely used as an extra tool in the decision making process, and not the final factor determining the outcomes of who does and who does not end up on a team's final roster.

With the way that our web-app is currently constructed and the original purpose of what the web-app was intended to do, there are not many ways to implement features that would in turn benefit the players instead. However, one way the program could benefit the players are the players that are labeled as strengths to a team. And as for the players that are labeled as potential areas of improvement for a team's roster, the program would still be able to inform those players on areas that they could improve upon. A player under performing could take it upon themselves to be the answer to their team's needs. A possible future feature that could be added to identify players as strengths to a team could be implemented in the future where it highlights players who are excelling in their roles along with the players who could be areas of improvement for a team.

Overall, data analytics in sports tend to have a dehumanizing effect where fans and those of us outside of the industry will view the stats produced by players as a summary of who they are. There is a habit of thinking about these players as numbers on a sheet or another chess piece ready to be sacrificed for the sake of winning the game. Yago Colás in his article "The Culture of Moving Dots, Toward a History of Counting and of What Counts in Basketball" does a great job of touching upon the dehumanizing tendencies of data analytics on the athlete in sports, as well as how most decisions made through these analytics heavily favor the financial interests of the team owners[8].

5.2 Data Concerns:

Another important ethical consideration that we need to keep in mind when creating our web-app is how the data we use influences the outcome, as well as gathering our data in an ethical way. First off, there are many conversations that go on about the ethics and practices of web-scraping [2]. A common consensus of agreement among the data community is that if the data you need can be found through an API, then the best practice would be to use the API instead of web-scraping. In our case for the project, there is an NBA API that holds all the stats that we end up using in our project, however we found the NBA API to be inconsistent and had troubles connecting to the data most likely due to the headers we mentioned previously when web-scraping the NBA stats site. While web-scraping, we also have to be aware of the policies behind the websites that we are scraping. Luckily, the use of web-scraping and sharing stats is prevalent in the sports community, as long as we are giving credit to the websites and places that our project is scraping the stats from.

Other ethical concerns about the use of data in our project is how we could not have enough data relative to what we want our project to do, along with the fact that our project is completely stats based and won't be incorporating the intangible features of players. We touched upon this topic a little when commenting on improvements that the project could make at the end of our methods section, but our project only focused on the main stats shown in the box score and a few other defensive stats. This leads to not having enough data relating to all of the Four Factors and the data that we do have could be receiving too much power when it comes to determining which players could help or not help a team. And like we said before, if this project were to actually be incorporated into the decision-making process, we need to make sure it is as accurate and paints as complete of a picture of a player and a team as possible. Stats can be influenced in many ways and be in charge of decisions that impact to these people's lives, [12] so it is our job to continue working on fleshing out the way our program observes team weaknesses along with the way players are suggested so that we have as many factors taken into consideration as possible.

5.3 Accessibility Factors:

Being that our project is a web-app, we must be aware of the many ways we can make it accessible to as many people as possible. One of the first topics of accessibility to focus on in our project is that for users to access it, they would need access to internet usage. The access to internet usage can be taken for granted in today's era of technology, but for many it is not the case. Internet access is a privilege in today's world, thus we must be aware that by creating our program as a web-app, we are limiting the audience of our potential users. As a possible solution, the info on our web-app could possibly be turned into a physical magazine or booklet that summarizes a team's areas of improvement at the end of a season or designated amount of time.

Another area of accessibility our web-app must be aware of is the audience of users that are color blind. Our program uses colors and the intensity of the colors to help the user identify players that are above, at, or below average in their stat rankings. We made sure to use shades of blue and orange/yellow to make sure the difference between the two colors is identifiable even for those that are color blind. If you are interested in finding colors that are color blind friendly or learning more on the science behind color blindness, you can see this website here [1].

Other areas of accessibility we must be aware of are how a web-reader processes our web-app for users that are blind, and currently our web-app is only in English so users of other languages might not be able to understand our program. In the future we would hope to make our program accessible to more languages, but were limited to English due to the time frame

of the project. For purposes of the paper, further details of these areas of accessibility will be omitted, however you can see more on website accessibility for persons with disabilities through this article [13].

6 Code Overview:

6.1 Organization of Code:

Being that our project is a web-app, there are two distinct sections to our project; these two sections are: the backend and the frontend of our web-app. And just like how any other web-app is produced, the backend comes before the frontend.

6.1.1 Backend:

The backend of our web-app is made up of four main python files along with the creation of our SQL database that is used to hold the player and team stats for the web-app. Each of the four python files has a different objective that is crucial to the making of our web-app:

- File 1 (Machine Learning): This file is in charge of the machine learning and KMeans clustering for player roles
- File 2 (Web Scraping Player and Team Data): This file does the web-scraping of the stats that will be shown on the web-app
- File 3 (Adding Player Percentile Ranks): This file assigns each player their percentile rankings within their role for every stat
- File 4 (Player Suggestion): This file observes the Four Factors of a designated team and determines the players that will be suggested for their roster construction

The organization of these files go in this order since each file builds off of the information or role of the previous. If users are interested in the coding side of the files and how it all works, the methods section and GitHub repository of the project are great resources where the code and comments on the code are readily available.

The last part of the backend side of the project comes with the SQL database holding each team's Four Factor stats as well as each of the players on current rosters and their stats. This database is build through DB Browser and csv files made through the python files listed above. A crucial organization detail is that these python files along with the database must be in the same repository as the frontend side of our web-app, which leads us to the organization of our frontend.

6.1.2 Frontend:

The frontend of our web-app is made up of two main sections. The first of the two main sections to our frontend is the python file in charge of using Flask to create the web-app, and the second section are all the html files that hold the format and information listed on each of the web pages for the web-app. It is also important that these files are kept in the same repository as each other, as the python file making the web-app needs access to the html files to properly make each of the web pages for the NBA teams. We also mentioned previously that the backend of the web-app needs to be in the same repository as the frontend; this is because the python file making the web-app also accesses the backend file that suggests players so that it can get the suggested players for each of the designated NBA teams.

The process to how a web page for an NBA team is loaded in the frontend is as follows:

- 1. Python file creating web-app initializes web-page for NBA team.
- 2. The information from our backend database related to the designated NBA team is recorded into variables.
- 3. The file the suggests players is ran for the designated NBA team and returns values for the suggested players.
- 4. Template for the web page is rendered of the team's html page while plugging in the recorded information from our backend database and suggested players.

Again, like the backend of the project, if users are interested in the coding side of the files and how it all works, the methods section and GitHub repository of the project are great resources where the code and comments on the code are readily available.

7 Reproducing Results:

Our section on reproducing the projects results can be found in the readme file of our project's GitHub repository.

8 Conclusion:

To conclude the paper, we were able to accomplish our goal of creating a web-app that is able to identify NBA teams' strengths and weaknesses through the Four Factors model, and based on the teams deficiencies, suggest NBA players that could positively impact the areas of weakness for the designated NBA team. It was through the usage of web-scraping and Flask documentation that we were able to bring the web-app to reality, and we hope to continue working on the project and further improve its features and accuracy of suggestions. Some of the immediate improvements we would like to make are to add more statistics in the backend of the program to further increase the accuracy of our player suggestions, as well as adding more features to the web-app that allows users to adjust filters that would impact the programs player suggestions (give a more interactive feel to our web-app instead of purely just relaying information).

The hope for this project is to one day be a resource used by NBA teams in their decision making process when adding or trading players, or in general making changes to their rosters. We also encourage others who are interested in this project and topic to use our web-app as a form of inspiration for their own projects and possibly improve upon what we have been able to achieve through this project. We hope that readers of this paper, as well as users of our web-app are able to learn as much from this project as we have.

References

[1] Coloring for colorblindness. https://davidmathlogic.com/colorblind/#%23D81B60-%231E88E5-%23FFC107-%23004D40. [Accessed: 2021-12-14].

[2] Ethics in web scraping. https://towardsdatascience.com/ethics-in-web-scraping-b96b18136f01. [Accessed: 2021-12-14].

[3] The four factors of basketball as a measure of success. https://statathlon.com/four-factors-basketball-success/. [Accessed: 2021-12-14].

[4] How different metrics correlate with winning in the nba over 30 years. https://towardsdatascience.com/how-different-metrics-correlate-with-winning-in-the-nba-over-30-years-57219d3d1c8. [Accessed: 2021-12-14].

[5] Medal tally ranking systems. https://www.topendsports.com/events/summer/medal-tally/rankings.htm. [Accessed: 2021-12-14].

[6] Nylon calculus: Grouping players by offensive role, again. https://fansided.com/2019/05/29/nylon-calculus-grouping-players-offensive-role-again/. [Accessed: 2021-12-14].

[7] Using machine learning to classify nba players, part i. https://www.thespax.com/nba/using-machine-learning-to-classify-nba-players-part-i/. [Accessed: 2021-12-14].

[8] Colás, Y. The culture of moving dots: Toward a history of counting and of what counts in basketball. *Journal of Sport History* (2017), 336–49.

[9] Hammond T, Galloreto C, K. H. H. D. H. t. The prevalence of failure-based depression among elite athletes. *Clin J Sport Med* (2013).

[10] Lewis, M. *Moneyball: The Art of Winning an Unfair Game*. W. W. Norton Company, 2004.

[11] Oliver, D. *Basketball on Paper: Rules and Tools for Performance Analysis*. Potomac Books, 2004.

[12] O'Neil, C. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. Crown, 2017.

[13] Simeone, J. Website accessibility and persons with disabilities. *Mental and Physical Disability Law Reporter, vol. 31, no. 4* (2007), pp. 507–11.

[14] Wilson, W. L. *Mathletics: How Gamblers, Managers, and Sports Enthusiasts Use Mathematics in Baseball, Basketball, and Football*. Princeton University Press, 2012.