Executive Summary:

I created a recommender system that takes in a basketball player and lists down the 10 most similar players. I intended this script to be used for fantasy basketball drafting. In fantasy a player of interest could already be taken, so I wanted a script that can recommend only players that are available. I wanted to solve the question "My favorite player got picked, who should I draft in his place?" I created this recommender using data from BasketballReference.com. After scraping the data I created the recommender using the library pairwise distances. I then created the script so I can use input/output to search players and to remove players.

Data Gathering+EDA:

I gathered my data from Basketball Reference. My methodology to scrape the information of the players systematically had 3 steps. After each of these steps I saved the data as a .csv to save myself the trouble of re-scraping the information. First step was to get each team and slugs of their team’s webpage. Then the next step was to take the active roster of each team and get the slugs of each player. The final step was to get the stats of the last 3 seasons of each player and their career stats.

To scrape the active teams and their slugs, I went to basketball reference team page. I made a for loop that scraped each element that had a <th> tag. The <th> tag contained the current name of the team and their slug.

For the second step I made a for loop that used the slugs gathered from the previous step, and went to the roster page for the upcoming season. It then scraped the name of the player, the team they are on, the slug to get to the players page, their position, and the number of years of experience. I needed to create a try/except for this function. If an error was raised, I had the loop print out the team which caused the error. The teams that caused this error were the Brooklyn Nets, Charlotte Hornets, and New Orleans Pelicans. This makes sense because these three teams recently went through rebranding. The Brooklyn Nets were previously the New Jersey Nets until 2013. The Charlotte Hornets were the Charlotte Bobcats until 2015. The New Orleans Pelicans were the New Orleans Hornets until 2014. To remedy the issue, I input the team roster link directly into my loop for the three teams. I combined the 4 different scrapes (the successful scrape and the 3 teams) into one dataframe. This gave me a dataframe with all active players.

The second step gave me a dataframe with with each player and their years of experience. Rookies were listed with an experience of “R”. Using this, I filtered these rookies from the list. Rookies do not have any professional experience so I decided not to include them in this recommender. I did not want to try to interpolate a rookies professional statistics from their collegate careers. If professional scouts could do that already there would never be any busts. After filtering out the rookies, I could then perform step 3.

For step 3 I created a for loop to go to the webpage of each player and grab their box scores for the last three years and their career stats. These boxscores included like games played, games started, three pointers attempted, three point percentages, blocks, boards, assists, etc. I had to create an if statement for these scrapes since the players stats can be grouped in these 3 categories: 3 or more years of experience, 2 years of experience, or 1 year of experience. I made two try/except statements. The first one checks if I am able to go to the players page at all. This catch would occur if i was accessing the website too much. The catch would print the index of where the error occurred so I can re-attempt to scrape them by the index at another time. I had a second try/except in the scraping of the boxscores. Occasionally a player is traded mid season and has two rows corresponding to one season. The try/except catches that situation. I created another if statement to check if the value of “3pt percentage” was empty. If the player did not attempt any threes the cell would be blank yet it would not show up as a null. This caused problems with the ordering of my cells. IE: data would be shifted one column over. The if statement imputes a value in case “3pt percentage” was empty. I needed to scrape this data multiple times since I was denied access to Basketball Reference a couple of times. I saved the data in two dataframes. One dataframe of the career data of each players and one dataframe of the last 3 seasons they had.

I ran into a number of issues after these steps. I forgot to attach the positions on the box score dataframes. I used the step 2 data to help impute the values for each player. Some players were listed as multiple positions like “F-C”,”C-F” or “G-F”,”F-G”. I simplified the positions as taking the first position they were listed as, as their primary position. IE F-C to F. To make these positions usable in the recommender, they needed to be numerical. I used the naming convention to translate them to numbers. Point guards are 1, shooting guards are 2, small forwards are 3, power forwards are 4, and centers are 5. Since they are only listed as guard, forward, or center I renamed them 1,3, and 5. Some players had null values since they played very little minutes. I imputed them as 0. I made sure that I did not have any duplicates. One person was scraped twice accidentally.

For me to combine the career dataframe and 3 season stats dataframe for the recommender, I needed to make them the same size. The shape of the career dataframe was 370 by 27. The shape of the 3 years dataframe was 908 by 27. I had to aggregate the last 3 seasons of each player and create a new dataframe. With the two sets of data the same size, I can make the recommendation system.

Recommender:

I first built the recommender system for the 3 season averages. The process was fairly straightforward. I used pairwise distances from the sklearn library metrics. I created a pairwise distances instance with a cosine distance metric to create the recommender dataframe. I had two versions, an unscaled version and a scaled version. The scaled version passed the eye test better. The unscaled version would recommend players with extremely different playstyles and focuses. It would put results that are unexpected with players that would not be normally compared to one another. The scaled version fits the searched players much better. I then built the recommender system for the career data frame. The career data frame was strange in the fact that some values were above 1. Since interpretation of those values do not affect my findings in a significant way they were left alone. I was then able to add the two data frames together after scaling them. One unintended consequence of combining the data frames was that games played and similar stats that were total counts were weighted a bit stronger since there was a much wider range for those. This lead to young players being grouped with young players and older players with old players. I thought it was useful to have.

I compared my recommender with FiveThirtyEights similar feature, CARMELO. CARMELO’s library of players is much more vast than the recommender I created. It would list players who have not been in the league for years and by what individual seasons the player is similar to. For MVP caliber players my recommender does not match theirs since they mainly list players who are not on my active player roster. It appears to work better with young players. Lonzo Ball for example has 3 matches: Marcus Smart, Elfrid Payton, and D’Angelo Russell. Marcus Smart is rank 1 on FiveThirtyEight and ranked 6 on mine. Payton is rank 2 on FiveThirtyEight and rank 5 on mine. Russell is rank 8 on FiveThirtyEight and rank 9 on mine.

Script:

I wanted to create a user friendly way to interact with the recommender. The main functions I wanted are: a search function where one can type in a players name and 10 similar players will be returned from the available player pool and a remove function where you can remove players from the available. The recommender first makes a copy of the master recommender to safeguard the master from changes. I created a function that acts as the main menu. It takes a query from the user to determine what option in the menu is chosen. Each menu option is its own function. The main menu can take 6 main options. Option 1 is the search feature. Option 2 is the remove function that removes a player from the pool. Option 3 adds players back into the pool. Option 4 prints the removed players. Option 5 prints the caveats of the recommender. The last option quits out of the system. Every time the user inputs an invalid option in any of the menus, I make the system print out a message and wait 1 second. I did this to give the user the opportunity to see what they did wrong prior booting them back to a menu screen.

The search feature takes a players name as a direct input. It searches a dataframe of names to see if the query is in the system. If the user inputs 9 the recommender goes back to the main menu. If there are no matches the function prints a suggestion to type out the name exactly as it appears on Basketball Reference. If there is one match, this could mean two things. They either spelled the name correctly and thus can properly have the similar players displayed or they did not spell the name fully correctly but the search returns one match. I had a try except statement to take care of these situations. If the name was not spelled fully, a recommended spelling will appear. What happens when there is a spelling match will be discussed in the next paragraph. If the query results with more than one match then that means multiple players can be called with that query. The function would display players that contain the users query. An invalid query will return the user back to the search prompt. After the result of the query I have the user prompt if they want to go back to the main menu or not. In this function I incorporated an easter egg. If the user types “jingles”, an image would open on the screen and return the most similar players to Joe Ingles, a player for the Utah Jazz. I specifically imported the library PIL to do this.

When the query is one player and it matches the spelling correctly, I have the system print a header I created which shows one column is player names and the other column is the similarities. It then takes the query and goes to the column in the recommender dataframe of that query. The dataframe then drops the rows of the players that have been added to the remove player list. Then the top 11 results are shown. I use the slice of the first 11 elements since first player will always be the player itself if no one is removed. Thus with top 11, the top 10 most similar are shown.

The remove feature takes a players name as a direct input. If the user types 9, the recommender goes back to the main menu. If the user types 1, it would print the list of removed players. It uses the same methods to look for players as the search feature. When there is a match of size 1 two results could happen. It would add the player to the removed player list or print the recommended spelling of the player. To do this I used a try except. The try would add the player to the removed players list. The except would print the suggested spelling of the query. I used the same feature used in the search feature to force the error. If there is a partial match the recommender would list the players the user could have meant. If there is no match the recommender prompts the user to spell the name of the person of interest correctly. After the result of the query I prompt the user if they want to continue removing players.

The add feature adds a player back into the player pool. If the user inputs 1, the recommender prints the list of removed players. If the user inputs 9, the recommender goes back to the main menu. If the input matches a value in the removed player list, the input query would be removed from the removed player list. If the input does not match a value, the recommender will print that the query is not in the removed players list. After the result of the query the recommender prompts if the user wants to continue adding players or to go back to the menu.

The print removed player option does exactly that. It prints the players in the removed list. If the list is empty it prints “List is Empty”

The last function prints out the caveats of the recommender. It prints that it the recommender only works for players who have been on teams prior mid-July 2019. It also prints that there are no retired players nor rookies in the system.

Next Steps:

This recommender is hopefully one part of many for future Fantasy Basketball applications. I would like to add rookies to future iterations of this recommender. Hopefully one day I will be able to predict the progression/regression of players as they age. One of my future goals is to make a model that predicts the fantasy scores. Then I would use that model be able to find who are the best players available to draft.